

[2-3. 미결제정 DB에 대한 CRUD 기능 구현]

1. CRUD 기능 구현 모듈 작성

1-1. 프롬프트 작성

먼저 작성된 쿼리문을 이용해서 pending_account 데이터베이스를 생성해 놓았어.

그리고, 아래와 같이 connector.py라는 이름으로 데이터베이스에 연결하는 모듈을 작성해서 같은 폴더에 저장해 놓았어.

```
# connector.py
import mysql.connector

class Connector:
    def __enter__(self):
        self.connection = mysql.connector.connect(
            host="localhost",
            user="root",
            port="3306",
            password="fastcampus1!",
            database="pending_account"
        )
        self.cursor = self.connection.cursor(dictionary=True)
        return self

    def __exit__(self, exc_type, exc_val, exc_tb):
        self.cursor.close()
        self.connection.close()
```

이때, TransactionManager와 CashflowManager 라는 이름의 클래스를 생성하고, 각각의 클래스 안에 pending_account DB에 대한 CRUD 기능을 구현하는 함수들을 생성하고 싶어. 각각의 CRUD 기능에 따라 함수가 생성되도록 코드 작성해줘.

1-2. ChatGPT 답변



`TransactionManager`와 `CashflowManager` 클래스는 각각 `transaction_records`와 `cashflow_records` 테이블에 대한 CRUD 기능을 제공합니다. 아래는 이러한 기능을 구현한 코드입니다.

python

코드 복사

```
# transaction_manager.py
from connector import Connector

class TransactionManager:
    def create_transaction(self, account_name, client_name, transaction_date, amount):
        with Connector() as db:
```

```

        query = """
            INSERT INTO transaction_records
            (account_name, client_name, transaction_date, amount_receivable, amount_payable, settlement_status, notes)
            VALUES (%s, %s, %s, %s, %s, %s, %s)
        """

        db.cursor.execute(query, (account_name, client_name, transaction_date,
                                   db.cursor.fetchone()[4], db.cursor.fetchone()[5], db.cursor.fetchone()[6], db.cursor.fetchone()[7]))
        db.connection.commit()
        return db.cursor.lastrowid

    def read_transaction(self, transaction_id):
        with Connector() as db:
            query = "SELECT * FROM transaction_records WHERE id = %s"
            db.cursor.execute(query, (transaction_id,))
            return db.cursor.fetchone()

```

```

    def update_transaction(self, transaction_id, account_name, client_name, transaction_date, amount_receivable, amount_payable, settlement_status, notes):
        with Connector() as db:
            query = """
                UPDATE transaction_records
                SET account_name = %s, client_name = %s, transaction_date = %s, amount_receivable = %s, amount_payable = %s, settlement_status = %s, notes = %s
                WHERE id = %s
            """
            db.cursor.execute(query, (account_name, client_name, transaction_date,
                                       amount_receivable, amount_payable, settlement_status, notes, transaction_id))
            db.connection.commit()

    def delete_transaction(self, transaction_id):
        with Connector() as db:
            query = "DELETE FROM transaction_records WHERE id = %s"
            db.cursor.execute(query, (transaction_id,))
            db.connection.commit()

# cashflow_manager.py
from connector import Connector

class CashflowManager:
    def create_cashflow(self, transaction_id, dw_date, deposit_amount, withdrawal_amount, notes):
        with Connector() as db:
            query = """
                INSERT INTO cashflow_records
                (transaction_id, dw_date, deposit_amount, withdrawal_amount, notes)
                VALUES (%s, %s, %s, %s, %s)
            """
            db.cursor.execute(query, (transaction_id, dw_date, deposit_amount, withdrawal_amount, notes))
            db.connection.commit()
            return db.cursor.lastrowid

```

```

def read_cashflow(self, cashflow_id):
    with Connector() as db:
        query = "SELECT * FROM cashflow_records WHERE id = %s"
        db.cursor.execute(query, (cashflow_id,))
        return db.cursor.fetchone()

def update_cashflow(self, cashflow_id, transaction_id, dw_date, deposit_amount, withdrawal_amount):
    with Connector() as db:
        query = """
            UPDATE cashflow_records
            SET transaction_id = %s, dw_date = %s, deposit_amount = %s, withdrawal_amount = %s
            WHERE id = %s
        """
        db.cursor.execute(query, (transaction_id, dw_date, deposit_amount, withdrawal_amount, cashflow_id))
        db.connection.commit()

def delete_cashflow(self, cashflow_id):
    with Connector() as db:
        query = "DELETE FROM cashflow_records WHERE id = %s"
        db.cursor.execute(query, (cashflow_id,))
        db.connection.commit()

```

이 코드에는 각 클래스에 대해 다음과 같은 CRUD 기능이 포함되어 있습니다:

1. `TransactionManager` 클래스:

- `create_transaction`: 새로운 거래 기록을 생성.
- `read_transaction`: 특정 거래 기록을 조회.
- `update_transaction`: 특정 거래 기록을 업데이트.
- `delete_transaction`: 특정 거래 기록을 삭제.

