



ÉCOLE CENTRALE LYON

MOD 11.1 : DATABASE SYSTEMS
REPORT

Data Management : Blue Jeans Factory

Students :
Neirouz BOUCHAIRA
Jeanne CATY

Supervisors :
Mohsen ARDABILIAN
Daniel MULLER

Contents

1	Introduction	2
2	Database Creation	2
2.1	Main Database - Company.db	2
2.1.1	Entity-Relationship Diagram	2
2.1.2	Definition of each table and integrity constraints	3
2.2	Access control - acces_control.db	6
3	Data generation	7
3.1	Employee account creation	7
3.2	Badge system	7
3.3	Python code for data generation	7
3.3.1	Principle	8
3.3.2	Algorithm - code features	8
3.4	Python code for calculating salaries per week <i>LOG_SEMAINE</i>	9
3.4.1	Algorithm and main functions	9
4	Requested views	10
5	Application interface	11
5.1	Streamlit	11
5.2	Access control	11
5.3	Application Features	12
5.3.1	Salary Tracking	12
5.3.2	Employees +3 Years of Seniority	13
5.3.3	Productivity	13
5.3.4	Bonuses	14
5.4	Creating an Employee Account	14
5.5	Adding views to the application	14
6	Conclusion	19
A	Calculation of salaries by week	20
B	Streamlit Application - Employee Management	22
C	Streamlit Application - Badging	28

1 Introduction

Cotton Blue is a jeans manufacturing plant. To make a pair of jeans, employees have to carry out three activities or production phases: cutting the back pieces, cutting the front pieces and assembling them.

In this factory, there are two types of employee. Each employee is identified by a numerical code. There are “production” employees and “workday” employees:

- Employees who work “in production” have an associated activity and a number of parts to produce.
- Employees who work “by the day” have an associated salary.

The working day is 9 hours from Monday to Friday and 3 hours on Saturday. On Saturdays, the working day is divided into three (three phases of three hours each).

Any employee arriving late at the plant will have his or her weekly wage cut by \$5. Daily” workers will suffer this reduction by the hour and up to a full day if they don’t show up for work. Production” workers will take this cut according to the number of pieces they fail to produce.

Employees will receive a double pay bonus if they work more during the week. In the case of “production” workers, this will depend on the number of pieces produced. For “daily” workers, it will depend on the number of overtime hours worked.

For each employee, we need to keep a record of the date he or she joined the company. Each time an employee reaches one year with the company, he or she will receive an extra week’s pay. Every employee who has been with the company for a year or more will receive an extra month’s salary as a Christmas present. Employees with less than a year with the company will receive the proportional part of this bonus.

2 Database Creation

2.1 Main Database - Company.db

2.1.1 Entity-Relationship Diagram

This database is made up of 7 tables (figure 1). The *EMPLOYEE* table groups all fixed information for each employee. In the case of an employee who leaves the company and returns, this employee will be assigned a new *NUMERO_EMPLOYE*.

The *GRILLE_SALAIRE_PROD* and *GRILLE_SALAIRE_HORAIRE* tables group together the various hourly or piecework salaries. Each employee is assigned an hourly or piecework wage code which, linked to these tables, defines his or her wage.

The *LOG_JOURNEE_JOUR* and *LOG_JOURNEE_PROD* tables indicate, for each day and for each employee, the work done per hour or per piece, as well as any late work or overtime.

The *SALAIREsEMAIN* table shows the total weekly salary per employee, plus any annual or Christmas bonuses.

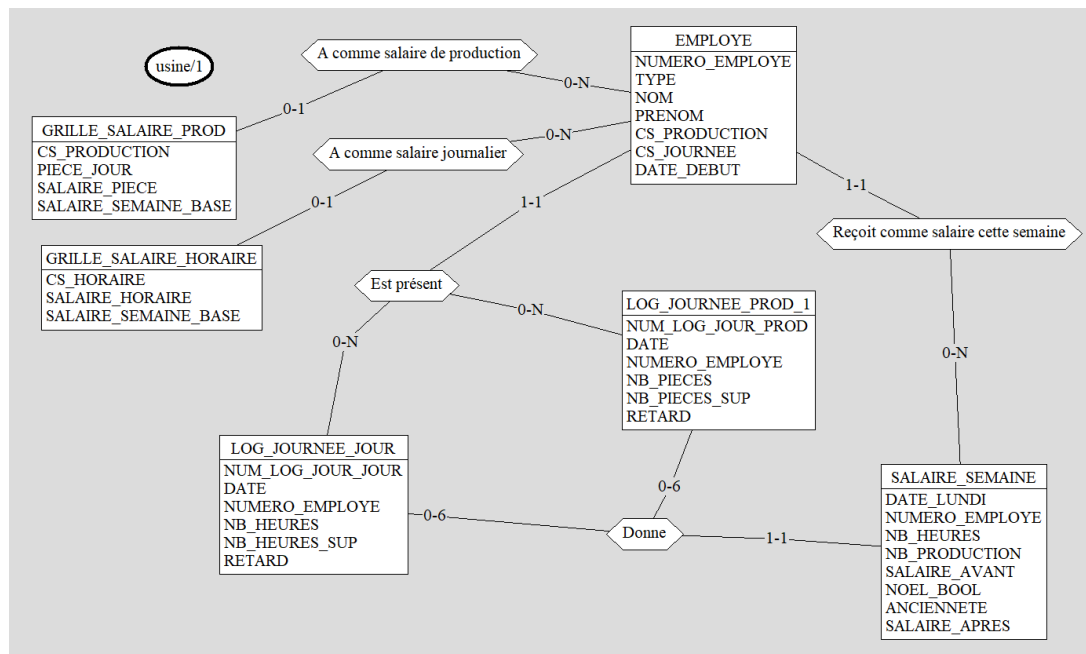


Figure 1: Entity-Association diagram of the database

2.1.2 Definition of each table and integrity constraints

Table *EMPLOYE*

```
CREATE TABLE EMPLOYE (
  NUMERO_EMPLOYE INT,
  TYPE INT CHECK (TYPE IN (1, 2) ),
  NOM CHAR (50) NOT NULL,
  PRENOM CHAR (50) NOT NULL,
  CS_PRODUCTION INT CHECK ( ("type" = 2 AND
                             "CS_PRODUCTION" IS NOT NULL) OR
                             ("type" = 1 AND
                             "CS_PRODUCTION" IS NULL) ),
  CS_JOURNEE INT CHECK ( ("type" = 1 AND
                          "CS_JOURNEE" IS NOT NULL) OR
                          ("type" = 2 AND
                          "CS_JOURNEE" IS NULL) ),
  DATE_DEBUT DATE NOT NULL,
  DATE_FIN DATE,
  PRIMARY KEY (
    NUMERO_EMPLOYE
  ),
  FOREIGN KEY (
    CS_JOURNEE
  )
  REFERENCES GRILLE_SALAIRE_HORAIRE,
  FOREIGN KEY (
    CS_PRODUCTION
  )
  REFERENCES GRILLE_SALAIRE_PROD);
```

In this table, the primary key is *NUMERO_EMPLOYE*, and is automatically auto-incremented. The TYPE is 1 if the employee is paid by the hour and 2 if paid by the

piece. If the employee is paid by the hour, he has a day wage code (*CS_JOURNEE*), otherwise he has a production wage code (*CS_PRODUCTION*). We have therefore implemented an integrity constraint to link the *TYPE* and the wage codes. Salary codes are foreign keys, as they are the primary keys of the two salary grid tables.

GRILLE_SALAIRE_PROD

```
CREATE TABLE GRILLE_SALAIRE_HORAIRE (
  CS_HORAIRE INTEGER PRIMARY KEY ASC AUTOINCREMENT,
  SALAIRE_HORAIRE FLOAT CHECK (typeof(SALAIRE_HORAIRE) IN ('integer', 'real'))
    NOT NULL,
  DEBUT_SAMEDI NOT NULL
    CHECK (DEBUT_SAMEDI IN (8, 11, 14)),
  SALAIRE_SEMAINE_BASE AS (SALAIRE_HORAIRE * (9 * 5 + 3));
```

The primary key of this table is the hourly wage code, which is self-creating. The hourly wage can be a FLOAT or an INT. *DEBUT_SAMEDI* corresponds to the Saturday start time, which can be 8am, 11am or 2pm. The *SALAIRE_SEMAINE_BASE* is automatically calculated for 9h work on weekdays and 3h work on Saturdays.

