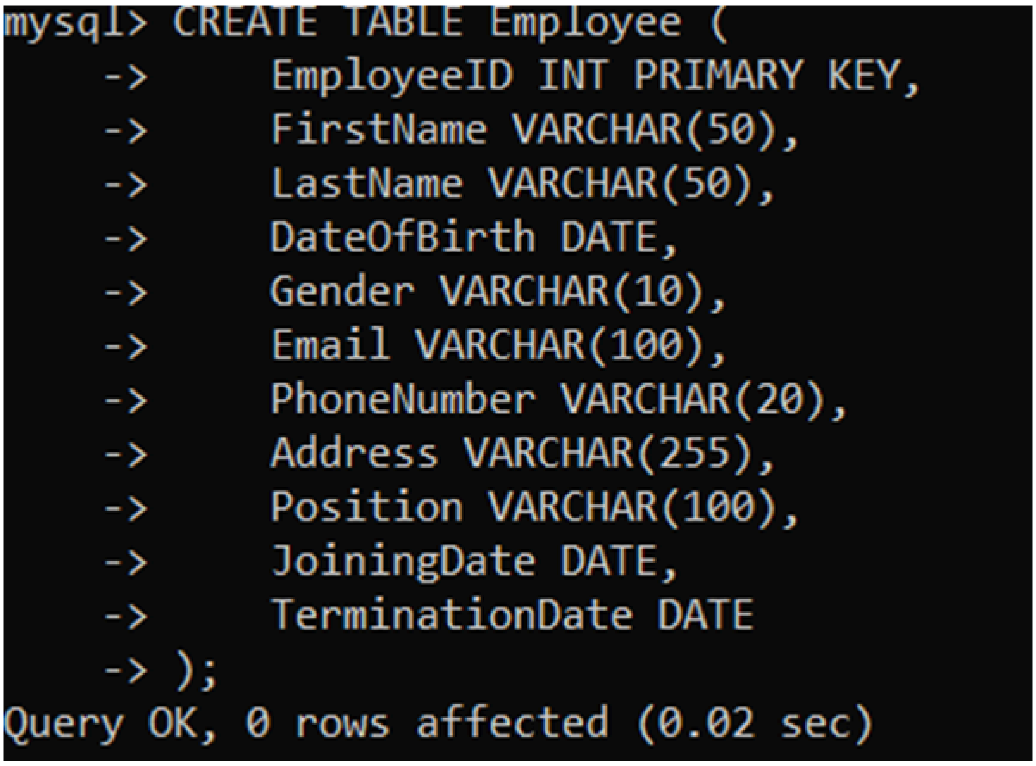
PayXpert – Payroll management system

Sivaprakas B M

SQL Tables:

1. Employee Table:

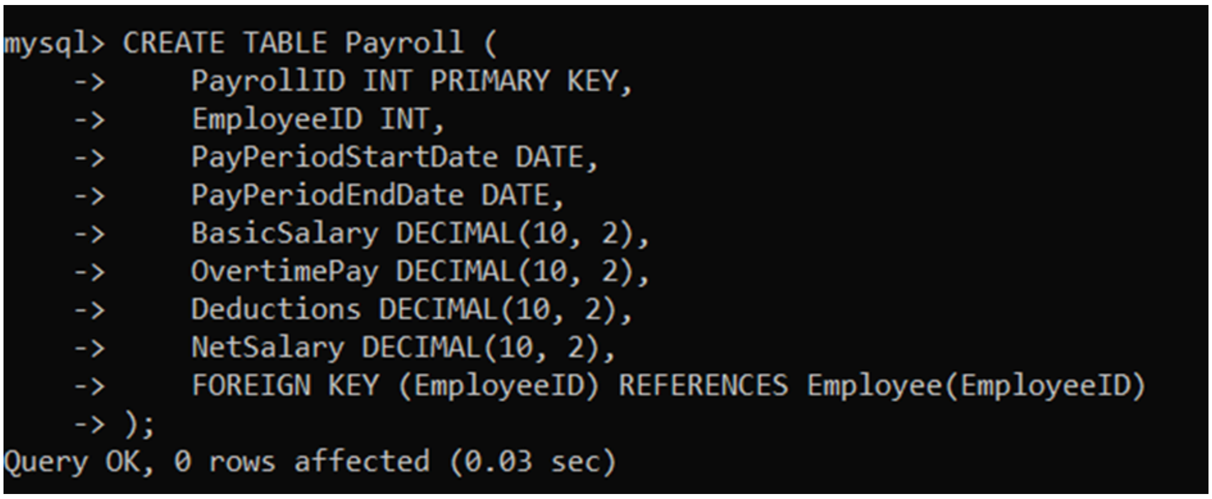
- EmployeeID (Primary Key): Unique identifier for each employee.
- FirstName: First name of the employee.
- LastName: Last name of the employee.
- DateOfBirth: Date of birth of the employee.
- Gender: Gender of the employee.
- Email: Email address of the employee.
- PhoneNumber: Phone number of the employee.
- Address: Residential address of the employee.
- Position: Job title or position of the employee.
- JoiningDate: Date when the employee joined the company.
- TerminationDate: Date when the employee left the company (nullable).



```
mysql> CREATE TABLE Employee (
    ->     EmployeeID INT PRIMARY KEY,
    ->     FirstName VARCHAR(50),
    ->     LastName VARCHAR(50),
    ->     DateOfBirth DATE,
    ->     Gender VARCHAR(10),
    ->     Email VARCHAR(100),
    ->     PhoneNumber VARCHAR(20),
    ->     Address VARCHAR(255),
    ->     Position VARCHAR(100),
    ->     JoiningDate DATE,
    ->     TerminationDate DATE
    -> );
Query OK, 0 rows affected (0.02 sec)
```

