

프로젝트 4: DBMS 프로그램 개발

2018007856 강민석

- import 및 mysql 접속

```
import random          # random과 string 라이브러리는 임의의 13자리 수 계좌번호를 생성하기 위해 필요
import string
import pymysql as pms
from getpass import getpass    # 비밀번호를 입력받을 때 비밀번호가 노출되지 않게 해주는 getpass 함수를 가지는 라이브러리

print("Accessing Bank Database ... ")
pw = getpass("Enter password: ")      # mysql에 접속하기 위해 비밀번호를 미리 입력받음.

connection = pms.connect(
    host='localhost',
    port=3306,
    user='root',
    password=pw,
    charset='utf8mb4',
    db='Bank'           # 생성한 Bank 데이터베이스에 접근.
)
```

코드 시작 부분에서는 필요한 라이브러리들을 import한다. 파이썬에서 mysql에 접근하는데 필요한 pymysql 라이브러리와 임의의 13자리 계좌번호를 생성하는데 필요한 random과 string 라이브러리, 마지막으로 비밀번호를 입력받을 때 비밀번호가 노출되지 않게 해주는 getpass 함수를 가지는 getpass 라이브러리를 import한다. 이 후에 사용자로부터 mysql의 root 비밀번호를 입력받은 후 connect 함수를 통해 mysql 서버에 접속한 후 Bank 데이터베이스에 접근한다.

- P3의 Relational Database Schema와 P2의 ER Diagram 수정

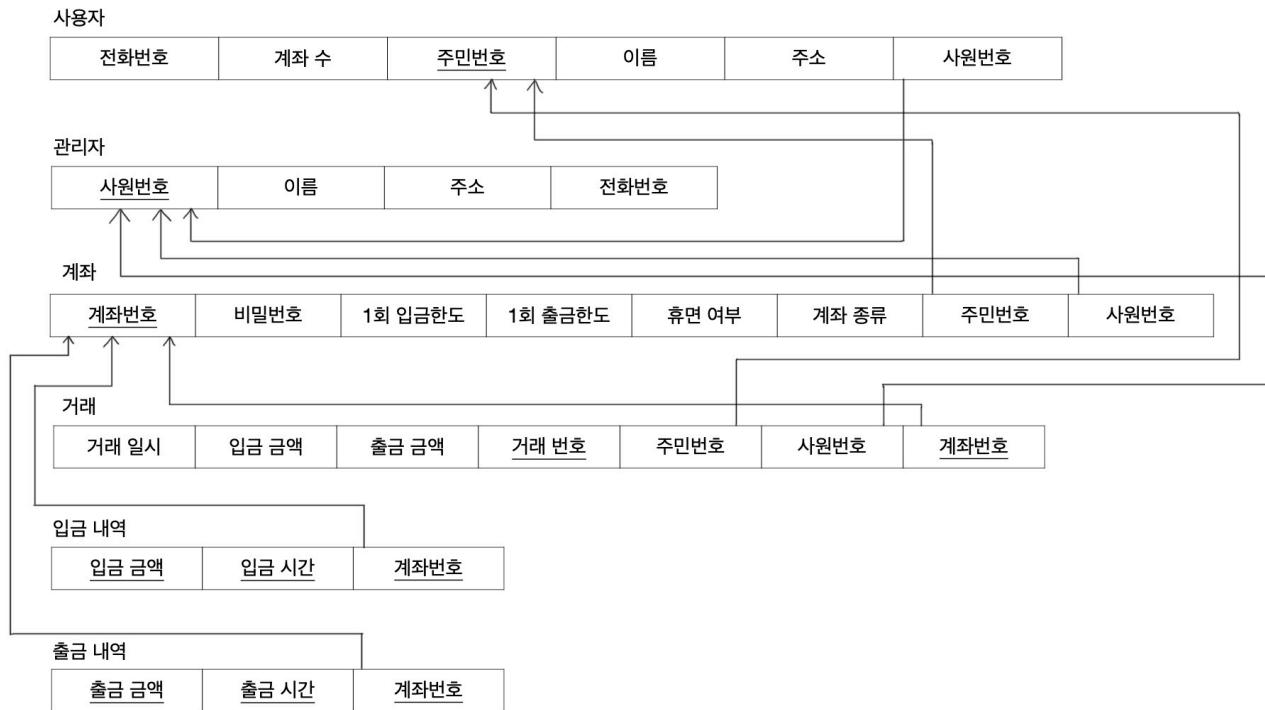
데이터베이스의 각 테이블 스키마를 작성하던 중 몇 가지 문제점을 발견하여 수정하였다.

첫째로 ‘거래’ Relation에서 ‘입금 금액’과 ‘출금 금액’ Attribute들이 있는데, 하나의 ‘거래’에서는 입금만 하거나 출금만 할 수 있기 때문에 이것을 실제 mysql로 구현시 ‘입금 금액’과 ‘출금 금액’ Attribute 둘 중 하나는 무조건 NULL값이 될 수밖에 없다. 따라서 tuple들이 불필요한 NULL값을 갖게 되므로 ‘입금 금액’과 ‘출금 금액’ Attribute들을 삭제하고 ‘입출금 금액’과 ‘거래 종류’ Attribute들을 새로 만들었다. 이렇게 하면 입금을 하든 출금을 하든 NULL값을 저장할 필요 없이 금액과 거래 종류만 저장하면 된다.

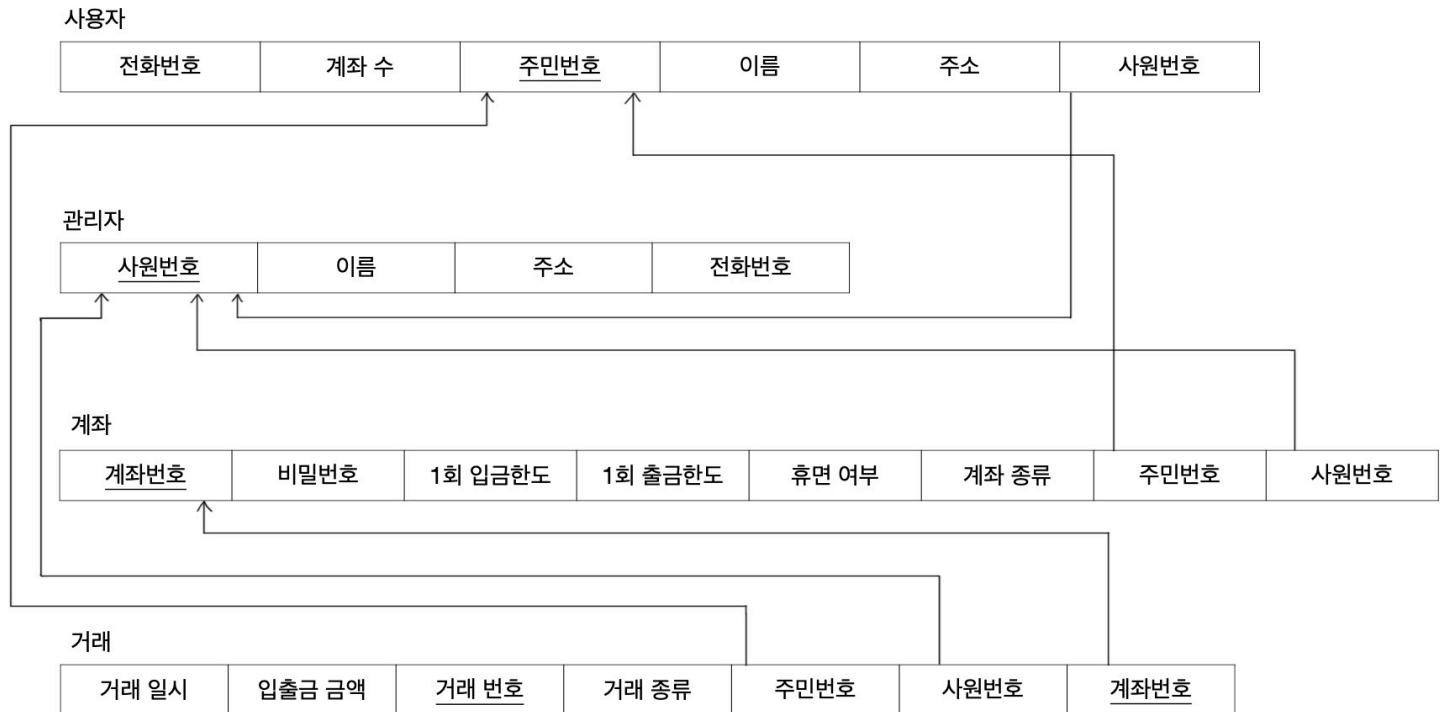
두번째 문제점은 ‘거래’ Relation과, ‘입금 내역’ 그리고 ‘출금 내역’ Relation의 Attribute들이 중복된다는 점이다. ‘거래’ Relation에 ‘거래 일시’와 ‘입출금 금액’ Attribute들이 이미 있는데 ‘입금 내역’과 ‘출금 내역’ Relation에도 각각 ‘입금 시간’과 ‘입금 금액’, 그리고 ‘출금 시간’과 ‘출금 금액’ Attribute들이 또 존재한다. 따라서 ‘입금 내역’과 ‘출금 내역’ Relation은 불필요하다고 생각되어 삭제하였다.

이 두 문제점을 반영하여 P3의 Relational Database Schema를 다음과 같이 수정하였다.

