# 데이터베이스 ESQL과제

- 고급언어 Python –

2315028 인공지능공학부 김성현

## 1. Progress Report

2024.11.09

PM 10:00 – 1:00 주제설정 및 데이터 생성, 연결

2024.11.10

PM 10:00 – 1:00 파이썬으로 함수를 구현하고, 프로그램 실행

2024.11.13~15
코드 보완 (오류 수정, 기능 추가)

2024. 11. 16

PM 10:00-11:00 보고서 작성

2024. 11. 18
PM 10:00-12:00 코드 수정 (크로스 체크하는 부분)

2024. 11 . 23

PM 8:00 – 9:00 보고서 수정

## 2. 주제

**눈송이 은행원 프로그램**

: 눈송이 은행에서 일하는 직원이 사용하는 프로그램을 구현해보았습니다.
고객의 계좌 관리부터 지점관리까지 모든 것이 가능한 프로그램을 구현하였습니다.

## 3. 함수

0. 데이터베이스 생성함수 : create_db()

```
def create_schema():
    schema_creation_query = """
CREATE TABLE 사용자 (
    사용자_ID INT PRIMARY KEY,
    이름 VARCHAR(50) NOT NULL,
    성별 CHAR(1) CHECK (성별 IN ('M','F')),
    연락처 VARCHAR(15) NOT NULL,
```

```
    주민번호 CHAR(15) UNIQUE NOT NULL
);

CREATE TABLE 계좌 (
    계좌번호 INT PRIMARY KEY,
    사용자_ID INT,
    계좌유형 VARCHAR(20) CHECK(계좌유형 IN ('저축','당좌','정기예금')),
    잔액 DECIMAL(15, 2) DEFAULT 0,
    FOREIGN KEY (사용자_ID) REFERENCES 사용자(사용자_ID)
);

CREATE TABLE 거래내역 (
    거래_ID INT PRIMARY KEY,
    계좌번호 INT,
    거래일시 TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    거래유형 VARCHAR(10) CHECK (거래유형 IN ('입금','출금')),
    거래금액 DECIMAL(15, 2) REFERENCES 계좌(계좌번호)
);

CREATE TABLE 지점 (
    지점_ID INT PRIMARY KEY,
    위치 VARCHAR(100) NOT NULL,
    지점장_이름 VARCHAR(50) NOT NULL,
    직원수 INT CHECK (직원수 >= 0)
);
    """
    try:
        connection = connect_db()
        if connection is None:
            print("데이터베이스와 연결이 안됩니다.")

        cursor = connection.cursor()
        cursor.execute(schema_creation_query, multi=True)
        connection.commit()
        print("데이터베이스가 성공적으로 만들어졌습니다.")

    except mysql.connector.Error as e:
        if "already exists" in str(e):
            print("\n 데이터베이스가 이미 존재합니다.")
        else:
            print(f"Unexpected error occurred: {e}")

    finally:
        if connection.is_connected():
            cursor.close()
            connection.close()
```

기능 : 새로운 데이터 베이스를 생성하는 함수입니다.

반환값 : 데이터베이스와 연동여부에 따라 메세지가 출력됩니다.

**오류 처리 :** 테이블이 이미 존재할 경우 발생하는 MySQL 오류 메시지에서 already exists를 확인하여 이를 사용자에게 알립니다. 그 외 예기치 못한 오류는 예외 메시지를 출력합니다.

### 1. 데이터베이스 연결함수 : connect_db()

```
def connect_db():
    return mysql.connector.connect(
        host="localhost",
        user="root",
        password="khyun0303",
        database="ESQL"
    )
```

**기능**: MySQL 데이터베이스와의 연결을 설정하는 함수입니다. mysql.connector.connect를 사용하여 연결 객체를 생성하고 반환합니다.

**반환값**: 데이터베이스 연결 객체를 반환하여 다른 함수에서 데이터베이스 작업을 수행할 수 있게 합니다.
**오류 :** 연결에 실패할 경우 이후 기능에서 None을 반환해 오류를 확인할 수 있도록 합니다.

### 2. 사용자 정보 조회 함수 : get_user_info(user_id)

```
def get_user_info(user_id):
    db = connect_db()
    cursor = db.cursor()
    query = """
    SELECT 사용자.이름, 사용자.성별, 사용자.연락처, 계좌.계좌번호
    FROM 사용자
    JOIN 계좌 ON 사용자.사용자_ID = 계좌.사용자_ID
    WHERE 사용자.사용자_ID = %s;
    """
    print("\n---------------SQL Code Test---------------\n", query)
    print("-------------------------------------------\n")
    cursor.execute(query, (user_id,))
    result = cursor.fetchall()
    db.close()
    return result
    cursor.execute(query, (user_id,))
    result = cursor.fetchall()
    db.close()
    return result
```

•**기능**: 특정 사용자 ID를 통해 사용자 정보와 연결된 계좌 번호를 조회하는 함수입니다.

