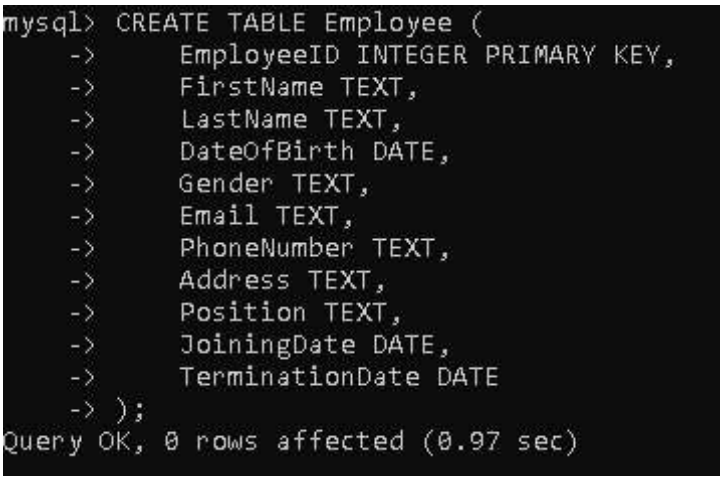
PayXpert, The Payroll Management System

SQL Tables:

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).



```
+-----------------+------+------+-----+---------+-------+
| Field           | Type | Null | Key | Default | Extra |
+-----------------+------+------+-----+---------+-------+
| EmployeeID      | int  | NO   | PRI | NULL    |       |
| FirstName       | text | YES  |     | NULL    |       |
| LastName        | text | YES  |     | NULL    |       |
| DateOfBirth     | date | YES  |     | NULL    |       |
| Gender          | text | YES  |     | NULL    |       |
| Email           | text | YES  |     | NULL    |       |
| PhoneNumber     | text | YES  |     | NULL    |       |
| Address         | text | YES  |     | NULL    |       |
| Position        | text | YES  |     | NULL    |       |
| JoiningDate     | date | YES  |     | NULL    |       |
| TerminationDate | date | YES  |     | NULL    |       |
+-----------------+------+------+-----+---------+-------+
```

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 INTEGER PRIMARY KEY,
    ->     EmployeeID INTEGER,
    ->     PayPeriodStartDate DATE,
    ->     PayPeriodEndDate DATE,
    ->     BasicSalary REAL,
    ->     OvertimePay REAL,
    ->     Deductions REAL,
    ->     NetSalary REAL,
    ->     FOREIGN KEY (EmployeeID) REFERENCES Employee(EmployeeID)
    -> );
Query OK, 0 rows affected (1.88 sec)
```

```
+--------------------+--------+------+-----+---------+-------+
| Field              | Type   | Null | Key | Default | Extra |
+--------------------+--------+------+-----+---------+-------+
| PayrollID          | int    | NO   | PRI | NULL    |       |
| EmployeeID         | int    | YES  | MUL | NULL    |       |
| PayPeriodStartDate | date   | YES  |     | NULL    |       |
| PayPeriodEndDate   | date   | YES  |     | NULL    |       |
| BasicSalary        | double | YES  |     | NULL    |       |
| OvertimePay        | double | YES  |     | NULL    |       |
| Deductions         | double | YES  |     | NULL    |       |
| NetSalary          | double | YES  |     | NULL    |       |
+--------------------+--------+------+-----+---------+-------+
```

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 INTEGER PRIMARY KEY,
    ->     EmployeeID INTEGER,
    ->     TaxYear INTEGER,
    ->     TaxableIncome REAL,
    ->     TaxAmount REAL,
    ->     FOREIGN KEY (EmployeeID) REFERENCES Employee(EmployeeID)
    -> );
Query OK, 0 rows affected (1.24 sec)
```

```
+---------------+--------+------+-----+---------+-------+
| Field         | Type   | Null | Key | Default | Extra |
+---------------+--------+------+-----+---------+-------+
| TaxID         | int    | NO   | PRI | NULL    |       |
| EmployeeID    | int    | YES  | MUL | NULL    |       |
| TaxYear       | int    | YES  |     | NULL    |       |
| TaxableIncome | double | YES  |     | NULL    |       |
| TaxAmount     | double | YES  |     | NULL    |       |
+---------------+--------+------+-----+---------+-------+
```