2. Payroll Table:

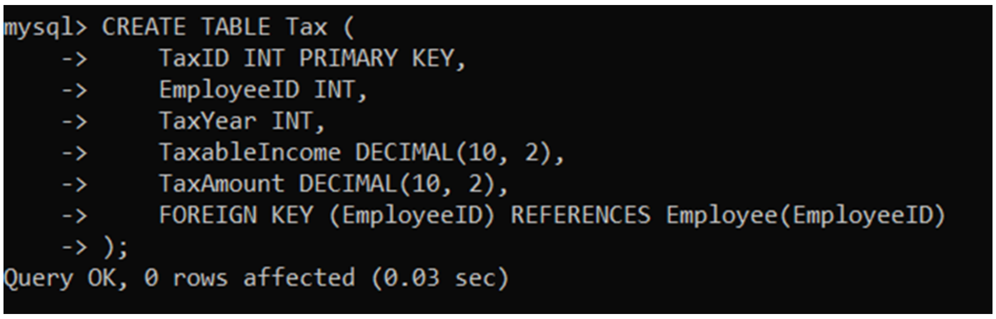
- PayrollID (Primary Key): Unique identifier for each payroll record.
- EmployeeID (Foreign Key): Foreign key referencing the Employee table.
- PayPeriodStartDate: Start date of the pay period.
- PayPeriodEndDate: End date of the pay period.
- BasicSalary: Base salary for the pay period.
- OvertimePay: Additional pay for overtime hours.
- Deductions: Total deductions for the pay period.
- NetSalary: Net salary after deductions.



```
mysql> CREATE TABLE Payroll (
    ->     PayrollID INT PRIMARY KEY,
    ->     EmployeeID INT,
    ->     PayPeriodStartDate DATE,
    ->     PayPeriodEndDate DATE,
    ->     BasicSalary DECIMAL(10, 2),
    ->     OvertimePay DECIMAL(10, 2),
    ->     Deductions DECIMAL(10, 2),
    ->     NetSalary DECIMAL(10, 2),
    ->     FOREIGN KEY (EmployeeID) REFERENCES Employee(EmployeeID)
    -> );
Query OK, 0 rows affected (0.03 sec)
```

3. Tax Table:

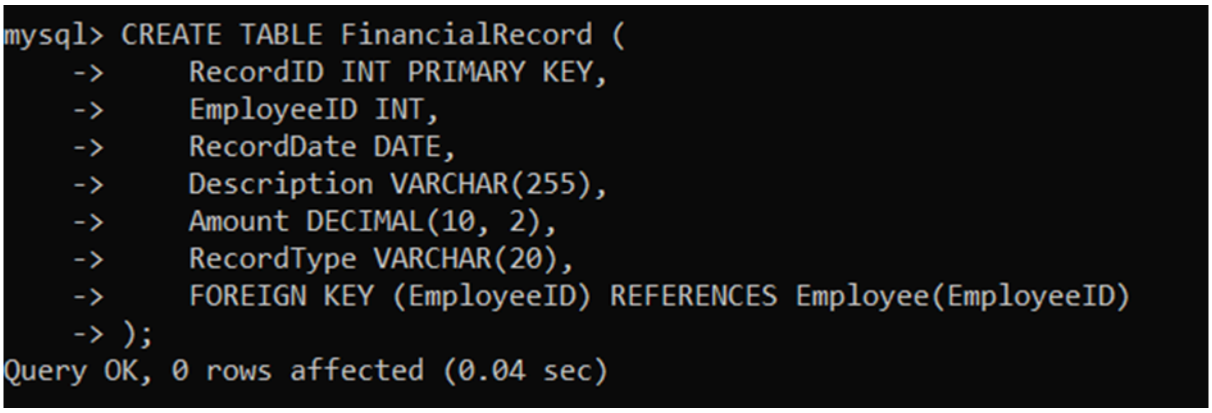
- TaxID (Primary Key): Unique identifier for each tax record.
- EmployeeID (Foreign Key): Foreign key referencing the Employee table.
- TaxYear: Year to which the tax information applies.
- TaxableIncome: Income subject to taxation.
- TaxAmount: Amount of tax to be paid.



```
mysql> CREATE TABLE Tax (
    ->     TaxID INT PRIMARY KEY,
    ->     EmployeeID INT,
    ->     TaxYear INT,
    ->     TaxableIncome DECIMAL(10, 2),
    ->     TaxAmount DECIMAL(10, 2),
    ->     FOREIGN KEY (EmployeeID) REFERENCES Employee(EmployeeID)
    -> );
Query OK, 0 rows affected (0.03 sec)
```