•**쿼리 설명**: 사용자와 계좌 테이블을 JOIN하여 사용자 ID에 맞는 이름, 성별, 연락처, 계좌번호를 조회합니다.

•**출력**: SQL 쿼리와 조회 결과를 반환하여 메뉴에서 출력할 수 있게 합니다.

### 3. 사용자 거래내역 조회 함수 : get_account_history(account_number)

```
def get_account_history(account_number):
    db = connect_db()
    cursor = db.cursor()
    query = """
    SELECT 사용자.이름, 거래내역.거래일시, 거래내역.거래유형, 거래내역.거래금액, 지점.위치
```

```
    FROM 거래내역
    JOIN 계좌 ON 거래내역.계좌번호 = 계좌.계좌번호
    JOIN 사용자 ON 계좌.사용자_ID = 사용자.사용자_ID
    JOIN 지점 ON 사용자.지점_ID = 지점.지점_ID
    WHERE 계좌.계좌번호 = %s;
    """
    print("\n---------------SQL Code Test---------------\n", query)
    print("-------------------------------------------\n")
    cursor.execute(query, (account_number,))
    result = cursor.fetchall()
    db.close()
    return result
```

- **기능**: 특정 계좌 번호를 통해 거래 내역과 지점 위치를 조회하는 함수입니다.
- **쿼리 설명**: 거래내역, 계좌, 사용자, 지점 테이블을 JOIN하여 계좌번호에 맞는 거래일시, 거래유형, 거래금액, 지점 위치 등을 가져옵니다.
- **출력**: SQL 쿼리와 조회 결과를 반환하여 메뉴에서 출력할 수 있게 합니다.

**오류 처리 :** SQL 쿼리 실행 결과가 없으면 빈 결과를 반환합니다. 데이터가 없거나 입력 값이 잘못된 경우 if result:로 결과 유무를 확인해 사용자에게 알립니다.

### 4. 계좌 잔액 조회 함수 (출금) : get_account_balance(account_number)

```
def get_account_balance(account_number):
    db = connect_db()
    cursor = db.cursor()
    query = "SELECT 잔액 FROM 계좌 WHERE 계좌번호 = %s;"
    print("\n---------------SQL Code Test---------------\n", query)
    print("-------------------------------------------\n")
    cursor.execute(query, (account_number,))
    result = cursor.fetchone()
    db.close()
    return result[0] if result else None
```

- **기능**: 특정 계좌의 잔액을 조회하는 함수로, 출금 시 잔액을 확인하기 위해 사용됩니다.
- **쿼리 설명**: 계좌 테이블에서 계좌번호에 맞는 잔액을 조회합니다.
  - **출력**: 조회된 잔액을 반환하여 출금 시 금액이 충분한지 확인할 수 있게 합니다.

### 5. 계좌 잔액 업데이트 함수(입금) : update_account(account_number, amount, transaction_type)

```
def update_account(account_number, amount, transaction_type):
    db = connect_db()
    cursor = db.cursor()
    if transaction_type == "입금":
        update_query = "UPDATE 계좌 SET 잔액 = 잔액 + %s WHERE 계좌번호 = %s;"
```

```
        print("\n---------------SQL Code Test---------------\n", update_query)
        print("-------------------------------------------\n")
    elif transaction_type == "출금":
        update_query = "UPDATE 계좌 SET 잔액 = 잔액 - %s WHERE 계좌번호 = %s;"
        print("\n---------------SQL Code Test---------------\n", update_query)
        print("-------------------------------------------\n")
    cursor.execute(update_query, (amount, account_number))

    query = "SELECT COALESCE(MAX(거래_ID), 2000) + 1 AS next_transaction_id FROM 거래내역;"
    print("\n----------------SQL Code Test--------------\n", query)
    print("-------------------------------------------\n")
    cursor.execute(query)

    transaction_id_result = cursor.fetchone()
    if transaction_id_result and transaction_id_result[0] is not None:
        transaction_id = transaction_id_result[0]
    else:
        transaction_id = 2001

    insert_query = """
    INSERT INTO 거래내역 (거래_ID, 계좌번호, 거래일시, 거래유형, 거래금액)
    VALUES (%s, %s, %s, %s, %s);
    """
    print("\n----------------SQL Code Test--------------\n, insert_query")
    print("-------------------------------------------\n")
    cursor.execute(insert_query, (transaction_id, account_number, datetime.now(),
transaction_type, amount))

    db.commit()
    db.close()
    print(f"{transaction_type} 완료: {amount}원이 계좌에 반영되었습니다.")
```

- **기능**: 입금 또는 출금 시 계좌 잔액을 업데이트하는 함수입니다.
- **조건**: transaction_type이 입금인지 출금인지에 따라 쿼리를 다르게 설정합니다.

