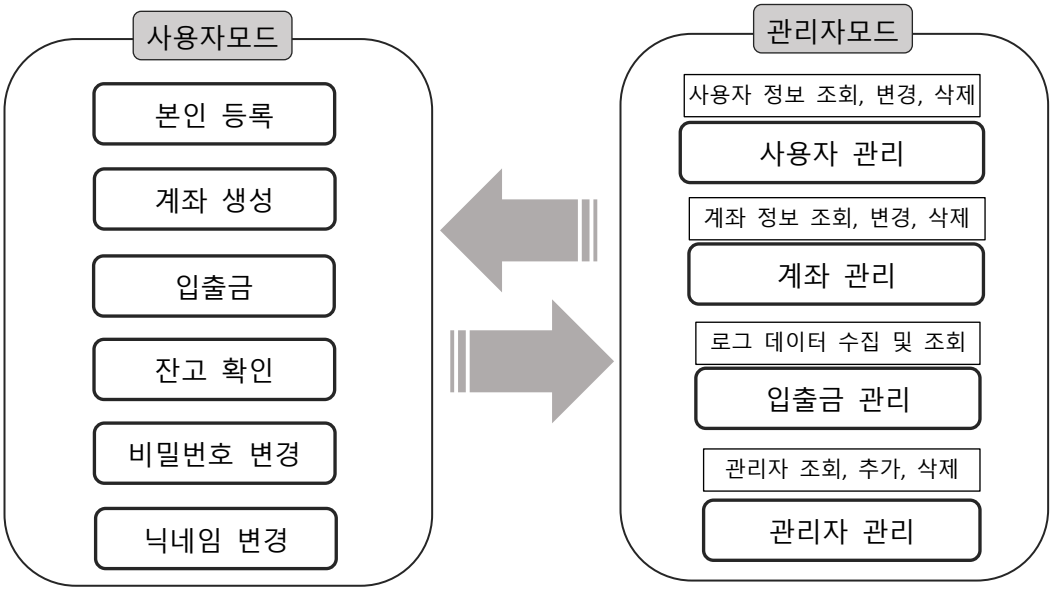
은행 *DB* 프로젝트 리포트

강 응 찬

1. 은행 애플리케이션 다이어그램



2. 프로그램 코드에 대한 설명

애플리케이션은 전체적으로 두 모드(사용자, 관리자)로 사용된다.

A. 사용자 모드

i. 사용자 정보(이름, 성, 주민번호, 생년월일) 등록
ii. 계좌 개설
iii. 입출금 진행
iv. 본인 잔고 확인

B. 관리자 모드

i. 사용자 정보 관리(정보 추출, 삭제, 등)
ii. 계좌 정보 관리(데이터 추출, 삭제, 등)
iii. 입출금 내역 로그 데이터 관리
iv. 관리자 정보 관리(추가, 삭제 등)

A. 사용자 모드 코드

DB연결 코드

```
def connect_db(self):
        try:
            self.DB_connector = mysql.connector.connect(host=self.host,
                                                        database=self.DB,
                                                        user=self.user,
                                                        password=self.pw)
            self.cursor = self.DB_connector.cursor()
```

```
        except mysql.connector.Error as error:
            print("Failed connecting DB {}".format(error))
```

A. 사용자 정보 등록 코드

```
# 사용자를 DB 에 추가하는 함수
    def insert_user_to_db(self):
        try:

            SSN = input("주민번호를 입력하세요: ")
            # 이름 성 순서로 띄어쓰기로 입력하면 이름과 성이 따로 저장된다.
            Name = str(input("이름을 입력하세요: ")).split(' ')
            Lname = Name[0] # 이름 저장
            Fname = Name[1] # 성 저장
            Bdate = input("생년월일을 입력하세요: ")
                # 입력을 위한 sql 문 작성
            sql = "insert into " + self.user_table + \
                  " values({},'{}','{}','{}');".format(SSN, Lname, Fname, Bdate)

            self.cursor.execute(sql)
            self.DB_connector.commit()

            print("사용자 계정이 생성되었습니다.")
        except mysql.connector.Error as error:
            print("Failed inserting user data into User table {}".format(error))
```