4. FinancialRecord Table:

- RecordID (Primary Key): Unique identifier for each financial record.
- EmployeeID (Foreign Key): Foreign key referencing the Employee table.
- RecordDate: Date of the financial record.
- Description: Description or category of the financial record.
- Amount: Monetary amount of the record (income, expense, etc.).
- RecordType: Type of financial record (income, expense, tax payment, etc.).



```
mysql> CREATE TABLE FinancialRecord (
    ->     RecordID INT PRIMARY KEY,
    ->     EmployeeID INT,
    ->     RecordDate DATE,
    ->     Description VARCHAR(255),
    ->     Amount DECIMAL(10, 2),
    ->     RecordType VARCHAR(20),
    ->     FOREIGN KEY (EmployeeID) REFERENCES Employee(EmployeeID)
    -> );
Query OK, 0 rows affected (0.04 sec)
```

Create the model/entity classes corresponding to the schema within package entity with variables declared private, constructors (default and parametrized) and getters,setters)

Classes:

Employee:

• Properties: EmployeeID, FirstName, LastName, DateOfBirth, Gender, Email, PhoneNumber, Address, Position, JoiningDate, TerminationDate

• Methods: CalculateAge()

```
from datetime import datetime, date

class Employee:
  def __init__(self, EmployeeID=None, FirstName=None, LastName=None, DateOfBirth=None,
Gender=None,
                Email=None, PhoneNumber=None, Address=None, Position=None, JoiningDate=None,
TerminationDate=None):
        self.__EmployeeID = EmployeeID
        self.__FirstName = FirstName
        self.__LastName = LastName
        self.__DateOfBirth = DateOfBirth
        self.__Gender = Gender
        self.__Email = Email
        self.__PhoneNumber = PhoneNumber
        self.__Address = Address
        self.__Position = Position
        self.__JoiningDate = JoiningDate
        self.__TerminationDate = TerminationDate

    # Getters
    def getEmployeeID(self):
        return self.__EmployeeID

    def getFirstName(self):
        return self.__FirstName

    def getLastName(self):
        return self.__LastName

    def getDateOfBirth(self):
        return self.__DateOfBirth

    def getGender(self):
        return self.__Gender

    def getEmail(self):
        return self.__Email

    def getPhoneNumber(self):
        return self.__PhoneNumber

    def getAddress(self):
        return self.__Address

    def getPosition(self):
        return self.__Position

    def getJoiningDate(self):
        return self.__JoiningDate

    def getTerminationDate(self):
        return self.__TerminationDate

    # Setters
    def setEmployeeID(self, EmployeeID):
        self.__EmployeeID = EmployeeID
```

```
  def setFirstName(self, FirstName):
    self.__FirstName = FirstName

  def setLastName(self, LastName):
    self.__LastName = LastName

  def setDateOfBirth(self, DateOfBirth):
    self.__DateOfBirth = DateOfBirth

  def setGender(self, Gender):
    self.__Gender = Gender

  def setEmail(self, Email):
    self.__Email = Email

  def setPhoneNumber(self, PhoneNumber):
    self.__PhoneNumber = PhoneNumber

  def setAddress(self, Address):
    self.__Address = Address

  def setPosition(self, Position):
    self.__Position = Position

  def setJoiningDate(self, JoiningDate):
    self.__JoiningDate = JoiningDate

  def setTerminationDate(self, TerminationDate):
    self.__TerminationDate = TerminationDate

  def calculateAge(self):
    if self.__DateOfBirth:
      today = date.today()
      age = today.year - self.__DateOfBirth.year - ((today.month, today.day) < (self.__DateOfBirth.month,
self.__DateOfBirth.day))
      return age
    else:
      return None
```

Payroll:

• Properties: PayrollID, EmployeeID, PayPeriodStartDate, PayPeriodEndDate, BasicSalary, OvertimePay, Deductions, NetSalary

```
class Payroll:
  def __init__(self, PayrollID=None, EmployeeID=None, PayPeriodStartDate=None, PayPeriodEndDate=None,
          BasicSalary=None, OvertimePay=None, Deductions=None, NetSalary=None):
    self.__PayrollID = PayrollID
    self.__EmployeeID = EmployeeID
    self.__PayPeriodStartDate = PayPeriodStartDate
    self.__PayPeriodEndDate = PayPeriodEndDate
    self.__BasicSalary = BasicSalary
    self.__OvertimePay = OvertimePay
    self.__Deductions = Deductions
    self.__NetSalary = NetSalary

# Getters
def getPayrollID(self):
    return self.__PayrollID

def getEmployeeID(self):
    return self.__EmployeeID

def getPayPeriodStartDate(self):
    return self.__PayPeriodStartDate

def getPayPeriodEndDate(self):
    return self.__PayPeriodEndDate

def getBasicSalary(self):
    return self.__BasicSalary

def getOvertimePay(self):
    return self.__OvertimePay

def getDeductions(self):
    return self.__Deductions

def getNetSalary(self):
    return self.__NetSalary

# Setters
def setPayrollID(self, PayrollID):
    self.__PayrollID = PayrollID

def setEmployeeID(self, EmployeeID):
    self.__EmployeeID = EmployeeID

def setPayPeriodStartDate(self, PayPeriodStartDate):
    self.__PayPeriodStartDate = PayPeriodStartDate

def setPayPeriodEndDate(self, PayPeriodEndDate):
    self.__PayPeriodEndDate = PayPeriodEndDate

def setBasicSalary(self, BasicSalary):
    self.__BasicSalary = BasicSalary

def setOvertimePay(self, OvertimePay):
    self.__OvertimePay = OvertimePay

def setDeductions(self, Deductions):
    self.__Deductions = Deductions
```

```
  def setNetSalary(self, NetSalary):
      self.__NetSalary = NetSalary
```