  입금: 잔액을 증가시키는 쿼리 실행

  출금: 잔액을 감소시키는 쿼리 실행
- **출력**: SQL 쿼리 실행 후 잔액이 업데이트됩니다.
- **오류 처리:**

출금 시 잔액이 부족하면 출금을 차단하고 오류 메시지를 출력합니다.

데이터베이스 트랜잭션(db.commit)을 사용해 잔액 변경 및 거래내역 기록이 동시에 처리되도록 보장합니다.

거래 ID는 MAX(거래_ID)로 자동 증가하며, 데이터가 없는 경우 기본값 2001을 설정합니다.

6. 신규 사용자 및 계좌 생성 함수 : create_user_account(user_data, account_data):

```
def create_user_account(user_data, account_data):
    db = connect_db()
    cursor = db.cursor()
    user_query = "INSERT INTO 사용자 (사용자_ID, 이름, 성별, 연락처, 주민번호, 지점_ID) VALUES
(%s, %s, %s, %s, %s, %s);"
    print("\n---------------SQL Code Test---------------\n", user_query)
    print("-------------------------------------------\n")
    cursor.execute(user_query, user_data)
    account_query = "INSERT INTO 계좌 (계좌번호, 사용자_ID, 계좌유형, 잔액, 지점_ID) VALUES
(%s, %s, %s, %s, %s);"
    print("\n---------------SQL Code Test---------------\n", account_query)
    print("-------------------------------------------\n")
    cursor.execute(account_query, account_data)
    db.commit()
    db.close()
    print("새로운 사용자 및 계좌가 성공적으로 등록되었습니다!")
```

•**기능**: 신규 사용자와 계좌를 생성하는 함수입니다.

•**쿼리 설명**:

첫 번째 쿼리: 사용자 테이블에 새로운 사용자 정보를 추가합니다.

두 번째 쿼리: 계좌 테이블에 새로운 계좌 정보를 추가하여 계좌를 개설합니다.

•**출력**: SQL 쿼리 실행 후 새로운 사용자와 계좌가 생성됩니다.

7. 지점별 사용자 조회 함수 : get_branch_accounts(branch_id)

```
def get_branch_accounts(branch_id):
    db = connect_db()
    cursor = db.cursor()
    query = """
    SELECT 지점.위치, 지점.지점장_이름, 지점.직원수, COUNT(사용자.사용자_ID) AS 사용자_수
    FROM 지점
    JOIN 사용자 ON 사용자.지점_ID = 지점.지점_ID
    WHERE 지점.지점_ID = %s
    """
    print("\n---------------SQL Code Test---------------\n", query)
    print("-------------------------------------------\n")
    cursor.execute(query, (branch_id,))
    result = cursor.fetchall()
    db.close()
    return result
```

• **기능**: 특정 지점 ID를 통해 해당 지점의 위치, 지점장 이름, 직원 수, 사용자 수를 조회하는 함수입니다.

• **쿼리 설명**: 지점 테이블을 사용자 테이블과 JOIN하여 지점 ID에 맞는 지점 정보와 사용자 수를 가져옵니다.

- **출력**: SQL 쿼리 실행 후 해당 지점의 정보를 반환합니다.

### 8. 사용자 출력화면 함수 : display_menu()

```
9.  def display_menu():
10.     print("\n<<<<<<<<<< 눈송이 은행에 오신걸 환영합니다!! >>>>>>>>>>\n")
11.     print("1. 사용자 정보 조회\n")
12.     print("2. 거래내역 조회\n")
13.     print("3. 입출금\n")
14.     print("4. 신규 가입 및 계좌 개설\n")
15.     print("5. 은행 지점 조회\n")
16.     print("6. 종료\n")
17.     choice = input("원하시는 번호를 입력해주세요 : ")
18.     return choice
```

- **기능**: 사용자에게 은행 메뉴를 보여주는 함수입니다.
- **입력**: 사용자가 원하는 기능에 해당하는 번호를 입력받습니다.
- **출력**: 입력된 번호에 따라 적절한 함수를 호출하고 그 결과를 화면에 출력합니다.