B. 계좌 개설 코드

```
# 사용자가 계좌를 생성하는 함수
    def create_account_to_db(self):
        try:
            account = input("계좌번호를 입력하세요: ")
            pw = input("암호를 설정하세요: ")
            balance = input("돈을 넣어주세요: ")
            check_count = 1     # 주민번호 오류 횟수
            while check_count <= 3: # 3 번 이상 틀리면 접근 불가
                Ussn = int(input("주민번호를 입력하세요: "))
                users = self.get_user_list(cursor)
                users = list(users['SSN'].values)
```

```
                    if Ussn in users:
                        sql = "insert into " + self.account + \
                            " values({}, {}, {}, {})".format(
                                account, balance, Ussn, pw)
                        self.cursor.execute(sql)
                        self.DB_connector.commit()
                        print("계좌 생성이 완료되었습니다.")
                        return
                    else:
                        print("해당 주민번호는 존재하지 않습니다. 다시 입력하세요. 남은 횟수
{}번".format(3-check_count))
                        check_count += 1
                print("계좌생성에 실패하였습니다.")
                return
            except mysql.connector.Error as error:
                print("Failed creating account into Account table {}".format(error))
```

c. 입출금 진행 코드

```
    def update_data(self, table):
        try:
            choose = int(input("입금 1, 출금 2: "))
            account = input("계좌번호를 입력하세요: ")
            login_count, i = 3, 1   # 세 번 이상 입력이 틀릴 경우 접속 제한
            while i <= login_count:
                # 비밀번호를 화면상에서 숨기기위해 getpass 사용
                pw = getpass.getpass('비밀번호를 입력하세요:')
                login_query = "Select PassWord, Balance from Account where AccNum =
{}".format(account)
                self.cursor.execute(login_query)
                # select 로 데이터를 불러오는 경우 ([...],[,,,])형식이기에 [0]을 취하면 리스트만 남는다.
                login = cursor.fetchall()[0]
                password = login[0]    # [password, balance]
                if int(password) == int(pw):
                    balance = login[1]
                    break
                elif i == login_count:
                    print("비밀번호 입력횟수를 초과하였습니다.")
                    return
                else:
                    print("비밀번호가 맞지 않습니다.")
```

```
                    print("다시 시도하세요. 시도횟수 {}번 남음".format(login_count - i))
                    i += 1

            if choose == 1:
                cash = input("입금할 금액을 입력하세요: ")
                balance = int(balance) + int(cash)
                # deposit 을 True 로 해 놓으면 로그 데이터에서 입금 여부를 확인한다.
                self.deposit = True
            else:
                cash = input("출금할 금액을 입력하세요: ")
                balance = (int(balance) - int(cash))
                if balance < 0:
                    print("잔고가 없습니다.")
                    return
                self.withdraw = True
            sql = "update " + table + \
                " set Balance = {} where AccNum = {}".format(balance, account)
            Self.cursor.execute(sql)
            # 로그 테이블을 호출하여 입출금 내용을 자동으로 기록
            self.write_log(cursor, account, cash)
            self.DB_connector.commit()
            if self.deposit == True:
                print("입금이 완료되었습니다. 현재 잔고는 {}원 입니다.".format(balance))
                self.deposit = False
            else:
                print("출금이 완료되었습니다. 현재 잔고는 {}입니다.".format(balance))
                self.withdraw = False
            return
        except mysql.connector.Error as error:
            print("Failed updating data into Account table {}".format(error))
            return
```

D. 잔고를 확인할 수 있는 코드

```
def check_balance(self):
        try:
            account = input("계좌번호를 입력하세요: ")
            counter, i = 3, 1
            while i <= counter:
```

```
                    pw = getpass.getpass('비밀번호를 입력하세요:')
                    login_query = "Select PassWord, Balance from Account where AccNum =
{}".format(account)
                    self.cursor.execute(login_query)
                    login = self.cursor.fetchall()
                    if bool(login):
                        login = login[0]
                    else:
                        print("계좌번호가 일치하지 않습니다.")
                        return
                    if int(login[0]) == int(pw):
                        print("현재 잔고는 {}원 입니다.".format(login[1]))
                        self.DB_connector.commit()
                        return
                    else:
                        print("비밀번호가 일치하지 않습니다. 남은 시도횟수 {}번".format(counter-i))
                        i += 1

            except mysql.connector.Error as error:
                print("Failed fetching trained_words from MySQL table {}".format(error))
                return
```