Tax:

• Properties: TaxID, EmployeeID, TaxYear, TaxableIncome, TaxAmount

```
class Tax:
  def __init__(self, TaxID=None, EmployeeID=None, TaxYear=None, TaxableIncome=None,
TaxAmount=None):
      self.__TaxID = TaxID
      self.__EmployeeID = EmployeeID
      self.__TaxYear = TaxYear
      self.__TaxableIncome = TaxableIncome
      self.__TaxAmount = TaxAmount

  # Getters
  def getTaxID(self):
      return self.__TaxID

  def getEmployeeID(self):
      return self.__EmployeeID

  def getTaxYear(self):
      return self.__TaxYear

  def getTaxableIncome(self):
      return self.__TaxableIncome

  def getTaxAmount(self):
      return self.__TaxAmount

  # Setters
  def setTaxID(self, TaxID):
      self.__TaxID = TaxID

  def setEmployeeID(self, EmployeeID):
      self.__EmployeeID = EmployeeID

  def setTaxYear(self, TaxYear):
      self.__TaxYear = TaxYear

  def setTaxableIncome(self, TaxableIncome):
      self.__TaxableIncome = TaxableIncome

  def setTaxAmount(self, TaxAmount):
      self.__TaxAmount = TaxAmount
```

FinancialRecord:

• Properties: RecordID, EmployeeID, RecordDate, Description, Amount, RecordType

```python
class FinancialRecord:
  def __init__(self, RecordID=None, EmployeeID=None, RecordDate=None, Description=None,
Amount=None, RecordType=None):
      self.__RecordID = RecordID
      self.__EmployeeID = EmployeeID
      self.__RecordDate = RecordDate
      self.__Description = Description
      self.__Amount = Amount
      self.__RecordType = RecordType

  # Getters
  def getRecordID(self):
      return self.__RecordID

  def getEmployeeID(self):
      return self.__EmployeeID

  def getRecordDate(self):
      return self.__RecordDate

  def getDescription(self):
      return self.__Description

  def getAmount(self):
      return self.__Amount

  def getRecordType(self):
      return self.__RecordType

  # Setters
  def setRecordID(self, RecordID):
      self.__RecordID = RecordID

  def setEmployeeID(self, EmployeeID):
      self.__EmployeeID = EmployeeID

  def setRecordDate(self, RecordDate):
      self.__RecordDate = RecordDate

  def setDescription(self, Description):
      self.__Description = Description

  def setAmount(self, Amount):
      self.__Amount = Amount

  def setRecordType(self, RecordType):
      self.__RecordType = RecordType
```

EmployeeService (implements IEmployeeService):

• Methods: GetEmployeeById, GetAllEmployees, AddEmployee, UpdateEmployee, RemoveEmployee

```python
class IEmployeeService(ABC):
   @abstractmethod
   def GetEmployeeById(self, EmployeeID):
       pass

   @abstractmethod
   def GetAllEmployees(self):
       pass

   @abstractmethod
   def AddEmployee(self, employee):
       pass

   @abstractmethod
   def UpdateEmployee(self, employee):
       pass

   @abstractmethod
   def RemoveEmployee(self, EmployeeID):
       pass

class EmployeeService(IEmployeeService):
   def __init__(self):
       # Initialize data store for employees
       self.__employees = {}

   def GetEmployeeById(self, EmployeeID):
       return self.__employees.get(EmployeeID)

   def GetAllEmployees(self):
       return list(self.__employees.values())

   def AddEmployee(self, employee):
       self.__employees[employee.getEmployeeID()] = employee

   def UpdateEmployee(self, employee):
       if employee.getEmployeeID() in self.__employees:
           self.__employees[employee.getEmployeeID()] = employee
       else:
           raise ValueError("Employee not found")

   def RemoveEmployee(self, EmployeeID):
       if EmployeeID in self.__employees:
           del self.__employees[EmployeeID]
       else:
           raise ValueError("Employee not found")
```

PayrollService (implements IPayrollService):
• Methods: GeneratePayroll, GetPayrollById, GetPayrollsForEmployee, GetPayrollsForPeriod

```
class IPayrollService(ABC):
    @abstractmethod
    def GeneratePayroll(self, employee, pay_period_start, pay_period_end):
        pass

    @abstractmethod
    def GetPayrollById(self, PayrollID):
        pass

    @abstractmethod
    def GetPayrollsForEmployee(self, EmployeeID):
        pass

    @abstractmethod
    def GetPayrollsForPeriod(self, start_date, end_date):
        pass

class PayrollService(IPayrollService):
    def __init__(self):
        # Initialize data store for payrolls
        self.__payrolls = {}

    def GeneratePayroll(self, employee, pay_period_start, pay_period_end):
        # Perform payroll generation logic here
        # This can include calculating basic salary, overtime pay, deductions,
net salary, etc.
        # For simplicity, we will just create a placeholder payroll object
with basic information
        payroll_id = len(self.__payrolls) + 1
        payroll = {
            'PayrollID': payroll_id,
            'EmployeeID': employee.getEmployeeID(),
            'PayPeriodStartDate': pay_period_start,
            'PayPeriodEndDate': pay_period_end,
            'BasicSalary': 5000.00,  # Placeholder values
            'OvertimePay': 1000.00,
            'Deductions': 500.00,
            'NetSalary': 5500.00
        }
        self.__payrolls[payroll_id] = payroll
        return payroll

    def GetPayrollById(self, PayrollID):
        return self.__payrolls.get(PayrollID)

    def GetPayrollsForEmployee(self, EmployeeID):
        return [payroll for payroll in self.__payrolls.values() if 
payroll['EmployeeID'] == EmployeeID]

    def GetPayrollsForPeriod(self, start_date, end_date):
        return [payroll for payroll in self.__payrolls.values() if start_date 
<= payroll['PayPeriodEndDate'] <= end_date]
```