동작

```
while True:
    create_schema()
    choice = display_menu()
    if choice == "1":
        user_id = input("사용자 ID를 입력하세요 : ")
        result = get_user_info(user_id)
        if result:
            for row in result:
                print(f"이름 : '{row[0]}'")
                print(f"성별 : '{row[1]}'")
                print(f"전화번호 : '{row[2]}'")
                print(f"계좌번호 : '{row[3]}'")
        else:
            print("입력하신 사용자아이디가 올바르지 않습니다.")

    elif choice == "2":
        account_number = input("계좌번호를 입력하세요 : ")
        result = get_account_history(account_number)
        if result:
            for row in result:
                print(f"이름 : '{row[0]}'")
                print(f"거래일시 : ({row[1].year}년, {row[1].month}월, {row[1].day}일,
{row[1].hour}시:{row[1].minute}분)")
                print(f"거래유형 : '{row[2]}'")
                print(f"거래금액 : '{row[3]}'")
                print(f"지점위치 : '{row[4]}'")
        else:
```

```
        print("입력하신 계좌번호가 올바르지 않습니다.")

    elif choice == "3":
        account_number = input("계좌번호를 입력하세요 : ")
        transaction_type = input("입금 / 출금 : ")
        if transaction_type in ["입금", "출금"]:
            amount = int(input(f"{transaction_type}할 금액을 입력하세요 : "))
            if transaction_type == "출금":
                balance = get_account_balance(account_number)
                if balance is not None and balance >= amount:
                    update_account(account_number, amount, transaction_type)
                    print(f"{amount}원이 출금되었습니다. 잔액: {balance - amount}원")
                else:
                    print("잔액이 부족합니다.")
            else:
                update_account(account_number, amount, transaction_type)
                print(f"{amount}원이 입금되었습니다.")
        else:
            print("잘못입력하셨습니다. (입금/출금) 정확히 입력해주세요.")

    elif choice == "4":
        user_data = (
            input("사용자 ID: "),
            input("이름: "),
            input("성별(M/F): "),
            input("연락처(010-0000-0000): "),
            input("주민번호(900101-1234567): "),
            input("지점 번호 : ")
        )
        account_data = (
            input("계좌번호(4 자리): "),
            user_data[0],
            input("계좌유형(저축/당좌/정기예금): "),
            input("초기 잔액: "),
            user_data[5]
        )
        create_user_account(user_data, account_data)

    elif choice == "5":
        branch_id = input("지점 ID 를 입력하세요 : ")
        result = get_branch_accounts(branch_id)
        if result:  # 결과가 존재하는지 확인
            for row in result:
                print(f"지점 위치 : '{row[0]}'")
                print(f"지점장 이름 : '{row[1]}'")
                print(f"직원 수 : '{row[2]}'")
                print(f"사용자 수 : '{row[3]}'")
        else:
            print("입력하신 지점은 존재하지 않습니다.")

    elif choice == "6":
        print(">>>>>>>>>> 프로그램을 종료합니다! <<<<<<<<<<")
        break
```

```
    else:
        print("잘못된 입력입니다. 다시 입력해주세요.")
```

### 0. 데이터베이스 생성 및 사용자 입력

데이터베이스를 생성합니다. 단 이미 존재하는 경우 이미 존재한다는 메세지를 출력합니다.

사용자 화면이 출력되어 사용자의 입력을 받습니다. 아래는 해당 번호를 입력받았을때의 각각의 경우입니다.

### 1. 사용자 정보 조회 (Option 1)

사용자 ID를 입력받아 get_user_info(user_id) 함수를 호출합니다.

반환된 결과가 있다면 사용자 이름, 성별, 전화번호, 계좌번호를 출력합니다.

결과가 없을 경우 "입력하신 사용자 ID가 올바르지 않습니다."라는 메시지를 표시합니다.

### 2. 계좌 거래 내역 조회 (Option 2)

계좌번호를 입력받아 get_account_history(account_number) 함수를 호출합니다.

거래 내역(거래일시, 거래유형, 거래금액, 지점위치)을 출력합니다. 날짜와 시간은 연도, 월, 일, 시, 분 형식으로 상세히 표시됩니다.

결과가 없으면 "입력하신 계좌번호가 올바르지 않습니다."라는 메시지를 출력합니다.

### 3. 입출금 처리 (Option 3)

계좌번호와 거래 유형("입금" 또는 "출금")을 입력받습니다.

출금의 경우, 먼저 get_account_balance(account_number)로 잔액을 확인합니다. 잔액이 부족하면 출금이 불가하며, 충분할 경우 출금 처리를 수행합니다.

입금의 경우 잔액 확인 없이 바로 update_account(account_number, amount, transaction_type) 함수로 처리됩니다.

결과에 따라 입출금 내역 및 잔액을 출력합니다.

### 4. 새 사용자 및 계좌 생성 (Option 4)

사용자 정보(ID, 이름, 성별, 연락처, 주민번호, 지점 번호)와 계좌 정보(계좌번호, 계좌유형, 초기 잔액, 지점 번호)를 입력받습니다.

create_user_account(user_data, account_data)를 호출하여 데이터베이스에 사용자 및 계좌 정보를 저장합니다.

### 5. 지점 정보 조회 (Option 5)

지점 ID를 입력받아 get_branch_accounts(branch_id)를 호출합니다.

결과로 반환된 지점 정보(지점 위치, 지점장 이름, 직원 수, 사용자 수)를 출력합니다.

