프로젝트 4: 은행 DBMS 프로그램 개발

컴퓨터소프트웨어학부 이혜민

1. 기존 과제 이후 변경 사항

project 3을 기반으로 추가되거나 변경된 사항만을 설명합니다.

1) Administrator

   관리자 등록시 8자리 수를 임의로 정할 수 있다. 하지만 이미 등록된 ID는 사용할 수 없다.

   계좌 개설 및 삭제는 사용자가 자유롭게 할 수 있다. 다만 관리자는 그 정보를 키를 통해 일부 민감한 정보를 제외하고 열람할 수 있다.

2) User

   사용자는 자유롭게 자신의 계좌를 개설하거나 삭제할 수 있다. 다만, 본인의 ID를 통해서만 자신의 계좌를 개설하거나 삭제, 정보를 수정할 수 있다. 조회도 자신의 계좌에 한해서만 할 수 있다.

   계좌 개설 시 개설한 BankBranch의 ID가 tuple에 정보로 추가된다. 관리자의 ID는 필요치 않다.

3) actransaction

   User에 대한 정보는 중복적이므로 가지지 않는다. 다만, AccountID를 통해 간접적으로 User를 SQL문으로 확인할 수 있다.

4) bankbranch

   비활성화가 가능하다. 그 경우 Manager ID는 삭제되며 NULL이 된다.

5) DBManager(추가)

   bankbranch를 삭제하거나, actransaction을 변경하는 등, 불가피하게 정보를 변경해야할 필요가 있는 경우 수정 권한을 가지는 객체이다. DBManagerMode에 접근하기 위해서는 ID와 PW에 기반한 Login이 필요하다.

2. Database 구조 설명

관리자(administrator)

- 은행에서 근무하는 직원
- Attributes

AdminID	Fname	Lname	phoneNum	Ad_state	Ad_details	BirthDate	AdBranchID
00000000	Gildong	Hong	000-0000-0000	Seoul	Seochogu	2000-01-01	0000
Administrator를 식별하는 ID	이름(first name)	성(last name)	휴대전화 번호. 대표값 하나만 기입.	주소(시/도). 대표값 하나만 기입.	주소(상세 주소). 대표값 하나만 기입.	생년월일	근무하는 Branch의 ID(FK)
8자리 수	공백 없이 최대 12자	공백 없이 최대 8자	000-0000-0000의 형식('-' 포함)	공백없이 최대 10자	공백없이 최대 16자	'2021-12-03' 등의 유효한 DATE 형식	4자리수

- Administrator는 다음의 동작을 수행할 수 있습니다.

1) User 정보 관리

User와 관련된 Data를 열람하거나 추가, 삭제, 변경할 수 있습니다.

A. User 정보 조회: User ID로 단일 User의 정보를 조회하거나, 전체 User을 조회할 수 있습니다.
B. User 추가: User의 Attributes를 입력하여 User을 새롭게 추가할 수 있습니다. 중복된 User ID는 사용할 수 없습니다.
C. User 삭제: User의 ID를 입력하여 User을 삭제할 수 있습니다. 이 경우, 해당 User가 가진 계좌도 함께 삭제됩니다.
D. User 정보 수정: User의 ID를 입력하여 User의 정보를 수정할 수 있습니다.

2) Administrator 정보 관리

Administrator와 관련된 Data를 열람하거나 추가, 삭제, 변경할 수 있습니다.

A. Administrator정보 조회: Administrator ID로 단일 Administrator의 정보를 조회하거나, 전체 Administrator을 조회할 수 있습니다.
B. Administrator 추가: Administrator의 Attributes를 입력하여 Administrator을 새롭게 추가할 수 있습니다. 중복된 Administrator ID는 사용할 수 없습니다.
C. Administrator 삭제: Administrator의 ID(00000001 이상)를 입력하여 Administrator을 삭제할 수 있습니다. 현재 Administrator mode에 접속하기 위해 사용된 계정 정보를 삭제한 경우, Main menu(select position)로 돌아갑니다. 한 명 이상의 Administrator가 존재해야 합니다.
D. Administrator 정보 수정: Administrator의 ID를 입력하여 Administrator의 정보를 수정할 수 있

습니다. Manager의 근무 branch는 변경할 수 없습니다. 변경을 원한다면 다른 직원을 Manager로 미리 배정해주어야 합니다.