TaxService (implements ITaxService):

• Methods: CalculateTax, GetTaxById, GetTaxesForEmployee, GetTaxesForYear

```
class ITaxService(ABC):
    @abstractmethod
    def CalculateTax(self, employee, taxable_income):
        pass

    @abstractmethod
    def GetTaxById(self, TaxID):
        pass

    @abstractmethod
    def GetTaxesForEmployee(self, EmployeeID):
        pass

    @abstractmethod
    def GetTaxesForYear(self, year):
        pass

class TaxService(ITaxService):
    def __init__(self):
        # Initialize data store for taxes
        self.__taxes = {}

    def CalculateTax(self, employee, taxable_income):
        # Perform tax calculation logic here
        # For simplicity, we will just create a placeholder tax object with 
basic information
        tax_id = len(self.__taxes) + 1
        tax_amount = taxable_income * 0.2  # Placeholder tax calculation
        tax = {
            'TaxID': tax_id,
            'EmployeeID': employee.getEmployeeID(),
            'TaxYear': datetime.now().year,  # Placeholder tax year (current 
year)
            'TaxableIncome': taxable_income,
            'TaxAmount': tax_amount
        }
        self.__taxes[tax_id] = tax
        return tax

    def GetTaxById(self, TaxID):
        return self.__taxes.get(TaxID)

    def GetTaxesForEmployee(self, EmployeeID):
        return [tax for tax in self.__taxes.values() if tax['EmployeeID'] ==
EmployeeID]

    def GetTaxesForYear(self, year):
        return [tax for tax in self.__taxes.values() if tax['TaxYear'] ==
year]
```

FinancialRecordService (implements IFinancialRecordService):

• Methods: AddFinancialRecord, GetFinancialRecordById, GetFinancialRecordsForEmployee, GetFinancialRecordsForDate

```
class IFinancialRecordService(ABC):
    @abstractmethod
    def AddFinancialRecord(self, financial_record):
        pass

    @abstractmethod
    def GetFinancialRecordById(self, RecordID):
        pass

    @abstractmethod
    def GetFinancialRecordsForEmployee(self, EmployeeID):
        pass

    @abstractmethod
    def GetFinancialRecordsForDate(self, record_date):
        pass

class FinancialRecordService(IFinancialRecordService):
    def __init__(self):
        # Initialize data store for financial records
        self.__financial_records = {}

    def AddFinancialRecord(self, financial_record):
        record_id = len(self.__financial_records) + 1
        financial_record.setRecordID(record_id)
        self.__financial_records[record_id] = financial_record

    def GetFinancialRecordById(self, RecordID):
        return self.__financial_records.get(RecordID)

    def GetFinancialRecordsForEmployee(self, EmployeeID):
        return [record for record in self.__financial_records.values() if
record.getEmployeeID() == EmployeeID]

    def GetFinancialRecordsForDate(self, record_date):
        return [record for record in self.__financial_records.values() if
record.getRecordDate() == record_date]
```

DatabaseContext:

• A class responsible for handling database connections and interactions.

```
class DatabaseContext:
   def __init__(self, host, user, password, database):
       self.host = host
       self.user = user
       self.password = password
       self.database = database
       self.connection = None

   def connect(self):
       try:
           self.connection = mysql.connector.connect(
               host=self.host,
               user=self.user,
               password=self.password,
               database=self.database
           )
           print("Connected to MySQL database")
       except mysql.connector.Error as err:
           print(f"Error: {err}")

   def disconnect(self):
       if self.connection:
           self.connection.close()
           print("Disconnected from MySQL database")

   def execute_query(self, query, params=None):
       cursor = self.connection.cursor()
       try:
           cursor.execute(query, params)
           self.connection.commit()
           print("Query executed successfully")
           return cursor
       except mysql.connector.Error as err:
           print(f"Error: {err}")
           self.connection.rollback()
       finally:
           cursor.close()

   def fetch_all(self, cursor):
       return cursor.fetchall()

   def fetch_one(self, cursor):
       return cursor.fetchone()
```

ValidationService:

• A class for input validation and business rule enforcement.

```
class ValidationService:
   @staticmethod
```

```
    def validate_employee(employee):
        errors = []

        if not employee.getFirstName():
            errors.append("First name is required")

        if not employee.getLastName():
            errors.append("Last name is required")

        if not employee.getDateOfBirth():
            errors.append("Date of birth is required")

        if not employee.getGender():
            errors.append("Gender is required")

        if not employee.getEmail():
            errors.append("Email is required")

        if not employee.getPhoneNumber():
            errors.append("Phone number is required")

        if not employee.getAddress():
            errors.append("Address is required")

        if not employee.getPosition():
            errors.append("Position is required")

        if not employee.getJoiningDate():
            errors.append("Joining date is required")

        return errors

    @staticmethod
    def validate_financial_record(financial_record):
        errors = []

        if not financial_record.getDescription():
            errors.append("Description is required")

        if financial_record.getAmount() is None or
financial_record.getAmount() <= 0:
            errors.append("Amount must be greater than zero")

        if not financial_record.getRecordDate():
            errors.append("Record date is required")
```

ReportGenerator:

• A class for generating various reports based on payroll, tax, and financial record data.

```
class ReportGenerator:
    @staticmethod
    def generate_payroll_report(payrolls):
        report = "Payroll Report:\n"
        for payroll in payrolls:
            report += f"Payroll ID: {payroll.getPayrollID()}, Employee ID: 
{payroll.getEmployeeID()}, " \
                      f"Period Start Date: {payroll.getPayPeriodStartDate()}, 
" \
                      f"Period End Date: {payroll.getPayPeriodEndDate()}, " \
                      f"Basic Salary: {payroll.getBasicSalary()}, " \
                      f"Overtime Pay: {payroll.getOvertimePay()}, " \
                      f"Deductions: {payroll.getDeductions()}, " \
                      f"Net Salary: {payroll.getNetSalary()}\n"
        return report

    @staticmethod
    def generate_tax_report(taxes):
        report = "Tax Report:\n"
        for tax in taxes:
            report += f"Tax ID: {tax.getTaxID()}, Employee ID: 
{tax.getEmployeeID()}, " \
                      f"Tax Year: {tax.getTaxYear()}, Taxable Income: 
{tax.getTaxableIncome()}, " \
                      f"Tax Amount: {tax.getTaxAmount()}\n"
        return report

    @staticmethod
    def generate_financial_record_report(financial_records):
        report = "Financial Record Report:\n"
        for record in financial_records:
            report += f"Record ID: {record.getRecordID()}, Employee ID: 
{record.getEmployeeID()}, " \
                      f"Record Date: {record.getRecordDate()}, Description: 
{record.getDescription()}, " \
                      f"Amount: {record.getAmount()}, Record Type: 
{record.getRecordType()}\n"
        return report
```

Interfaces/Abstract class:
IEmployeeService:
• GetEmployeeById(employeeId)
• GetAllEmployees()
• AddEmployee(employeeData)
• UpdateEmployee(employeeData)
• RemoveEmployee(employeeId)

```
from abc import ABC, abstractmethod

class IEmployeeService(ABC):
    @abstractmethod
    def GetEmployeeById(self, employeeId):
        pass

    @abstractmethod
    def GetAllEmployees(self):
        pass

    @abstractmethod
    def AddEmployee(self, employeeData):
        pass

    @abstractmethod
    def UpdateEmployee(self, employeeData):
        pass

    @abstractmethod
    def RemoveEmployee(self, employeeId):
        pass
```

IPayrollService:
• GeneratePayroll(employeeId, startDate, endDate)
• GetPayrollById(payrollId)
• GetPayrollsForEmployee(employeeId)
• GetPayrollsForPeriod(startDate, endDate)

```
class IPayrollService(ABC):
    @abstractmethod
    def GeneratePayroll(self, employeeId, startDate, endDate):
        pass

    @abstractmethod
    def GetPayrollById(self, payrollId):
        pass

    @abstractmethod
    def GetPayrollsForEmployee(self, employeeId):
        pass

    @abstractmethod
    def GetPayrollsForPeriod(self, startDate, endDate):
        pass
```

ITaxService:
• CalculateTax(employeeId, taxYear)
• GetTaxById(taxId)
• GetTaxesForEmployee(employeeId)
• GetTaxesForYear(taxYear)

```
class ITaxService(ABC):
    @abstractmethod
    def CalculateTax(self, employeeId, taxYear):
        pass

    @abstractmethod
    def GetTaxById(self, taxId):
        pass

    @abstractmethod
    def GetTaxesForEmployee(self, employeeId):
        pass

    @abstractmethod
    def GetTaxesForYear(self, taxYear):
        pass
```

IFinancialRecordService:
• AddFinancialRecord(employeeId, description, amount, recordType)
• GetFinancialRecordById(recordId)
• GetFinancialRecordsForEmployee(employeeId)
• GetFinancialRecordsForDate(recordDate)

```
class IFinancialRecordService(ABC):
    @abstractmethod
    def AddFinancialRecord(self, employeeId, description, amount, recordType):
        pass

    @abstractmethod
    def GetFinancialRecordById(self, recordId):
        pass

    @abstractmethod
    def GetFinancialRecordsForEmployee(self, employeeId):
        pass

    @abstractmethod
    def GetFinancialRecordsForDate(self, recordDate):
        pass
```