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 INTEGER PRIMARY KEY,
    ->     EmployeeID INTEGER,
    ->     RecordDate DATE,
    ->     Description TEXT,
    ->     Amount REAL,
    ->     RecordType TEXT,
    ->     FOREIGN KEY (EmployeeID) REFERENCES Employee(EmployeeID)
    -> );
Query OK, 0 rows affected (0.94 sec)
```

```
+-------------+--------+------+-----+---------+-------+
| Field       | Type   | Null | Key | Default | Extra |
+-------------+--------+------+-----+---------+-------+
| RecordID    | int    | NO   | PRI | NULL    |       |
| EmployeeID  | int    | YES  | MUL | NULL    |       |
| RecordDate  | date   | YES  |     | NULL    |       |
| Description | text   | YES  |     | NULL    |       |
| Amount      | double | YES  |     | NULL    |       |
| RecordType  | text   | YES  |     | NULL    |       |
+-------------+--------+------+-----+---------+-------+
```

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()

```python
class Employee:
    def __init__(self, employee_id, first_name, last_name, date_of_birth, gender, email, phone_number, address,
                 position, joining_date, termination_date=None):
        self.__employee_id = employee_id
        self.__first_name = first_name
        self.__last_name = last_name
        self.__date_of_birth = date_of_birth
        self.__gender = gender
        self.__email = email
        self.__phone_number = phone_number
        self.__address = address
        self.__position = position
        self.__joining_date = joining_date
        self.__termination_date = termination_date
```

```python
    def get_employee_id(self):
        return self.__employee_id

    def set_employee_id(self, employee_id):
        self.__employee_id = employee_id

    def get_first_name(self):
        return self.__first_name

    def set_first_name(self, first_name):
        self.__first_name = first_name

    def get_last_name(self):
        return self.__last_name

    def set_last_name(self, last_name):
        self.__last_name = last_name

    def get_date_of_birth(self):
        return self.__date_of_birth

    def set_date_of_birth(self, date_of_birth):
        self.__date_of_birth = date_of_birth
```

```python
    def get_gender(self):
        return self.__gender

    def set_gender(self, gender):
        self.__gender = gender
```

```
def get_email(self):
    return self.__email

def set_email(self, email):
    self.__email = email

def get_phone_number(self):
    return self.__phone_number

def set_phone_number(self, phone_number):
    self.__phone_number = phone_number

def get_address(self):
    return self.__address

def set_address(self, address):
    self.__address = address

def get_position(self):
    return self.__position

def set_position(self, position):
    self.__position = position
```

```
def get_joining_date(self):
    return self.__joining_date

def set_joining_date(self, joining_date):
    self.__joining_date = joining_date

def get_termination_date(self):
    return self.__termination_date

def set_termination_date(self, termination_date):
    self.__termination_date = termination_date
```

Payroll:

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

```
class Payroll:
    def __init__(self, payroll_id, employee_id, pay_period_start_date, pay_period_end_date, basic_salary, overtime_pay,
                 deductions, net_salary):
        self.__payroll_id = payroll_id
        self.__employee_id = employee_id
        self.__pay_period_start_date = pay_period_start_date
        self.__pay_period_end_date = pay_period_end_date
        self.__basic_salary = basic_salary
        self.__overtime_pay = overtime_pay
        self.__deductions = deductions
        self.__net_salary = net_salary
```

```
def get_payroll_id(self):
    return self.__payroll_id

def set_payroll_id(self, payroll_id):
    self.__payroll_id = payroll_id

def get_employee_id(self):
    return self.__employee_id

def set_employee_id(self, employee_id):
    self.__employee_id = employee_id

def get_pay_period_start_date(self):
    return self.__pay_period_start_date

def set_pay_period_start_date(self, pay_period_start_date):
    self.__pay_period_start_date = pay_period_start_date

def get_pay_period_end_date(self):
    return self.__pay_period_end_date

def set_pay_period_end_date(self, pay_period_end_date):
    self.__pay_period_end_date = pay_period_end_date
```

```
def get_basic_salary(self):
    return self.__basic_salary

def set_basic_salary(self, basic_salary):
    self.__basic_salary = basic_salary

def get_overtime_pay(self):
    return self.__overtime_pay

def set_overtime_pay(self, overtime_pay):
    self.__overtime_pay = overtime_pay

def get_deductions(self):
    return self.__deductions

def set_deductions(self, deductions):
    self.__deductions = deductions

def get_net_salary(self):
    return self.__net_salary

def set_net_salary(self, net_salary):
    self.__net_salary = net_salary
```

Tax:

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