3) 계좌 정보 조회
   A. 계좌 번호로 특정 단일 계좌 정보를 조회할 수 있습니다.
   B. User ID로 해당 User가 소유한 계좌 정보를 조회할 수 있습니다.
   C. 전체 계좌 정보를 조회해볼 수 있습니다.
4) 입출금 내역 조회
   A. 계좌 번호로 특정 단일 계좌를 검색하여 해당 계좌의 입출금 내역을 조회할 수 있습니다.
   B. User ID로 해당 User가 소유한 계좌의 입출금 내역을 조회할 수 있습니다.
5) 은행 관리
   A. 은행 소유 금액 조회: DB를 구동할 때 선택했던 현재 Branch가 소유한 돈을 조회하거나, 전체 유효한 은행이 보유한 돈을 조회해볼 수 있습니다.
   B. 전체 지점 조회: 전체 유효한 은행 Branch 목록을 조회할 수 있습니다.
   C. 적자 은행 조회: 적자를 내고 있는 은행만 따로 조회할 수 있습니다.
   D. 지점 추가: Bank Branch의 attribute를 입력하여 Bank Branch를 새로 추가할 수 있습니다. Bank Branch ID는 코드에 의해 주어집니다.
   E. 지점 정보 수정: Bank Branch 정보를 수정할 수 있습니다.

- Administrator mode에 접근하기 위해서는 유효한 AdminID를 입력해야 합니다.

사용자(user) = 고객

- 은행 서비스를 이용하는 고객
- Attributes

UserID	Fname	Lname	phoneNum	Ad_state	Ad_details	BirthDate
00000000	Gildong	Hong	000-0000-0000	Seoul	Seochogu	2000-01-01
User를 식별하는 ID	이름(first name)	성(last name)	휴대전화 번호. 대표값 하나만 기입.	주소(시/도). 대표값 하나만 기입.	주소(상세 주소). 대표값 하나만 기입.	생년월일
8자리 수	공백 없이 최대 12자	공백 없이 최대 8자	000-0000-0000의 형식('-' 포함)	공백없이 최대 10자	공백없이 최대 16자	'2021-12-03' 등의 유효한 DATE 형식

- User는 다음의 동작을 수행할 수 있습니다.

1) 계좌에 입금하기
   A. 입금할 계좌 번호를 입력 받고, 해당하는 계좌 번호가 DB에 존재한다면, 입금할 금액을 입력 받아서

해당 계좌에 입력한 금액만큼 입금할 수 있습니다.

2) 계좌로부터 출금하기
   A. 출금할 계좌 번호를 입력 받고, 해당하는 계좌 번호가 DB에 존재한다면, 계좌 비밀번호를 입력 받아 일치할 경우 출금할 금액을 입력 받아서 해당 계좌에 입력한 금액만큼 출금할 수 있습니다.
   B. 마이너스 통장의 경우 계좌 잔액이 음수가 되어도 상관이 없으나, 일반 통장인 경우 계좌 잔액을 0 이하로 만드는 금액을 출금할 수 없습니다.
3) 내 계좌 조회
   A. 내 계좌 정보 조회: 내가 가진 계좌의 정보를 확인할 수 있습니다.
   B. 내 계좌 거래 내역 조회: 내가 가진 계좌 정보를 보여준 뒤, 원하는 계좌를 선택하면 해당 계좌의 거래 내역을 보여줍니다.
4) 내 계좌 개설
   A. 통장의 비밀번호, 마이너스 여부를 입력하여 통장을 개설할 수 있습니다.
   B. 통장의 계좌 번호는 코드에 의해 주어집니다.
5) 내 계좌 삭제
   A. 고객님께서 소유한 계좌를 보여준 뒤, 삭제할 계좌 번호와 해당 계좌의 비밀번호를 입력 받아 계좌를 삭제합니다.
6) 내 계좌 정보 수정

