Certainly! Below is a basic database schema for storing employee details persistently. We will use a relational database schema as an example.

**Tables:**

1. **Employee** - Contains basic information about the employee.
2. **Department** - Contains information about different departments in the company.
3. **Designation** - Contains information about various designations/roles.
4. **EmployeeAddress** - Stores multiple addresses for an employee, such as home address, temporary address, etc.
5. **EmployeeContacts** - To keep track of multiple contact numbers.

**CREATE TABLE Department (**

**DepartmentID INT PRIMARY KEY AUTO\_INCREMENT,**

**DepartmentName VARCHAR(255) NOT NULL,**

**Description TEXT**

**);**

**CREATE TABLE Designation (**

**DesignationID INT PRIMARY KEY AUTO\_INCREMENT,**

**DesignationName VARCHAR(255) NOT NULL,**

**Description TEXT**

**);**

**CREATE TABLE Employee (**

**EmployeeID INT PRIMARY KEY AUTO\_INCREMENT,**

**FirstName VARCHAR(255) NOT NULL,**

**LastName VARCHAR(255) NOT NULL,**

**Email VARCHAR(255) UNIQUE NOT NULL,**

**DateOfBirth DATE,**

**DateOfJoining DATE NOT NULL,**

**DepartmentID INT,**

**DesignationID INT,**

**Salary DECIMAL(10, 2),**

**FOREIGN KEY (DepartmentID) REFERENCES Department(DepartmentID),**

**FOREIGN KEY (DesignationID) REFERENCES Designation(DesignationID)**

**);**

**CREATE TABLE EmployeeAddress (**

**AddressID INT PRIMARY KEY AUTO\_INCREMENT,**

**EmployeeID INT,**

**AddressType VARCHAR(255), -- Example values: 'Home', 'Temporary'**

**AddressLine1 TEXT NOT NULL,**

**AddressLine2 TEXT,**

**City VARCHAR(255) NOT NULL,**

**State VARCHAR(255) NOT NULL,**

**PostalCode VARCHAR(10) NOT NULL,**

**Country VARCHAR(255) NOT NULL,**

**FOREIGN KEY (EmployeeID) REFERENCES Employee(EmployeeID)**

**);**

**CREATE TABLE EmployeeContacts (**

**ContactID INT PRIMARY KEY AUTO\_INCREMENT,**

**EmployeeID INT,**

**ContactType VARCHAR(255), -- Example values: 'Mobile', 'Home'**

**ContactNumber VARCHAR(20) NOT NULL,**

**FOREIGN KEY (EmployeeID) REFERENCES Employee(EmployeeID)**

**);**

**Notes:**

* The above schema provides a simple relational structure for the HRMS system.
* Employee table references both Department and Designation tables, allowing for easy modifications of department names or designations without affecting the primary employee records.
* EmployeeAddress and EmployeeContacts allow for the storage of multiple addresses and contacts respectively for each employee.
* This schema can be extended with additional features like attendance, leave management, payroll details, etc., as required.
* When implementing the above schema in a real-world scenario, always consider normalizing the database to eliminate any redundancy and to ensure the integrity of the data.
* The SQL provided is generic and may need to be adjusted based on the specifics of your chosen RDBMS (e.g., MySQL, PostgreSQL, SQL Server, etc.).



Database schema for attendance, leave management, payroll, appraisals

ChatGPT

Certainly! Let's expand upon our previous schema and add tables related to attendance, leave management, payroll, and appraisals.

**1. Attendance:**

This will record daily attendance for each employee.

CREATE TABLE Attendance (

AttendanceID INT PRIMARY KEY AUTO\_INCREMENT,

EmployeeID INT,

Date DATE NOT NULL,

TimeIn TIME,

TimeOut TIME,

AttendanceStatus VARCHAR(255), -- Example values: 'Present', 'Absent', 'Half-day'

Remarks TEXT,

FOREIGN KEY (EmployeeID) REFERENCES Employee(EmployeeID)

);

**2. Leave Management:**

**CREATE TABLE LeaveType (**

**LeaveTypeID INT PRIMARY KEY AUTO\_INCREMENT,**

**TypeName VARCHAR(255) NOT NULL,**

**Description TEXT**

**);**

**CREATE TABLE EmployeeLeave (**

**LeaveID INT PRIMARY KEY AUTO\_INCREMENT,**

**EmployeeID INT,**

**LeaveTypeID INT,**

**StartDate DATE NOT NULL,**

**EndDate DATE NOT NULL,**

**Status VARCHAR(255), -- Example values: 'Pending', 'Approved', 'Rejected'**

**Remarks TEXT,**

**FOREIGN KEY (EmployeeID) REFERENCES Employee(EmployeeID),**

**FOREIGN KEY (LeaveTypeID) REFERENCES LeaveType(LeaveTypeID)**

**);**

**3. Payroll:**

For the sake of simplicity, we'll create a basic table for payroll. Complex payroll systems require more tables and relations.