<수정 전 Relational Database Schema>

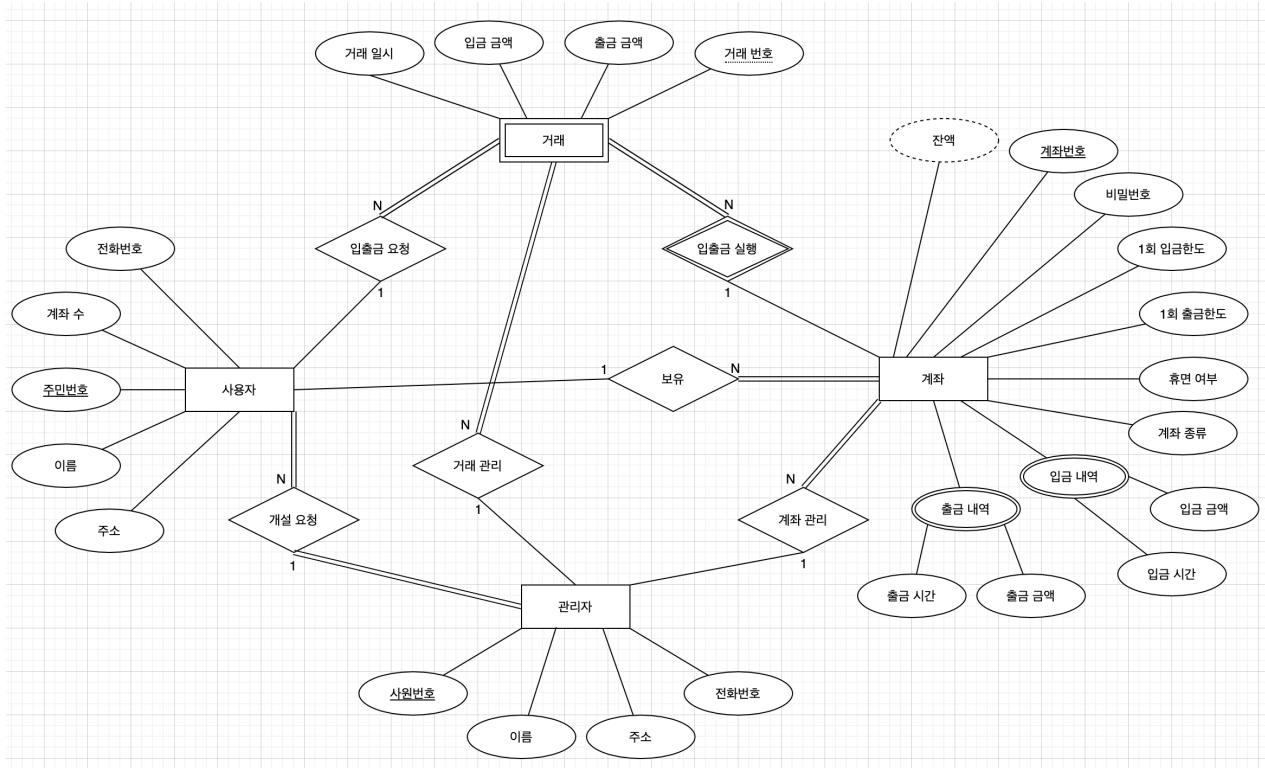


<수정 후 Relational Database Schema>

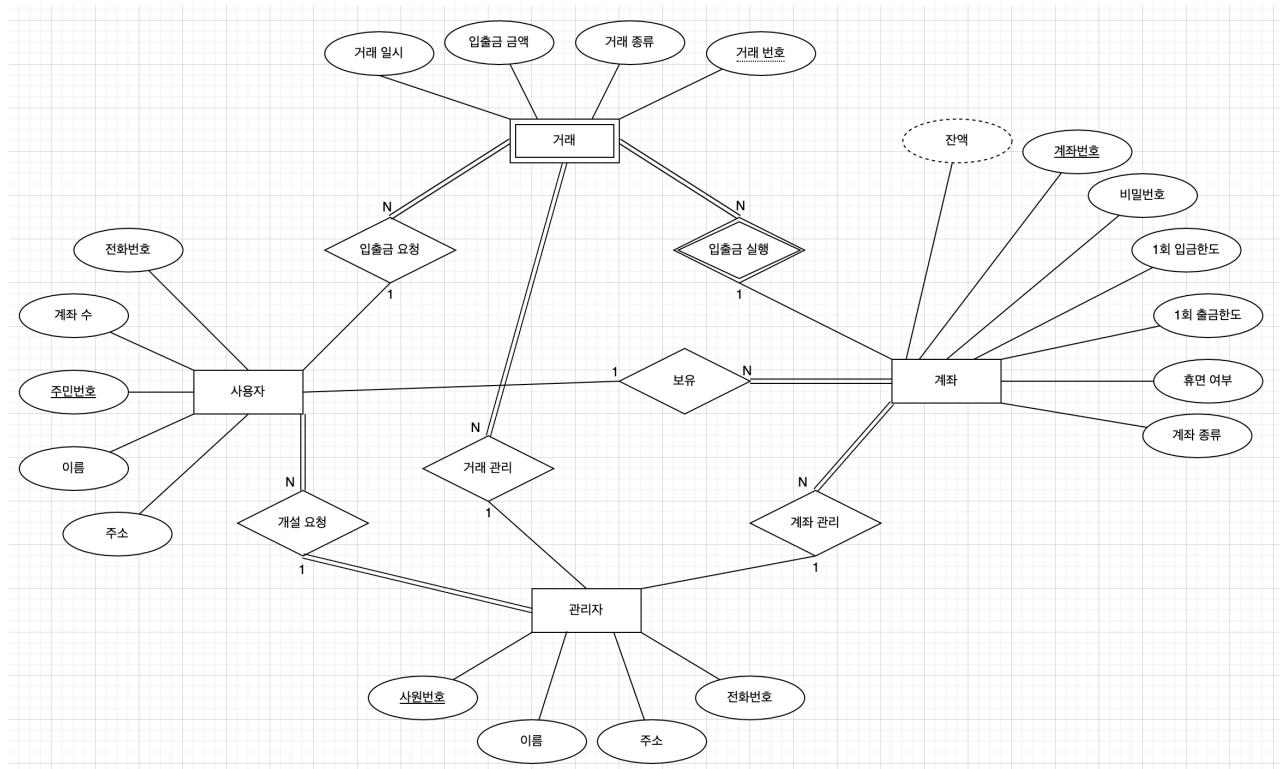


수정사항을 P2의 ER Diagram에도 반영하였다. ‘거래’ Entity의 ‘입금 금액’과 ‘출금 금액’ Attribute들을 삭제하고 ‘입출금 금액’과 ‘거래 종류’ Attribute들을 생성하였다. 그리고 ‘계좌’ Entity의 ‘입금 내역’과 ‘출금 내역’ Multi-valued Attribute들을 삭제하였다.

<수정 전 ER Diagram>



<수정 후 ER Diagram>



- 테이블 스키마 생성

초기 dump를 만들기 위해 P3에서 작성한 Relational Database Schema를 바탕으로 테이블 스키마를 CREATE TABLE sql문을 이용해 생성하였다. 모든 테이블에는 CREATE TABLE 뒤에 IF NOT EXISTS가 붙어서 dump가 생성된 이후에는 프로그램이 실행돼도 CREATE TABLE 부분은 실행되지 않고 넘어가도록 하였다.

- a) 관리자 Relation

```
try:  
    with connection.cursor() as cursor:  
        sql = """CREATE TABLE IF NOT EXISTS ADMIN (  
            AdminSsn CHAR(13) NOT NULL,  
            Name VARCHAR(20) NOT NULL,  
            Address VARCHAR(30),  
            Phone CHAR(11),  
            PRIMARY KEY (AdminSsn)  
        )"""  
        cursor.execute(sql)
```

관리자 Relation을 ADMIN이라는 테이블로 만들었다. 사원번호는 AdminSsn이라는 고정길이(13자리) 문자열이고 NULL값을 가져서는 안된다는 NOT NULL constraint를 붙였다. 이름은 Name이라는 가변길이(최대 20자리) 문자열이고 NOT NULL constraint를 붙였다. 주소는 Address라는 가변길이(최대 30자리) 문자열이다. 전화번호는 Phone이라는 고정길이(11자리) 문자열이다. Primary Key는 AdminSsn(사원번호)로 지정하였다.

- b) 사용자 Relation

```
sql = """CREATE TABLE IF NOT EXISTS USER (  
    Phone CHAR(11),  
    NumOfAccounts INT NOT NULL default 0,  
    Ssn CHAR(13) NOT NULL,  
    Name VARCHAR(20) NOT NULL,  
    Address VARCHAR(30),  
    ASsn CHAR(13) NOT NULL,  
    PRIMARY KEY (Ssn),  
    FOREIGN KEY (ASsn) REFERENCES ADMIN(AdminSsn)  
)"""  
cursor.execute(sql)
```

사용자 Relation을 USER라는 테이블로 만들었다. 전화번호는 Phone이라는 고정길이(11자리) 문자열이다. 계좌 수는 NumOfAccounts라는 INT형 숫자값으로 NULL 값을 가져서는 안된다는 NOT NULL constraint를 붙였고 DEFAULT 값은 0이다. DEFAULT는 따로 Attribute에 값을 지정해주지 않았을 때 자동으로 지정되는 값이다. 주민 번호는 Ssn이라는 고정길이(13자리) 문자열이고 NOT NULL constraint를 붙였다. 이름은 Name이라는 가변길이(최대 20자리) 문자열이고 NOT NULL constraint를 붙였다. 주소는 Address라는 가변길이(최대 30자리) 문자열이다. 사원번호는 ASsn이라는 고정길이(13자리) 문자열이고 NOT NULL constraint를 붙였다. Primary Key는 Ssn(주민번호)이다. FOREIGN KEY (ASsn) REFERENCES ADMIN(AdminSsn)은 ASsn이 ADMIN 테이블의 Primary Key인 AdminSsn을 참조하는 Foreign Key라는 것을 의미하는 구문이다.

c) 계좌 Relation

```
sql = """CREATE TABLE IF NOT EXISTS ACCOUNT (
    AccNum CHAR(13) NOT NULL,
    Password CHAR(4) NOT NULL,
    DepositLimit INT NOT NULL,
    WithdrawLimit INT NOT NULL,
    Dormant BOOLEAN NOT NULL default 0,
    AccountType VARCHAR(10) NOT NULL,
    AccSsn CHAR(13) NOT NULL,
    AdSsn CHAR(13) NOT NULL,
    PRIMARY KEY (AccNum),
    FOREIGN KEY (AccSsn) REFERENCES USER(Ssn) ON DELETE CASCADE,
    FOREIGN KEY (AdSsn) REFERENCES ADMIN(AdminSsn)
)"""
cursor.execute(sql)
```

계좌 Relation을 ACCOUNT라는 테이블로 만들었다. ACCOUNT의 모든 Attribute들은 NULL 값을 가져서는 안 된다는 NOT NULL constraint를 가진다. 계좌번호는 AccNum이라는 고정길이(13자리) 문자열이다. 비밀번호는 Password라는 고정길이(4자리) 문자열이다. 1회 입금한도는 DepositLimit이라는 INT형 숫자값이다. 1회 출금한도는 WithdrawLimit라는 INT형 숫자값이다. 휴면 여부는 Dormant라는 BOOLEAN형 1(True) or 0(False) 값이고 DEFAULT는 0(False)이다(따로 지정 안했을 시 자동으로 0을 지정해준다는 의미). 계좌 종류는 AccountType이라는 가변길이(최대 10자리) 문자열이다. 주민번호는 AccSsn이라는 고정길이(13자리) 문자열이다. 사원번호는 AdSsn이라는 고정길이(13자리) 문자열이다. Primary Key는 AccNum(계좌번호)이다. FOREIGN KEY (AccSsn) REFERENCES USER(Ssn)은 AccSsn이 USER 테이블의 Primary Key인 Ssn을 참조하는 Foreign Key라는 것을 의미하고 그 뒤에 붙은 ON DELETE CASCADE는 AccSsn이 참조하는 USER 테이블의 튜플이 삭제되면 AccSsn을 가지는 튜플도 삭제한다는 의미다. FOREIGN KEY (AdSsn) REFERENCES ADMIN(AdminSsn)은 AdSsn이 ADMIN 테이블의 Primary Key인 AdminSsn을 참조하는 Foreign Key라는 것을 의미하는 구문이다.

d) 거래 Relation

```
sql = """CREATE TABLE IF NOT EXISTS TRANSACTION (
    Date TIMESTAMP NOT NULL DEFAULT CURRENT_TIMESTAMP,
    Amount INT NOT NULL,
    TransType VARCHAR(10) NOT NULL,
    TNum INT NOT NULL AUTO_INCREMENT,
    TSsn CHAR(13) NOT NULL,
    TAdSsn CHAR(13) NOT NULL,
    TAccNum CHAR(13) NOT NULL,
    PRIMARY KEY (TNum, TAccNum),
    FOREIGN KEY (TSsn) REFERENCES USER(Ssn) ON DELETE CASCADE,
    FOREIGN KEY (TAdSsn) REFERENCES ADMIN(AdminSsn),
    FOREIGN KEY (TAccNum) REFERENCES ACCOUNT(AccNum) ON DELETE CASCADE
)"""
cursor.execute(sql)
connection.commit()
```

거래 Relation을 TRANSACTION이라는 테이블로 만들었다. TRANSACTION의 모든 Attribute들은 NULL 값을 가져서는 안된다는 NOT NULL constraint를 가진다. 거래 일시는 Date라는 TIMESTAMP형 날짜-시간 값이고 DEFAULT는 CURRENT_TIMESTAMP, 즉 현재 시간이다(따로 지정 안됐을 시 자동으로 현재 시간을 지정해준다는 의미). 입출금 금액은 Amount라는 INT형 숫자값이다. 거래 종류는 TransType라는 가변길이(최대 10자리) 문자열이다. 거래 번호는 TNum이라는 INT형 숫자값이고 뒤에 붙은 AUTO_INCREMENT는 TRANSACTION 테이블에 튜플이 새로 생성될 때마다 1부터 시작하여 값을 1씩 증가하면서 지정해주는 속성이다. TRANSACTION 테이블에 처음 생기는 튜플은 TNum이 1이고, 두번째 생기는 튜플은 TNum이 2가 되는 식으로 값이 증가한다. 주민번호는 TSsn, 사원번호는 TAdSsn, 계좌번호는 TAccNum이라는 고정길이(13자리) 문자열이다. FOREIGN KEY (TSsn) REFERENCES USER(Ssn)는 TSsn이 USER 테이블의 Primary Key인 Ssn을 참조한다는 의미이고 뒤에 붙은 ON DELETE CASCADE는 TSsn이 참조하는 USER 테이블의 튜플이 삭제되면 TSsn을 가지는 튜플도 삭제한다는 의미다. FOREIGN KEY (TAdSsn) REFERENCES ADMIN(AdminSsn)는 TAdSsn이 ADMIN 테이블의 Primary Key인 AdminSsn을 참조한다는 의미다. FOREIGN KEY (TAccNum) REFERENCES ACCOUNT(AccNum)는 TAccNum이 ACCOUNT 테이블의 Primary Key인 AccNum을 참조한다는 의미이고 뒤에 붙은 ON DELETE CASCADE는 TAccNum이 참조하는 ACCOUNT 테이블의 튜플이 삭제되면 TAccNum을 가지는 튜플도 삭제한다는 의미다.

모든 테이블 스키마를 생성하면 commit 함수를 호출해서 mysql에 반영시킨다.