   변경할 계좌의 계좌 번호를 입력 받습니다.
   A. 마이너스 통장 여부 변경: 마이너스 통장 여부를 변경할 수 있습니다. 하지만, 금액이 음수인 통장을 일반 통장으로 바로 변경할 수는 없습니다. 대출된 금액만큼 입금 후 정보 변경이 가능합니다.
   B. Password 변경: 기존의 Password를 입력 받아 일치하면 새로운 4자리 Password를 설정할 수 있습니다.

- 3) ~ 6)의 동작을 수행하기 위해서는 동작을 수행하기 전 유효한 UserID를 입력해야 합니다.

계좌(account)

- 은행의 DB 내에 저장된 계좌
- Attributes

AccountID	Balance	Password	isMinus	AcBranchID	AcUserID	StartDate
123-4567-8901	0	1234	1 또는 -1	0000	00000001	2021-12-01
Account를 식별하는 계좌 번호	계좌 잔액	계좌 비밀번호	마이너스 통장 가능 여부(1: 가능)	계좌를 개설한 Branch의 ID	계좌를 보유한 User의 ID	계좌 개설일
000-0000-0000의	최대 2억	4자리수	1 또는 -1	4자리 수	8자리수	'2021-12-03'

형식('-' 포함)						등의 유효한 DATE 형식

- Administrator에 의해 조회할 수 있습니다.
- User에 의해 해당 User가 소유한 Account에 한해 조회할 수 있습니다.
- AccountID는 11자리 수로, {000-0000-0000}의 형태입니다. 처음에는 sequential하게 주어지고, 이후에는 heuristic하게 주어집니다.

입출금 기록(actransaction)

- 계좌의 입출금 기록
- Attirubtes

acTimeStamp	acType	amount	TBranchID	TAccountID
2021-12-01 01:01:01	1(입금) or -1(출금)	1000	0000	000-0000-0007
해당 거래 기록이 이루어진 시간	해당 거래가 입금인지 출금인지 구분	해당 거래의 거래 금액	거래가 이루어진 Branch ID	해당 거래가 이루어진 Account ID
2021-12-01-01:01:01과 같은 유효한 SQL의 CURRENT_TIMESTAMP 타입	1 또는 -1	최대 1천만원	0 이상의 4자리수	000-0000-0000의 형식('-' 포함)

- 계좌의 단일 거래 내역 하나 하나를 나타냅니다.

은행 지점(bankbranch)

- 'OO은행 서울 종로구점'과 같은 은행의 각 지점을 나타냅니다.
- Attributes

BranchID	Lo_state	Lo_details	ManagerID
0001	Seoul	Gangnamgu	01010101
Bank Branch을 식별하는 ID	해당 Branch의 주소(시/도)	해당 Branch의 상세 주소(종로구, 강남구 등)	Branch를 대표하는 Administrator ID
0이상의 4자리수	공백 없이 최대 12자	공백 없이 최대 16자	8자리 수

DB Manager(dbmanager)

- Database에 잘못된 정보가 기입되는 경우 이를 수정할 수 있는 권한을 가진 사람
- Attributes

DBManagerID	Fname	Lname	phoneNum	Ad_state	Ad_details	BirthDate	Password
00000000	Gildong	Hong	000-0000-0000	Seoul	Seochogu	2000-01-01	1a2b3c
DB Manager을 식별하는 ID	이름(first name)	성(last name)	휴대전화 번호. 대표값 하나만 기입.	주소(시/도). 대표값 하나만 기입.	주소(상세 주소). 대표값 하나만 기입.	생년월일	로그인을 위한 비밀 번호
8자리 수	공백 없이 최대 12자	공백 없이 최대 8자	000-0000-0000의 형식('-' 포함)	공백없이 최대 10자	공백없이 최대 16자	'2021-12-03' 등의 유효한 DATE 형식	공백 없이 최대 15자