GRILLE_SALAIRE_HORAIRE

```
CREATE TABLE GRILLE_SALAIRE_PROD (
  CS_PRODUCTION INTEGER PRIMARY KEY ASC AUTOINCREMENT,
  PIECE_JOUR INT CHECK (typeof(PIECE_JOUR) IN ('integer', 'real') AND
    PIECE_JOUR > 0)
    NOT NULL,
  SALAIRE_PIECE FLOAT CHECK (typeof(SALAIRE_PIECE) IN ('integer', 'real') AND
    SALAIRE_PIECE > 0)
    NOT NULL,
  DEBUT_SAMEDI INTEGER NOT NULL
    CHECK (DEBUT_SAMEDI IN (8, 11, 14)),
  SALAIRE_SEMAINE_BASE AS (PIECE_JOUR * SALAIRE_PIECE * (5 + 1 / 3));
```

The primary key of this table is the hourly wage code, which is self-creating. The hourly wage can be a FLOAT or an INT. *DEBUT_SAMEDI* corresponds to the Saturday start time, which can be 8am, 11am or 2pm. The *SALAIRE_SEMAINE_BASE* is automatically calculated for 9h work on weekdays and 3h work on Saturdays.

The primary key of this table is the production wage code, which is self-creating. The piecework wage can be a FLOAT or an INT and must be greater than 0. *PIECE_JOUR* corresponds to the number of pieces to be produced per day. This must be an integer or a float and must be greater than 0. *DEBUT_SAMEDI* is the Saturday start time, which can be 8am, 11am or 2pm. The *SALAIRE_SEMAINE_BASE* is automatically calculated for *PIECE_JOUR* produced on weekdays and *PIECE_JOUR*/3 produced on Saturdays, as workers work 9 hours on weekdays and 3 hours on Saturdays.

LOG_JOURNEE_PROD

```
CREATE TABLE LOG_JOURNEE_PROD (
  NUM_LOG_JOUR_PROD INT PRIMARY KEY,
  DATE DATE NOT NULL,
```

```

WEEK_DAY INTEGER CHECK (WEEK_DAY <= 6 AND
                        WEEK_DAY >= 0)
                        NOT NULL,
NUMERO_EMPLOYE INT NOT NULL
                        CHECK (NUMERO_EMPLOYE >= 0),
NB_PIECES INT CHECK (NB_PIECES >= 0)
                        NOT NULL,
NB_PIECES_SUP INT CHECK (NB_PIECES_SUP >= 0)
                        NOT NULL,
HEURE_ARRIVEE NOT NULL,
RETARD BOOLEAN NOT NULL,
FOREIGN KEY (
    NUMERO_EMPLOYE)
REFERENCES EMPLOYE (NUMERO_EMPLOYE) );

```

NUM_LOG_JOUR_PROD is the primary key of this table. The date and employee number are mandatory. *WEEK_DAY* corresponds to the day of the week. 0 corresponds to Monday and 6 to Sunday. The number of pieces and the number of extra pieces correspond to the production of pieces up to the expected minimum, then to the extra pieces which are paid at double. Both numbers must be greater than 0. The arrival time is used to check whether the employee has been late or not. *RETARD* is a Boolean if the arrival time is after 8 a.m. on weekdays and after the start time on Saturdays.

LOG_JOURNEE_JOUR

```

CREATE TABLE LOG_JOURNEE_JOUR (
    NUM_LOG_JOUR_JOUR INT,
    DATE DATE NOT NULL,
    WEEK_DAY INTEGER CHECK (WEEK_DAY >= 0 AND
                          WEEK_DAY <= 6)
                          NOT NULL,
    NUMERO_EMPLOYE INT NOT NULL,
    HEURE_ARRIVEE NOT NULL,
    HEURE_FIN NOT NULL,
    NB_HEURES NUMERIC CHECK (NB_HEURES >= 0 AND
                          NB_HEURES <= 9)
                          NOT NULL,
    NB_HEURES_SUP NUMERIC CHECK (NB_HEURES_SUP >= 0)
                          NOT NULL,
    RETARD BOOLEAN NOT NULL,
    PRIMARY KEY (
        NUM_LOG_JOUR_JOUR),
    FOREIGN KEY (
        NUMERO_EMPLOYE)
REFERENCES EMPLOYE (NUMERO_EMPLOYE) );

```

NUM_LOG_DAY is the primary key of this table. The date and employee number are mandatory. *WEEK_DAY* corresponds to the day of the week. 0 corresponds to Monday and 6 to Sunday. The number of hours and the number of overtime hours correspond to hours worked up to the expected minimum, then to overtime hours which are paid at double time. Both numbers must be greater than 0. The arrival time is used to check whether the employee has been late or not. *RETARD* is a Boolean if the arrival time is after 8 a.m. on weekdays and after the start time on Saturdays. The end time is used with the end time to calculate the number of hours worked per day.

SALAIRE_SEMAINE

```
CREATE TABLE SALAIRE_SEMAINE (
  DATE_LUNDI DATE,
  NUMERO_EMPLOYE INT,
  NB_HEURES INT CHECK (NB_HEURES >= 0),
  NB_PRODUCTION INT CHECK (NB_PRODUCTION >= 0 AND
    ( (NB_PRODUCTION IS NULL AND
      NB_HEURES IS NOT NULL) OR
      (NB_PRODUCTION IS NOT NULL AND
      NB_HEURES IS NULL) ) ),
  SALAIRE_AVANT FLOAT NOT NULL
    CHECK (SALAIRE_AVANT >= 0),
  NOEL BOOLEAN NOT NULL,
  ANCIENNETE NOT NULL
    CHECK (ANCIENNETE >= 0),
  SALAIRE_APRES FLOAT CHECK (SALAIRE_APRES >= SALAIRE_AVANT)
    NOT NULL,
  PRIMARY KEY (
    DATE_LUNDI,
    NUMERO_EMPLOYE),
  FOREIGN KEY (
    NUMERO_EMPLOYE)
  REFERENCES EMPLOYE (NUMERO_EMPLOYE) );
```

The primary key of this table is the pair *DATE_LUNDI* and *NUMERO_EMPLOYE*. *NUMERO_EMPLOYE* is a foreign key of the *EMPLOYE* table. *NB_HEURES* and *NB_PRODUCTION* correspond to the hours worked and the number of pieces produced during a week. These two numbers are therefore positive, and as an employee cannot work by the piece or by the day, one of them is NULL. *SALARY_BEFOR* corresponds to salary without Christmas or birthday bonuses. *NOEL_BOOLEAN* is a non-zero Boolean when the week in question is Christmas. Seniority is a float corresponding to the number of years worked on the Monday date. *SALARY_APRES* corresponds to the full salary with bonuses.

2.2 Access control - acces_control.db

A SQLite database has been set up to manage access control, based on employee roles. By creating an “employees” table with the columns “employee_code” (primary key), “password” and “status”, we define a structure for storing user identification information and privileges. The constraint CHECK(status IN ('HR', 'admin', 'employee')) ensures that each employee is associated with a predefined role: HR, administrator or standard employee.

This approach to role-based access control (RBAC) simplifies permissions management by grouping users by function and assigning them rights specific to each role. For example, an administrator has access to all data, an employee **HR** has information relating to human resources (**our application in this case**), while a standard employee would have limited access. This structure enables efficient, centralized management of access rights, improving data security and organization.

```
CREATE TABLE IF NOT EXISTS employees (
  employee_code TEXT PRIMARY KEY,
  mot_de_passe TEXT NOT NULL,
```

```
status TEXT CHECK(status IN ('RH', 'admin', 'employee')) NOT NULL)
```

3 Data generation

3.1 Employee account creation

The procedures in place for managing elementary modification operations (insertion, deletion and modification) are as follows:

1. Inserting a line

A new line is inserted into the EMPLOYEE table via a dedicated form accessible from the user interface. This action triggers the following operations:

- **Employee table:** The new entry is added to the EMPLOYEE table using data supplied via the form.
- **Log tables :** A record is created in the log table, tracing the insertion. This record is associated with the employee performing the operation via his or her badge. This traceability enables precise tracking of actions and identification of responsibilities.
- **Weekly wage table:** Updating this table should be automated by a cron job d'u code (appendix A) in theory. The latter periodically executes a wage calculation code, based on the information contained in the logs. This automation guarantees consistency between employee data and salary calculations.

2. Deleting and modifying a row:

Deleting and modifying a row in the EMPLOYEE table should be subject to access restrictions. Only an administrator has the necessary rights to perform these actions. The user interface does not allow standard users to directly delete or modify database entries. This part is therefore not in the HR application.

3.2 Badge system

Our database model is based on the principle that employees would have a personal badge which they would swipe through a machine as they entered and left the factory to indicate their arrival and departure times. Since we don't have such a system on hand, we've set up a tab in the interface where an employee can select himself and "badge". The actual system would be much more secure, as it could not be accessed from outside the factory, and a colleague could not badge in place of another because of the personal badge.

A badging simulation interface (figure ??) was created by Streamlit via the code in appendix C.