E. 계좌 비밀번호 변경 코드

```
def password_update(self):
            try:
                account = input("계좌번호를 입력하세요: ")
                counter, i = 3, 1
                while i <= counter:
                    pw = getpass.getpass('비밀번호를 입력하세요:')
                    login_query = "Select PassWord, Balance from Account where AccNum =
{}".format(account)
                    self.cursor.execute(login_query)
                    login = self.cursor.fetchall()
                    if bool(login):
                        login = login[0]
                    else:
                        print("계좌번호가 일치하지 않습니다.")
                        return
                    if int(login[0]) == int(pw):
                        new_pw = input("새로운 비밀번호 입력: ")
```

```
                    pw_update = "update Account set PassWord = {} where AccNum =
'{}'".format(new_pw, account)
                    self.cursor.execute(pw_update)
                    self.DB_connector.commit()
                    print("비밀번호 재설정 완료!")
                    return
                else:
                    print("비밀번호가 일치하지 않습니다. 남은 시도횟수 {}번".format(counter-i))
                    i += 1
        except mysql.connector.Error as error:
            print("Failed update password from MySQL table {}".format(error))
            return
```

F. 닉네임 변경 코드

```
def change_nickname(self):
        try:
            ssn = input("주민번호를 입력하세요: ")
            old_nick = input('기존 닉네임을 입력하세요:')
            login_query = "Select NickName from User where SSN = {} and NickName = '{}'".format(ssn,
old_nick)
            self.cursor.execute(login_query)
            login = self.cursor.fetchall()
            if bool(login):
                login = login[0]
            else:
                print("해당 정보가 존재하지 않습니다.")
                return

            new_nick = input("새로운 닉네임 입력: ")
            pw_update = "update User set NickName = '{}' where SSN = {}".format(
                new_nick, ssn)
            self.cursor.execute(pw_update)
            self.DB_connector.commit()
            print("닉네임 재설정 완료!")
            return

        except mysql.connector.Error as error:
            print("Failed update NickName from MySQL table {}".format(error))
            return
```

B. 관리자 모드

A. 사용자 정보 관리

```
# 관리자모드에서 전체 사용자 명단을 보게 하는 함수
    def get_user_list(self, cursor):
        sql_user_cnt = "select count(*) as cnt from User"
        cursor.execute(sql_user_cnt)
        user_count = cursor.fetchall()[0]
        print("\n 전체 유저 인원수: ", user_count[0], '명\n')
        sql_user_account = "select Lname, count(*) as numOfaccount from User, Account where SSN = Ussn
group by Lname"
        cursor.execute(sql_user_account)
        user_accounts = cursor.fetchall()
        print("각 유저당 보유 계좌 개수:")
        for acc_num in user_accounts:
            print(acc_num[0], ': ', acc_num[1])
        print()
        print("전체 사용자 목록:")
        sql_user_list = "Select * from User"
        cursor.execute(sql_user_list)
        user_data = cursor.fetchall()
        column = ['SSN', 'Lname', 'Fname', 'Bdate']
        user_data = pd.DataFrame(user_data, columns=column)
        return user_data

# 사용자를 삭제할 수 있는 함수
    def delete_user(self, cursor):
        ssn = int(input("주민번호 입력: "))
        users = self.get_user_list(cursor)
        users = list(users['SSN'].values)
        if ssn in users:
            # SSN 이 Account table 의 FK 이므로 Account 계정을 먼저 삭제해줘야 한다.
            sql1 = "Delete from Account where Ussn = {}".format(ssn)
            cursor.execute(sql1)
            sql2 = "Delete from User Where SSN = {}".format(ssn)
            cursor.execute(sql2)
            print("사용자 {} 삭제 완료".format(ssn))
            return
        else:
            print("일치하는 사용자 데이터가 존재하지 않습니다.")
```

```
                return
```