```
class Tax:
    def __init__(self, tax_id, employee_id, tax_year, taxable_income, tax_amount):
        self.__tax_id = tax_id
        self.__employee_id = employee_id
        self.__tax_year = tax_year
        self.__taxable_income = taxable_income
        self.__tax_amount = tax_amount
```

```
def get_tax_id(self):
    return self.__tax_id

def set_tax_id(self, tax_id):
    self.__tax_id = tax_id
```

```
def get_employee_id(self):
    return self.__employee_id

def set_employee_id(self, employee_id):
    self.__employee_id = employee_id

def get_tax_year(self):
    return self.__tax_year

def set_tax_year(self, tax_year):
    self.__tax_year = tax_year

def get_taxable_income(self):
    return self.__taxable_income

def set_taxable_income(self, taxable_income):
    self.__taxable_income = taxable_income

def get_tax_amount(self):
    return self.__tax_amount

def set_tax_amount(self, tax_amount):
    self.__tax_amount = tax_amount
```

FinancialRecord:

- Properties: RecordID, EmployeeID, RecordDate, Description, Amount, RecordType
- EmployeeService (implements IEmployeeService):
- Methods: GetEmployeeById, GetAllEmployees, AddEmployee, UpdateEmployee, RemoveEmployee

```python
class FinancialRecord:
    def __init__(self, record_id, employee_id, record_date, description, amount, record_type):
        self.__record_id = record_id
        self.__employee_id = employee_id
        self.__record_date = record_date
        self.__description = description
        self.__amount = amount
        self.__record_type = record_type
```

```python
def get_record_id(self):
    return self.__record_id

def set_record_id(self, record_id):
    self.__record_id = record_id

def get_employee_id(self):
    return self.__employee_id

def set_employee_id(self, employee_id):
    self.__employee_id = employee_id

def get_record_date(self):
    return self.__record_date

def set_record_date(self, record_date):
    self.__record_date = record_date

def get_description(self):
    return self.__description

def set_description(self, description):
    self.__description = description
```

```python
def get_amount(self):
    return self.__amount

def set_amount(self, amount):
    self.__amount = amount

def get_record_type(self):
    return self.__record_type

def set_record_type(self, record_type):
    self.__record_type = record_type
```

PayrollService (implements IPayrollService):

- Methods: GeneratePayroll, GetPayrollById, GetPayrollsForEmployee, GetPayrollsForPeriod

```
class EmployeeService(IEmployeeService):
    def __init__(self):
        self.connection = DBConnUtil.get_connection()

    2 usages
    def get_employee_by_id(self, employee_id):
        try:
            cursor = self.connection.cursor()
            cursor.execute("SELECT * FROM Employee WHERE EmployeeID = %s", (employee_id,))
            employee_data = cursor.fetchone()
            cursor.close()

            if employee_data:
                return Employee(*employee_data)
            else:
                raise EmployeeNotFoundException("Employee not found with ID: " + str(employee_id))
        except Exception as e:
            raise DatabaseConnectionException("Error retrieving employee from database: " + str(e))
```

```
def get_all_employees(self):
    try:
        cursor = self.connection.cursor()
        cursor.execute("SELECT * FROM Employee")
        employees_data = cursor.fetchall()
        cursor.close()


        employees = []
        for employee_data in employees_data:
            employees.append(Employee(*employee_data))

        return employees
    except Exception as e:
        raise DatabaseConnectionException("Error retrieving employees from database: " + str(e))
```

```python
def add_employee(self, employee_data):
    try:
        if not employee_data:
            raise InvalidInputException("Employee data is empty")

        cursor = self.connection.cursor()
        insert_query = ("INSERT INTO Employee (FirstName, LastName, DateOfBirth, Gender, "
                        "Email, PhoneNumber, Address, Position, JoiningDate, TerminationDate) VALUES (%s, %s, %s, %s, %s, %s, %s, %s, %s, %s)")
        cursor.execute(insert_query, (
            employee_data.first_name, employee_data.last_name, employee_data.date_of_birth, employee_data.gender,
            employee_data.email, employee_data.phone_number, employee_data.address, employee_data.position,
            employee_data.joining_date, employee_data.termination_date))
        self.connection.commit()
        cursor.close()
        self.connection.close()

        return "Employee added successfully"
    except Exception as e:
        raise DatabaseConnectionException("Error adding employee to database: " + str(e))
```