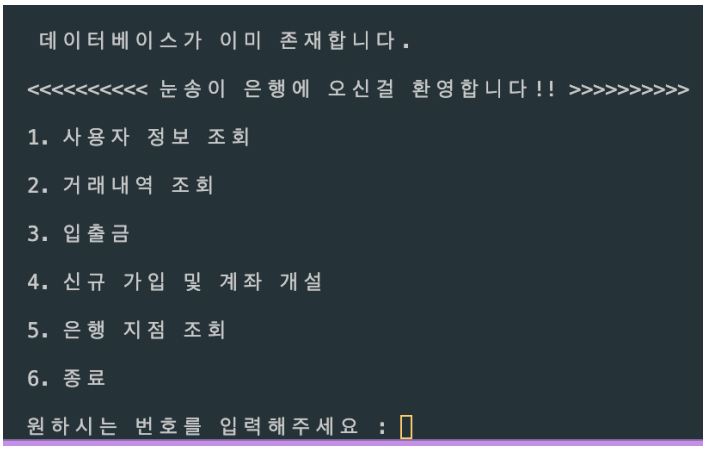
해당 지점이 없을 경우 "입력하신 지점은 존재하지 않습니다."라는 메시지를 표시합니다.

**6. 프로그램 종료 (Option 6)**

"프로그램을 종료합니다!"라는 메시지를 출력하고, break를 사용해 루프를 종료합니다.

**기타 입력 처리** : 사용자가 메뉴에서 잘못된 입력을 할 경우, "잘못된 입력입니다. 다시 입력해주세요."라는 메시지를 출력하며 다시 메뉴를 표시합니다.

## 4. 출력 결과




이미 데이터베이스가 존재하기에 데이터베이스가 존재한다는 문구가 출력됩니다.

사용자 화면이 동작되며, 서비스의 이름과 번호가 출력됩니다.

원하는 서비스의 번호를 입력받습니다.

```
<<<<<<<<<< 눈송이 은행에 오신걸 환영합니다!! >>>>>>>>>>

1. 사용자 정보 조회

2. 거래내역 조회

3. 입출금

4. 신규 가입 및 계좌 개설

5. 은행 지점 조회

6. 종료

원하시는 번호를 입력해주세요 : 1
사용자 ID를 입력하세요 : 5

---------------SQL Code Test---------------

    SELECT 사용자.이름, 사용자.성별, 사용자.연락처, 계좌.계좌번호
    FROM 사용자
    JOIN 계좌 ON 사용자.사용자_ID = 계좌.사용자_ID
    WHERE 사용자.사용자_ID = %s;

-----------------------------------------

이름 : '정다영'
성별 : 'F'
전화번호 : '010-5678-9012'
계좌번호 : '1005'
```

1번 사용자 정보 조회 서비스를 선택하였을 경우입니다. 우선, 사용자의 아이디를 입력합니다.

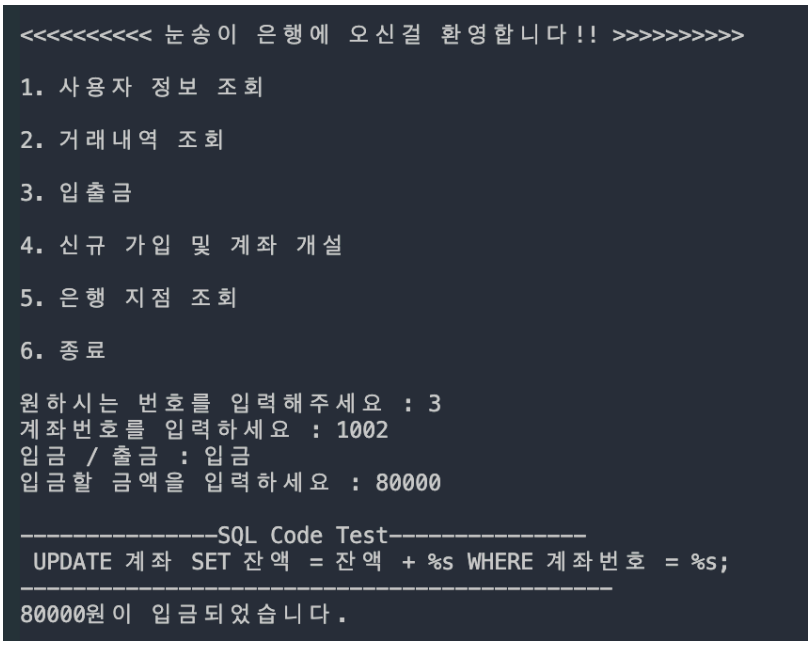
SQL CODE TEST가 진행되고 나서, 입력한 사용자의 이름, 성별, 전화번호, 계좌번호 순으로 출력됩니다.

```
<<<<<<<<<< 눈송이 은행에 오신걸 환영합니다!! >>>>>>>>>>

1. 사용자 정보 조회

2. 거래내역 조회

3. 입출금

4. 신규 가입 및 계좌 개설

5. 은행 지점 조회

6. 종료

원하시는 번호를 입력해주세요 : 2
계좌번호를 입력하세요 : 1001

---------------SQL Code Test---------------

    SELECT 사용자.이름, 거래내역.거래일시, 거래내역.거래유형, 거래내역
.거래금액, 지점.위치
    FROM 거래내역
    JOIN 계좌 ON 거래내역.계좌번호 = 계좌.계좌번호
    JOIN 사용자 ON 계좌.사용자_ID = 사용자.사용자_ID
    JOIN 지점 ON 사용자.지점_ID = 지점.지점_ID
    WHERE 계좌.계좌번호 = %s;

-----------------------------------------

이름 : '김민수'
거래일시 : (2024년, 11월, 1일, 10시:15분)
거래유형 : '입금'
거래금액 : '200000.00'
지점위치 : '서울 종로구'
```

2번 거래내역 조회 서비스를 선택하였을 경우입니다. 우선, 계좌번호를 입력합니다.