- 프로그램 코드 설명

```
while True:  
    print("-----")  
    print("1. Register User")  
    print("2. Register Admin")  
    print("3. Login as User")  
    print("4. Login as Admin")  
    print("5. Exit")  
    start = input("Choose an instruction: ")
```

1. Register User - USER 테이블에 사용자를 등록하는 옵션
2. Register Admin - ADMIN 테이블에 관리자를 등록하는 옵션
3. Login as User - USER 테이블에 있는 사용자 중 한 명으로 로그인하는 옵션
4. Login as Admin - ADMIN 테이블에 있는 관리자 중 한 명으로 로그인하는 옵션
5. Exit - 프로그램 종료하는 옵션

start 변수에 input을 받아 해당 옵션을 실행한다.

참고사항 - 프로그램에서 사용되는 돈은 원화이며 단위는 원이다. 예를 들어 1000을 입금하면 1000원이 입금된다.

1. Register User

```
if start == '1':
    while True:
        print("-----")
        print("Registering User...")
        name = input("Name: ")                                # Name 입력 받음.
        if len(name) > 20:                                  # Name은 최대 20자리여야함.
            print("-----")
            print("Name must be within 20 characters.")
            continue
        ssn = input("Ssn (13 digits): ")                   # Ssn 입력 받음.
        if not (len(ssn) == 13 and ssn.isdigit()):          # Ssn은 13자리 숫자여야함.
            print("-----")
            print("Ssn must be a 13 digit number.")
            continue
        phone = input("Phone Number (11 digits): ")         # Phone 입력 받음.
        if not (len(phone) == 11 and phone.isdigit()):       # Phone은 11자리 숫자여야함.
            print("-----")
            print("Phone number must be a 11 digit number.")
            continue
        address = input("Address: ")                          # Address 입력 받음.
        if len(address) > 30:                               # Address는 최대 30자리여야함.
            print("-----")
            print("Address must be within 30 characters.")
            continue

# ADMIN에서 튜플 하나를 무작위로 선정하고 튜플의 AdminSsn 값을 가져오는 SQL문.
sql = "SELECT AdminSsn FROM ADMIN ORDER BY rand() LIMIT 1"
cursor.execute(sql)
adssn = cursor.fetchall()
# USER 테이블에 튜플을 INSERT할 때 위에서 가져온 AdminSsn값을 튜플의 ASsn값으로 지정.
# USER 테이블에 INSERT한 튜플이 ADMIN 테이블의 튜플 하나를 참조하게 됨.
sql = "INSERT INTO USER(Name, Ssn, Phone, Address, ASsn) VALUES (%s, %s, %s, %s, %s)"
cursor.execute(sql, (name, ssn, phone, address, adssn))
print("-----")
print("Register complete.")
connection.commit()
break
```

USER 테이블에 사용자를 등록하는 옵션이다. USER 테이블의 Name, Ssn, Phone, Address Attribute들에 대한 값을 입력으로 받는다. while문을 돌면서 Attribute의 조건에 맞지 않는 값이 들어오면 continue를 통해 다시 처음부터 값을 입력받는다. 그리고 SELECT AdminSsn FROM ADMIN ORDER BY rand() LIMIT 1라는 sql문을 통해 ADMIN 테이블에서 튜플 하나를 무작위로 선정하고 튜플의 AdminSsn 값을 가져와서 adssn 변수에 저장 한다. ORDER BY rand()는 테이블에 있는 튜플의 순서들을 무작위로 정한다는 의미이고 LIMIT 1는 그중 하나만 고른다는 의미다. INSERT INTO USER(Name, Ssn, Phone, Address, ASsn) VALUES(%s, %s, %s, %s, %s)라는 sql문을 통해 튜플을 저장하는데, 이때 ASsn에는 ADMIN 테이블에서 가져온 AdminSsn값을 넣어준다. 이렇게 하면 USER 테이블에 삽입한 튜플이 ADMIN 테이블의 튜플 하나를 참조하게 된다. 정확히는 USER 테이블에 있는 튜플의 ASsn이라는 Foreign Key가 ADMIN 테이블에 있는 튜플의 Primary Key인 AdminSsn을 가리키게 된다. 튜플이 생성되면 mysql에 commit하고 while문을 빠져나온다.

2. Register Admin

```
elif start == '2':
    while True:
        print("-----")
        print("Registering Admin...")
        ssn = input("Ssn (13 digits): ")           # AdminSsn 입력받음.
        if not (len(ssn) == 13 and ssn.isdigit()):   # AdminSsn은 13자리 숫자여야함.
            print("-----")
            print("Ssn must be a 13 digit number.")
            continue
        name = input("Name: ")                      # Name 입력 받음.
        if len(name) > 20:                         # Name은 최대 20자리여야함.
            print("-----")
            print("Name must be within 20 characters.")
            continue
        address = input("Address: ")                # Address 입력 받음.
        if len(address) > 30:                       # Address는 최대 30자리여야함.
            print("-----")
            print("Address must be within 30 characters.")
            continue
        phone = input("Phone Number (11 digits): ")  # Phone 입력 받음.
        if not (len(phone) == 11 and phone.isdigit()): # Phone은 11자리 숫자여야함.
            print("-----")
            print("Phone number must be a 11 digit number.")
            continue
        # ADMIN 테이블에 입력받은 정보들이 담긴 새로운 튜플 삽입.
        sql = "INSERT INTO ADMIN/AdminSsn, Name, Address, Phone) VALUES (%s, %s, %s, %s)"
        cursor.execute(sql, (ssn, name, address, phone))
        print("-----")
        print("Register complete.")
        connection.commit()
        break
```

ADMIN 테이블에 관리자를 등록하는 옵션이다. ADMIN 테이블의 AdminSsn, Name, Address, Phone Attribute들에 대한 값을 입력으로 받는다. while문을 돌면서 Attribute의 조건에 맞지 않는 값이 들어오면 continue를 통해 다시 처음부터 값을 입력받는다. INSERT INTO ADMIN/AdminSsn, Name, Address, Phone) VALUES(%s, %s, %s, %s)라는 sql문을 통해 ADMIN 테이블에 입력받은 정보들이 담긴 새로운 튜플을 삽입한다. 마지막으로 mysql에 commit하고 while문을 빠져나온다.

3. Login as User

```
elif start == '3':
    print("-----")
    ssn = input("Enter Ssn: ")                                # 사용자로부터 Ssn을 입력 받음.
    sql = "SELECT Ssn FROM USER WHERE Ssn = %s"            # 사용자가 입력한 Ssn과 일치하는 Ssn을 갖는 튜플을 찾아서 Ssn값 반환.
    cursor.execute(sql, ssn)
    select = cursor.fetchall()
    if not select:                                         # 비어있는 row가 반환되면 사용자가 입력한 Ssn이 존재하지 않는다는 것으로.
        print("-----")                                     # 등록되지 않은 사용자라는 메시지를 출력하고 다시 메뉴로 복귀.
        print("You are not a registered user.")
        continue
    sql = "SELECT Name FROM USER WHERE Ssn = %s"          # 사용자가 입력한 Ssn을 갖는 튜플의 Name값을 반환.
    cursor.execute(sql, ssn)
    select = cursor.fetchall()
    print("-----")
    print(f"Welcome, {select[0][0]}")                         # Name값을 이용해 사용자를 맞이하는 문장 출력.
    while True:
        print("-----")
        print("1. Create Account")
        print("2. Select Account")
        print("3. Update User Info")
        print("4. View User Info")
        print("5. Delete User")
        print("6. Exit")
        login_user = input("Choose an instruction: ")
```

USER로 로그인할 수 있는 옵션이다. 사용자로부터 Ssn을 입력 받고 “SELECT Ssn FROM USER WHERE Ssn = %s”라는 sql문을 통해 사용자가 입력한 Ssn과 일치하는 Ssn을 갖는 튜플을 찾아서 Ssn값을 반환한다. 비어있는 row가 반환되면 사용자가 입력한 Ssn이 존재하지 않는다는 것으로 등록되지 않은 사용자라는 메시지를 출력하고 다시 메뉴로 복귀한다. 등록된 사용자면 “SELECT Name FROM USER WHERE Ssn = %s”라는 sql문을 통해 사용자가 입력한 Ssn을 갖는 튜플의 Name값을 반환한다. 이 Name값을 이용해 사용자를 맞이하는 문장을 출력하고 USER로 로그인한 상태에서 가능한 옵션들을 나열한 메뉴를 출력한다.

3. Login as User

1. Create Account

```
if login_user == '1':  # 계좌 생성
    # 로그인한 사용자의 Ssn을 갖는 튜플에서 NumOfAccounts를 가져옴으로써 사용자의 계좌 수를 알아낸다.
    sql = "SELECT NumOfAccounts FROM USER WHERE Ssn = %s"
    cursor.execute(sql, ssn)
    numofAccounts = cursor.fetchall()
    if numofAccounts[0][0] >= 10:                           # 계좌 수가 10개 이상이면 계정 생성을 막는다.
        print("-----")
        print("You can only have 10 accounts maximum.")
        continue
```

새로운 ACCOUNT(계좌)를 생성하는 옵션이다. “SELECT NumOfAccounts FROM USER WHERE Ssn = %s”라는 sql문을 통해 로그인한 사용자의 Ssn을 갖는 튜플에서 NumOfAccounts(계좌 수)를 가져온다. 계좌 수가 10개 이상이면 계좌는 최대 10개까지만 보유할 수 있다는 문구를 출력하고 continue를 통해 Login as User 메뉴로 복귀한다.

```

while True:
    print("-----")
    print("Creating account...")

    # Password를 입력 받음. Password는 4자리 숫자여야함.
    password = getpass("Password (4 digits): ")
    if not(len(password) == 4 and password.isdigit()):
        print("-----")
        print("Password must be 4 digits.")
        continue

    # DepositLimit 입력 받음. DepositLimit은 숫자여야함.
    depositLimit = input("Deposit Limit (for a single transaction): ")
    if not depositLimit.isdigit():
        print("-----")
        print("Deposit Limit must be a number.")
        continue

    # WithdrawalLimit 입력 받음. WithdrawalLimit은 숫자여야함.
    withdrawLimit = input("Withdrawal Limit (for a single transaction): ")
    if not withdrawLimit.isdigit():
        print("-----")
        print("Withdrawal Limit must be a number.")
        continue

    # AccountType 입력 받음. AccountType은 'checking' 혹은 'savings' 둘 중 하나만 가능함.
    accountType = input("Account Type (Type 'checking' or 'savings'): ")
    if not (accountType == "checking" or accountType == "savings"):
        print("-----")
        print("Account Type must be either 'checking' or 'savings'.")
        continue

    while True:      # 중복되지 않는 계좌번호가 나올 때까지 while문 반복.
        # 임의의 13자리 숫자로 된 문자열을 반환해주는 함수. 이 함수로 계좌번호를 생성한다.
        accnum = ''.join(random.choice(string.digits) for _ in range(13))
        # 임의로 생성한 계좌번호(AccNum)가 이미 ACCOUNT 테이블에 존재하는지 확인하는 sql문.
        sql = "SELECT EXISTS (SELECT * FROM ACCOUNT WHERE AccNum = %s)"
        cursor.execute(sql, accnum)
        if not cursor.fetchall()[0][0]:      # 존재하지 않으면 while문 빠져나옴.
            break

    # 로그인한 사용자의 ASsn을 반환. ASsn은 ADMIN의 AdminSsn을 참조하는 Foreign Key이다.
    sql = "SELECT ASsn FROM USER WHERE Ssn = %s"
    cursor.execute(sql, ssn)
    adssn = cursor.fetchall()

    sql = """INSERT INTO ACCOUNT(AccNum, Password, DepositLimit, WithdrawLimit,
                           AccountType, AccSsn, AdSsn) VALUES (%s, %s, %s, %s, %s, %s)"""
    cursor.execute(sql, (accnum, password, depositLimit, withdrawLimit, accountType, ssn,
                        adssn))

    # ACCOUNT 테이블에 튜플을 생성하면 사용자의 계좌가 하나 늘어난 것이므로 사용자의 NumOfAccounts 업데이트.
    sql = "UPDATE USER SET NumOfAccounts = NumOfAccounts + 1 WHERE Ssn = %s"
    cursor.execute(sql, ssn)
    print("-----")
    print("Account successfully created.")
    print(f"Your account number: {accnum}")
    connection.commit()
    break

```

ACCOUNT 테이블에 들어갈 튜플을 만들기 위한 Attribute 값들을 입력으로 받는다. Password, DepositLimit, WithdrawLimit, AccountType을 순서대로 받으면서 조건에 맞지 않는 값이 들어오면 continue를 통해 처음부터 다시 입력을 받는다.