Connect your application to the SQL database:
• Create a connection string that includes the necessary information to connect to your SQL Server
database. This includes the server name, database name, authentication credentials, and any other relevant settings.
• Use the SqlConnection class to establish a connection to the SQL Server database.
• Once the connection is open, you can use the SqlCommand class to execute SQL queries.

```
import mysql.connector

class DatabaseContext:
    def __init__(self, host, user, password, database):
```

```python
        self.host = host
        self.user = user
        self.password = password
        self.database = database
        self.connection = None

    def connect(self):
        try:
            self.connection = mysql.connector.connect(
                host=self.host,
                user=self.user,
                password=self.password,
                database=self.database
            )
            print("Connected to MySQL database")
        except mysql.connector.Error as err:
            print(f"Error: {err}")

    def disconnect(self):
        if self.connection:
            self.connection.close()
            print("Disconnected from MySQL database")

    def execute_query(self, query, params=None):
        cursor = self.connection.cursor()
        try:
            cursor.execute(query, params)
            self.connection.commit()
            print("Query executed successfully")
            return cursor
        except mysql.connector.Error as err:
            print(f"Error: {err}")
            self.connection.rollback()
        finally:
            cursor.close()

    def fetch_all(self, cursor):
        return cursor.fetchall()

    def fetch_one(self, cursor):
        return cursor.fetchone()

# Example usage:
db_context = DatabaseContext(host='localhost', user='root',
password='Siva@2003', database='payxpert')
db_context.connect()
```

Custom Exceptions:

EmployeeNotFoundException:

• Thrown when attempting to access or perform operations on a non-existing employee.

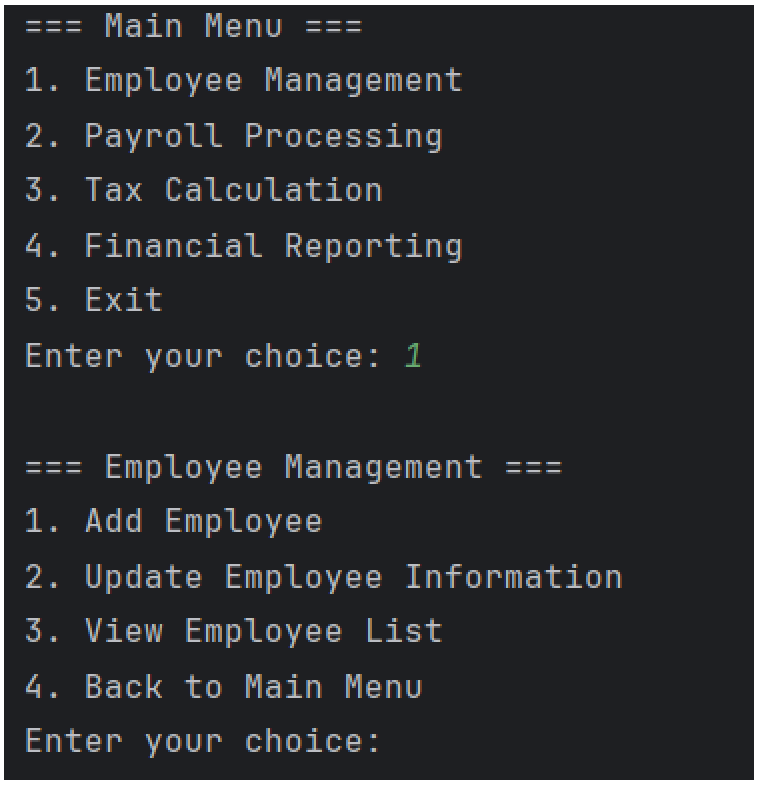
PayrollGenerationException:

• Thrown when there is an issue with generating payroll for an employee.

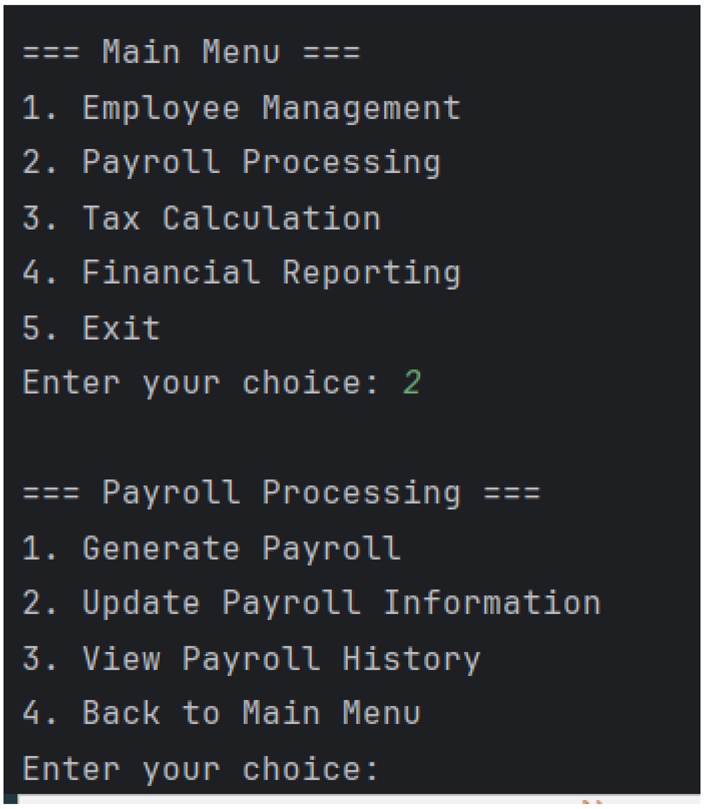
TaxCalculationException:
• Thrown when there is an error in calculating taxes for an employee.
FinancialRecordException:
• Thrown when there is an issue with financial record management.
InvalidInputException:
• Thrown when input data doesn't meet the required criteria.
DatabaseConnectionException:
• Thrown when there is a problem establishing or maintaining a connection with the database.

```
class EmployeeNotFoundException(Exception):
   pass

class PayrollGenerationException(Exception):
   pass

class TaxCalculationException(Exception):
   pass

class FinancialRecordException(Exception):
   pass

class InvalidInputException(Exception):
   pass

class DatabaseConnectionException(Exception):
   pass
```