3.3 Python code for data generation

To have a lot of data, the badging system requires each employee to be badged twice a day. This is not viable for us to have enough data. So a python code was created to automate the creation of daily work data.

Machine de Badgeage

Passez une bonne journée!

Numéro Employé

2

Badge In

Badge Out

Figure 2: Interface de badgeage

3.3.1 Principe

This code takes the list of employees in the *EMPLOYEE* table with their number and start date. For an employee, starting from their date of employment, it creates a line for each day of the week except Sunday in either *LOG_JOURNEE_JOUR* if the employee works by the hour, or *LOG_JOURNEE_PROD* if the employee works by the piece, arriving on time in the morning and working either by 9 a.m. or the minimum number of pieces required.

A line with a delay is then added by hand to test this borderline case.

3.3.2 Algorithm - code features

1- Connection and initialization:

- The code connects to the “company.db” database using sqlite3.- A cursor object is created to execute SQL queries.

2- Data retrieval:

- It retrieves all employee information (including employee number, production/daily code, start date) from the “EMPLOYEE” table. - It also retrieves the highest existing log numbers for production and daily logs.

3- Log generation loop :

- The code iterates over each employee in the retrieved list. - For each employee, it determines the start date and calculates the total number of days worked to date. According to the employee’s category (production or day laborer)

- Daily employees :

- o It retrieves the Saturday start time from the “GRILLE_SALAIRE_HORAIRE” table according to the employee’s daily work code. o It iterates over all days worked (including the start date and today):

- Excludes Saturdays and Sundays from classic log generation.
- For weekdays, inserts a new log entry in the “LOG_JOURNEE_JOUR” table and assigns the predefined number of working hours (assumed to be 9 hours according to the statement) and the actual working hours (identical to the scheduled hours).
- The arrival and scheduled arrival times are identical (8h).
- The “DELAY” flag is set to False (no delay is simulated).

- Production employees :
 - It retrieves the predefined number of pieces per day and the Saturday start time from the “GRILLE_SALAIRE_PROD” table according to the employee’s production code.
 - It iterates over all days worked (including start date and today):
 - Excludes Saturdays and Sundays from classic log generation.
 - For weekdays, inserts a new log entry in the “LOG_JOURNEE_PROD” table and assigns the predefined number of pieces as planned and actual production.
 - Arrival and scheduled arrival times are identical (8h).
 - The “DELAY” flag is set to False (no delay is simulated).

4- Validating changes : After iterating over all employees, the code validates the log data inserted in the database using `conn.commit()`.

3.4 Python code for calculating salaries per week *LOG_SEMAINE*

This script is designed to calculate employees’ weekly wages and record them in the *LOGWEEK* database according to their contract type and days worked. It begins by retrieving employee data and work logs. Next, it determines the earliest date in the logs and the number of days between that date and today’s date. The script then calculates the weekly wages for all Mondays between the oldest date and today’s date. Employees are separated into hourly and piece-rate contracts. For hourly employees, we use the hourly wage from the pay scale, checking the days and hours worked during the week. This gives the salary without bonuses. Christmas and seniority bonuses are then added, giving the final salary. The process is similar for piecework contracts, except that the hourly wage is replaced by the piecework wage. The wage without bonus is calculated according to the pieces produced. Bonuses are then added. The Christmas bonus is added when one of the days of the week is December 25, and is calculated according to seniority, capped at one month for employees with more than one year’s seniority. Employees with more than one year’s seniority receive a bonus on the anniversary week of their start of employment.

3.4.1 Algorithm and main functions

Step 1: Connection and data recovery

- Connect to SQLite database.
- Retrieve employee data (*EMPLOYEE*) and daily logs (*LOG_JOURNEE_JOUR*, *LOG_JOURNEE_PROD*).
- Identification of the earliest date in the daily logs to determine the analysis period.

Step 2: Iteration by date and selection of Mondays

- The loop traverses each day between the earliest date and today.
- Calculations are made for Mondays only.

Step 3: Weekly data calculation

- For each employee, according to category (*CS_JOURNEE* or *CS_PRODUCTION*):
 - Data for the 6 days preceding Monday are retrieved and analyzed.
 - Salaries are calculated as follows:
 - * **For hourly employees:**
 - Base salary = hours worked \times hourly rate.
 - Total salary = base salary + overtime \times hourly rate – late penalties.
 - Christmas bonus and seniority bonus added if applicable.
 - * **For production employees:**
 - Base salary = pieces produced \times rate per piece.
 - Total wage = base wage + additional pieces \times rate per piece – penalties for late work.
 - Christmas bonus and seniority bonus added if applicable.
 - A line is inserted in the *WEEKLY_SALARY* table if it does not already exist.

Step 4: Premium management

- **Christmas bonus:**
 - Calculated for Christmas Day (December 25, 2025).
 - Based on proportionality to time worked and a defined ceiling (4.5 times (4.5 weeks in a month) the hourly or piecework wage for a day's work).
- **Seniority bonus:**
 - Added if the employee has worked for more than one year and if the date corresponds to the anniversary of his/her employment.

Step 5: Validate and save data

- The calculated data is saved to the database via *conn.commit()*.

4 Requested views

The requested views are :

1. List of each employee's weekly salary
2. List of daily employees with more than 3 years seniority
3. List of workers who have had wage cuts during a week, each week
4. List for a worker, hours worked or parts produced per week
5. List of total number of employees who will receive an annual or Christmas bonus
6. List of the total number of employees who will receive only the proportional part of the Christmas bonus

7. List of the number of employees receiving their annual bonus per week

The first view is a join of the *SALAIRE_SEMAINE* table with the *EMPLOYEE* table by employee number, keeping only the columns for employee number, surname, first name, Monday date and salary after bonus.

The second view is a projection of the *EMPLOYEE* table for which the date difference between the current date and the start date is more than 3 years.

The third view is a restriction of the *SALAIRE_SEMAINE* table when salary before bonus is less than base salary, and a projection of this table to leave only the employee number and Monday date.

The fourth view is also a projection of the *SALAIRE_SEMAINE* table, retaining only the employee number, Monday date, number of hours and number of parts produced.

The last three views are views of future bonuses, whether partial or total Christmas bonuses, or birthday bonuses. We've taken one year as the duration to visualize bonuses in the future. We define the list of Mondays for one year, and test for each Monday and each employee whether the week includes Christmas or the employee's anniversary.

5 Application interface

5.1 Streamlit

For the user interface, a Streamlit application has been developed (code in appendix B) to simplify the management of a company's employee information. It offers an interactive interface for consulting salaries, seniority, productivity, bonuses and creating new employee accounts, akin to a human resources platform. The application uses a SQLite database for data storage, and relies on Python libraries such as Pandas, Matplotlib and Datetime for data processing and visualization. Streamlit is an open source Python library that greatly simplifies the creation and sharing of interactive web applications, particularly for data science and machine learning. Its main strength lies in its ability to transform Python scripts into functional web applications with a minimum of code, making it particularly accessible to data scientists and developers who are not necessarily experts in web development.

5.2 Access control

The access control implemented in this Streamlit application uses a SQLite database to authenticate users. A login form prompts users to enter their employee code and password (figure 3). When the form is submitted, a SQL query is executed to check that the identifiers match in the "employees" table of the "access_control.db" database. Specifically, the query `SELECT status FROM employees WHERE employee_code = ? AND mot_de_passe = ?` searches for records where the employee code and password match the values entered.

If a match is found and the "status" field of this record is equal to **'RH'** or **'admin'**, access is authorized: a session variable `st.session_state.logged_in` is set to `True` and the status is stored in `st.session_state.status`, allowing protected content to be displayed. Otherwise, an "Access denied" error message is displayed. The use of session variables (`st.session_state`) enables the user's connection state to be maintained between different interactions with the application. Once the verification is complete, the database connection is closed with `conn.close()`.

Page de gestion des employés

Bienvenue! 😊

Login

Code Employé

Mot de passe

Login

Figure 3: Page de connexion

→ It is important to note that, for security reasons, storing passwords in clear text in the database is strongly discouraged. It would be preferable to use a cryptographic hash function to store passwords securely, which unfortunately was not done in this project.

```
if not st.session_state.logged_in:
    # Create a login form
    st.header("Login")
    employee_code = st.text_input("Code_Employ")
    mot_de_passe = st.text_input("Mot_de_passe", type="password")
    login_button = st.button("Login")

    if login_button:
        cursor.execute("SELECT status FROM employees WHERE employee_code=? AND mot_de_passe=?", (employee_code, mot_de_passe))
        result = cursor.fetchone()
        if result and (result[0] == 'RH' or result[0] == 'admin'):
            st.session_state.logged_in = True
            st.session_state.status = result[0]
        else:
            st.error("Accès refusé. Veuillez vérifier vos informations de connexion.")

conn.close()
```

Voici une traduction en anglais des descriptions des fonctionnalités de l'application, en conservant les noms des variables et des modules/fonctions tels quels :

5.3 Application Features

5.3.1 Salary Tracking

This tab allows visualizing employee salaries per week. It offers filtering options to display either all salaries, or only those that have experienced a decrease compared to the base salary.