B. 계좌정보 관리

```
# 데이터베이스에 저장된 전체 계좌 목록을 보여주는 함수
    def get_account_list(self, cursor):
        print()
        account_summary = "select count(AccNum), sum(Balance) from Account"
        cursor.execute(account_summary)
        account_summary = cursor.fetchall()
        summary_column = ['total_cnt', 'total_sum']
        account_summary = pd.DataFrame(account_summary, columns=summary_column)
        print("전체 계좌 개수: ", account_summary['total_cnt'].values[0])
        print("전체 은행 잔고: ", account_summary['total_sum'].values[0])
        # 예금이 100000 원 이상이면 vip
        balance = 100000
        vip_members = "select Lname, Fname, Balance from User, Account where Ussn = SSN and
Balance >= {} order by Balance".format(
            balance)
        cursor.execute(vip_members)
        vip = cursor.fetchall()
        vip_column = ['Lname', 'Fname', 'Balance']
        vip = pd.DataFrame(vip, columns=vip_column)
        print("\nvip 명단")
        print(vip)
        print()
        print("\n 전체 계좌")
        sql_total_list = "Select * from Account"
        cursor.execute(sql_total_list)
        account_data = cursor.fetchall()
        column = ['AccNum', 'Balance', 'Ussn', 'Password']
        account_data = pd.DataFrame(account_data, columns=column)
        return account_data

# 계좌를 삭제할 수 있는 함수
    def delete_account(self, cursor):
        account = int(input("계좌번호: "))

        accounts = self.get_account_list(cursor)
```

```
        accounts = list(accounts['AccNum'].values)
        if account in accounts:
            sql = "Delete from Account Where AccNum = {}".format(account)
            cursor.execute(sql)
            print("계좌 {} 삭제 완료".format(account))
            return
        else:
            print("일치하는 계좌가 존재하지 않습니다.")
            return
```

C. 입출금 데이터 관리

```
# 로그 데이터를 보여주는 함수
    def get_log_data(self, cursor):
        print("\nSummary of Log:")
        sql1 = "select Name, sum(withdraw), sum(deposit) from Log group by Name"
        cursor.execute(sql1)
        summary = cursor.fetchall()
        column1 = ['이름', '출금액', '입금액']
        summary = pd.DataFrame(summary, columns=column1)
        print(summary)
        print("\nTotal of Log:")
        sql2 = "Select * from Log"
        cursor.execute(sql2)
        log = cursor.fetchall()
        column = ["LogID", "Account", "Name", "Withdraw", "Deposit", "LogDate"]
        log = pd.DataFrame(log, columns=column)
        return log
```

D. 관리자 정보 관리

```
# 관리자를 추가해주는 함수
    def add_admin_member(self, cursor):
        ID = input("아이디 혹은 이메일주소 입력: ")
        PW = input("비밀번호 입력: ")
        sql = "insert into Admin values('{}', {})".format(ID, PW)
        cursor.execute(sql)
        print("관리자 {}님이 추가되었습니다.".format(ID))
```

```
# 관리자 목록을 보여주는 함수
def print_admin_list(self, cursor):
        sql = "Select * from Admin"
        cursor.execute(sql)
        admin_list = cursor.fetchall()
        column = ["아이디", "비밀번호"]
        admin_list = pd.DataFrame(admin_list, columns=column)
        return admin_list

# 관리자 정보를 삭제하는 있는 함수
def delete_admin(self, cursor):
        ID = input("관리자 아이디: ")
        admin = self.print_admin_list(cursor)

        admins = list(admin['아이디'].values)
        if ID in admins:
                sql = "Delete from Admin Where Admin_ID = '{}'".format(ID)
                cursor.execute(sql)
                print("관리자 {} 삭제 완료".format(ID))
                return
        else:
                print("일치하는 관리자 데이터가 존재하지 않습니다.")
                return
```

3. 수행되는 기능 스크린샷

메인 화면

```
Welcom to HanYang Bank.

은행 서비스 메뉴얼
============================================================
사용자 모드
    본인 등록
    계좌 생성
    입출금
    잔고 확인
    비밀번호 변경

관리자 모드
    전체 사용자목록 확인
    전체 계좌목록 확인
    입출금 내역 확인
    사용자 정보 삭제
    계좌 삭제
    관리자 목록
    관리자 추가
    관리자 삭제
============================================================


    0. 서비스 종료
    1. 본인 등록
    2. 계좌 생성
    3. 입출금
    4. 잔고 확인
    5. 계좌 비밀번호 변경
    6. 사용자 닉네임 변경
    7. 관리자 모드
```

A. 사용자 모드

i. 본인 등록

```
        0. 서비스 종료
        1. 본인 등록
        2. 계좌 생성
        3. 입출금
        4. 잔고 확인
        5. 계좌 비밀번호 변경
        6. 사용자 닉네임 변경
        7. 관리자 모드

원하는 서비스를 선택하세요: 1
주민번호를 입력하세요: 1234
이름을 입력하세요: bobby Kim
생년월일을 입력하세요: 1989-11-09
닉네임을 설정하세요: chan
사용자 계정이 생성되었습니다.

        0. 서비스 종료
        1. 본인 등록
        2. 계좌 생성
        3. 입출금
        4. 잔고 확인
        5. 계좌 비밀번호 변경
        6. 사용자 닉네임 변경
        7. 관리자 모드
```