CREATE TABLE Payroll (

PayrollID INT PRIMARY KEY AUTO\_INCREMENT,

EmployeeID INT,

MonthYear DATE NOT NULL,

BasicSalary DECIMAL(10, 2),

Allowances DECIMAL(10, 2),

Deductions DECIMAL(10, 2),

NetSalary DECIMAL(10, 2),

PaidOn DATE,

FOREIGN KEY (EmployeeID) REFERENCES Employee(EmployeeID)

);

**4. Appraisals:**

Here, we define criteria for appraisals and a table to record employee appraisals.

CREATE TABLE AppraisalCriteria (

CriteriaID INT PRIMARY KEY AUTO\_INCREMENT,

CriteriaName VARCHAR(255) NOT NULL,

Description TEXT

);

CREATE TABLE EmployeeAppraisal (

AppraisalID INT PRIMARY KEY AUTO\_INCREMENT,

EmployeeID INT,

Date DATE NOT NULL,

CriteriaID INT,

Score DECIMAL(5, 2), -- Assuming it's on a scale, e.g., 0-100

Feedback TEXT,

FOREIGN KEY (EmployeeID) REFERENCES Employee(EmployeeID),

FOREIGN KEY (CriteriaID) REFERENCES AppraisalCriteria(CriteriaID)

);

**Notes:**

* As with the previous schema, this SQL is generic and might require adjustments based on the specifics of the RDBMS you're using.
* These tables are simplified and may not cover all requirements for a real-world HRMS system. A complete HRMS system would have more fields, constraints, and relations to cover all scenarios.
* For more advanced systems, consider integrating features like:
  + Leave balances for each type of leave per employee.
  + Various payroll components like tax calculations, bonuses, overtime, etc.
  + More comprehensive appraisal features, including peer reviews, 360-degree feedback, etc.



Code snipet for User authentication and authorization mechanisms.

ChatGPT

Certainly! For user authentication and authorization, I'll outline a basic implementation using Python's Flask web framework with Flask-Login for authentication and a simple role-based mechanism for authorization.

**1. Set up:**

First, install the necessary packages: BASH

pip install Flask Flask-SQLAlchemy Flask-Login

**2. Basic App and Database Models:**

pythonCopy code

from flask import Flask, render\_template, redirect, url\_for, request, flash

from flask\_sqlalchemy import SQLAlchemy

from flask\_login import LoginManager, UserMixin, login\_user, login\_required, logout\_user, current\_user

from werkzeug.security import generate\_password\_hash, check\_password\_hash

app = Flask(\_\_name\_\_)

app.config['SECRET\_KEY'] = 'your\_secret\_key'

app.config['SQLALCHEMY\_DATABASE\_URI'] = 'sqlite:///hrms.db'

db = SQLAlchemy(app)

login\_manager = LoginManager(app)

login\_manager.login\_view = 'login'

class User(UserMixin, db.Model):

id = db.Column(db.Integer, primary\_key=True)

username = db.Column(db.String(150), unique=True, nullable=False)

password = db.Column(db.String(150), nullable=False)

role = db.Column(db.String(150), nullable=False) # Example values: 'admin', 'employee'

@login\_manager.user\_loader

def load\_user(user\_id):

return User.query.get(int(user\_id))

@app.route('/login', methods=['GET', 'POST'])

def login():

if request.method == 'POST':

username = request.form.get('username')

password = request.form.get('password')

user = User.query.filter\_by(username=username).first()

if user and check\_password\_hash(user.password, password):

login\_user(user)

return redirect(url\_for('dashboard'))

flash('Login Unsuccessful. Please check username and password.', 'danger')

return render\_template('login.html')

@app.route('/dashboard')

@login\_required

def dashboard():

return f"Hello {current\_user.username}, Role: {current\_user.role}"

@app.route('/admin\_page')

@login\_required

def admin\_page():

if current\_user.role != 'admin':

flash('You do not have permission to view this page.', 'danger')

return redirect(url\_for('dashboard'))

return "Welcome to the Admin Page!"