Modules and functions:

- streamlit (st) : Used for the user interface (displaying the title, filters with st.popover and st.checkbox, and the DataFrame with st.dataframe).
- sqlite3 : Used for the connection to the SQLite database (sqlite3.connect).

- pandas (pd) : Used for manipulating data extracted from the database and displaying it as a DataFrame (pd.DataFrame). The selection of data to display salary decreases is done with Pandas boolean filtering (`df[df['Salaire avant primes'] < df['Salaire base']`)).
- SQL Query : A SELECT query with JOIN is executed to retrieve salary data and employee information.

Algorithm: Reading data from the database, creating a Pandas DataFrame, applying optional filters, displaying the DataFrame.

5.3.2 Employees +3 Years of Seniority

This tab displays the list of employees with more than three years of seniority. It distinguishes daily employees from production employees. A toggle allows displaying either only employees with more than 3 years of seniority, or the complete list of employees.

Modules and functions:

- streamlit (st) : Used for displaying the title, the toggle (st.toggle) and the DataFrames.
- sqlite3 : Used for the connection to the database.
- pandas (pd) : Used for data manipulation and DataFrame creation.
- datetime : Used for seniority calculation (datetime.today(), date conversion with pd.to_datetime and calculation of the difference in days with dt.days).

Algorithm: Reading data from the database, converting the start date to datetime format, calculating seniority in days, filtering employees with more than 3 years of seniority, displaying DataFrames filtered by contract type.

5.3.3 Productivity

This tab allows visualizing the productivity of a selected employee. For daily employees, the number of hours worked per week is displayed, while for production employees, it is the number of pieces produced. A matplotlib graph displays the productivity evolution over the last 10 weeks, with an indication of the average.

Modules and functions:

- streamlit (st) : Used for employee selection (st.selectbox), displaying the DataFrame and the graph (st.pyplot).
- sqlite3 : Used for the connection to the database.
- pandas (pd) : Used for data manipulation and DataFrame creation.
- matplotlib.pyplot (plt) : Used for graph creation.
- datetime : Used for date management.

Algorithm: Reading data from the database, selecting an employee, performing calculations based on the employee's contract type, generating and displaying a matplotlib graph.

5.3.4 Bonuses

This tab calculates and displays Christmas bonuses (full and partial) and birthday bonuses for the upcoming year. The user can choose to display each bonus type separately.

Modules and functions:

- streamlit (st) : Used for displaying the title and DataFrames, and the checkboxes for selecting bonus types (st.checkbox).
- sqlite3 : Used for the connection to the database.
- pandas (pd) : Used for the creation of the final DataFrame.
- datetime : Used for date management and bonus calculation.

Algorithm: 1- Generating a list of Mondays for the upcoming year 2- Reading employee information from the database 3- Calculating Christmas bonuses (full and partial) and birthday bonuses based on the employee's start date and salary grids 4- Storing the results in a list and creating a Pandas DataFrame then displaying the DataFrames filtered by selected bonus type

→ Important point: The bonus calculation algorithm uses a nested loop for each employee and each Monday, which can be optimized for better performance with a large number of employees.

5.4 Creating an Employee Account

Modules and functions:

- streamlit (st) : Used for the input form (st.text_input, st.selectbox, st.number_input, st.date_input) and the submission button (st.button).
- sqlite3 : Used for the connection to the database and the execution of SELECT (to check for employee existence) and INSERT (to add the new employee) queries.
- datetime : Used for managing the start date.

Algorithm: Retrieving the maximum employee number, incrementing it for the new employee, entering employee information via a form, checking for the existence of an employee with the same information, inserting the new employee into the database.

5.5 Adding views to the application

The application allows you to see all the requested views.

Création de compte employé

Nom

Prénom

Type de contrat

Journalier

Code salarial journalier

1

date de début

2024/11/13

Create Account

Figure 4: Interface page for creating an employee account

Liste de la totalité des employés

Liste des employés journaliers

	Numéro employé	Nom	Prénom	Type du contrat	Date de début
0	1	Dubois	Michel	1	2020-11-13 00:00:00
1	2	Bouchaira	Neirouz	1	2021-11-13 00:00:00
4	6	Bowers	Matthew	1	2024-11-20 00:00:00
7	8	Noullant	Noura	1	2023-08-16 00:00:00

Liste des employés par production

	Numéro employé	Nom	Prénom	Type du contrat	Date de début
2	3	Caty	Jeanne	2	2019-11-03 00:00:00
3	5	Mourin	Julie	2	2023-11-13 00:00:00
5	7	Jura	Charlie	2	2022-11-28 00:00:00
6	4	Dupont	Natalie	2	2022-02-20 00:00:00

Figure 5: Interface page for viewing all employees (view 2)

[Suivi des salaires](#)
[Employés +3 ans d'ancienneté](#)
[Productivité](#)
[Primes](#)
[Création d'un compte emp](#)

Suivi Employés

Filtre employés à afficher ▼

Salaires des employés par semaine (question 1)

	Date du lundi	Numéro employé	Nom	Prénom	Salaire base	Salaire avant primes	Salaire
0	2020-11-09 00:00:00	1	Dubois	Michel	120	110	
1	2020-11-09 00:00:00	3	Caty	Jeanne	990	990	
2	2020-11-16 00:00:00	1	Dubois	Michel	480	480	

Salaires des employés ayant eu une diminution par semaine (question 3)

	Date du lundi	Numéro employé	Nom	Prénom	Salaire base	Salaire avant primes	Salaire ré
0	2020-11-09 00:00:00	1	Dubois	Michel	120	110	1

Figure 6: Interface page for viewing the salary of all employees per week (views 1/3)

5 - Mourin Julie

Nombre de production par semaine pour l'employé sélectionné (question 4)

	DATE_LUNDI	NB_PRODUCTION
0	2023-11-13 00:00:00	126
1	2023-11-20 00:00:00	126
2	2023-11-27 00:00:00	126
3	2023-12-04 00:00:00	126
4	2023-12-11 00:00:00	126
5	2023-12-18 00:00:00	126
6	2023-12-25 00:00:00	126
7	2024-01-01 00:00:00	126
8	2024-01-08 00:00:00	126
9	2024-01-15 00:00:00	126

Figure 7: Interface page for viewing an employee's production (view 4)

Employé

8 - Noullant Noura

Nombre d'heures travaillées par semaine pour l'employé sélectionné

	DATE_LUNDI	NB_HEURES
0	2023-08-14 00:00:00	30
1	2023-08-21 00:00:00	48
2	2023-08-28 00:00:00	48
3	2023-09-04 00:00:00	48
4	2023-09-11 00:00:00	48
5	2023-09-18 00:00:00	48
6	2023-09-25 00:00:00	48
7	2023-10-02 00:00:00	48
8	2023-10-09 00:00:00	48
9	2023-10-16 00:00:00	48

Figure 8: Interface page for viewing an employee's hourly production (view 4)

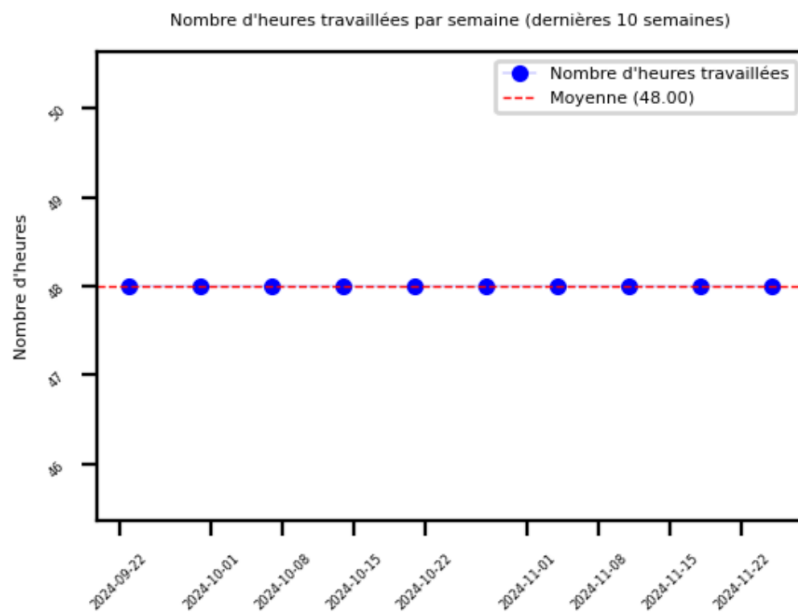


Figure 9: Interface page for viewing weekly hourly work over 10 weeks

Primes d'anniversaire (question 5)