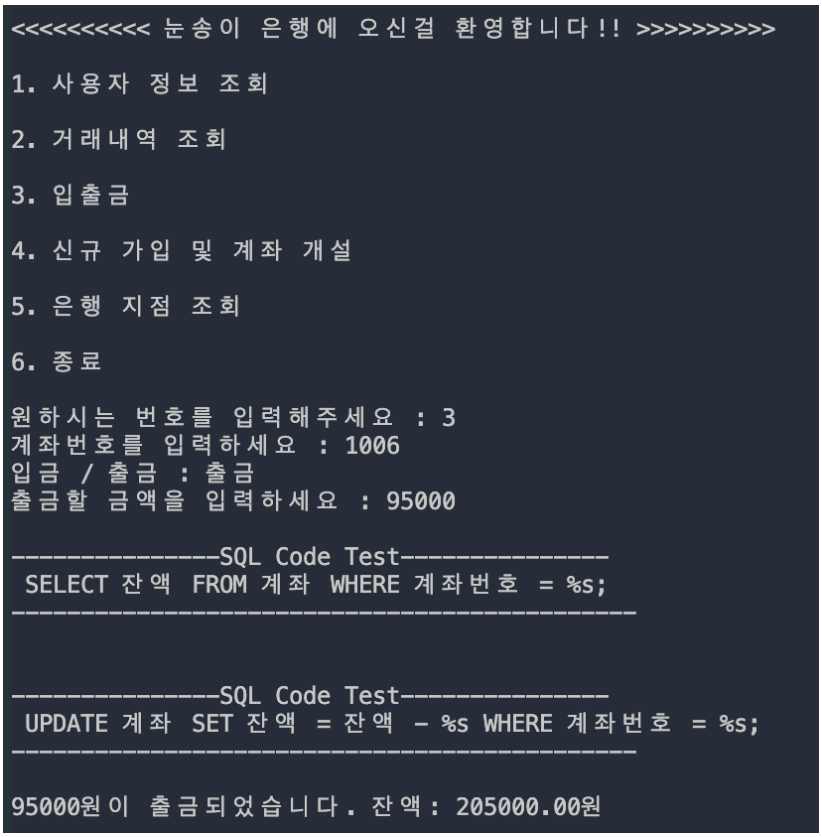
SQL CODE TEST가 진행되고 나서, 계좌번호의 주인인 사용자의 이름, 거래가 이루어진 시간, 거래유형,

거래 금액, 은행 지점 위치가 출력됩니다. 계좌의 거래내역을 조회할 수 있습니다.




3번 입출금 서비스를 선택하였을 경우입니다. 계좌번호를 입력하고, 입금과 출금 중 원하는 서비스를 입력합니다. 입금을 선택하였을 경우 입금할 금액을 입력합니다.

SQL CODE TEST가 진행되고나서, 입금이 완료되었다는 메세지가 뜨게 됩니다.



3번 입출금 서비스를 선택하였을 경우입니다. 계좌번호를 입력하고, 입금과 출금 중 원하는 서비스를 입력합니다. 출금을 선택하였을 경우 출금할 금액을 입력합니다.

SQL CODE TEST가 진행되고 나서, 출금이 완료되었다는 메세지와 함께 계좌의 잔액이 계산되어져 출력됩니다.

```
원하시는 번호를 입력해주세요 : 4
사용자 ID: 15
이름: 최보라
성별(M/F): F
연락처(010-0000-0000): 010-5589-7720
주민번호(900101-1234567): 9010155-885412
지점 번호 : 3003
계좌번호(4자리): 1015
계좌유형(저축/당좌/정기예금): 정기예금
초기 잔액: 50000000

---------------SQL Code Test---------------
 INSERT INTO 사용자 (사용자_ID, 이름, 성별, 연락처, 주민
번호, 지점_ID) VALUES (%s, %s, %s, %s, %s, %s);
-------------------------------------------


---------------SQL Code Test---------------
 INSERT INTO 계좌 (계좌번호, 사용자_ID, 계좌유형, 잔액,
지점_ID) VALUES (%s, %s, %s, %s, %s);
-------------------------------------------

새로운 사용자 및 계좌가 성공적으로 등록되었습니다!
```

4번 신규가입 및 계좌개설 서비스를 선택했을 경우입니다.