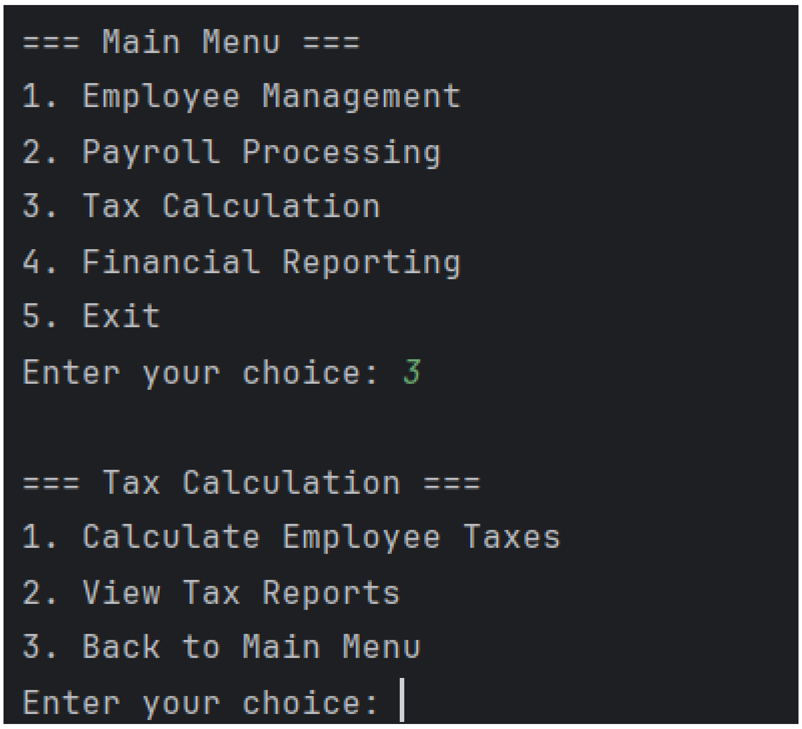
OUTPUT :



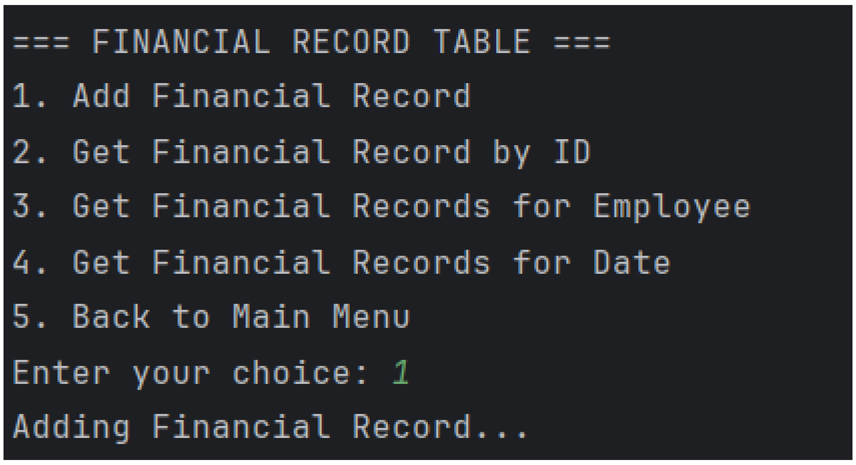
```
=== Main Menu ===
1. Employee Management
2. Payroll Processing
3. Tax Calculation
4. Financial Reporting
5. Exit
Enter your choice: 1

=== Employee Management ===
1. Add Employee
2. Update Employee Information
3. View Employee List
4. Back to Main Menu
Enter your choice:
```



```
=== Main Menu ===
1. Employee Management
2. Payroll Processing
3. Tax Calculation
4. Financial Reporting
5. Exit
Enter your choice: 2

=== Payroll Processing ===
1. Generate Payroll
2. Update Payroll Information
3. View Payroll History
4. Back to Main Menu
Enter your choice:
```



```
=== Main Menu ===
1. Employee Management
2. Payroll Processing
3. Tax Calculation
4. Financial Reporting
5. Exit
Enter your choice: 3

=== Tax Calculation ===
1. Calculate Employee Taxes
2. View Tax Reports
3. Back to Main Menu
Enter your choice: 
```



```
=== FINANCIAL RECORD TABLE ===
1. Add Financial Record
2. Get Financial Record by ID
3. Get Financial Records for Employee
4. Get Financial Records for Date
5. Back to Main Menu
Enter your choice: 1
Adding Financial Record...
```