	Lundi	Numéro employé	Nom	Prénom	Date de début	Prime anniversaire
45	2025-11-17	1	Dubois	Michel	2020-11-13	480
97	2025-11-17	2	Bouchaira	Neirouz	2021-11-13	480
147	2025-11-03	3	Caty	Jeanne	2019-11-03	825
201	2025-11-17	5	Mourin	Julie	2023-11-13	472.5
254	2025-11-24	6	Bowers	Matthew	2024-11-20	480
307	2025-12-01	7	Jura	Charlie	2022-11-28	825
319	2025-02-24	4	Dupont	Natalie	2022-02-20	472.5
396	2025-08-18	8	Noullant	Noura	2023-08-16	480
416	2025-01-06	9	Francis	Phillipe	2025-01-03	720

Figure 10: Interface page for viewing employee birthday/employment start date bonuses (views 5/7)

Primes de Noël complètes (question 5/7)

	Lundi	Numéro employé	Nom	Prénom	Date de début	Prime Noël
51	2025-12-29	1	Dubois	Michel	2020-11-13	2,160
103	2025-12-29	2	Bouchaira	Neirouz	2021-11-13	2,160
155	2025-12-29	3	Caty	Jeanne	2019-11-03	3,712.5
207	2025-12-29	5	Mourin	Julie	2023-11-13	2,126.25
259	2025-12-29	6	Bowers	Matthew	2024-11-20	2,160
311	2025-12-29	7	Jura	Charlie	2022-11-28	3,712.5
363	2025-12-29	4	Dupont	Natalie	2022-02-20	2,126.25
415	2025-12-29	8	Noullant	Noura	2023-08-16	2,160

Figure 11: Interface page for viewing employees receiving full Christmas bonuses (views 5/7)

Primes de Noël et d'anniversaire pour les employés pour l'an à venir

- ☒ Primes d'anniversaire
- ☒ Primes de Noël complètes
- ☒ Primes de Noël partielles

Primes de Noël partielles (question 6)

	Lundi	Numéro employé	Nom	Prénom	Date de début	Prime Noël partielles
467	2025-12-29	9	Francis	Phillipe	2025-01-03	710.137

Figure 12: Interface page for viewing employees receiving partial Christmas bonuses (views 6/7)

6 Conclusion

This project made it possible to design and develop a database solution for managing company employees. Thanks to Streamlit, we were able to provide an interactive and intuitive application. This application makes it possible to centralize and analyze the data of employees of a Jeans industrial company, such as salaries, productivity, seniority and employee bonuses.

Among its functionalities, we can highlight:

- Weekly salary tracking with filtering options.
- Employee management.
- Detailed analysis of employee productivity, adapted to different types of contracts.
- Automated calculation of Christmas and seniority bonuses, with anticipation of future periods.
- Simplified creation of new employee accounts, for rapid database updates.

In conclusion, this application constitutes a solid basis for digital human resources management, with a potential for evolution towards more advanced functionalities such as access for each employee or predictive analyses, automated reports or integration with third-party management tools.

A Calculation of salaries by week

```
# Realized by Neirouz Bouchaira
import datetime
cursor.execute("SELECT NUMERO_EMPLOYE, NOM, PRENOM, CS_PRODUCTION, CS_JOURNEE, DATE_DEBUT FROM EMPLOYE")
employees = cursor.fetchall()

cursor.execute("SELECT * FROM LOG_JOURNEE_JOUR")
LOG_JOURNEE_JOUR=cursor.fetchall()

# Find the smallest date
smallest_date = min(row[1] for row in LOG_JOURNEE_JOUR)
today = datetime.datetime.now().strftime("%Y-%m-%d")
delta = datetime.datetime.strptime(today, "%Y-%m-%d") - datetime.datetime.strptime(smallest_date, "%Y-%m-%d")
for i in range(delta.days + 1):
    date = (datetime.datetime.strptime(smallest_date, "%Y-%m-%d") + datetime.timedelta(days=i)).strftime("%Y-%m-%d")
    if datetime.datetime.strptime(date, "%Y-%m-%d").weekday() == 0:
        for employee in employees:
            NUMERO_EMPLOYE, NOM, PRENOM, CS_PRODUCTION, CS_JOURNEE, DATE_DEBUT = employee
            cursor.execute("SELECT * FROM SALAIRE_SEMAINE where DATE_LUNDI=? and NUMERO_EMPLOYE=?", (date, NUMERO_EMPLOYE))
            semaine=cursor.fetchall()
            print(semaine)
            if semaine == [] and CS_JOURNEE is not None:
                cursor.execute(f"SELECT SALAIRE_HORAIRE FROM GRILLE_SALAIRE_HORAIRE WHERE CS_HORAIRE={CS_JOURNEE}")
                salaire_horaire=cursor.fetchone()[0]
                liste_jours_travail=[]
                for j in range(2,8):
                    jour_semaine=(datetime.datetime.strptime(date, "%Y-%m-%d") - datetime.timedelta(days=j)).strftime("%Y-%m-%d")
                    jour_semaine_date = datetime.datetime.strptime(jour_semaine, "%Y-%m-%d")
                    if jour_semaine_date.month == 12 and jour_semaine_date.day == 25:
                        noel=round(min(max(0, (datetime.datetime.strptime(DATE_DEBUT, "%Y-%m-%d") - datetime.datetime.strptime(jour_semaine, "%Y-%m-%d")).days / 365 * salaire_horaire * 48), salaire_horaire * 48 * 4.5))
                    else:
                        noel=0
                    if (datetime.datetime.strptime(date, "%Y-%m-%d") - datetime.datetime.strptime(DATE_DEBUT, "%Y-%m-%d")).days >= 365 and (datetime.datetime.strptime(date, "%Y-%m-%d") - datetime.datetime.strptime(DATE_DEBUT, "%Y-%m-%d")).days % 365 in range(0, 6):
                        anciennete=round(48*salaire_horaire)
                    else:
                        anciennete=0
                    cursor.execute("SELECT * FROM LOG_JOURNEE_JOUR where DATE=? and NUMERO_EMPLOYE=?", (jour_semaine, NUMERO_EMPLOYE))
                    jour=cursor.fetchall()
                    liste_jours_travail.insert(0, jour)
                print(liste_jours_travail)
            if liste_jours_travail != [[], [], [], [], [], []]:
                NB_heures=0
```

```

heures_sup=0
retard=0
heures_base=0
for k in liste_jours_travail:
    if k!=[]:
        heures_base=heures_base+k[0][7]
        NB_heures=NB_heures+k[0][8]
        heures_sup=heures_sup+k[0][9]
        if k[0][10]==True or k[0][10]==1:
            retard+=1

#print(datetime.datetime.strptime(date, "%Y-%m-%d") - datetime.
#timedelta(days=7),NUMERO_EMPLOYE,NB_PIECES,0,salaire_horaire*
#NB_PIECES,noel,anciennete,salaire_horaire*Nb_PIECES+noel+
#anciennete)
cursor.execute("SELECT DATE_LUNDI,SALAIRE_SEMAINE.NUMERO_EMPLOYE,
from SALAIRE_SEMAINE where DATE_LUNDI=? and NUMERO_EMPLOYE=?", (
datetime.datetime.strptime(date, "%Y-%m-%d") - datetime.
timedelta(days=7),NUMERO_EMPLOYE))
semaine_verif=cursor.fetchall()
if semaine_verif==[]:
    cursor.execute("INSERT INTO SALAIRE_SEMAINE (DATE_LUNDI,
NUMERO_EMPLOYE,NB_HEURES,NB_PRODUCTION,RETARD,HEURES_SUP,
PIECES_SUP,SALAIRE_BASE,SALAIRE_AVANT,NOEL,ANCIENNETE,
SALAIRE_APRES) VALUES (?,?,?,?,?,?,?,?,?,?,?,?,?)",
((datetime.datetime.strptime(date, "%Y-%m-%d") -
datetime.datetime.timedelta(days=7)),NUMERO_EMPLOYE,
NB_heures,0,retard,heures_sup,0,
salaire_horaire*heures_base, salaire_horaire*(
NB_heures+heures_sup)-retard*5,noel,anciennete
,salaire_horaire*(NB_heures+heures_sup)-retard
*5+noel+anciennete))
if semaine ==[] and CS_PRODUCTION is not None:
    cursor.execute(f"SELECT SALAIRE_PIECE,PIECE_JOUR FROM
GRILLE_SALAIRE_PROD WHERE CS_PRODUCTION={CS_PRODUCTION}")
salaire_PROD ,PIECE_JOUR =cursor.fetchone()
liste_jours_travail=[]
for j in range(2,8):
    jour_semaine=(datetime.datetime.strptime(date, "%Y-%m-%d") -
datetime.datetime.timedelta(days=j)).strftime("%Y-%m-%d")
    jour_semaine_date = datetime.datetime.strptime(jour_semaine, "%Y-%m-%d")
    if jour_semaine_date.month == 12 and jour_semaine_date.day == 25:
        noel=round(min(max(0, (datetime.datetime.strptime(DATE_DEBUT, "%Y-%m-%d")-datetime.datetime.strptime(jour_semaine, "%Y-%m-%d")
).days /365 *salaire_PROD*PIECE_JOUR ),salaire_PROD*
PIECE_JOUR*4.5))
    else:
        noel=0
    if (datetime.datetime.strptime(date, "%Y-%m-%d") - datetime.datetime.
strptime(DATE_DEBUT, "%Y-%m-%d")).days>=365 and (datetime.
datetime.strptime(date, "%Y-%m-%d") - datetime.datetime.strptime
(DATE_DEBUT, "%Y-%m-%d")).days % 365 in range(0, 6):
        anciennete=round(PIECE_JOUR*salaire_PROD)
    else:
        anciennete=0