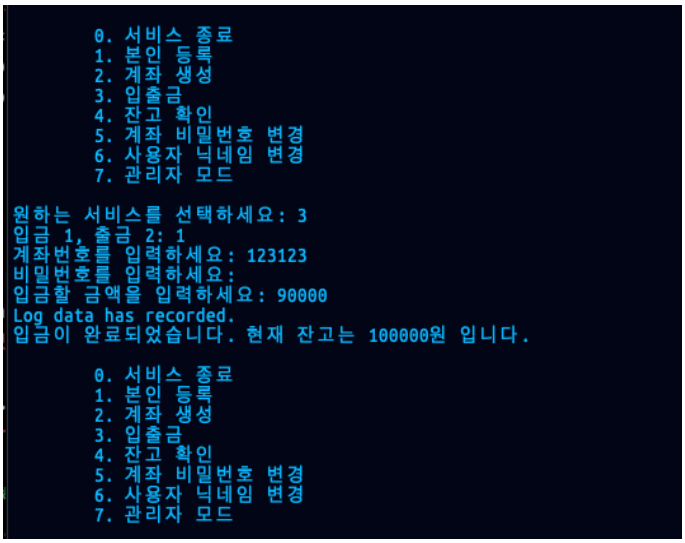
ii. 계좌 생성

```
        0. 서비스 종료
        1. 본인 등록
        2. 계좌 생성
        3. 입출금
        4. 잔고 확인
        5. 계좌 비밀번호 변경
        6. 사용자 닉네임 변경
        7. 관리자 모드

원하는 서비스를 선택하세요: 2
계좌번호를 입력하세요: 123123
비밀번호를 설정하세요: 1234
돈을 넣어주세요: 10000
주민번호를 입력하세요: 1234
계좌 생성이 완료되었습니다.

        0. 서비스 종료
        1. 본인 등록
        2. 계좌 생성
        3. 입출금
        4. 잔고 확인
        5. 계좌 비밀번호 변경
        6. 사용자 닉네임 변경
        7. 관리자 모드
```