3. 컴파일 및 실행 방법

1) 준비물

A. mysql-connector-java-8.0.27.jar

B. Main.java (Bank DBMS code file)

C. BankDump.sql (Schema of Bank Database and dump data file)

2) 실행 방법

① BankDump.sql 파일을 import 합니다. cmd나 mysql CLI에서 진행해도 가능하며, heidisql.exe 〉 파일 〉 SQL 파일 실행 〉 BankDump.sql (인코딩: UTF-8)로 실행해도 실행할 수 있습니다.

② 준비물 A~C를 한 폴더 안에 준비합니다.

③ cmd를 열어 준비물 A~C를 준비한 폴더까지의 절대 경로를 입력합니다.

```
> cd C:\Users\loveg\2021_ite2038_2020028586\04_BankDBMS
```

④ 다음 명령어를 cmd에 입력합니다.

```
> javac -classpath mysql-connector-java-8.0.27.jar; Main.java -encoding UTF-8
> java -cp mysql-connector-java-8.0.27.jar; Main
```

4. 프로그램 설명

1) 각 함수에 대한 자세한 설명 및 수행 스크린샷

Invalid input에 대해서는 접근이 금지되거나 예외가 발생할 수 있습니다.

① void main(String[] args): mysql 및 jdbc에 연결을 시도합니다. 최종적으로 bank DB에 접근을 시도하여 유효하다면 이후의 프로그램을 정상적으로 실행합니다. 먼저 현재 bank DB를 구동하고 있는 Bank Branch 지점을 입력 받아 DB이 기본 정보를 꾸립니다.

```
Enter ID: ID입력
Enter PW: PW입력
. . . DB 접속 성공

--------Bank Database System---------
Bank Application에 오신 것을 환영합니다..

Bank의 전체 Branch 목록:

BranchID Location                ManagerID 사원 수
-------------------------------------------------
0000     online -                00000001  2
0001     Seoul Junggu            00000010  1
0002     Busan Sasanggu          00000005  2
0003     Busan Gijanggun         11221122  2

현재 BranchID(4자리수) 입력: 1

현재 지점 정보는 다음과 같습니다.

BranchID Location                ManagerID 사원 수
-------------------------------------------------
0001     Seoul Junggu            00000010  1
```

② find●●By○○ID(input): ○○의 ID 정보를 바탕으로 해당하는 ●●가 DB에 존재하는지 탐색합니다. SELECT 구문을 이용하며, 결과를 result set에 저장합니다.

③ show○○ByID(): ○○의 정보를 보여줍니다. ID는 생략되는 경우도 있습니다.

④ ○○LoginByID(): User/Administrator/DB Manager에 각각 로그인하기 위해 사용되는 함수입니다.

```
<Administrator Login>
 관리자 님의 Administrator ID(8자리 수)를 입력해주세요: 00000001
 안녕하세요, Lee Jintae님.
```

⑤ menuSelectPosition(): User/Administrator/DB Manager 중 어떤 사용자가 DB에 접근하려고 하는지 확인

합니다.

```
--------Select your position--------
 0. Exit
 1. Administrator mode
 2. User mode
 3. DB Manager mode
-----------------------------------
 Input: 1
-----------------------------------
```

⑥ AdminMainMenu(): Administrator로 로그인 시 사용가능한 기능을 보여줍니다.

```
------------ADMIN MODE-------------
-----------Select action-----------
 0. Return to previous menu
 1. User 정보 관리
 2. Administrator 정보 관리
 3. 계좌 정보 조회
 4. 입출금 내역 조회
 5. 은행 관리
-----------------------------------
 Input: 1
-----------------------------------

 < User 정보 관리 >
  0. Return to previous menu
  1. User 정보 조회
  2. User 추가
  3. User 삭제
  4. User 정보 수정
 Input: 1
```

⑦ adminManage○○(): Administrator가 User/Administrator/Bank의 정보를 조회하거나 관리(추가/삭제/수정)합니다. Bank를 관리하는 경우 추가적으로 총 금액이나 적자를 내고 있는 유의해야할 은행을 조회할 수도 있습니다. 다만 Bank Branch의 삭제는 불가능합니다(DB Manager의 권한).