2. `CashflowManager` 클래스:

- `create_cashflow`: 새로운 현금 흐름 기록을 생성.
- `read_cashflow`: 특정 현금 흐름 기록을 조회.
- `update_cashflow`: 특정 현금 흐름 기록을 업데이트.
- `delete_cashflow`: 특정 현금 흐름 기록을 삭제.

각 함수는 데이터베이스 연결을 열고 닫으며, 데이터베이스 작업을 수행합니다. 필요한 경우 매개변수 값을 함수 호출 시 전달하여 데이터베이스에 필요한 데이터를 삽입, 조회, 업데이트, 삭제할 수 있습니다.

1-3. ChatGPT 작성 코드

```

# transaction_manager.py
from connector import Connector

class TransactionManager:

```

```
def create_transaction(
    self, account_name, client_name,
    transaction_date, amount_receivable,
    amount_payable, settlement_status, notes
):
    with Connector() as db:
        query = """
            INSERT INTO transaction_records
                (account_name, client_name,
                 transaction_date, amount_receivable,
                 amount_payable, settlement_status,
                 notes)
            VALUES (%s, %s, %s, %s, %s, %s, %s)
        """
        db.cursor.execute(
            query,
            (account_name, client_name,
             transaction_date, amount_receivable,
             amount_payable, settlement_status, notes)
        )
        db.connection.commit()
        return db.cursor.lastrowid

def read_transaction(self, transaction_id):
    with Connector() as db:
        query = """
            SELECT *
            FROM transaction_records
            WHERE id = %s
        """
        db.cursor.execute(query, (transaction_id,))
        return db.cursor.fetchone()

def update_transaction(
    self, transaction_id, account_name,
    client_name, transaction_date,
    amount_receivable, amount_payable,
    settlement_status, notes
):
    with Connector() as db:
        query = """
            UPDATE transaction_records
            SET
                account_name = %s,
                client_name = %s,
                transaction_date = %s,
                amount_receivable = %s,
                amount_payable = %s,
                settlement_status = %s,
                notes = %s
            WHERE id = %s
        """
```

```
        """
        db.cursor.execute(
            query,
            (account_name, client_name,
             transaction_date, amount_receivable,
             amount_payable, settlement_status,
             notes, transaction_id)
        )
        db.connection.commit()

    def delete_transaction(self, transaction_id):
        with Connector() as db:
            query = """
                DELETE
                FROM transaction_records
                WHERE id = %s
            """
            db.cursor.execute(query, (transaction_id,))
            db.connection.commit()

# cashflow_manager.py
from connector import Connector

class CashflowManager:
    def create_cashflow(
        self, transaction_id, dw_date, deposit_amount,
        withdrawal_amount, notes
    ):
        with Connector() as db:
            query = """
                INSERT INTO cashflow_records
                (transaction_id, dw_date,
                 deposit_amount,
                 withdrawal_amount, notes)
                VALUES (%s, %s, %s, %s, %s)
            """
            db.cursor.execute(
                query,
                (transaction_id, dw_date, deposit_amount,
                 withdrawal_amount, notes)
            )
            db.connection.commit()
            return db.cursor.lastrowid

    def read_cashflow(self, cashflow_id):
        with Connector() as db:
            query = """
                SELECT *
                FROM cashflow_records
                WHERE id = %s
            """
```

```

        db.cursor.execute(query, (cashflow_id,))
        return db.cursor.fetchone()

    def update_cashflow(
        self, cashflow_id, transaction_id,
        dw_date, deposit_amount,
        withdrawal_amount, notes
    ):
        with Connector() as db:
            query = """
                UPDATE cashflow_records
                SET
                    transaction_id = %s,
                    dw_date = %s,
                    deposit_amount = %s,
                    withdrawal_amount = %s,
                    notes = %s
                WHERE id = %s
            """
            db.cursor.execute(
                query,
                (transaction_id, dw_date, deposit_amount,
                 withdrawal_amount, notes, cashflow_id)
            )
            db.connection.commit()

    def delete_cashflow(self, cashflow_id):
        with Connector() as db:
            query = """
                DELETE FROM cashflow_records
                WHERE id = %s
            """
            db.cursor.execute(query, (cashflow_id,))
            db.connection.commit()

```

1-4. 코드 수정 반영

- 함수 이름 축약
- read_all 함수 작성

```

In [ ]: # transaction_manager
        from connector import Connector

        class TransactionManager:
            def create(
                self, account_name, client_name, transaction_date,
                amount_receivable, amount_payable, settlement_status,
                notes
            ):
                with Connector() as db:

```

```
query = """
    INSERT INTO transaction_records
        (account_name, client_name, transaction_date,
         amount_receivable, amount_payable,
         settlement_status, notes)
    VALUES (%s, %s, %s, %s, %s, %s, %s)
    """
db.cursor.execute(
    query,
    (account_name, client_name, transaction_date,
     amount_receivable, amount_payable,
     settlement_status, notes)
)
db.connection.commit()
return db.cursor.lastrowid

def read_all(self):
    with Connector() as db:
        query = "SELECT * FROM transaction_records"
        db.cursor.execute(query)
        return db.cursor.fetchall()

def read(self, transaction_id):
    with Connector() as db:
        query = "SELECT * FROM transaction_records WHERE id = %s"
        db.cursor.execute(query, (transaction_id,))
        return db.cursor.fetchone()

def update(
    self, transaction_id, account_name, client_name,
    transaction_date, amount_receivable,
    amount_payable, settlement_status, notes
):
    with Connector() as db:
        query = """
            UPDATE transaction_records
            SET account_name = %s, client_name = %s,
                transaction_date = %s, amount_receivable = %s,
                amount_payable = %s, settlement_status = %s,
                notes = %s
            WHERE id = %s
            """
        db.cursor.execute(
            query,
            (account_name, client_name, transaction_date,
             amount_receivable, amount_payable,
             settlement_status, notes, transaction_id)
        )
        db.connection.commit()

def delete(self, transaction_id):
    with Connector() as db:
```

```
        query = "DELETE FROM transaction_records WHERE id = %s"
        db.cursor.execute(query, (transaction_id,))
        db.connection.commit()

# cashflow_manager
from connector import Connector

class CashflowManager:
    def create(
        self, transaction_id, dw_date, deposit_amount,
        withdrawal_amount, notes
    ):
        with Connector() as db:
            query = """
                INSERT INTO cashflow_records
                (transaction_id, dw_date, deposit_amount,
                withdrawal_amount, notes)
                VALUES (%s, %s, %s, %s, %s)
            """
            db.cursor.execute(
                query,
                (transaction_id, dw_date, deposit_amount,
                withdrawal_amount, notes)
            )
            db.connection.commit()
            return db.cursor.lastrowid

    def read_all(self):
        with Connector() as db:
            query = "SELECT * FROM cashflow_records"
            db.cursor.execute(query)
            return db.cursor.fetchall()

    def read(self, cashflow_id):
        with Connector() as db:
            query = "SELECT * FROM cashflow_records WHERE id = %s"
            db.cursor.execute(query, (cashflow_id,))
            return db.cursor.fetchone()

    def update(
        self, cashflow_id, transaction_id, dw_date,
        deposit_amount, withdrawal_amount, notes
    ):
        with Connector() as db:
            query = """
                UPDATE cashflow_records
                SET transaction_id = %s, dw_date = %s,
                deposit_amount = %s, withdrawal_amount = %s,
                notes = %s
                WHERE id = %s
            """
            db.cursor.execute(
                query,
                (transaction_id, dw_date, deposit_amount,
```



```
        withdrawal_amount, notes, cashflow_id)
    )
    db.connection.commit()

def delete(self, cashflow_id):
    with Connector() as db:
        query = "DELETE FROM cashflow_records WHERE id = %s"
        db.cursor.execute(query, (cashflow_id,))
        db.connection.commit()
```

2. CRUD 기능을 이용하여 데이터 입력 및 출력 하기

2-1. 클래스 인스턴스 선언

```
In [ ]: tm = TransactionManager()
        cm = CashflowManager()
```

2-2. 샘플 데이터 생성

```
In [ ]: tm.create(
        account_name = "외상매출금",
        client_name = "ABC전자",
        transaction_date = "2024-01-10",
        amount_receivable = 1_000_000,
        amount_payable = 0,
        settlement_status = None,
        notes = None
    )
```

```
Out[ ]: 1
```

```
In [ ]: tm.read_all()
```

```
Out[ ]: [{'id': 1,
          'account_name': '외상매출금',
          'client_name': 'ABC전자',
          'transaction_date': datetime.date(2024, 1, 10),
          'amount_receivable': 1000000,
          'amount_payable': 0,
          'settlement_status': None,
          'notes': None}]
```

```
In [ ]: tm.create(
    account_name = "외상매출금",
    client_name = "AG상사",
    transaction_date = "2024-01-10",
    amount_receivable = 2_000_000,
    amount_payable = 0,
    settlement_status = None,
    notes = None
)
```

```
Out[ ]: 2
```

```
In [ ]: tm.read_all()
```

```
Out[ ]: [{ 'id': 1,
  'account_name': '외상매출금',
  'client_name': 'ABC전자',
  'transaction_date': datetime.date(2024, 1, 10),
  'amount_receivable': 1000000,
  'amount_payable': 0,
  'settlement_status': None,
  'notes': None},
{ 'id': 2,
  'account_name': '외상매출금',
  'client_name': 'AG상사',
  'transaction_date': datetime.date(2024, 1, 10),
  'amount_receivable': 2000000,
  'amount_payable': 0,
  'settlement_status': None,
  'notes': None}]
```