사용자 아이디, 이름, 성별, 연락처, 주민번호, 지점번호, 계좌번호, 계좌유형, 초기잔액을 입력받아 신규 사용자와 계좌를 개설합니다.

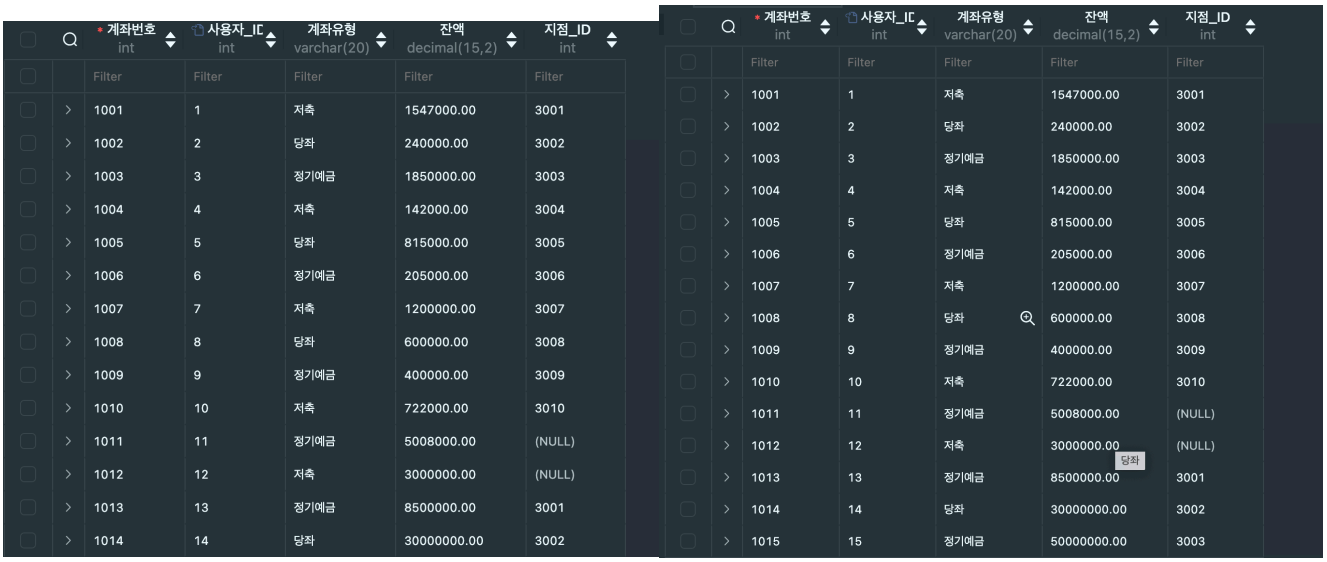
SQL Code Test가 진행되고나서, '새로운 사용자 및 계좌가 성공적으로 등록되었다'는 메세지가 출력됩니다.

아래 사진은 신규가입 및 계좌재설 하기 전과 후의 사진입니다.

| 사용자_ID int | 이름 varchar(50) | 성별 char(1) | 연락처 varchar(15) | 주민번호 char(15) | 지점_ID int |
|---|---|---|---|---|---|
| 1 | 김민수 | M | 010-1234-5678 | 900101-1234567 | 3001 |
| 2 | 이영희 | F | 010-2345-6789 | 910202-2345678 | 3002 |
| 3 | 박철수 | M | 010-3456-7890 | 920303-3456789 | 3003 |
| 4 | 최지은 | F | 010-4567-8901 | 930404-4567890 | 3004 |
| 5 | 정다영 | F | 010-5678-9012 | 940505-5678901 | 3005 |
| 6 | 오준호 | M | 010-6789-0123 | 950606-6789012 | 3006 |
| 7 | 한지민 | F | 010-7890-1234 | 960707-7890123 | 3007 |
| 8 | 김서준 | M | 010-8901-2345 | 970808-8901234 | 3008 |
| 9 | 신혜진 | F | 010-9012-3456 | 980909-9012345 | 3009 |
| 10 | 장우혁 | M | 010-0123-4567 | 990101-0123456 | 3010 |
| 11 | 김보배 | F | 010-5946-5123 | 905623- | (NULL) |
| 12 | 이가은 | F | 010-4453-2993 | 900213-2438594 | (NULL) |
| 13 | 구하리 | M | 010-8546-2213 | 9445126-7789526 | 3001 |
| 14 | 이효현 | M | 010-3049-2204 | 9011234-588493 | 3002 |