코드 시작 부분에서 import한 random과 string 라이브러리에 있는 함수들을 이용해 임의의 13자리 숫자로 된 문자열을 생성한다. 사용자의 새 계좌에 계좌번호를 부여하기 위해 임의의 계좌번호를 생성하고 “SELECT EXISTS (SELECT * FROM ACCOUNT WHERE AccNum = %s)”라는 sql문을 통해 중복이 있는지 확인한다. 만약 nested query가 비어있는 row를 반환하면 최종적으로 False(0)을 반환하고, 생성한 계좌번호와 일치하는 AccNum을 가지는 튜플을 반환하면 최종적으로 True(1)를 반환한다. 중복이 있으면 다시 계좌번호를 생성하고 그렇지 않다면 while문을 빠져나온다.

그리고 나서 “SELECT ASsn FROM USER WHERE Ssn = %s”라는 sql문을 통해 로그인한 사용자의 ASsn, 즉 참조하는 ADMIN의 AdminSsn을 반환한다. 이 값이 ACCOUNT의 AdSsn, 그리고 사용자의 Ssn이 ACCOUNT의 AccSsn이 된다. 앞서 입력받은 Password, DepositLimit, WithdrawLimit, AccountType과 AdSsn, AccSsn을 “INSERT INTO ACCOUNT(…) VALUES (…)”라는 sql문을 통해 ACCOUNT 테이블에 삽입한다. 이때 사용자의 계좌가 하나 더 생긴 것이므로 “UPDATE USER SET NumOfAccounts = NumOfAccounts + 1 WHERE Ssn = %s”라는 sql문을 통해 사용자의 NumOfAccounts를 1만큼 증가시킨다. 모든 과정이 완료되면 계좌 생성이 완료되었다는 문구와 계좌번호를 출력한다. 마지막으로 mysql에 commit하여 반영시킨다.

3. Login as User

2. Select Account

```
elif login_user == '2':
    print("-----")
    sql = "SELECT AccNum, DepositLimit, WithdrawLimit, Dormant, AccountType FROM ACCOUNT WHERE " \
          "AccSsn = %s"
    cursor.execute(sql, ssn)
    select = cursor.fetchall()
    print("{:<15} {:<20} {:<20} {:<15} {:<20}".format("Account Number", "Deposit Limit",
                                                       "Withdrawal Limit", "Status", "Account Type"))
    for row in select:      # Dormant값이 0이면 Active, 1이면 Dormant인 상태.
        if row[3] == 0:
            status = "Active"
        else:
            status = "Dormant"
        print("{:<15} {:<20} {:<20} {:<15} {:<20}".format(row[0], row[1], row[2], status, row[4]))
    print("-----")
```

사용자가 가지고 있는 계좌들을 모두 보여준 뒤 하나를 선택할 수 있게 해주는 옵션이다. SELECT문을 활용해 ACCOUNT 테이블에서 로그인한 사용자의 Ssn과 일치하는 AccSsn을 가지는 튜플들만을 고르고 그 튜플들의 AccNum(계좌번호), DepositLimit(1회 입금한도), WithdrawLimit(1회 출금한도), Dormant(휴면 여부), AccountType(계좌 종류)을 보여주는 row들을 반환한다. format함수를 이용해 적절히 표를 만들어 for문을 돌면서 row들을 전부 출력하는데 이때 Dormant(휴면 여부)값이 0이면 ‘Active’, 1이면 ‘Dormant’를 출력해 계좌의 휴면 여부를 표시한다.

```

accnum = input("Enter account number: ")           # 계좌번호를 입력 받음.
sql = "SELECT AccNum FROM ACCOUNT WHERE AccNum = %s" # 입력한 계좌번호와 일치하는 AccNum 찾아서 반환.
cursor.execute(sql, accnum)
select = cursor.fetchall()
if not select:                                     # 일치하는 AccNum이 없으면 직전 메뉴로 복귀.
    print("-----")
    print("Wrong account number.")
    continue
password = getpass("Enter password: ")            # 비밀번호를 입력 받음.
sql = "SELECT Password FROM ACCOUNT WHERE Accnum = %s" # 선택한 계좌의 Password를 반환.
cursor.execute(sql, accnum)
select = cursor.fetchall()
if password != select[0][0]:                      # 입력한 비밀번호가 계좌의 Password와 일치하지 않으면 직전 메뉴로 복귀.
    print("-----")
    print("Wrong password.")
    continue
while True:                                         # 비밀번호가 계좌의 Password와 일치하면 다음 메뉴로 진행.
    print("-----")
    print("1. Deposit")
    print("2. Withdrawal")
    print("3. Check Balance")
    print("4. View Transactions")
    print("5. Update Account Info")
    print("6. Delete Account")
    print("7. Exit")
    trans = input("Choose instruction: ")

```

계좌들을 전부 출력한 후 사용자로부터 계좌번호를 입력받는다. 사용자가 입력한 계좌번호를 accnum 변수에 저장하고, “SELECT AccNum FROM ACCOUNT WHERE AccNum = %s”라는 sql문을 통해 accnum과 일치하는 AccNum을 ACCOUNT 테이블에서 찾아서 AccNum을 반환한다. 만약 일치하는 AccNum이 없으면 존재하지 않는 계좌이므로 직전 메뉴로 복귀한다. 이후 사용자로부터 비밀번호를 입력받고 password 변수에 저장한다. “SELECT Password FROM ACCOUNT WHERE AccNum = %s”라는 sql문을 통해 accnum과 일치하는 AccNum을 갖는 튜플을 찾아 그 튜플의 Password를 반환하고 이 Password를 입력받은 password와 비교한다. 일치하지 않으면 직전 메뉴로 복귀하고 일치하면 다음 메뉴로 진행한다.

```

# TRANSACTION의 튜플 중에서 입력받은 계좌번호와 TAccNum이 일치하고 TransType(거래 종류)가 Deposit(입금)에
# 해당하는 튜플들을 골라 Amount(금액)의 합산을 구함.
sql = "SELECT SUM(Amount) FROM TRANSACTION WHERE TAccNum = %s AND TransType = 'Deposit'"
cursor.execute(sql, accnum)
deposit = cursor.fetchall()[0][0]
# TRANSACTION의 튜플 중에서 입력받은 계좌번호와 TAccNum이 일치하고 TransType(거래 종류)가 Withdrawal(출금)에
# 해당하는 튜플들을 골라 Amount(금액)의 합산을 구함.
sql = "SELECT SUM(Amount) FROM TRANSACTION WHERE TAccNum = %s AND TransType = 'Withdrawal'"
cursor.execute(sql, accnum)
withdraw = cursor.fetchall()[0][0]

balance = 0                                         # 입금과 출금 모두 없다면 balance(잔액)은 0.
if deposit and withdraw:                          # 입금과 출금 둘다 존재하면 balance는 입금 금액에서 출금 금액을 뺀 값.
    balance = deposit - withdraw
elif deposit:                                      # 입금만 존재하고 출금 기록은 없다면 balance는 입금 금액의 합산임.
    balance = deposit

```

다음 메뉴로 진행하기 전에 balance, 즉 계좌의 잔액을 설정해준다. “SELECT SUM(Amount) FROM TRANSACTION WHERE TAccNum = %s AND TransType = ‘Deposit’”라는 sql문을 통해 TRANSACTION의 튜플 중에서 입력받은 계좌번호와 TAccNum이 일치하고 TransType(거래 종류)가 Deposit(입금)에 해당하는 튜플들을 골라 Amount(금액)의 합산을 SUM이라는 Aggregate Function을 이용해서 구한다. 입금 금액의 합산을 deposit이라는 변수에 저장한다. 마찬가지 방법으로 출금 금액의 합산을 withdraw라는 변수에 저장한다. 만약 입금 내역과 출금 내역이 모두 없으면 balance는 0이고, 둘다 존재하면 deposit에서 withdraw를 뺀 값이다. 만약 입금 내역만 존재하고 출금 내역은 없으면, balance는 deposit, 즉 입금 금액의 합산이다.

3. Login as User

2. Select Account

1. Deposit

```

if trans == "1":
    print("-----")
    deposit = input("Enter deposit amount: ")           # Amount(입금 금액)을 입력 받음.
    if not deposit.isdigit():                          # 숫자가 아니라면 continue.
        print("-----")
        print("Deposit amount must be a number.")
        continue
    sql = "SELECT DepositLimit FROM ACCOUNT WHERE AccNum = %s"  # 계좌의 입금 한도 확인
    cursor.execute(sql, accnum)
    depositLimit = cursor.fetchall()[0][0]
    if int(deposit) > depositLimit:                  # 입금 금액이 한도를 초과하면 continue.
        print("-----")
        print("Deposit amount exceeds Deposit Limit.")
        continue

# ASsn은 USER가 참조하는 ADMIN의 AdminSsn. TRANSACTION의 TAdSsn은 ADMIN을 참조하는 Foreign
# Key인데, TAdSsn을 USER의 ASsn으로 지정함으로써 TRANSACTION과 USER 모두 같은 ADMIN을 가리키게 함.
sql = "SELECT ASsn FROM USER WHERE Ssn = %s"
cursor.execute(sql, ssn)
adssn = cursor.fetchall()
sql = "INSERT INTO TRANSACTION (Amount, TransType, TSsn, TAdSsn, TAccNum) VALUES" \
      "(%s, %s, %s, %s, %s)"
# 입금이므로 TransType은 Deposit
cursor.execute(sql, (deposit, "Deposit", ssn, adssn, accnum))
connection.commit()

```

사용자가 선택한 계좌에 입금하는 옵션이다. 입금 금액을 입력 받는데 이때 입력값이 숫자가 아니면 메뉴로 돌아간다. “SELECT DepositLimit FROM ACCOUNT WHERE AccNum = %s”라는 sql문을 통해 선택한 계좌의 입금 한도를 알아낸다. 만약 입력받은 금액이 입금 한도를 초과하면 메뉴로 돌아간다. 조건을 모두 통과하면 “SELECT ASsn FROM USER WHERE Ssn = %s”라는 sql문을 통해 로그인한 USER의 ASsn, 즉 USER가 참조하는 ADMIN의 AdminSsn을 가져온다. TRANSACTION의 TAdSsn은 ADMIN의 AdminSsn을 참조하는 Foreign Key인데, TAdSsn을 USER의 ASsn으로 지정함으로써 TRANSACTION과 USER 모두 같은 ADMIN을 가리키게 한다. “INSERT INTO TRANSACTION (...) VALUES (...)”라는 sql문을 통해 입금 내역을 TRANSACTION 테이블에 저장하는데, 이때 TransType(거래 종류)는 Deposit(입금)으로 지정한다. 완료되면 commit을 통해 mysql에 반영한다.

3. Login as User

2. Select Account

2. Withdrawal

```
elif trans == "2":  
    print("-----")  
    withdraw = input("Enter withdrawal amount: ")           # Amount(출금 금액)을 입력 받음.  
    if not withdraw.isdigit():  
        print("-----")  
        print("Withdrawal amount must be a number.")  
        continue  
    sql = "SELECT WithdrawLimit FROM ACCOUNT WHERE AccNum = %s"      # 계좌의 출금 한도 확인  
    cursor.execute(sql, accnum)  
    withdrawLimit = cursor.fetchall()[0][0]  
    if balance - int(withdraw) < 0:                         # 출금 금액이 balance보다 크면 continue.  
        print("-----")  
        print("Withdrawal amount exceeds balance.")  
        continue  
    if int(withdraw) > withdrawLimit:                      # 입금 금액이 한도를 초과하면 continue.  
        print("-----")  
        print("Withdrawal amount exceeds Withdraw Limit.")  
        continue  
  
    # ASsn은 USER가 참조하는 ADMIN의 AdminSsn. TRANSACTION의 TAdSsn은 ADMIN을 참조하는 Foreign  
    # Key인데, TAdSsn을 USER의 ASsn으로 지정함으로써 TRANSACTION과 USER 모두 같은 ADMIN을 가리키게 함.  
    sql = "SELECT ASsn FROM USER WHERE Ssn = %s"  
    cursor.execute(sql, ssn)  
    adssn = cursor.fetchall()  
    sql = """INSERT INTO TRANSACTION (Amount, TransType, TSsn, TAdSsn, TAccNum) VALUES  
    (%s, %s, %s, %s, %s)"""  
    # 출금이므로 TransType은 Withdrawal  
    cursor.execute(sql, (withdraw, "Withdrawal", ssn, adssn, accnum))  
    connection.commit()
```

사용자가 선택한 계좌에서 출금하는 옵션이다. 입금하는 옵션과 코드가 거의 동일한데 입력받은 출금 금액이 적합한지 판단하는 조건이 하나 더 추가되었다. 만약 입력받은 출금 금액이 balance(잔액)보다 많으면 더 이상 진행하지 않고 continue를 통해 메뉴로 돌아간다. 이를 제외한 나머지 코드는 입금과 동일하다.