2-3. 데이터 조회 및 DataFrame으로 변환

```
In [ ]: import pandas as pd
tmddf = pd.DataFrame(tm.read_all())
```

```
In [ ]: tmddf
```

```
Out[ ]:
```

	id	account_name	client_name	transaction_date	amount_receivable	amount_payable
0	1	외상매출금	ABC전자	2024-01-10	1000000	0
1	2	외상매출금	AG상사	2024-01-10	2000000	0

2-4. Cashflow 데이터 입력

```
In [ ]: cm.create(
        transaction_id = 1,
        dw_date = "2024-01-20",
        deposit_amount = 500_000,
        withdrawal_amount = 0,
        notes = None
    )
cm.create(
    transaction_id = 1,
    dw_date = "2024-01-30",
    deposit_amount = 500_000,
    withdrawal_amount = 0,
    notes = None
)
```

Out[]: 2

```
In [ ]: cm.read_all()
```

```
Out[ ]: [{'id': 1,
          'transaction_id': 1,
          'dw_date': datetime.date(2024, 1, 20),
          'deposit_amount': 500000,
          'withdrawal_amount': 0,
          'notes': None},
         {'id': 2,
          'transaction_id': 1,
          'dw_date': datetime.date(2024, 1, 30),
          'deposit_amount': 500000,
          'withdrawal_amount': 0,
          'notes': None},
         {'id': 3,
          'transaction_id': 2,
          'dw_date': datetime.date(2024, 1, 15),
          'deposit_amount': 1000000,
          'withdrawal_amount': 0,
          'notes': None},
         {'id': 4,
          'transaction_id': 2,
          'dw_date': datetime.date(2024, 1, 25),
          'deposit_amount': 500000,
          'withdrawal_amount': 0,
          'notes': None}]
```

```
In [ ]: cm.create(
        transaction_id = 2,
        dw_date = "2024-01-15",
        deposit_amount = 1_000_000,
        withdrawal_amount = 0,
        notes = None
    )
cm.create(
    transaction_id = 2,
    dw_date = "2024-01-25",
    deposit_amount = 500_000,
    withdrawal_amount = 0,
    notes = None
)
```

Out[]: 6

```
In [ ]: cmdf = pd.DataFrame(cm.read_all())
```

```
In [ ]: cmdf
```

```
Out[ ]:
```

	id	transaction_id	dw_date	deposit_amount	withdrawal_amount	notes
0	1	1	2024-01-20	500000	0	None
1	2	1	2024-01-30	500000	0	None
2	3	2	2024-01-15	1000000	0	None
3	4	2	2024-01-25	500000	0	None

2-5. pending_account 모듈 작성

- pending_account.py 파일 생성
- TranasctionManager 클래스와 CashflowManager 클래스 복사

```
In [ ]: from pending_account import (
        TransactionManager as TM,
        CashflowManager as CM
    )

tm = TM()
cm = CM()
```

```
In [ ]: tm.read_all()
```

```
Out[ ]: [{ 'id': 1,
            'account_name': '외상매출금',
            'client_name': 'ABC전자',
            'transaction_date': datetime.date(2024, 1, 10),
            'amount_receivable': 1000000,
            'amount_payable': 0,
            'settlement_status': None,
            'notes': None},
          { 'id': 2,
            'account_name': '외상매출금',
            'client_name': 'AG상사',
            'transaction_date': datetime.date(2024, 1, 10),
            'amount_receivable': 2000000,
            'amount_payable': 0,
            'settlement_status': None,
            'notes': None}]
```

```
In [ ]: cm.read_all()
```

```
Out[ ]: [{ 'id': 1,
            'transaction_id': 1,
            'dw_date': datetime.date(2024, 1, 20),
            'deposit_amount': 500000,
            'withdrawal_amount': 0,
            'notes': None},
          { 'id': 2,
            'transaction_id': 1,
            'dw_date': datetime.date(2024, 1, 30),
            'deposit_amount': 500000,
            'withdrawal_amount': 0,
            'notes': None},
          { 'id': 3,
            'transaction_id': 2,
            'dw_date': datetime.date(2024, 1, 15),
            'deposit_amount': 1000000,
            'withdrawal_amount': 0,
            'notes': None},
          { 'id': 4,
            'transaction_id': 2,
            'dw_date': datetime.date(2024, 1, 25),
            'deposit_amount': 500000,
            'withdrawal_amount': 0,
            'notes': None}]
```