```

```

        cursor.execute("SELECT * FROM LOG_JOURNEE_PROD where DATE=? and NUMERO_EMPLOYE=?", (jour_semaine, NUMERO_EMPLOYE))
        jour=cursor.fetchall()
        liste_jours_travail.insert(0, jour)
    print(liste_jours_travail)
    if liste_jours_travail!=([], [], [], [], [], []):
        NB_PIECES=0
        retard=0
        pieces_sup=0
        pieces_base=0
        for k in liste_jours_travail:
            if k!=[]:
                pieces_base=pieces_base+k[0][4]
                NB_PIECES=NB_PIECES+k[0][5]
                pieces_sup=pieces_sup+k[0][6]
                if k[0][9]==True or k[0][9]==1:
                    retard+=1
        #print(datetime.datetime.strptime(date, "%Y-%m-%d") - datetime.
        timedelta(days=7),NUMERO_EMPLOYE,NB_PIECES,0,salaire_horaire*
        NB_PIECES,noel,anciennete,salaire_horaire*NB_PIECES+noel+
        anciennete)
        cursor.execute("SELECT DATE_LUNDI,SALAIRE_SEMAINE.NUMERO_EMPLOYE,
        from SALAIRE_SEMAINE where DATE_LUNDI=? and NUMERO_EMPLOYE=?", (
        datetime.datetime.strptime(date, "%Y-%m-%d") - datetime.
        timedelta(days=7),NUMERO_EMPLOYE))
        semaine_verif=cursor.fetchall()
        if semaine_verif==[]:
            cursor.execute("INSERT INTO SALAIRE_SEMAINE (DATE_LUNDI,
            NUMERO_EMPLOYE,NB_HEURES,NB_PRODUCTION,RETARD,HEURES_SUP,
            PIECES_SUP,SALAIRE_BASE,SALAIRE_AVANT,NOEL,ANCIENNETE,
            SALAIRE_APRES) VALUES(?,?,?,?,?,?,?,?,?,?,?,?,?)",
            ((datetime.datetime.strptime(date, "%Y-%m-%d") -
            datetime.datetime.strptime(date, "%Y-%m-%d"),NUMERO_EMPLOYE,0,
            NB_PIECES,retard,0,pieces_sup,salaire_PROD*
            pieces_base,salaire_PROD*(NB_PIECES+pieces_sup)
            -retard*5,noel,anciennete,salaire_PROD*(
            NB_PIECES+pieces_sup)-retard*5+noel+anciennete)
            )

conn.commit()
conn.close()

```

B Streamlit Application - Employee Management

Below is the code used to implement the application:

```

# Created by Neirouz Bouchaira
import streamlit as st
from datetime import date, datetime, timedelta
import sqlite3
import pandas as pd
import numpy as np # Add this line to import numpy
st.title("Page de gestion des employs")
st.write("Bienvenue!")

```

```

if 'page' not in st.session_state:
    st.session_state.page = None
tab1, tab2, tab3, tab4, tab5 = st.tabs(["Suivi des salaires", "Employs +3 ans d'anciennet",
    "Productivité", "Primes", "Création d'un compte employé"])
with tab1:
    st.header("Suivi Employs")
    popover = st.popover("Filtre employs afficher")
    total = popover.checkbox("Salaires des employs par semaine", True)
    dimin = popover.checkbox("Salaires des employs ayant eu une diminution par semaine", True)
    conn = sqlite3.connect(r'donnees\company.db')
    cursor = conn.cursor()
    cursor.execute("SELECT DATE_LUNDI, SALAIRE_SEMAINE.NUMERO_EMPLOYE, NOM, PRENOM, SALAIRE_BASE, SALAIRE_AVANT, SALAIRE_APRES, NB_HEURES, NB_PRODUCTION, RETARD, ANCIENNETE, NOEL, DATE_DEBUT FROM SALAIRE_SEMAINE JOIN EMPLOYE ON EMPLOYE.NUMERO_EMPLOYE = SALAIRE_SEMAINE.NUMERO_EMPLOYE")
    employees = cursor.fetchall()
    # Convert the employee records to a DataFrame
    df = pd.DataFrame(employees, columns=[desc[0] for desc in cursor.description])
    df.columns = ['Date du lundi', 'Numéro employé', 'Nom', 'Prénom', 'Salaire base', 'Salaire avant primes', 'Salaire rel', 'Nombre d\'heures', 'Nombre de production', 'Retard', 'Ancienneté', 'Bonus de Noël', 'Date de début']
    if total:
        # Add code for "Suivi Employs" page here
        st.write("Salaires des employs par semaine (question 1)")
        st.dataframe(df)
    if dimin:
        st.write("Salaires des employs ayant eu une diminution par semaine (question 3)")
        st.dataframe(df[df['Salaire avant primes'] < df['Salaire base']])

    conn.close()
with tab2:
    conn = sqlite3.connect(r'donnees\company.db')
    cursor = conn.cursor()
    on = st.toggle("Employs avec 3 ans d'ancienneté")
    cursor.execute("SELECT NUMERO_EMPLOYE, NOM, PRENOM, TYPE, DATE_DEBUT FROM EMPLOYE")
    employees = cursor.fetchall()
    df = pd.DataFrame(employees, columns=[desc[0] for desc in cursor.description])
    from datetime import datetime
    today = datetime.today()
    df['DATE_DEBUT'] = pd.to_datetime(df['DATE_DEBUT'])
    if on:
        st.header("Liste des employs avec 3 ans d'ancienneté")
        df = df[(today - df['DATE_DEBUT']).dt.days > 3 * 365]
        df.columns = ['Numéro employé', 'Nom', 'Prénom', 'Type du contrat', 'Date de début']
        st.write("Liste des employs journaliers (question 2)")
        st.dataframe(df[df['Type du contrat'] == 1])
        st.write("Liste des employs par production")
        st.dataframe(df[df['Type du contrat'] == 2])
    else:
        df.columns = ['Numéro employé', 'Nom', 'Prénom', 'Type du contrat', 'Date de début']
        st.header("Liste de la totalité des employs")
        st.write("Liste des employs journaliers")