- User 정보 관리

```
 < User 정보 관리 >
  0. Return to previous menu
  1. User 정보 조회
  2. User 추가
  3. User 삭제
  4. User 정보 수정
 Input: 1

 < User 정보 조회 >
  1. User ID로 검색하기
  2. 전체 User 조회하기
 Input: 1

 정보를 검색할 User ID 입력(8자리 수): 00000002
UserID   Name               phoneNum      Address                    BirthDate
00000002 Gildong Hong       010-0000-5678 Ulsan Namgu                1998-01-25
```

```
 Input: 1

 < User 정보 조회 >
  1. User ID로 검색하기
  2. 전체 User 조회하기
 Input: 2

--------------전체 user--------------
UserID   Name               phoneNum      Address                    BirthDate
00000001 Younghee Kim       010-1234-5678 Seoul Gangnamgu            2001-01-01
00000002 Gildong Hong       010-0000-5678 Ulsan Namgu                1998-01-25
00000008 Jenny Kim          010-6535-3481 Busan Gijanggun            1988-03-06
00112233 Thomas Campbell    010-5821-4697 Seoul Gangnamgu            2001-08-23
12341234 Luke Watson        010-3584-1274 Seoul Mapogu               1987-10-02
```

```
 Input: 2

 < User 추가 등록 >
  1. 등록할 User ID(8자리 수) 입력: 32322121
  2. 이름(first name) 입력: Sangwon
  3. 성(last name) 입력: Lee
  4. 전화번호(000-0000-0000) 입력: 010-4578-9800
  5. 주소(특별시/광역시/도) 입력: Daegu
  6. 주소(나머지 주소) 입력: Suseonggu
  7. 생년월일(yyyy-MM-dd) 입력: 2000-01-12

 다음의 User을 추가하였습니다:
UserID   Name               phoneNum      Address                    BirthDate
32322121 Sangwon Lee        010-4578-9800 Daegu Suseonggu            2000-01-12
```

```
 < User 정보 관리 >
  0. Return to previous menu
  1. User 정보 조회
  2. User 추가
  3. User 삭제
  4. User 정보 수정
 Input: 3

 < User 삭제 >
  삭제할 User ID(8자리 수) 입력: 32322121
 User 삭제 성공: 이전 메뉴 선택 창으로 돌아갑니다.
```

```
  3. User 삭제
  4. User 정보 수정
 Input: 4

 < User 정보 수정 >
  수정할 User ID(8자리 수) 입력: 00000001
  1. 이름(first name) 입력: Younghee
  2. 성(last name) 입력: Lee
  3. 전화번호(000-0000-0000) 입력: 010-1234-5678
  4. 주소(특별시/광역시/도) 입력: Seoul
  5. 주소(나머지 주소) 입력: Gangnamgu
 User 정보 수정을 완료하였습니다.
```

- Administrator 정보 관리: 정보 조회, 추가는 동일

```
< Administrator 정보 관리 >
 0. Return to previous menu
 1. Administrator 정보 조회
 2. Administrator 추가
 3. Administrator 삭제
 4. Administrator 정보 수정
Input: 3

< Administrator 삭제 >
 삭제할 Administrator ID(8자리 수) 입력: 00000001
 Manager는 삭제할 수 없습니다. Branch의 Manager의 수정을 원하신다면 '은행 관리'창에서 Manager를 변경해주세요.
이전 메뉴로 돌아갑니다.
```

```
< Administrator 정보 관리 >
 0. Return to previous menu
 1. Administrator 정보 조회
 2. Administrator 추가
 3. Administrator 삭제
 4. Administrator 정보 수정
Input: 4

< Administrator 정보 수정 >
 수정할 Administrator ID(8자리 수) 입력: 00000001
 1. 이름(first name) 입력: Bae
 2. 성(last name) 입력: Kim
 3. 전화번호(000-0000-0000) 입력: 010-7813-9160
 4. 주소(특별시/광역시/도) 입력: Seoul
 5. 주소(나머지 주소) 입력: Junggu
 6. 근무 branch ID(4자리 수) 입력: 0002
Administrator 정보 수정 실패: Manager의 근무 branch는 변경할 수 없습니다.
Branch의 Manager 수정을 원하신다면 '은행 관리'창에서 Manager를 변경해주세요.
이전 메뉴 선택 창으로 돌아갑니다.
```