3. Login as User

2. Select Account

3. Check Balance

```
elif trans == "3":  
    print("-----")  
    print(f"Balance: {balance}")
```

잔액을 확인하는 옵션이다. balance를 출력해준다.

3. Login as User

2. Select Account

4. View Transactions

```
elif trans == "4":  
    print("-----")  
    # 선택한 계좌의 AccNum과 TRANSACTION 튜플들의 TAccNum을 비교하고 일치하면 금액, 거래종류, 거래일시를 가져옴.  
    sql = "SELECT Amount, TransType, Date FROM TRANSACTION WHERE TAccNum = %s"  
    cursor.execute(sql, accnum)  
    select = cursor.fetchall()  
    print("{:<10} {:<20} {:<25}".format("Amount", "Transaction Type", "Date"))  
    for row in select:  
        print("{:<10} {:<20} {:<25}".format(row[0], row[1], str(row[2])))
```

선택한 계좌에서 발생한 거래들을 보여주는 옵션이다. “SELECT Amount, TransType, Date FROM TRANSACTION WHERE TAccNum = %s”라는 sql문을 통해 선택한 계좌의 AccNum과 TRANSACTION 튜플들의 TAccNum을 비교하고 일치하면 Amount(금액), TransType(거래 종류), Date(거래 일시)를 가져온다. format 함수를 통해 표 형태를 만든 다음 차례대로 row를 출력해서 어떤 거래들이 이루어졌는지 보여준다.

3. Login as User

2. Select Account

5. Update Account Info

```
elif trans == "5":  
    while True:  
        print("-----")  
        print("1. Update Password")  
        print("2. Update Deposit Limit")  
        print("3. Update Withdrawal Limit")  
        print("4. Update Account Type")  
        print("5. Exit")  
        update_account = input("Choose instruction: ")
```

선택한 계좌의 정보를 변경할 수 있는 옵션이다.

3. Login as User

2. Select Account

5. Update Account Info

1. Update Password

```
if update_account == '1':
    password = getpass("Password (4 digits): ")           # Password 입력 받음.
    if not (len(password) == 4 and password.isdigit()):   # 4자리 숫자가 아니면 continue.
        print("-----")
        print("Password must be 4 digits.")
        continue
    # ACCOUNT 테이블에서 선택한 계좌에 해당하는 튜플 찾아서 Password를 변경함.
    sql = "UPDATE ACCOUNT SET Password = %s WHERE AccNum = %s"
    cursor.execute(sql, (password, accnum))
    print("-----")
    print("Password updated.")
```

선택한 계좌의 비밀번호를 변경하는 옵션이다. 입력받은 비밀번호는 4자리 숫자여야 한다. “UPDATE ACCOUNT SET Password = %s WHERE AccNum = %s”라는 sql문을 통해 ACCOUNT 테이블에서 선택한 계좌에 해당하는 튜플을 찾아서 그 튜플의 Password(비밀번호)를 변경한다.

3. Login as User

2. Select Account

5. Update Account Info

2. Update Deposit Limit

```
elif update_account == '2':
    # DepositLimit 입력 받음. 숫자가 아니면 continue.
    depositLimit = input("Deposit Limit (for a single transaction): ")
    if not depositLimit.isdigit():
        print("-----")
        print("Deposit Limit must be a number.")
        continue
    # ACCOUNT 테이블에서 선택한 계좌에 해당하는 튜플 찾아서 DepositLimit을 변경함.
    sql = "UPDATE ACCOUNT SET DepositLimit = %s WHERE AccNum = %s"
    cursor.execute(sql, (depositLimit, accnum))
    print("-----")
    print("Deposit Limit updated.")
```

선택한 계좌의 입금 한도를 변경하는 옵션이다. 입력받은 입금 한도는 숫자여야 한다. “UPDATE ACCOUNT SET DepositLimit = %s WHERE AccNum = %s”라는 sql문을 통해 ACCOUNT 테이블에서 선택한 계좌에 해당하는 튜플을 찾아서 그 튜플의 DepositLimit(입금 한도)를 변경한다.

3. Login as User

2. Select Account

5. Update Account Info

3. Update Withdrawal Limit

```
elif update_account == '3':
    # WithdrawLimit 입력 받음. 숫자가 아니면 continue.
    withdrawLimit = input("Withdrawal Limit (for a single transaction): ")
    if not withdrawLimit.isdigit():
        print("-----")
        print("Withdrawal Limit must be a number.")
        continue
    # ACCOUNT 테이블에서 선택한 계좌에 해당하는 튜플 찾아서 WithdrawLimit을 변경함.
    sql = "UPDATE ACCOUNT SET WithdrawLimit = %s WHERE AccNum = %s"
    cursor.execute(sql, (withdrawLimit, accnum))
    print("-----")
    print("Withdrawal Limit updated.")
```

선택한 계좌의 출금 한도를 변경하는 옵션이다. 입력받은 출금 한도는 숫자여야 한다. “UPDATE ACCOUNT SET WithdrawLimit = %s WHERE AccNum = %s”라는 sql문을 통해 ACCOUNT 테이블에서 선택한 계좌에 해당하는 튜플을 찾아서 그 튜플의 WithdrawLimit(출금 한도)를 변경한다.

3. Login as User

2. Select Account

5. Update Account Info

4. Update Account Type

```
elif update_account == '4':
    # AccountType 입력 받음. 'checking' 혹은 'savings'가 아니면 continue.
    accountType = input("Account Type (Type 'checking' or 'savings'): ")
    if not (accountType == "checking" or accountType == "savings"):
        print("-----")
        print("Account Type must be either 'checking' or 'savings'.")
        continue
    # ACCOUNT 테이블에서 선택한 계좌에 해당하는 튜플 찾아서 AccountType을 변경함.
    sql = "UPDATE ACCOUNT SET AccountType = %s WHERE AccNum = %s"
    cursor.execute(sql, (accountType, accnum))
    print("-----")
    print("Account Type updated.")
```

선택한 계좌의 계좌 종류를 변경하는 옵션이다. 입력받은 계좌 종류는 ‘checking’ 혹은 ‘savings’ 둘 중 하나여야 한다. “UPDATE ACCOUNT SET AccountType = %s WHERE AccNum = %s”라는 sql문을 통해 ACCOUNT 테이블에서 선택한 계좌에 해당하는 튜플을 찾아서 그 튜플의 AccountType(계좌 종류)를 변경한다.

3. Login as User

2. Select Account

5. Update Account Info

5. Exit

```
elif update_account == '5':  
    connection.commit()      # mysql에 반영하고 while문 빠져나옴.  
    break
```

commit을 통해 변경사항을 mysql에 반영하고 while문을 빠져나오는 옵션이다.

3. Login as User

2. Select Account

6. Delete Account

```
elif trans == "6":  
    print("-----")  
    # ACCOUNT 테이블에서 선택한 계좌에 해당하는 튜플 찾아서 삭제.  
    sql = "DELETE FROM ACCOUNT WHERE AccNum = %s"  
    cursor.execute(sql, accnum)  
    # USER 테이블에서 로그인한 사용자에 해당하는 튜플 찾아서 NumOfAccounts를 1만큼 감소시킴.  
    sql = "UPDATE USER SET NumOfAccounts = NumOfAccounts - 1 WHERE Ssn = %s"  
    cursor.execute(sql, ssn)  
    print("Account deleted.")  
    connection.commit()  
    break
```

선택한 계좌를 삭제할 수 있는 옵션이다. “DELETE FROM ACCOUNT WHERE AccNum = %s”라는 sql문을 통해 ACCOUNT 테이블에서 선택한 계좌에 해당하는 튜플을 찾은 다음 삭제한다. 사용자의 계좌 수가 줄어든 것이므로 “UPDATE USER SET NumOfAccounts = NumOfAccounts - 1 WHERE Ssn = %s”라는 sql문을 통해 USER 테이블에서 사용자에 해당하는 튜플을 찾아서 NumOfAccounts를 1만큼 감소시킨다. 이후에 commit을 통해 mysql에 반영하고, 계좌가 삭제되었으므로 break를 통해 while문을 빠져나온다.

3. Login as User

2. Select Account

7. Exit

```
elif trans == "7":  
    break
```

작업을 마쳤으면 while문을 빠져나올 수 있는 옵션이다.

3. Login as User

3. Update User Info

```
elif login_user == '3':
    while True:
        print("-----")
        print("1. Update Phone Number")
        print("2. Update Address")
        print("3. Update Name")
        print("4. Exit")
        update_user = input("Choose instruction: ")
```

로그인한 사용자의 정보를 변경할 수 있는 옵션이다. 전화번호, 주소, 이름을 변경할 수 있다.

3. Login as User

3. Update User Info

1. Update Phone Number

```
if update_user == "1":
    print("-----")
    phone = input("Enter phone number: ")           # Phone 입력 받음.
    if not (len(phone) == 11 and phone.isdigit()):   # 11자리 숫자가 아니면 continue.
        print("-----")
        print("Phone number must be a 11 digit number.")
        continue
    # USER 테이블에서 로그인한 사용자에 해당하는 튜플을 찾아서 Phone을 변경함.
    sql = "UPDATE USER SET Phone = %s WHERE Ssn = %s"
    cursor.execute(sql, (phone, ssn))
    print("-----")
    print("Phone number updated.")
```

로그인한 사용자의 전화번호를 바꾸는 옵션이다. 전화번호는 11자리 숫자여야 한다. “UPDATE USER SET Phone = %s WHERE Ssn = %s”라는 sql문을 통해 USER 테이블에서 로그인한 사용자에 해당하는 튜플을 찾아서 Phone값을 입력받은 전화번호로 변경한다.

3. Login as User

3. Update User Info

2. Update Address

```
elif update_user == "2":
    print("-----")
    address = input("Enter address: ")               # Address 입력 받음.
    if len(address) > 30:                          # 30자리 초과하면 continue.
        print("-----")
        print("Address must be within 30 characters.")
        continue
    # USER 테이블에서 로그인한 사용자에 해당하는 튜플을 찾아서 Address를 변경함.
    sql = "UPDATE USER SET Address = %s WHERE Ssn = %s"
    cursor.execute(sql, (address, ssn))
    print("-----")
    print("Address updated.")
```

로그인한 사용자의 주소를 바꾸는 옵션이다. 주소는 최대 30자리여야 한다. “UPDATE USER SET Address = %s WHERE Ssn = %s”라는 sql문을 통해 USER 테이블에서 로그인한 사용자에 해당하는 튜플을 찾아서 Address 값을 입력받은 주소로 변경한다.

3. Login as User

3. Update User Info

3. Update Name

```
elif update_user == "3":  
    print("-----")  
    name = input("Enter name: ") # Name 입력 받음.  
    if len(name) > 20: # 20자리 초과하면 continue.  
        print("-----")  
        print("Name must be within 20 characters.")  
        continue  
    # USER 테이블에서 로그인한 사용자에 해당하는 튜플을 찾아서 Name을 변경함.  
    sql = "UPDATE USER SET Name = %s WHERE Ssn = %s"  
    cursor.execute(sql, (name, ssn))  
    print("-----")  
    print("Name updated.")
```

로그인한 사용자의 이름을 바꾸는 옵션이다. 이름은 최대 20자리여야 한다. “UPDATE USER SET Name = %s WHERE Ssn = %s”라는 sql문을 통해 USER 테이블에서 로그인한 사용자에 해당하는 튜플을 찾아서 Name값을 입력받은 이름으로 변경한다.

3. Login as User

3. Update User Info

4. Exit

```
elif update_user == "4":  
    connection.commit() # mysql에 반영하고 while문 빠져나옴.  
    break
```

작업을 마친 후 commit을 통해 mysql에 반영하고 while문을 빠져나오는 옵션이다.

3. Login as User

4. View User Info

```
elif login_user == '4':  
    print("-----")  
    # USER 테이블에서 로그인한 사용자에 해당하는 튜플을 찾아서 이름, 주민번호, 계좌 수, 전화번호, 주소를 가져옴.  
    sql = "SELECT Name, Ssn, NumOfAccounts, Phone, Address FROM USER WHERE Ssn = %s"  
    cursor.execute(sql, ssn)  
    print("{:<25} {:<15} {:<20} {:<13} {:<35}".format("Name", "Ssn", "Number of Accounts",  
                                                       "Phone Number", "Address"))  
    info = cursor.fetchall()[0]  
    print("{:<25} {:<15} {:<20} {:<13} {:<35}".format(info[0], info[1], info[2], info[3], info[4]))
```

로그인한 사용자의 정보를 볼 수 있는 옵션이다. “SELECT Name, Ssn, NumOfAccounts, Phone, Address FROM USER WHERE Ssn = %s”라는 sql문을 통해 USER 테이블에서 로그인한 사용자에 해당하는 튜플을 찾아서 Name(이름), Ssn(주민번호), NumOfAccounts(계좌 수), Phone(전화번호), Address(주소)를 가져온다. format 함수를 이용해 정보를 한 줄에 모두 출력한다.