```python
def update_employee(self, employee_data):
    try:
        if not employee_data:
            raise InvalidInputException("Employee data is empty")

        #connection = get_connection()
        cursor = self.connection.cursor()
        update_query = "UPDATE Employee SET FirstName=%s, LastName=%s, DateOfBirth=%s, Gender=%s, Email=%s, PhoneNumber=%s, Address=%s,
        cursor.execute(update_query, (
            employee_data.first_name, employee_data.last_name, employee_data.date_of_birth, employee_data.gender,
            employee_data.email, employee_data.phone_number, employee_data.address, employee_data.position,
            employee_data.joining_date, employee_data.termination_date, employee_data.employee_id))
        self.connection.commit()
        cursor.close()
        self.connection.close()

        return "Employee updated successfully"
    except Exception as e:
        raise DatabaseConnectionException("Error updating employee in database: " + str(e))
```

```python
def remove_employee(self, employee_id):
    try:
        #connection = get_connection()
        cursor = self.connection.cursor()
        delete_query = "DELETE FROM Employee WHERE EmployeeID = %s"
        cursor.execute(delete_query, (employee_id,))
        self.connection.commit()
        cursor.close()
        self.connection.close()

        return "Employee removed successfully"
    except Exception as e:
        raise DatabaseConnectionException("Error removing employee from database: " + str(e))
```

TaxService (implements ITaxService):

- Methods: CalculateTax, GetTaxById, GetTaxesForEmployee, GetTaxesForYear

```python
class TaxService(ITaxService):
    def __init__(self):
        self.connection=DBConnUtil.get_connection()

    1 usage
    def calculate_tax(self, employee_id, tax_year):
        try:
            #connection = get_connection()
            cursor = self.connection.cursor()
            tax_data = cursor.execute("SELECT * FROM Tax WHERE EmployeeID = %s AND TaxYear = %s", (employee_id, tax_year))
            cursor.close()
            self.connection.close()

            if tax_data:
                return Tax(*tax_data)
            else:
                raise TaxCalculationException("Tax data not found for employee ID: " + str(employee_id) + " and Tax Year: " + str(tax_year))
        except Exception as e:
            raise DatabaseConnectionException("Error calculating tax: " + str(e))
```

```python
def get_tax_by_id(self, tax_id):
    try:
        #connection = get_connection()
        cursor = self.connection.cursor()
        cursor.execute("SELECT * FROM Tax WHERE TaxID = %s", (tax_id,))
        tax_data = cursor.fetchone()
        cursor.close()
        self.connection.close()

        if tax_data:
            return Tax(*tax_data)
        else:
            raise TaxCalculationException("Tax not found with ID: " + str(tax_id))
    except Exception as e:
        raise DatabaseConnectionException("Error retrieving tax from database: " + str(e))
```

```python
def get_taxes_for_employee(self, employee_id):
    try:
        #connection = get_connection()
        cursor = self.connection.cursor()
        cursor.execute("SELECT * FROM Tax WHERE EmployeeID = %s", (employee_id,))
        taxes_data = cursor.fetchall()
        cursor.close()
        self.connection.close()

        taxes = []
        for tax_data in taxes_data:
            taxes.append(Tax(*tax_data))

        return taxes
    except Exception as e:
        raise DatabaseConnectionException("Error retrieving taxes from database: " + str(e))
```

```python
def get_taxes_for_year(self, tax_year):
    try:
        #connection = get_connection()
        cursor = self.connection.cursor()
        cursor.execute("SELECT * FROM Tax WHERE TaxYear = %s", (tax_year,))
        taxes_data = cursor.fetchall()
        cursor.close()
        self.connection.close()

        taxes = []
        for tax_data in taxes_data:
            taxes.append(Tax(*tax_data))

        return taxes
    except Exception as e:
        raise DatabaseConnectionException("Error retrieving taxes from database: " + str(e))
```

FinancialRecordService (implements IFinancialRecordService):

- Methods: AddFinancialRecord, GetFinancialRecordById, GetFinancialRecordsForEmployee, GetFinancialRecordsForDate

```python
class FinancialRecordService(IFinancialRecordService):
    def __init__(self):
        self.connection = DBConnUtil.get_connection()