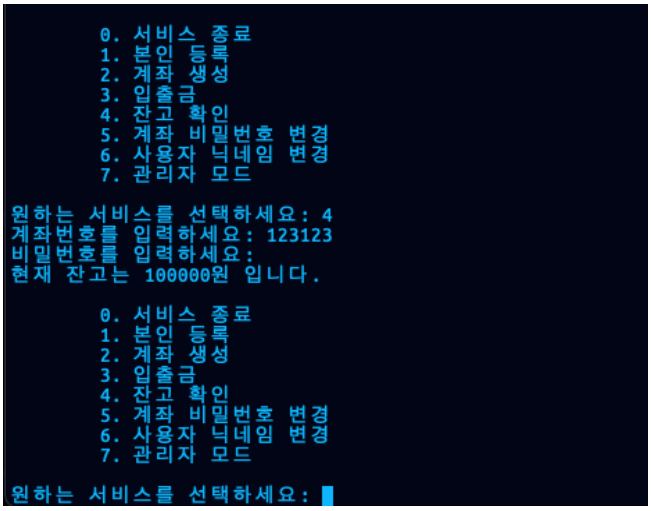
iii. 입출금



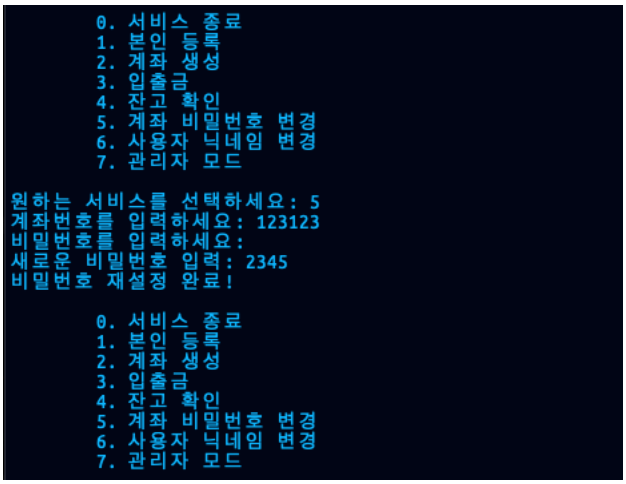
비밀번호를 입력할 때는 보안상 외부에 보이지 않게 설정

비밀번호는 3회까지 재시도 가능

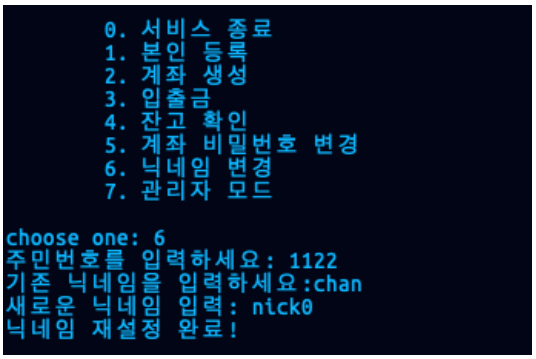
iv. 현재 잔고 확인



v. 계좌 비밀번호 변경



vi. 사용자 닉네임 변경



B. 관리자 모드 진입

관리자 모드는 계정아이디와 비밀번호를 입력해야 접근 가능. 만일 아이디나 비밀번호가 틀릴 경우 3번까지 재시도 허용.

i. 사용자 정보 관리

```
choose one: 1

전체 유저 인원수:  7 명

각 유저당 보유 계좌 개수:
hanyang :  1
Miffy :  1
Henry :  2
beranar :  1
Jack :  2
cosmos :  1

전체 사용자 목록:
      SSN     Lname    Fname       Bdate NickName
0    1122      mola     kang  1991-10-10    nick0
1    7890   hanyang      kim  1939-01-01    nick1
2    8899     Miffy     Kang  2000-02-02    nick2
3   34567     Henry      Kim  1990-10-05    nick3
4  202100   beranar   berber  1995-01-01    nick4
5  456456      Jack    Black  1960-05-05    nick5
6  890890    cosmos      kim  1995-01-19    nick6
```

화면에 보이는 대로 DB에 저장된 전체 유저, 각 유저가 보유하고 있는 계좌 수, 전체 사용자 데이터를 구체적으로 확인 가능

ii. 계좌 정보 관리

```
choose one: 2

전체 계좌 개수:  10
전체 은행 잔고:  693000

vip 명단
    Lname    Fname  Balance
0   Henry      Kim   100000
1    Hong  kildong   110000
2    Jack    Black   140000
3  cosmos      kim   190000


전체 계좌
   AccNum  Balance    Ussn  Password
0    1212     2000    7890      1234
1    1232    90000  202100      1234
2    3454   190000  890890      1234
3    6787     5000   34567      1234
4    8989    11000  456456      1234
5    9090    15000    8899      1234
6   12321   110000  123456      1234
7  222333   100000   34567      1109
8  404040   140000  456456      1234
9  456456    30000  123456      1234
```

DB에 등록된 전체 계좌 개수, 현재 전체 은행 잔고, 잔고가 일정 금액(100000)이상인 유저들을 vip로 지정하고 명단 확보, 구체적인 계좌 상황 확인

iii. 입출금 내역 관리

```
Summary of Log:
              이름     출금액     입금액
0  beranar berber       0    9000
1      cosmos kim  200000  600000
2   eungchan Kang  220000   50000
3     hanyang kim       0    5000
4       Henry Kim   50000       0
5    Hong kildong       0   60000
6      Jack Black   10000  100000
7      Miffy Kang       0    5000

Total of Log:
    LogID  Account            Name  Withdraw  Deposit     LogDate
0       2   222333       Henry Kim     50000        0  2021-11-27
1       3  3334444   eungchan Kang    100000        0  2021-11-27
2       4  5559999   eungchan kang         0    50000  2021-11-27
3       5   505050   eungchan kang     50000        0  2021-11-27
4       6  9876500  beranar berber         0     9000  2021-11-27
5       7   123321      cosmos kim         0   100000  2021-11-28
6       8   123321      cosmos kim         0   500000  2021-11-28
7       9   123321      cosmos kim    200000        0  2021-11-28
8      10   404040      Jack Black     10000        0  2021-12-02
9      11   404040      Jack Black         0   100000  2021-12-02
10     12   102804     hanyang kim         0     5000  2021-12-02
11     13  5559999   eungchan kang     50000        0  2021-12-02
12     14     9090      Miffy Kang         0     5000  2021-12-02
13     15  5559999   eungchan kang     20000        0  2021-12-02
14     16    12321    Hong kildong         0    60000  2021-12-03
```