@app.route('/logout')

@login\_required

def logout():

logout\_user()

return redirect(url\_for('login'))

if \_\_name\_\_ == "\_\_main\_\_":

db.create\_all()

app.run(debug=True)

**3. Basic HTML for Login:**

Create a template named **login.html**:

htmlCopy code

<!DOCTYPE html>

<html>

<head>

<title>Login</title>

</head>

<body>

<form action="/login" method="post">

<label for="username">Username:</label>

<input type="text" name="username" required>

<label for="password">Password:</label>

<input type="password" name="password" required>

<button type="submit">Login</button>

</form>

</body>

</html>

**Explanation:**

* We set up Flask, Flask-SQLAlchemy for database handling, and Flask-Login for session management.
* We created a simple **User** model with a role attribute.
* The **/login** route handles the login process. It checks the hashed password and logs in the user if the credentials are valid.
* The **@login\_required** decorator ensures that certain routes can only be accessed by logged-in users.
* For authorization, the **/admin\_page** route checks the role of the **current\_user** to determine whether they can access the page or not.

Note: This is a very basic implementation for demonstration purposes. In real-world applications, additional security measures like salting passwords, handling CSRF, etc., should be implemented. Furthermore, one might consider using Flask-Principal or other packages for a more advanced role/permission system.



Code snipet for Reporting and analytics capabilities

ChatGPT

Implementing reporting and analytics capabilities is a broad topic. Here, I'll provide a basic implementation using Python's Flask framework with SQLAlchemy for database interaction and Plotly for visualization.

For simplicity, we'll assume a system where we wish to visualize the number of users added to our system per month.

**1. Set up:**

Install necessary packages:

bashCopy code

pip install Flask Flask-SQLAlchemy pandas plotly

**2. Basic App, Database Model, and Visualization:**

pythonCopy code

from flask import Flask, render\_template

from flask\_sqlalchemy import SQLAlchemy

import pandas as pd

import plotly.express as px

app = Flask(\_\_name\_\_)

app.config['SQLALCHEMY\_DATABASE\_URI'] = 'sqlite:///hrms.db'

db = SQLAlchemy(app)

class User(db.Model):

id = db.Column(db.Integer, primary\_key=True)

username = db.Column(db.String(150), unique=True, nullable=False)

joined\_date = db.Column(db.DateTime, nullable=False)

@app.route('/user\_report')

def user\_report():

# Using SQLAlchemy to get data

results = db.session.query(User.joined\_date).all()

# Convert the results to a pandas DataFrame

df = pd.DataFrame(results, columns=['joined\_date'])

# Extract the month and year from the date and count the occurrences

df['month\_year'] = df['joined\_date'].dt.to\_period('M')

monthly\_counts = df['month\_year'].value\_counts().sort\_index()

# Create a bar chart with Plotly

fig = px.bar(monthly\_counts, x=monthly\_counts.index.astype(str), y=monthly\_counts.values, labels={'x': 'Month', 'y': 'Number of Users'})

# Convert the figure to HTML

graph\_html = fig.to\_html(full\_html=False)

return render\_template('report.html', graph=graph\_html)

if \_\_name\_\_ == "\_\_main\_\_":

db.create\_all()

app.run(debug=True)

**3. HTML for the report:**

Create a template named **report.html**:

htmlCopy code

<!DOCTYPE html>

<html>

<head>

<title>User Report</title>

</head>

<body>

{{ graph|safe }}

</body>

</html>

**Explanation:**

* We've set up Flask and SQLAlchemy as before and introduced **pandas** for data processing and **plotly** for visualization.
* We have a basic **User** model with a **joined\_date** attribute.
* The **/user\_report** route fetches user data from the database, processes it with **pandas** to get monthly counts of new users, and then visualizes it using **plotly**.
* The resultant plot is embedded into an HTML page.

For larger datasets or complex queries, consider optimizing database interactions and possibly using caching. Additionally, while this example focuses on user growth, you can extend this method to any metrics of interest, such as sales growth, product usage, and more.