    1 usage
    def add_financial_record(self, employee_id, description, amount, record_type):
        try:
            if not description or not amount or not record_type:
                raise InvalidInputException("Invalid input data for adding financial record")

            #connection = get_connection()
            cursor = self.connection.cursor()
            insert_query = "INSERT INTO FinancialRecord (EmployeeID, RecordDate, Description, Amount, RecordType) VALUES (%s, NOW(), %s, %s, %s)"
            cursor.execute(insert_query, (employee_id, description, amount, record_type))
            self.connection.commit()
            cursor.close()
            self.connection.close()

            return "Financial record added successfully"
        except Exception as e:
            raise DatabaseConnectionException("Error adding financial record: " + str(e))
```

```python
def get_financial_record_by_id(self, record_id):
    try:
        cursor = self.connection.cursor()
        cursor.execute("SELECT * FROM FinancialRecord WHERE RecordID = %s", (record_id,))
        financial_record_data = cursor.fetchone()
        cursor.close()
        self.connection.close()

        if financial_record_data:
            financialrec = FinancialRecord(*financial_record_data)
            return financialrec
        else:
            raise FinancialRecordException("Financial record not found with ID: " + str(record_id))
    except Exception as e:
        raise DatabaseConnectionException("Error retrieving financial record from database: " + str(e))
```

```python
def get_financial_records_for_employee(self, employee_id):
    try:
        cursor = self.connection.cursor()
        cursor.execute("SELECT * FROM FinancialRecord WHERE EmployeeID = %s", (employee_id,))
        financial_records_data = cursor.fetchall()
        cursor.close()
        self.connection.close()

        financial_records = []
        for financial_record_data in financial_records_data:
            financial_records.append(FinancialRecord(*financial_record_data))

        return financial_records
    except Exception as e:
        raise DatabaseConnectionException("Error retrieving financial records from database: " + str(e))
```

```python
def get_financial_records_for_date(self, record_date):
    try:
        cursor = self.connection.cursor()
        cursor.execute("SELECT * FROM FinancialRecord WHERE RecordDate = %s", (record_date,))
        financial_records_data = cursor.fetchall()
        cursor.close()
        self.connection.close()

        financial_records = []
        for financial_record_data in financial_records_data:
            financial_records.append(FinancialRecord(*financial_record_data))

        return financial_records
    except Exception as e:
        raise DatabaseConnectionException("Error retrieving financial records from database: " + str(e))
```

Interfaces/Abstract class:

IEmployeeService:

- GetEmployeeById(employeeId)
- GetAllEmployees()
- AddEmployee(employeeData)
- UpdateEmployee(employeeData)
- RemoveEmployee(employeeId)

```python
from abc import ABC, abstractmethod
from entity.employee import Employee
2 usages
class IEmployeeService(ABC):
    @abstractmethod
    def get_employee_by_id(self, employee_id: int) -> Employee:
        pass

    @abstractmethod
    def get_all_employees(self) -> list[Employee]:
        pass

    @abstractmethod
    def add_employee(self, employee_data: dict) -> bool:
        pass

    @abstractmethod
    def update_employee(self, employee_data: dict) -> bool:
        pass
```

```python
    @abstractmethod
    def remove_employee(self, employee_id: int) -> bool:
        pass
```

IPayrollService:

- GeneratePayroll(employeeId, startDate, endDate)
- GetPayrollById(payrollId)
- GetPayrollsForEmployee(employeeId)
- GetPayrollsForPeriod(startDate, endDate)

```python
from abc import ABC, abstractmethod
from entity.payroll import Payroll
2 usages
class IPayrollService(ABC):
    @abstractmethod
    def generate_payroll(self, employee_id: int, start_date: str, end_date: str) -> bool:
        pass

    @abstractmethod
    def get_payroll_by_id(self, payroll_id: int) -> Payroll:
        pass

    @abstractmethod
    def get_payrolls_for_employee(self, employee_id: int) -> list[Payroll]:
        pass

    @abstractmethod
    def get_payrolls_for_period(self, start_date: str, end_date: str) -> list[Payroll]:
        pass
```

ITaxService:

- CalculateTax(employeeId, taxYear)
- GetTaxById(taxId)
- GetTaxesForEmployee(employeeId)
- GetTaxesForYear(taxYear)