```



```

st.dataframe(df[df['Type_du_contrat']==1])
st.write("Liste_des_employs_par_production")
st.dataframe(df[df['Type_du_contrat'] == 2])

with tab3:
    conn= sqlite3.connect(r'donnees\company.db')
    cursor = conn.cursor()
    cursor.execute("SELECT_NUMERO_EMPLOYE,NOM,PRENOM,TYPE_FROM_EMPLOYE")
    employees = cursor.fetchall()
    liste_employees=sorted([ (str(employees[i][0]) + "-" + employees[i][1] + "-" +
        employees[i][2]) for i in range(len(employees)))])
    employe = st.selectbox("Employ", options=liste_employees, index=None,placeholder="
        Choisir_un_employ")
    if employe is None:
        st.write("")
    else:
        df_employees = pd.DataFrame(employees, columns=[desc[0] for desc in cursor.
            description])
        if int(df_employees[df_employees['NUMERO_EMPLOYE']==int(employe.split("-")[
            0]])['TYPE']) == 1:
            cursor.execute("SELECT_DATE_LUNDI,NB_HEURES_FROM_SALAIRE_SEMAINE_WHERE_
                NUMERO_EMPLOYE=?", (int(employe.split("-")[0]),))
            df=pd.DataFrame(cursor.fetchall(), columns=[desc[0] for desc in cursor.
                description])
            st.write("Nombre_d'heures_travaillées_par_semaine_pour_l'employ_slectionn")
            st.dataframe(df)
            import matplotlib.pyplot as plt

            # Filter the last 10 weeks
            df['DATE_LUNDI'] = pd.to_datetime(df['DATE_LUNDI'])
            last_10_weeks = df.sort_values(by='DATE_LUNDI').tail(10)

            # Calculate the average hours per week
            average_hours = last_10_weeks['NB_HEURES'].mean()

            # Plot the data
            fig, ax = plt.subplots()
            ax.plot(last_10_weeks['DATE_LUNDI'], last_10_weeks['NB_HEURES'], marker='.',
                , linestyle='--',linewidth=0.1, label='Nombre_d\'heures_travaillées',
                color='blue')
            ax.axhline(y=average_hours, color='red', linestyle='--',linewidth=0.5,
                label=f'Moyenne_{average_hours:.2f}')
            ax.set_xlabel('Date', fontsize=4)
            ax.set_ylabel('Nombre_d\'heures', fontsize=4)
            ax.set_title('Nombre_d\'heures_travaillées_par_semaine_(dernieres_10_semaines
                )', fontsize=4)
            ax.tick_params(axis='both', which='major', labels=3, rotation=45)
            fig.set_size_inches(3, 2)
            ax.legend(fontsize=4)

            # Display the plot in Streamlit
            st.pyplot(fig)
        else:
            cursor.execute("SELECT_DATE_LUNDI,NB_PRODUCTION_FROM_SALAIRE_SEMAINE_WHERE_
                NUMERO_EMPLOYE=?", (int(employe.split("-")[0]),))
            df=pd.DataFrame(cursor.fetchall(), columns=[desc[0] for desc in cursor.
                description])

```

```

st.write("Nombre de production par semaine pour l'employé sélectionné(
question_4)")
st.dataframe(df)
import matplotlib.pyplot as plt

# Filter the last 10 weeks
df['DATE_LUNDI'] = pd.to_datetime(df['DATE_LUNDI'])
last_10_weeks = df.sort_values(by='DATE_LUNDI').tail(10)

# Calculate the average production per week
average_production = last_10_weeks['NB_PRODUCTION'].mean()

# Plot the data
fig, ax = plt.subplots()
ax.plot(last_10_weeks['DATE_LUNDI'], last_10_weeks['NB_PRODUCTION'], marker=
'.', linestyle='-', linewidth=0.1, label='Nombre de pièces produites',
color='blue')
ax.axhline(y=average_production, color='red', linestyle='--', linewidth
=0.5, label=f'Moyenne ({average_production:.2f})')
ax.set_xlabel('Date', fontsize=4)
ax.set_ylabel('Nombre de pièces produites', fontsize=4)
ax.set_title('Nombre de pièces produites par semaine (dernières 10 semaines)',
, fontsize=4)
ax.tick_params(axis='both', which='major', labelsize=3, rotation=45)
fig.set_size_inches(3, 2)
ax.legend(fontsize=4)

# Display the plot in Streamlit
st.pyplot(fig)

with tab4:
    # Fonction pour générer les lundis
    def generate_mondays(start_date):
        """
        Génère une liste de lundis à partir d'une date donnée jusqu'à un an après.

        :param start_date: Date de début au format 'YYYY-MM-DD'
        :return: Liste de lundis au format 'YYYY-MM-DD'
        """
        start = datetime.strptime(start_date, '%Y-%m-%d')
        end_date = start + timedelta(days=365)

        # Ajuster au premier lundi
        if start.weekday() != 0:
            start += timedelta(days=(7 - start.weekday()))

        mondays = []
        current_date = start
        while current_date <= end_date:
            mondays.append(current_date.strftime('%Y-%m-%d'))
            current_date += timedelta(weeks=1)

        return mondays

    # Connexion à la base de données
    conn = sqlite3.connect('donnees\company.db')
    cursor = conn.cursor()

```

```
# Génération des lundis
mondays = generate_mondays(datetime.now().strftime('%Y-%m-%d'))

# Exemple d'opérations avec les employés
christmas_day = datetime(2024, 12, 25)
cursor.execute("SELECT NUMERO_EMPLOYE, NOM, PRENOM, CS_PRODUCTION, CS_JOURNEE, DATE_DEBUT FROM EMPLOYE")
employees = cursor.fetchall()

prime_totale = []
prime_partielle = []
Bonus=[]
for employee in employees:
    NUMERO_EMPLOYE, NOM, PRENOM, CS_PRODUCTION, CS_JOURNEE, DATE_DEBUT = employee
    start_date = datetime.strptime(DATE_DEBUT, '%Y-%m-%d')

    for LUNDI in mondays:
        lundi = datetime.strptime(LUNDI, '%Y-%m-%d')
        delta = lundi - start_date
        delta_days = delta.days
        delta_noel = lundi - christmas_day
        delta_noel_days = delta_noel.days
        prime_partielle=0
        prime_totale=0
        prime_anniversaire=0

        if 0 <= delta_noel_days%365<= 6:
            if delta_days >= 365:
                if CS_JOURNEE is not None:
                    cursor.execute(f"SELECT SALAIRE_SEMAINE_BASE FROM GRILLE_SALAIRE_HORAIRE WHERE CS_HORAIRE={CS_JOURNEE}")
                    salaire_base=cursor.fetchone()[0]
                    prime_totale= salaire_base*4.5
                if CS_PRODUCTION is not None:
                    cursor.execute(f"SELECT SALAIRE_SEMAINE_BASE FROM GRILLE_SALAIRE_PROD WHERE CS_PRODUCTION={CS_PRODUCTION}")
                    salaire_base=cursor.fetchone()[0]
                    prime_totale= salaire_base*4.5
            else:
                if CS_JOURNEE is not None:
                    cursor.execute(f"SELECT SALAIRE_SEMAINE_BASE FROM GRILLE_SALAIRE_HORAIRE WHERE CS_HORAIRE={CS_JOURNEE}")
                    salaire_base=cursor.fetchone()[0]
                    prime_partielle= max(0, salaire_base * delta_days / 365 )
                if CS_PRODUCTION is not None:
                    cursor.execute(f"SELECT SALAIRE_SEMAINE_BASE FROM GRILLE_SALAIRE_PROD WHERE CS_PRODUCTION={CS_PRODUCTION}")
                    salaire_base=cursor.fetchone()[0]
                    prime_partielle=max(0, salaire_base*100 * delta_days / 365 )

        if 0 <= abs(delta_days%365)<= 6:
            if CS_JOURNEE is not None:
                cursor.execute(f"SELECT SALAIRE_SEMAINE_BASE FROM GRILLE_SALAIRE_HORAIRE WHERE CS_HORAIRE={CS_JOURNEE}")
                salaire_base=cursor.fetchone()[0]
                prime_anniversaire= salaire_base
```

```

        if CS_PRODUCTION is not None:
            cursor.execute(f"SELECT SALAIRE_SEMAINE_BASE FROM GRILLE_SALAIRE_PROD WHERE CS_PRODUCTION={CS_PRODUCTION}")
            salaire_base=cursor.fetchone()[0]
            prime_anniversaire= salaire_base
            Bonus.append((NUMERO_EMPLOYE, NOM, PRENOM, DATE_DEBUT, LUNDI, prime_totale,
                prime_partielle, prime_anniversaire, prime_partielle+prime_totale+
                prime_anniversaire ))
            Bonus = pd.DataFrame(Bonus, columns=['NUMERO_EMPLOYE', 'NOM', 'PRENOM', 'DATE_DEBUT',
                'LUNDI', 'PRIME_NOEL', 'PRIME_NOEL_PARTIELLES', 'PRIME_ANNIVERSAIRE', 'PRIMES'])
            Bonus = Bonus[['LUNDI', 'NUMERO_EMPLOYE', 'NOM', 'PRENOM', 'DATE_DEBUT', 'PRIME_NOEL', 'PRIME_NOEL_PARTIELLES', 'PRIME_ANNIVERSAIRE', 'PRIMES']]
            Bonus.columns = ['Lundi', 'Numro_employ', 'Nom', 'Prnom', 'Date_debut', 'Prime_Nol', 'Prime_Nol_partielles', 'Prime_anniversaire', 'Primes']
            Bonus[Bonus['Primes']>0].head(50)
            st.header("Primes de Nol et d'anniversaire pour les employs pour l'an venir")
            options = [ "Primes d'anniversaire", "Primes de Nol compltes", "Primes de Nol partielles" ]
            selections = {option: st.checkbox(option) for option in options}
            if selections["Primes de Nol partielles"]:
                st.write("Primes de Nol partielles (question 6)")
                st.dataframe(Bonus[Bonus['Prime_Nol_partielles']>0][["Lundi", "Numro_employ", "Nom", "Prnom", "Date_debut", "Prime_Nol_partielles"]])
            if selections["Primes de Nol compltes"]:
                st.write("Primes de Nol compltes (question 5/7)")
                st.dataframe(Bonus[Bonus['Prime_Nol']>0][["Lundi", "Numro_employ", "Nom", "Prnom", "Date_debut", "Prime_Nol"]])
            if selections["Primes d'anniversaire"]:
                st.write("Primes d'anniversaire (question 5)")
                st.dataframe(Bonus[Bonus['Prime_anniversaire']>0][["Lundi", "Numro_employ", "Nom", "Prnom", "Date_debut", "Prime_anniversaire"]])

with tab5:
    st.header("Cratation de compte employ")
    conn= sqlite3.connect(r'donnees\company.db')
    cursor = conn.cursor()
    cursor.execute("SELECT MAX(NUMERO_EMPLOYE) FROM EMPLOYE")
    max_numero_employe = cursor.fetchone()[0]

    # Set the employee number to the next available number
    NUMERO_EMPLOYE = max_numero_employe + 1 if max_numero_employe is not None else 1

    # Employee type

    # Employee name
    NOM = st.text_input("Nom", value='')

    # Employee first name
    PRENOM = st.text_input("Prnom", value='')
    TYPE = st.selectbox("Type de contrat", options=[1, 2], format_func=lambda x: "Journalier" if x == 1 else "Production", index=0)

    # Conditional fields based on employee type
    if TYPE == 2:
        CS_PRODUCTION = st.number_input("Code salarial de production", min_value=1,