| 사용자_ID int | 이름 varchar(50) | 성별 char(1) | 연락처 varchar(15) | 주민번호 char(15) | 지점_ID int |
|---|---|---|---|---|---|
| 1 | 김민수 | M | 010-1234-5678 | 900101-1234567 | 3001 |
| 2 | 이영희 | F | 010-2345-6789 | 910202-2345678 | 3002 |
| 3 | 박철수 | M | 010-3456-7890 | 920303-3456789 | 3003 |
| 4 | 최지은 | F | 010-4567-8901 | 930404-4567890 | 3004 |
| 5 | 정다영 | F | 010-5678-9012 | 940505-5678901 | 3005 |
| 6 | 오준호 | M | 010-6789-0123 | 950606-6789012 | 3006 |
| 7 | 한지민 | F | 010-7890-1234 | 960707-7890123 | 3007 |
| 8 | 김서준 | M | 010-8901-2345 | 970808-8901234 | 3008 |
| 9 | 신혜진 | F | 010-9012-3456 | 980909-9012345 | 3009 |
| 10 | 장우혁 | M | 010-0123-4567 | 990101-0123456 | 3010 |
| 11 | 김보배 | F | 010-5946-5123 | 905623- | (NULL) |
| 12 | 이가은 | F | 010-4453-2993 | 900213-2438594 | (NULL) |
| 13 | 구하리 | M | 010-8546-2213 | 9445126-7789526 | 3001 |
| 14 | 이효현 | M | 010-3049-2204 | 9011234-588493 | 3002 |
| 15 | 최보라 | F | 010-5589-7720 | 9010155-885412 | 3003 |

(신규 사용자 개설 전과 후)



| 계좌번호 int | 사용자_ID int | 계좌유형 varchar(20) | 잔액 decimal(15,2) | 지점_ID int |
|---|---|---|---|---|
| 1001 | 1 | 저축 | 1547000.00 | 3001 |
| 1002 | 2 | 당좌 | 240000.00 | 3002 |
| 1003 | 3 | 정기예금 | 1850000.00 | 3003 |
| 1004 | 4 | 저축 | 142000.00 | 3004 |
| 1005 | 5 | 당좌 | 815000.00 | 3005 |
| 1006 | 6 | 정기예금 | 205000.00 | 3006 |
| 1007 | 7 | 저축 | 1200000.00 | 3007 |
| 1008 | 8 | 당좌 | 600000.00 | 3008 |
| 1009 | 9 | 정기예금 | 400000.00 | 3009 |
| 1010 | 10 | 저축 | 722000.00 | 3010 |
| 1011 | 11 | 정기예금 | 5008000.00 | (NULL) |
| 1012 | 12 | 저축 | 3000000.00 | (NULL) |
| 1013 | 13 | 정기예금 | 8500000.00 | 3001 |
| 1014 | 14 | 당좌 | 30000000.00 | 3002 |
| 1015 | 15 | 정기예금 | 50000000.00 | 3003 |

(신규 계좌 개설 전과 후)

```
원하시는 번호를 입력해주세요 : 5
지점 ID를 입력하세요 : 3001

---------------SQL Code Test---------------

    SELECT 지점.위치, 지점.지점장_이름, 지점.직원수, COUN
T(사용자.사용자_ID) AS 사용자_수
    FROM 지점
    JOIN 사용자 ON 사용자.지점_ID = 지점.지점_ID
    WHERE 지점.지점_ID = %s

-------------------------------------------

지점 위치 : '서울 종로구'
지점장 이름 : '김지훈'
직원 수 : '15'
사용자 수 : '2'
```

5번 은행지점 조회 서비스를 입력했을 경우입니다.

지점 아이디를 입력받습니다.

SQL Code Test를 진행하고나서.. 입력한 지점의 위치, 지점장 이름, 직원 수, 사용자 수(고객 수)를 출력합니다.

```
<<<<<<<<<< 눈송이 은행에 오신걸 환영합니다!! >>>>>>>>>>

1. 사용자 정보 조회

2. 거래내역 조회

3. 입출금

4. 신규 가입 및 계좌 개설

5. 은행 지점 조회

6. 종료

원하시는 번호를 입력해주세요 : 6
>>>>>>>>>> 프로그램을 종료합니다! <<<<<<<<<<
```

6번 종료를 입력했을 경우입니다.

프로그램을 종료합니다의 메세지가 뜨면서 프로그램이 종료됩니다.

## 5. Cross check

(시나리오) 웹 언어에서 바뀐 입출금 내역 확인


```
원하시는 번호를 입력해주세요 : 2
계좌번호를 입력하세요 : 1017

---------------SQL Code Test---------------

    SELECT 사용자.이름, 거래내역.거래일시, 거래내
역.거래유형, 거래내역.거래금액, 지점.위치
    FROM 거래내역
    JOIN 계좌 ON 거래내역.계좌번호 = 계좌.계좌번호
    JOIN 사용자 ON 계좌.사용자_ID = 사용자.사용자_
ID
    JOIN 지점 ON 사용자.지점_ID = 지점.지점_ID
    WHERE 계좌.계좌번호 = %s;

-----------------------------------------

이름 : '김영진'
거래일시 : (2024년, 11월, 23일, 15시:4분)
거래유형 : '입금'
거래금액 : '80000.00'
지점위치 : '인천 연수구'
```