3. Login as User

5. Delete User

```
elif login_user == '5':  
    print("-----")  
    # USER 테이블에서 로그인한 사용자에 해당하는 튜플을 찾아서 삭제함.  
    sql = "DELETE FROM USER WHERE Ssn = %s"  
    cursor.execute(sql, ssn)  
    print("User deleted.")  
    connection.commit()  
    break
```

로그인한 사용자 자신을 삭제할 수 있는 옵션이다. “DELETE FROM USER WHERE Ssn = %s”라는 sql문을 통해 USER 테이블에서 로그인한 사용자에 해당하는 튜플을 찾아서 삭제한다. 삭제가 완료되면 commit을 통해 mysql에 반영하고 break를 해서 while문을 빠져나온다.

3. Login as User

6. Exit

```
elif login_user == '6':  
    break
```

작업을 마친 후 while문을 빠져나올 수 있는 옵션이다.

4. Login as Admin

```
elif start == '4':
    print("-----")
    adssn = input("Enter Ssn: ")          # 관리자의 AdminSsn을 입력으로 받아서 adssn에 저장.
    # ADMIN 테이블에서 adssn과 일치하는 AdminSsn을 갖는 튜플이 있다면 그 튜플의 AdminSsn 반환.
    sql = "SELECT AdminSsn FROM ADMIN WHERE AdminSsn = %s"
    cursor.execute(sql, adssn)
    select = cursor.fetchall()
    if not select:                      # AdminSsn 일치하는 튜플 없으면 등록된 관리자가 아니므로 continue.
        print("-----")
        print("You are not a registered admin.")
        continue
    # ADMIN 테이블에서 로그인한 관리자에 해당하는 튜플에서 Name을 가져옴.
    sql = "SELECT Name FROM ADMIN WHERE AdminSsn = %s"
    cursor.execute(sql, adssn)
    select = cursor.fetchall()
    print("-----")
    print(f"Welcome, {select[0][0]}")      # Name을 이용해서 관리자를 맞이하는 문구 출력.
    while True:
        print("-----")
        print("1. View Managing Users")
        print("2. Update Admin Info")
        print("3. View Admin Info")
        print("4. Delete Admin")
        print("5. Exit")
        login_admin = input("Choose instruction: ")
```

관리자로 로그인하는 옵션이다. ADMIN 테이블에 존재하는 관리자인지 확인하기 위해 관리자의 AdminSsn을 입력으로 받는다. “SELECT AdminSsn FROM ADMIN WHERE AdminSsn = %s”라는 sql문을 통해 ADMIN 테이블에서 adssn과 일치하는 AdminSsn을 갖는 튜플이 있으면 그 튜플의 AdminSsn을 반환한다. 일치하는 튜플이 없다면 등록된 관리자가 아니라는 뜻이므로 직전 메뉴로 돌아간다. 등록된 관리자라면 “SELECT Name FROM ADMIN WHERE AdminSsn = %s”라는 sql문을 통해 ADMIN 테이블에서 로그인한 관리자에 해당하는 튜플을 찾아 Name을 가져온다. 이 값을 이용해 관리자를 맞이하는 문구를 출력한 후 메뉴를 출력한다.

4. Login as Admin

1. View Managing Users

```
if login_admin == '1':
    while True:
        print("-----")
        # USER 테이블에서 ASsn이 관리자의 AdminSsn과 일치하는 튜플들을 찾아서 이름, 주민번호, 계좌 수, 전화번호, 주소 가져옴.
        sql = "SELECT Name, Ssn, NumOfAccounts, Phone, Address FROM USER WHERE ASsn = %s"
        cursor.execute(sql, adssn)

        select = cursor.fetchall()
        print("{:<25} {:<15} {:<20} {:<13} {:<35}".format("Name", "Ssn", "Number of Accounts",
                                                       "Phone Number", "Address"))

        for row in select:
            print("{:<25} {:<15} {:<20} {:<13} {:<35}".format(row[0], row[1], row[2], row[3],
                                                       row[4]))
        print("-----")
        print("1. Select User")
        print("2. Delete User")
        print("3. Exit")
        select_user = input("Choose instruction: ")
```

로그인한 관리자가 관리하는 사용자들을 볼 수 있는 옵션이다. “SELECT Name, Ssn, NumOfAccounts, Phone, Address FROM USER WHERE ASsn = %s”라는 sql문을 통해 USER 테이블에서 ASsn이 관리자의 AdminSsn과 일치하는 튜플들을 찾아서 Name(이름), Ssn(주민번호), NumOfAccounts(계좌 수), Phone(전화번호), Address(주소)를 가져온다. 가져온 튜플들을 format 함수로 정리해서 한 줄씩 출력한다. 출력이 완료되면 마지막으로 메뉴를 출력한다.

4. Login as Admin

1. View Managing Users

1. Select User

```
if select_user == '1':
    print("-----")
    user_ssn = input("Enter User Ssn to select user: ")      # 관리하는 사용자의 Ssn을 입력 받음.
    # 입력받은 사용자의 Ssn을 가지는 튜플이 USER 테이블에 존재하는지 확인.
    sql = "SELECT EXISTS (SELECT * FROM USER WHERE Ssn = %s)"
    cursor.execute(sql, user_ssn)
    if not cursor.fetchall()[0][0]:                         # 존재하지 않으면 continue.
        print("-----")
        print("User does not exist.")
        continue
    while True:
        print("-----")
        print("1. View Accounts")
        print("2. View Transactions")
        print("3. Set Account Status (Dormant or Active)")
        print("4. Delete Account")
        print("5. Exit")
        admin_user = input("Choose instruction: ")
```

관리자가 관리하는 사용자들 중 한 명을 고를 수 있는 옵션이다. 사용자의 Ssn을 입력으로 받은 후 “SELECT EXISTS (SELECT * FROM USER WHERE Ssn = %s)”라는 sql문을 통해 입력받은 사용자의 Ssn을 가지는 튜플이 USER 테이블에 존재하는지 확인한다. 존재하지 않는다면 더 이상 진행하지 않고 직전 메뉴로 돌아간다.

4. Login as Admin

1. View Managing Users

1. Select User

1. View Accounts

관리자가 선택한 사용자의 계좌들을 볼 수 있는 옵션이다. “SELECT AccNum, DepositLimit, WithdrawLimit, Dormant, AccountType FROM ACCOUNT WHERE AccSsn = %s”라는 sql문을 통해 ACCOUNT 테이블에서 선택한 사용자의 Ssn과 튜플의 AccSsn이 일치하는 튜플들을 골라서 AccNum(계좌번호), DepositLimit(입금한도), WithdrawLimit(출금한도), Dormant(휴면 여부), AccountType(계좌 종류)를 가져온다. format 함수를 통해 각 row를 출력하는데 이때 Dormant(휴면 여부)는 값이 0이면 ‘Active’, 10이면 ‘Dormant’를 출력한다.

4. Login as Admin

1. View Managing Users

1. Select User

2. View Transactions

관리자가 선택한 사용자의 거래들을 볼 수 있는 옵션이다. “SELECT TAccNum, Amount, TransType, Date FROM TRANSACTION WHERE TSsn = %s”라는 sql문을 통해 TRANSACTION 테이블에서 선택한 사용자의 Ssn과 튜플의 TSsn이 일치하는 튜플들을 골라서 TAccNum(계좌번호), Amount(금액), TransType(거래 종류), Date(거래 일시)를 가져온다. format 함수를 이용해 각 row를 출력한다.

4. Login as Admin

1. View Managing Users

1. Select User

3. Set Account Status (Dormant or Active)

```
elif admin_user == '3':
    print("-----")
    # 휴면 여부 설정하기 전 사용자가 가지고 있는 계좌 목록을 계좌번호와 최근 거래일시와 함께 출력함.

    # ACCOUNT 테이블에서 선택한 사용자에 해당하는 튜플들을 골라서 계좌번호를 가져옴.
    sql = "SELECT AccNum FROM ACCOUNT WHERE AccSsn = %s"
    cursor.execute(sql, user_ssn)
    accounts = cursor.fetchall()
    # 각 계좌의 계좌번호와 마지막 거래 시간을 출력.
    print("{:<20} {:<20}".format("Account Number", "Last Transaction"))
    for account in accounts:
        # TRANSACTION 테이블에서 각 계좌의 계좌번호와 거래일시를 가져옴. 거래는 TNum이 클수록
        # 나중에 이루어진 것으로 DESC과 LIMIT 1을 통해 TNum이 가장 큰 하나만 가져옴.
        sql = "SELECT TAccNum, Date FROM TRANSACTION WHERE TAccNum = %s " \
              "ORDER BY TNum DESC LIMIT 1"
        cursor.execute(sql, account[0])
        select = cursor.fetchall()
        if select:
            row = select[0]
            print("{:<20} {:<20}".format(row[0], str(row[1])))
        else:    # 거래가 아직 안 이루어졌다면 No Transaction 출력.
            print("{:<20} {:<20}".format(account[0], "No Transaction"))
```

관리자가 선택한 사용자의 계좌 중 하나를 골라서 휴면 여부를 설정할 수 있는 옵션이다. 휴면 여부를 설정하기 전에 사용자가 가지고 있는 계좌 목록을 계좌번호와 최근 거래일시와 함께 먼저 출력한다. “SELECT AccNum FROM ACCOUNT WHERE AccSsn = %s”라는 sql문을 통해 ACCOUNT 테이블에서 선택한 사용자에 해당하는 튜플들을 골라서 계좌번호를 가져온다. 그리고 for문을 돌면서 “SELECT TAccNum, Date FROM TRANSACTION WHERE TAccNum = %s ORDER BY TNum DESC LIMIT 1”라는 sql문을 통해 TRANSACTION 테이블에서 각 계좌의 계좌번호와 거래일시를 가져온다. 이때 ORDER BY TNum DESC LIMIT 1는 TNum이 큰 순서대로 나열한 뒤 1개만 고르라는 의미다. 거래는 TNum이 클수록 나중에 이루어진 것으로 TNum이 가장 큰 하나의 튜플만 골라서 계좌번호와 마지막 거래일시를 가져온다. 가져온 계좌번호와 마지막 거래일시를 차례로 출력한다. 거래가 아직 안 이루어졌다면 거래일시 대신 No Transaction을 출력한다.

```

print("-----")
acc_num = input("Enter Account Number: ")           # 계좌번호 입력 받음.
if not (len(acc_num) == 13 and acc_num.isdigit()):
    print("-----")
    print("Account Number must be a 13 digit number.")
    continue
# ACCOUNT 테이블에서 관리자가 선택한 사용자의 계좌들 중에서 입력받은 계좌가 존재하는지 확인.
sql = "SELECT EXISTS (SELECT * FROM ACCOUNT WHERE AccSsn = %s AND "
      "AccNum = %s)"
cursor.execute(sql, (user_ssn, acc_num))
if not cursor.fetchall()[0][0]:                      # 존재하지 않으면 continue.
    print("-----")
    print("Account does not exist.")
    continue
status = input("Enter status (0 for Active, 1 for Dormant): ")
if not (status == '0' or status == '1'):
    print("-----")
    print("Type only 0 or 1.")
    continue

# ACCOUNT 테이블에서 계좌를 찾아 계좌의 휴면 여부를 0 또는 1의 int값으로 설정.
sql = "UPDATE ACCOUNT SET Dormant = %s WHERE AccNum = %s"
cursor.execute(sql, (int(status), acc_num))
print("-----")
print("Account Status Updated.")
connection.commit()

```

계좌 목록이 출력되었으면 이제 관리자로부터 계좌번호를 입력 받는다. 입력받은 계좌번호가 13자리 숫자가 아니면 continue한다. 또 “SELECT EXISTS (SELECT * FROM ACCOUNT WHERE AccSsn = %s AND AccNum = %s)”라는 sql문을 통해 ACCOUNT 테이블에서 관리자가 선택한 사용자의 계좌들 중에서 입력받은 계좌가 존재하는지 확인하고, 만약 존재하지 않는다면 continue한다. 마지막으로 관리자로부터 휴면 여부를 0 또는 1로 입력 받는다. 0이면 Active이고, 1이면 Dormant이다. “UPDATE ACCOUNT SET Dormant = %s WHERE AccNum = %s”라는 sql문을 통해 해당 계좌의 Dormant(휴면 여부)를 0 또는 1의 int형 값으로 설정한다. 이후 commit하여 mysql에 반영한다.