```

```
        step=1, value=None)
    CS_JOURNEE = None
else:
    CS_JOURNEE = st.number_input("Code_salarial_journalier", min_value=1, step=1,
                                value=1)
    CS_PRODUCTION = None

# Start date
date_debut = st.date_input("date_de_debut", value=date(2024, 11, 13))

# Submit button
if st.button("Create_Account"):
    # Connect to the database
    conn=sqlite3.connect(r'donnees\company.db')
    cursor = conn.cursor()
    cursor.execute("""
        SELECT NUMERO_EMPLOYE FROM EMPLOYE
        WHERE NOM = ? AND PRENOM = ? AND DATE_DEBUT = ?
    """, (NOM, PRENOM, date_debut))
    existing_employee = cursor.fetchone()

    if existing_employee:
        st.write(f"Cet_employe_est_dj_enregistr_avec_le_numro_employ:{existing_employee[0]}")
        conn.close()

    else:
        cursor.execute("""
            INSERT INTO EMPLOYE (NUMERO_EMPLOYE, TYPE, NOM, PRENOM, CS_PRODUCTION,
                                CS_JOURNEE, DATE_DEBUT)
            VALUES (?, ?, ?, ?, ?, ?, ?)
        """, (NUMERO_EMPLOYE, TYPE, NOM, PRENOM, CS_PRODUCTION if CS_PRODUCTION is
              not None else None, CS_JOURNEE if CS_JOURNEE is not None else None,
              date_debut))
        # Commit the transaction
        conn.commit()

        st.write("Nouvel_employe_enregistr!")
        st.write(f"Numro_employ:{NUMERO_EMPLOYE}")
        st.write(f"Type_de_contrat:{'Journalier' if TYPE==1 else 'Production'}")
        st.write(f"Nom:{NOM}")
        st.write(f"Prnom:{PRENOM}")
        if TYPE == 2:
            st.write(f"Code_salarial_de_production:{CS_PRODUCTION}")
        else:
            st.write(f"code_salarial_journalier:{CS_JOURNEE}")
        st.write(f"Date_du_debut_du_contrat:{date_debut}")
        conn.close()

st.stop()
```

C Streamlit Application - Badging

```
# Created by Neirouz Bouchaira
import streamlit as st
```

```
import sqlite3
import datetime
# Connect to the database
conn = sqlite3.connect(r'C:\Users\neiro\Documents\+\GitHub\sgbd-usine\donnees\company.
    db')
cursor = conn.cursor()
# Set the title of the Streamlit app
st.title("Machine de Badgeage")
# Initialize session state for buttons
if 'badge_in_clicked' not in st.session_state:
    st.session_state.badge_in_clicked = False
if 'badge_out_clicked' not in st.session_state:
    st.session_state.badge_out_clicked = False

# Disable buttons if already clicked
badge_in_disabled = st.session_state.badge_in_clicked
badge_out_disabled = st.session_state.badge_out_clicked
# Display a welcome message
st.write("Passez une bonne journe!")
# Get the next NUM_LOG_JOUR_JOUR
cursor.execute("SELECT MAX(NUM_LOG_JOUR_JOUR) FROM LOG_JOURNEE_JOUR")
max_num_log_jour_jour = cursor.fetchone()[0]
NUM_LOG_JOUR_JOUR = max_num_log_jour_jour + 1 if max_num_log_jour_jour is not None
    else 1

# Get today's date
DATE = datetime.date.today().strftime('%Y-%m-%d')

# Employee number input
NUMERO_EMPLOYE = st.number_input("Numro Employ", min_value=1, step=1)

# Badge in button
if st.button("Badge In", disabled=badge_in_disabled, key="badge_in"):
    st.session_state.badge_in_clicked = True
    st.experimental_rerun() # Force rerun to update button state
    HEURE_ARRIVEE = datetime.datetime.now().strftime('%H:%M:%S')
    cursor.execute('''
        INSERT INTO LOG_JOURNEE_JOUR (NUM_LOG_JOUR_JOUR, DATE, NUMERO_EMPLOYE,
            HEURE_ARRIVEE)
        VALUES (?, ?, ?, ?)
    ''', (NUM_LOG_JOUR_JOUR, DATE, NUMERO_EMPLOYE, HEURE_ARRIVEE))
    conn.commit()
    st.success(f"Arrive enregistre {HEURE_ARRIVEE}")

# Badge out button
if st.button("Badge Out", disabled=badge_out_disabled, key="badge_out"):
    st.session_state.badge_out_clicked = True
    st.experimental_rerun() # Force rerun to update button state
    HEURE_FIN = datetime.datetime.now().strftime('%H:%M:%S')
    cursor.execute('''
        UPDATE LOG_JOURNEE_JOUR
        SET HEURE_FIN = ?
        WHERE NUMERO_EMPLOYE = ? AND DATE = ?
    ''', (HEURE_FIN, NUMERO_EMPLOYE, DATE))
    conn.commit()
    st.success(f"Dpart enregistri {HEURE_FIN}")
# Calculate working hours and update the database
```

```
cursor.execute('''
SELECT HEURE_ARRIVEE, HEURE_FIN FROM LOG_JOURNEE_JOUR
WHERE NUMERO_EMPLOYE = ? AND DATE = ?
''', (NUMERO_EMPLOYE, DATE))
result = cursor.fetchone()

if result:
    HEURE_ARRIVEE, HEURE_FIN = result
    if HEURE_ARRIVEE and HEURE_FIN:
        # Calculate total hours worked
        heure_arrivee = datetime.datetime.strptime(HEURE_ARRIVEE, '%H:%M:%S')
        heure_fin = datetime.datetime.strptime(HEURE_FIN, '%H:%M:%S')
        nb_heures = (heure_fin - heure_arrivee).seconds / 3600.0

        # Calculate overtime hours
        nb_heures_sup = max(nb_heures - 9, 0)

        # Determine if there was a delay
        retard = heure_arrivee.time() > datetime.time(8, 0, 0)

        # Update the database with calculated values
        cursor.execute('''
UPDATE LOG_JOURNEE_JOUR
SET NB_HEURES = ?, NB_HEURES_SUP = ?, RETARD = ?
WHERE NUMERO_EMPLOYE = ? AND DATE = ?
''', (nb_heures, nb_heures_sup, retard, NUMERO_EMPLOYE, DATE))
        conn.commit()

# Display a summary of the day's work
st.write("Rcapitulatif de la journée:")
st.write(f"Numro Employ: {NUMERO_EMPLOYE}")
# Fetch employee's name and surname
cursor.execute('''
SELECT NOM, PRENOM FROM EMPLOYE
WHERE NUMERO_EMPLOYE = ?
''', (NUMERO_EMPLOYE,))
employee = cursor.fetchone()
if employee:
    NOM, PRENOM = employee
    st.write(f"Nom: {NOM}")
    st.write(f"Prnom: {PRENOM}")
st.write(f"Nombre d'heures travaillées: {nb_heures:.2f}")
st.write(f"Nombre d'heures supplémentaires: {nb_heures_sup:.2f}")
st.write(f"Retard: {'Oui' if retard else 'Non'}")
```