로그 데이트를 통하여 입출금 내역 확인. 각 유저가 입금과 출금을 모두 얼마 진행했는지를 확인. 입출금에 대한 모든 내역을 유저 이름과 계좌번호, 날짜를 기준으로 정리.

iv. 관리자 목록 확인

```
choose one: 4
                    아이디  비밀번호
0        abcd@naver.com  1234
1  eungchan@hanyang.com  5097
2       hanyang@abc.com  1234

        0. 이전 단계
        1. 사용자 정보
        2. 계좌 정보
        3. 입출금 내역
        4. 관리자 목록
        5. 관리자 추가
        6. 사용자 데이터 삭제
        7. 계좌 삭제
        8. 관리자 삭제

choose one:
```

v. 관리자 추가

```
choose one: 5
아이디 혹은 이메일주소 입력:  kildong@gmail.com
비밀번호 입력: 1234
관리자 kildong@gmail.com님이 추가되었습니다.

        0. 이전 단계
        1. 사용자 정보
        2. 계좌 정보
        3. 입출금 내역
        4. 관리자 목록
        5. 관리자 추가
        6. 사용자 데이터 삭제
        7. 계좌 삭제
        8. 관리자 삭제

choose one:
```

vi. 사용자 데이터 삭제

```
choose one: 6
주민번호 입력: 123456

전체 유저 인원수:  7 명

각 유저당 보유 계좌 개수:
hanyang :  1
Miffy :  1
Henry :  2
Hong :  2
beranar :  1
Jack :  2
cosmos :  1

전체 사용자 목록:
사용자 123456 삭제 완료
```

사용자의 주민번호는 계좌 테이블의 FK로 쓰이고 있다. 따라서 사용자데이터를 삭제하기 위해서 계좌 테이블에서 사용자가 가지고 있는 계좌를 모두 삭제한 후에 처리된다.

vii. 관리자 삭제

```
choose one: 8
관리자 아이디: kildong@gmail.com
관리자 kildong@gmail.com 삭제 완료

        0. 이전 단계
        1. 사용자 정보
        2. 계좌 정보
        3. 입출금 내역
        4. 관리자 목록
        5. 관리자 추가
        6. 사용자 데이터 삭제
        7. 계좌 삭제
        8. 관리자 삭제

choose one:
```

관리자 테이블에 대한 Update 기능을 구현하였지만 메뉴에는 보이지 않게 설정

4. 사용되는 SQL query 명세

A. Insert

i. 사용자 추가

```
"insert into " + user_table + \
                    " values({},'{}','{}','{}','{}');".format(
                        SSN, Lname, Fname, Bdate, NickName)
```

ii. 계좌 생성

```
"insert into " + account + \
                            " values({}, {}, {}, {})".format(
                                account, balance, Ussn, pw)
```

iii. 로그 데이터 기록

```
"insert into " + Log + "(Account, Name, Withdraw, Deposit,
LogDate) values({}, '{}', {}, {},'{}')".format(account,
name, cash, '0', curdate)
```

iv. 관리자 추가

```
"insert into Admin values('{}', {})".format(ID, PW)
```

B. Select

i. 유저 정보

```
"select Lname, count(*) as numOfaccount from User,
```

```
 Account where SSN = Ussn group by Lname"
```

ii. 계좌 정보

```
"select Lname, Fname, Balance from User, Account where Ussn =
 SSN and    Balance >= {} order by Balance".format(balance)
```

iii. 로그 데이터

```
"select Name, sum(withdraw), sum(deposit) from Log group by Name"
```

iv. 관리자 정보

```
"Select * from Admin"
```

C. Update

i. 사용자 닉네임 변경

```
"update User set NickName = '{}' where SSN = {}".format(nick_name, ssn)
```

ii. 계좌 입출력 진행

```
"update " + table + \
                        " Set Balance = {} where AccNum = }".format(balance, account)
```

D. Delete

i. 사용자 정보 삭제

```
"Delete from User Where SSN = {}".format(ssn)
```

ii. 계좌 정보 삭제

```
"Delete from Account Where AccNum = {}".format(account)
```

iii. 관리자 정보 삭제

```
"Delete from Admin Where Admin_ID = '{}'".format(ID)
```