4. Login as Admin

1. View Managing Users

1. Select User

4. Delete Account

```
elif admin_user == '4':
    print("-----")
    acc_num = input("Enter Account Number: ")           # 계좌번호 입력 받음.
    if not (len(acc_num) == 13 and acc_num.isdigit()):
        print("-----")
        print("Account Number must be a 13 digit number.")
        continue
    # ACCOUNT 테이블에서 관리자가 선택한 사용자의 계좌들 중에서 입력받은 계좌가 존재하는지 확인.
    sql = "SELECT EXISTS (SELECT * FROM ACCOUNT WHERE AccSsn = %s AND "
    sql += "AccNum = %s)"
    cursor.execute(sql, (user_ssn, acc_num))
    if not cursor.fetchall()[0][0]:
        print("-----")
        print("Account does not exist.")
        continue

    # ACCOUNT 테이블 내에서 해당 계좌를 삭제.
    sql = "DELETE FROM ACCOUNT WHERE AccNum = %s"
    cursor.execute(sql, acc_num)
    # USER 테이블에서 관리자가 선택한 사용자를 찾아 계좌 수를 하나 줄임.
    sql = "UPDATE USER SET NumOfAccounts = NumOfAccounts - 1 WHERE Ssn = %s"
    cursor.execute(sql, user_ssn)
    print("-----")
    print("Account Deleted.")
    connection.commit()
```

관리자가 선택한 사용자의 계좌 중 하나를 삭제할 수 있는 옵션이다. 관리자로부터 계좌번호를 입력 받은 후 만약 13자리 숫자가 아니면 continue한다. “SELECT EXISTS (SELECT * FROM ACCOUNT WHERE AccSsn = %s AND AccNum = %s)”라는 sql문을 통해 ACCOUNT 테이블에서 관리자가 선택한 사용자의 계좌들 중에서 입력받은 계좌가 존재하는지 확인하고 존재하지 않으면 continue한다. 존재한다면 “DELETE FROM ACCOUNT WHERE AccNum = %s”라는 sql문을 통해 ACCOUNT 테이블 내에서 해당 계좌를 찾아 삭제한다. 마지막으로 “UPDATE USER SET NumOfAccounts = NumOfAccounts - 1 WHERE Ssn = %s”라는 sql문을 통해 USER 테이블에서 관리자가 선택한 사용자를 찾아 계좌 수를 하나 줄인다. 이후 commit을 통해 mysql에 반영한다.

4. Login as Admin

1. View Managing Users

1. Select User

5. Exit

```
elif admin_user == '5':
    break
```

while문을 빠져나오는 옵션이다.

4. Login as Admin

1. View Managing Users

2. Delete User

```
elif select_user == '2':  
    print("-----")  
    user_ssn = input("Enter User Ssn to delete user: ")           # 삭제할 사용자 Ssn 입력받음.  
    # USER 테이블 내에 입력받은 Ssn을 가지는 사용자가 존재하는지 확인.  
    sql = "SELECT EXISTS (SELECT * FROM USER WHERE Ssn = %s)"  
    cursor.execute(sql, user_ssn)  
    if not cursor.fetchall()[0][0]:  
        print("-----")  
        print("User does not exist.")  
        continue  
  
    # USER 테이블에서 해당 사용자를 삭제.  
    sql = "DELETE FROM USER WHERE Ssn = %s"  
    cursor.execute(sql, user_ssn)  
    print("-----")  
    print("User Deleted.")  
    connection.commit()
```

관리자가 사용자 한 명을 삭제할 수 있는 옵션이다. 삭제할 사용자의 Ssn을 입력으로 받은 다음 “SELECT EXISTS (SELECT * FROM USER WHERE Ssn = %s)”라는 sql문을 통해 USER 테이블 내에 입력받은 Ssn을 가지는 사용자가 존재하는지 확인한다. 존재하지 않으면 continue하고 존재하면 “DELETE FROM USER WHERE Ssn = %s”라는 sql문을 통해 USER 테이블에서 해당 사용자를 삭제한다. 이후 commit하여 mysql에 반영한다.

4. Login as Admin

1. View Managing Users

3. Exit

```
elif select_user == '3':  
    break
```

while문을 빠져나오는 옵션이다.

4. Login as Admin

2. Update Admin Info

```
elif login_admin == '2':  
    while True:  
        print("-----")  
        print("1. Update Phone Number")  
        print("2. Update Address")  
        print("3. Update Name")  
        print("4. Exit")  
        update_admin = input("Choose instruction: ")
```

관리자의 전화번호, 주소, 이름을 변경할 수 있는 옵션이다.

```

if update_admin == "1":
    print("-----")
    phone = input("Enter phone number: ") # 전화번호를 입력받음.
    if not (len(phone) == 11 and phone.isdigit()): # 전화번호는 11자리 숫자여야함.
        print("-----")
        print("Phone number must be a 11 digit number.")
        continue
    # ADMIN 테이블에서 해당 관리자 찾아서 Phone(전화번호)를 변경함.
    sql = "UPDATE ADMIN SET Phone = %s WHERE AdminSsn = %s"
    cursor.execute(sql, (phone, adssn))
    print("-----")
    print("Phone number updated.")

elif update_admin == "2":
    print("-----")
    address = input("Enter address: ") # 주소를 입력받음.
    if len(address) > 30: # 주소는 30자리를 초과하면 안됨.
        print("-----")
        print("Address must be within 30 characters.")
        continue
    # ADMIN 테이블에서 해당 관리자 찾아서 Address(주소)를 변경함.
    sql = "UPDATE ADMIN SET Address = %s WHERE AdminSsn = %s"
    cursor.execute(sql, (address, adssn))
    print("-----")
    print("Address updated.")

elif update_admin == "3":
    print("-----")
    name = input("Enter name: ") # 이름을 입력받음.
    if len(name) > 20: # 이름은 20자리를 초과하면 안됨.
        print("-----")
        print("Name must be within 20 characters.")
        continue
    # ADMIN 테이블에서 해당 관리자 찾아서 Name(이름)을 변경함.
    sql = "UPDATE ADMIN SET Name = %s WHERE AdminSsn = %s"
    cursor.execute(sql, (name, adssn))
    print("-----")
    print("Name updated.")

elif update_admin == "4":
    connection.commit()
    break

```

3. Login as User — 3. Update User Info와 똑같은 방식으로 정보를 업데이트하므로 설명은 생략한다.

4. Login as Admin

3. View Admin Info

```
elif login_admin == '3':
    print("-----")
    # ADMIN 테이블에서 해당 관리자 찾아서 이름, 사원번호, 전화번호, 주소 가져옴.
    sql = "SELECT Name, AdminSsn, Phone, Address FROM ADMIN WHERE AdminSsn = %s"
    cursor.execute(sql, adssn)
    print("{:<25} {:<15} {:<13} {:<35}".format("Name", "Ssn", "Phone Number", "Address"))
    info = cursor.fetchall()[0]
    print("{:<25} {:<15} {:<13} {:<35}".format(info[0], info[1], info[2], info[3]))
```

로그인한 관리자의 정보를 볼 수 있는 옵션이다. “SELECT Name, AdminSsn, Phone, Address FROM ADMIN WHERE AdminSsn = %s”라는 sql문을 통해 Name(이름), AdminSsn(사원번호), Phone(전화번호), Address(주소)를 가져온 후 이를 format 함수를 이용해 출력한다.

4. Login as Admin

4. Delete Admin

```
elif login_admin == '4':
    print("-----")
    sql = "SELECT COUNT(*) FROM ADMIN"          # ADMIN 테이블에 있는 튜플의 개수를 반환.
    cursor.execute(sql)
    count = cursor.fetchall()[0][0]
    if count == 1:                                # 한 개밖에 없다면 continue.
        print("There should be at least 1 Admin.")
        continue

    while True:        # 관리자가 삭제되면 그가 관리하던 모든 튜플들의 관리자가 바뀌어야 하므로 새로운 관리자를 찾아서 지정.
        # ADMIN 테이블 내에서 무작위로 관리자를 뽑아 AdminSsn을 가져옴.
        sql = "SELECT AdminSsn FROM ADMIN ORDER BY rand() LIMIT 1"
        cursor.execute(sql)
        new_admin = cursor.fetchall()[0][0]
        if adssn != new_admin:
            break

    # USER, ACCOUNT, TRANSACTION 테이블에서 참조하는 관리자가 삭제되는 튜플들의 관리자를 변경해줌.
    sql = "UPDATE USER SET ASsn = %s WHERE ASsn = %s"
    cursor.execute(sql, (new_admin, adssn))
    sql = "UPDATE ACCOUNT SET AdSsn = %s WHERE AdSsn = %s"
    cursor.execute(sql, (new_admin, adssn))
    sql = "UPDATE TRANSACTION SET TAdSsn = %s WHERE TAdSsn = %s"
    cursor.execute(sql, (new_admin, adssn))
    # ADMIN 테이블에서 해당 관리자를 찾아 삭제.
    sql = "DELETE FROM ADMIN WHERE AdminSsn = %s"
    cursor.execute(sql, adssn)
    print("Admin Deleted.")
    connection.commit()
    break
```

로그인한 관리자를 삭제할 수 있는 옵션이다. 우선 “SELECT COUNT(*) FROM ADMIN”이라는 sql문을 통해 ADMIN 테이블에 있는 튜플의 개수를 COUNT를 통해 반환한다. 관리자는 최소한 한 명 있어야 하므로 만약 관리자가 한 명밖에 없다면 삭제하지 않고 continue한다. 관리자가 삭제되면 그가 관리하던 모든 튜플들의 관리자가 바뀌어야 하므로 새로운 관리자를 찾아서 지정한다. 이를 위해 while문을 돌면서 “SELECT AdminSsn FROM ADMIN ORDER BY rand() LIMIT 1”라는 sql문을 통해 ADMIN 테이블 내에서 무작위로 관리자를 뽑아 AdminSsn 을 가져온다. 무작위로 뽑은 관리자가 삭제될 관리자와 다른 사람이면 while문을 빠져나온다. 이후 “UPDATE <테이블> SET <Foreign Key> WHERE <Foreign Key> = %s”라는 sql문을 통해 USER, ACCOUNT, TRANSACTION 테이블에서 참조하는 관리자가 삭제되는 튜플들의 관리자를 변경해준다. 마지막으로 “DELETE FROM ADMIN WHERE AdminSsn = %s”라는 sql문을 통해 ADMIN 테이블에서 해당 관리자를 찾아 삭제한다. commit하고 mysql에 반영한다.

4. Login as Admin

5. Exit

```
elif login_admin == '5':  
    break
```

while문을 빠져나오는 옵션이다.

5. Exit

```
elif start == '5':  
    break
```

가장 첫 while문을 빠져나올 수 있는 옵션이다.

```
        connection.commit()  
finally:  
    connection.close()
```

while문을 빠져나오면 commit을 호출한 후 finally문을 이용해 무조건 connection을 close한다. 이후 프로그램이 종료된다.

- 수행되는 기능 스크린샷

```
[base] Minui-MacBook-Pro-2:db_p4 minsuk$ python3 main.py
Accessing Bank Database ...
[Enter password:
Database Access Authorized.

-----
1. Register User
2. Register Admin
3. Login as User
4. Login as Admin
5. Exit
Choose an instruction: ]
```

프로그램을 처음 실행하면 비밀번호를 입력하라는 문구가 뜬다. 비밀번호를 입력하면 데이터베이스 접근이 허용되고 선택할 수 있는 메뉴가 출력된다.

1. Register User

```
-----
1. Register User
2. Register Admin
3. Login as User
4. Login as Admin
5. Exit
Choose an instruction: 1
-----
Registering User...
Name: Messi
Ssn (13 digits): 8706241234567
Phone Number (11 digits): 01012348765
Address: Rosario, Argentina
-----
Register complete.
```

Name, Ssn, Phone Number, Address를 입력하면 사용자 등록이 완료된다.

2. Register Admin

```
1. Register User
2. Register Admin
3. Login as User
4. Login as Admin
5. Exit
Choose an instruction: 2
-----
Registering Admin...
Ssn (13 digits): 6301261234567
Name: Mourinho
Address: Lisbon, Portugal
Phone Number (11 digits): 01012345678
-----
Register complete.
```

Ssn, Name, Address, Phone Number를 입력하면 관리자 등록이 완료된다.