```python
from abc import ABC, abstractmethod
from entity.tax import Tax
2 usages
class ITaxService(ABC):
    @abstractmethod
    def calculate_tax(self, employee_id: int, tax_year: int) -> bool:
        pass

    @abstractmethod
    def get_tax_by_id(self, tax_id: int) -> Tax:
        pass

    @abstractmethod
    def get_taxes_for_employee(self, employee_id: int) -> list[Tax]:
        pass

    @abstractmethod
    def get_taxes_for_year(self, tax_year: int) -> list[Tax]:
        pass
```

IFinancialRecordService:

- AddFinancialRecord(employeeId, description, amount, recordType)
- GetFinancialRecordById(recordId)
- GetFinancialRecordsForEmployee(employeeId)
- GetFinancialRecordsForDate(recordDate)

```python
from abc import ABC, abstractmethod
from entity.FinancialRecord import FinancialRecord
2 usages
class IFinancialRecordService(ABC):
    @abstractmethod
    def add_financial_record(self, employee_id: int, description: str, amount: float, record_type: str) -> bool:
        pass

    @abstractmethod
    def get_financial_record_by_id(self, record_id: int) -> FinancialRecord:
        pass

    @abstractmethod
    def get_financial_records_for_employee(self, employee_id: int) -> list[FinancialRecord]:
        pass

    @abstractmethod
    def get_financial_records_for_date(self, record_date: str) -> list[FinancialRecord]:
        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.

```python
from abc import ABC, abstractmethod
from entity.FinancialRecord import FinancialRecord
2 usages
class IFinancialRecordService(ABC):
    @abstractmethod
    def add_financial_record(self, employee_id: int, description: str, amount: float, record_ty
        pass

    @abstractmethod
    def get_financial_record_by_id(self, record_id: int) -> FinancialRecord:
        pass

    @abstractmethod
    def get_financial_records_for_employee(self, employee_id: int) -> list[FinancialRecord]:
        pass

    @abstractmethod
    def get_financial_records_for_date(self, record_date: str) -> list[FinancialRecord]:
        pass
```

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.

```
import mysql.connector
from exceptions.custom_exceptions import DatabaseConnectionException

5 usages
class DBConnUtil:
    4 usages
    @staticmethod
    def get_connection():
        try:
            connection = mysql.connector.connect(
                host="localhost",
                database="payxpert",
                user="root",
                password="root",port="3306"
            )
            return connection
        except mysql.connector.Error as e:
            raise DatabaseConnectionException("Failed to connect to database")
```

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

=== EMPLOYEE TABLE ===
1. Get Employee by ID
2. Get All Employees
3. Add Employee
4. Update Employee
5. Remove Employee
6. Back to Main Menu
Enter your choice:
```

```
=== PAYROLL TABLE ===
1. Generate Payroll
2. Get Payroll by ID
3. Get Payrolls for Employee
4. Get Payrolls for Period
5. Back to Main Menu
Enter your choice:
```

```
=== TAX TABLE ===
1. Calculate Tax
2. Get Tax by ID
3. Get Taxes for Employee
4. Get Taxes for Year
5. Back to Main Menu
```

```
=== 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:
```

```
Enter your choice: 1
Enter Employee ID: 2
Employee: Employee Details:
Employee ID: 2
First Name: Jane
Last Name: Smith
Date of Birth: 1985-08-22
Gender: Female
Email: jane.smith@example.com
Phone Number: 987-654-3210
Address: 456 Elm St, City, Country
Position: Developer
Joining Date: 2018-03-20
Termination Date: 2022-06-30
```

```
Enter your choice: 2
Enter Tax ID: 2
Tax: Tax Details:
Tax ID: 2
Employee ID: 2
Tax Year: 2
Taxable Income: 55000.0
Tax Amount: 2
```

```
Enter your choice: 2
Enter Payroll ID: 4
Payroll: Payroll Details:
Payroll ID: 4
Employee ID: 4
Pay Period Start Date: 2024-04-01
Pay Period End Date: 2024-04-15
Basic Salary: 5200.0
Overtime Pay: 220.0
Deductions: 550.0
Net Salary: 4870.0
```

```
Enter your choice: 2
Enter Record ID: 6
Financial Record: Financial Record Details:
Record ID: 6
Employee ID: 6
Record Date: 2024-04-05
Description: Training Course
Amount: 1000.0
Record Type: Expense
```