3. Login as User

```
1. Register User
2. Register Admin
3. Login as User
4. Login as Admin
5. Exit
Choose an instruction: 3
-----
Enter Ssn: 8706241234567
-----
Welcome, Messi
-----
1. Create Account
2. Select Account
3. Update User Info
4. View User Info
5. Delete User
6. Exit
Choose an instruction: ■
```

사용자의 Ssn을 입력하면 로그인이 완료되고 선택할 수 있는 메뉴가 출력된다.

3. 1. Create Account

```
-----  
1. Create Account  
2. Select Account  
3. Update User Info  
4. View User Info  
5. Delete User  
6. Exit  
Choose an instruction: 1  
-----  
Creating account...  
[Password (4 digits): ]  
Deposit Limit (for a single transaction): 15000000  
Withdrawal Limit (for a single transaction): 15000000  
Account Type (Type 'checking' or 'savings'): checking  
-----  
Account successfully created.  
Your account number: 1690142074062  
-----
```

Password, Deposit Limit, Withdrawal Limit, Account Type을 입력하면 계좌가 생성되고 계좌번호를 출력한다. Password는 getpass함수를 사용해서 노출이 되지 않는다.

3. 2. Select Account

```
-----  
1. Create Account  
2. Select Account  
3. Update User Info  
4. View User Info  
5. Delete User  
6. Exit  
Choose an instruction: 2  
-----  
Account Number   Deposit Limit      Withdrawal Limit    Status      Account Type  
1690142074062   15000000          15000000          Active     checking  
5647272933363   10000000          10000000          Active     checking  
-----  
Enter account number: 1690142074062  
[Enter password: ]  
-----  
1. Deposit  
2. Withdrawal  
3. Check Balance  
4. View Transactions  
5. Update Account Info  
6. Delete Account  
7. Exit  
Choose instruction: [ ]
```

계좌 목록이 출력된다. 계좌 번호와 비밀번호를 입력하면 선택할 수 있는 메뉴가 출력된다.

3. 2. 1. Deposit

```
-----  
1. Deposit  
2. Withdrawal  
3. Check Balance  
4. View Transactions  
5. Update Account Info  
6. Delete Account  
7. Exit  
Choose instruction: 1  
-----  
Enter deposit amount: 5000  
-----
```

금액을 입력하면 입금이 완료된다.

3. 2. 2. Withdrawal

```
-----  
1. Deposit  
2. Withdrawal  
3. Check Balance  
4. View Transactions  
5. Update Account Info  
6. Delete Account  
7. Exit  
Choose instruction: 2  
-----  
Enter withdrawal amount: 3000  
-----
```

금액을 입력하면 출금이 완료된다.

3. 2. 3. Check Balance

```
-----  
1. Deposit  
2. Withdrawal  
3. Check Balance  
4. View Transactions  
5. Update Account Info  
6. Delete Account  
7. Exit  
Choose instruction: 3  
-----  
Balance: 2000  
-----
```

잔액을 출력한다.

3. 2. 4. View Transactions

```
-----  
1. Deposit  
2. Withdrawal  
3. Check Balance  
4. View Transactions  
5. Update Account Info  
6. Delete Account  
7. Exit  
Choose instruction: 4  
-----  
Amount      Transaction Type      Date  
5000        Deposit            2021-12-02 01:46:31  
3000        Withdrawal         2021-12-02 01:46:41  
-----
```

계좌의 거래 목록을 출력한다.

3. 2. 5. Update Account Info

```
-----  
1. Deposit  
2. Withdrawal  
3. Check Balance  
4. View Transactions  
5. Update Account Info  
6. Delete Account  
7. Exit  
Choose instruction: 5  
-----
```

```
1. Update Password  
2. Update Deposit Limit  
3. Update Withdrawal Limit  
4. Update Account Type  
5. Exit  
Choose instruction: 
```

비밀번호, 입금한도, 출금한도, 계좌종류를 변경하는 옵션들을 출력한다.

```
-----  
1. Update Password  
2. Update Deposit Limit  
3. Update Withdrawal Limit  
4. Update Account Type  
5. Exit  
Choose instruction: 1  
Password (4 digits):  
-----  
Password updated.  
-----  
1. Update Password  
2. Update Deposit Limit  
3. Update Withdrawal Limit  
4. Update Account Type  
5. Exit  
Choose instruction: 2  
Deposit Limit (for a single transaction): 50000000  
-----  
Deposit Limit updated.  
-----  
1. Update Password  
2. Update Deposit Limit  
3. Update Withdrawal Limit  
4. Update Account Type  
5. Exit  
Choose instruction: 3  
Withdrawal Limit (for a single transaction): 50000000  
-----  
Withdrawal Limit updated.  
-----  
1. Update Password  
2. Update Deposit Limit  
3. Update Withdrawal Limit  
4. Update Account Type  
5. Exit  
Choose instruction: 4  
Account Type (Type 'checking' or 'savings'): savings  
-----  
Account Type updated.  
-----
```

위와 같이 각각의 info들을 변경할 수 있다.

3. 2. 6. Delete Account

```
-----  
1. Deposit  
2. Withdrawal  
3. Check Balance  
4. View Transactions  
5. Update Account Info  
6. Delete Account  
7. Exit  
Choose instruction: 6  
-----
```

```
Account deleted.  
-----
```

계좌가 삭제된다.

3. 3. Update User Info

```
-----  
1. Create Account  
2. Select Account  
3. Update User Info  
4. View User Info  
5. Delete User  
6. Exit  
Choose an instruction: 3  
-----  
1. Update Phone Number  
2. Update Address  
3. Update Name  
4. Exit  
Choose instruction: ■
```

사용자의 전화번호, 주소, 이름을 변경할 수 있다.

```
-----  
1. Update Phone Number  
2. Update Address  
3. Update Name  
4. Exit  
Choose instruction: 1  
-----  
Enter phone number: 01012345867  
-----  
Phone number updated.  
-----  
1. Update Phone Number  
2. Update Address  
3. Update Name  
4. Exit  
Choose instruction: 2  
-----  
Enter address: Barcelona, Spain  
-----  
Address updated.  
-----  
1. Update Phone Number  
2. Update Address  
3. Update Name  
4. Exit  
Choose instruction: 3  
-----  
Enter name: Lionel Messi  
-----  
Name updated.  
-----
```

위와 같이 각각의 info들을 변경할 수 있다.

3. 4. View User Info

```
-----  
1. Create Account  
2. Select Account  
3. Update User Info  
4. View User Info  
5. Delete User  
6. Exit  
Choose an instruction: 4  
-----  
Name          Ssn      Number of Accounts  Phone Number  Address  
Lionel Messi  8706241234567   1                  01012345867  Barcelona, Spain  
-----
```

사용자 정보가 출력된다.

3. 5. Delete User

```
-----  
1. Create Account  
2. Select Account  
3. Update User Info  
4. View User Info  
5. Delete User  
6. Exit  
Choose an instruction: 5  
-----
```

```
User deleted.  
-----
```

사용자가 삭제된다.

4. Login as Admin

```
-----  
1. Register User  
2. Register Admin  
3. Login as User  
4. Login as Admin  
5. Exit  
Choose an instruction: 4  
-----  
Enter Ssn: 6301261234567  
-----  
Welcome, Mourinho  
-----  
1. View Managing Users  
2. Update Admin Info  
3. View Admin Info  
4. Delete Admin  
5. Exit  
Choose instruction: █
```

관리자의 Ssn을 입력하면 로그인이 완료되고 선택할 수 있는 메뉴가 출력된다.

4. 1. View Managing Users

```
-----  
1. View Managing Users  
2. Update Admin Info  
3. View Admin Info  
4. Delete Admin  
5. Exit  
Choose instruction: 1  
-----  
Name          Ssn      Number of Accounts  Phone Number  Address  
Ronaldo       8502051234567   2                  01087651234  Funchal, Portugal  
-----  
1. Select User  
2. Delete User  
3. Exit  
Choose instruction: █
```

관리하는 사용자들이 출력되고 선택할 수 있는 메뉴가 출력된다.

4. 1. 1. Select User

```
-----  
1. Select User  
2. Delete User  
3. Exit  
Choose instruction: 1  
-----  
Enter User Ssn to select user: 8502051234567  
-----  
1. View Accounts  
2. View Transactions  
3. Set Account Status (Dormant or Active)  
4. Delete Account  
5. Exit  
Choose instruction: ■
```

사용자의 Ssn을 입력하면 사용자에 대해 수행 가능한 옵션들이 있는 메뉴가 출력된다.

4. 1. 1. 1. View Accounts

```
-----  
1. View Accounts  
2. View Transactions  
3. Set Account Status (Dormant or Active)  
4. Delete Account  
5. Exit  
Choose instruction: 1  
-----  
Account Number Deposit Limit Withdrawal Limit Status Account Type  
2692307431611 15000000 15000000 Active savings  
6632407386188 20000000 20000000 Active checking  
-----
```

사용자가 가지고 있는 계좌 목록이 출력된다.

4. 1. 1. 2. View Transactions

```
-----  
1. View Accounts  
2. View Transactions  
3. Set Account Status (Dormant or Active)  
4. Delete Account  
5. Exit  
Choose instruction: 2  
-----  
Account Number Amount Transaction Type Date  
2692307431611 5000 Deposit 2021-12-02 02:04:50  
2692307431611 3000 Deposit 2021-12-02 02:04:52  
2692307431611 2000 Deposit 2021-12-02 02:04:55  
6632407386188 10000 Deposit 2021-12-02 02:05:09  
6632407386188 15000 Deposit 2021-12-02 02:05:12  
6632407386188 3000 Withdrawal 2021-12-02 02:05:16  
-----
```

사용자의 모든 계좌에서 이루어진 거래 목록이 출력된다.

4. 1. 1. 3. Set Account Status (Dormant or Active)

```
-----  
1. View Accounts  
2. View Transactions  
3. Set Account Status (Dormant or Active)  
4. Delete Account  
5. Exit  
Choose instruction: 3  
-----  
Account Number Last Transaction  
2692307431611 2021-12-02 02:04:55  
6632407386188 2021-12-02 02:05:16  
-----  
Enter Account Number: 2692307431611  
Enter status (0 for Active, 1 for Dormant): 1  
-----  
Account Status Updated.  
-----
```

각 계좌의 마지막 거래 일시가 출력되고 계좌번호와 status를 입력하면 휴면 여부가 변경된다.

4. 1. 1. 4. Delete Account

```
-----  
1. View Accounts  
2. View Transactions  
3. Set Account Status (Dormant or Active)  
4. Delete Account  
5. Exit  
Choose instruction: 4  
-----  
Enter Account Number: 6632407386188  
-----  
Account Deleted.  
-----
```

계좌번호를 입력하면 계좌가 삭제된다.

4. 1. 2. Delete User

```
-----  
1. Select User  
2. Delete User  
3. Exit  
Choose instruction: 2  
-----  
Enter User Ssn to delete user: 8502051234567  
-----  
User Deleted.  
-----
```

사용자의 Ssn을 입력하면 사용자가 삭제된다.

4. 2. Update Admin Info

```
-----  
1. View Managing Users  
2. Update Admin Info  
3. View Admin Info  
4. Delete Admin  
5. Exit  
Choose instruction: 2  
-----  
1. Update Phone Number  
2. Update Address  
3. Update Name  
4. Exit  
Choose instruction: ■
```

관리자의 전화번호, 주소, 이름을 변경할 수 있다.

```
-----  
1. Update Phone Number  
2. Update Address  
3. Update Name  
4. Exit  
Choose instruction: 1  
-----  
Enter phone number: 01012345678  
-----  
Phone number updated.  
-----  
1. Update Phone Number  
2. Update Address  
3. Update Name  
4. Exit  
Choose instruction: 2  
-----  
Enter address: Paris, France  
-----  
Address updated.  
-----  
1. Update Phone Number  
2. Update Address  
3. Update Name  
4. Exit  
Choose instruction: 3  
-----  
Enter name: Jose Mourinho  
-----  
Name updated.
```

위와 같이 각각의 info들을 변경할 수 있다.

4. 3. View Admin Info

```
-----  
1. View Managing Users  
2. Update Admin Info  
3. View Admin Info  
4. Delete Admin  
5. Exit  
Choose instruction: 3  
-----  
Name Ssn Phone Number Address  
Jose Mourinho 63012612345678 01012345678 Paris, France
```

관리자의 정보가 출력된다.

4. 4. Delete Admin

```
-----  
1. View Managing Users  
2. Update Admin Info  
3. View Admin Info  
4. Delete Admin  
5. Exit  
Choose instruction: 4  
-----  
Admin Deleted.
```

관리자가 삭제된다.