- 은행 관리: 적자 은행은 있는 경우 보여주며, 지점 추가, 지점 정보 수정은 Manager update 제한이 걸린 채로 다른 table에 대한 동작과 비슷하게 수행된다.

```
<은행 관리>
 1. 은행 소유 금액 조회
 2. 전체 지점 조회
 3. 적자 은행 조회
 4. 지점 추가
 5. 지점 정보 수정
Input: 2

<은행 지점 전체 조회>

BranchID Location                ManagerID 사원 수
--------------------------------------------------
0000     online -                00000001  2
0001     Seoul Junggu            00000010  1
0002     Busan Sasanggu          00000005  2
0003     Busan Gijanggun         11221122  2
```

```
<은행 관리>
 1. 은행 소유 금액 조회
 2. 전체 지점 조회
 3. 적자 은행 조회
 4. 지점 추가
 5. 지점 정보 수정
Input: 1

<은행 소유 금액 조회>
 1. 현재 지점 소유 금액 조회
 2. 전체 은행 소유 금액 조회
Input: 1
 현재 지점의 전체 소유 금액: 79150
```

⑧ adminShow○○(): User의 계좌나 계좌 거래 내역을 보여줍니다.

- 계좌 정보 조회

```
< 계좌 정보 조회 >
 0. Return to previous menu
 1. 계좌 번호(Account ID)로 검색하기
 2. User ID로 검색하기
 3. 전체 계좌 조회하기
Input: 1
검색할 Account ID(000-0000-0000) 입력: 000-0000-0001

AccountID     Balance      Overdraft    BranchID OwnerID  OpeningDate
000-0000-0001 14600        possible     0001     00000002 2021-12-01
```

```
< 계좌 정보 조회 >
 0. Return to previous menu
 1. 계좌 번호(Account ID)로 검색하기
 2. User ID로 검색하기
 3. 전체 계좌 조회하기
Input: 2
검색할 User ID(8자리 수) 입력: 00000002

AccountID     Balance      Overdraft     Branch                          OpeningDate
-------------------------------------------------------------------------------------
000-0000-0001 14600        possible      Seoul Junggu지점                2021-12-01
000-0000-0006 4500         possible      Busan Sasanggu지점              2021-12-01
```

```
 Input: 3

AccountID     Balance      Overdraft    BranchID OwnerID  OpeningDate
000-0000-0001 14600        possible     0001     00000002 2021-12-01
000-0000-0002 64550        impossible   0001     00000008 2021-12-01
000-0000-0004 29500        possible     0000     00000001 2021-12-01
000-0000-0006 4500         possible     0002     00000002 2021-12-01
000-0000-0008 0            possible     0002     00000001 2021-12-01
000-0000-0009 0            impossible   0002     00112233 2021-12-01
000-0000-0010 3450         possible     0003     00112233 2021-12-03
```

- 입출금 내역 조회

```
< 거래 내역 조회 >
 0. Return to previous menu
 1. 계좌 번호(Account ID)로 검색하기
 2. User ID로 검색하기
Input: 1

거래 내역을 검색할 Account ID(000-0000-0000) 입력: 000-0000-0004
-----------------------------------------------------------
TimeStamp               Type Amount           BranchID
-----------------------------------------------------------
2021-12-03 03:51:30.0   입금  10000            0000
2021-12-03 03:51:50.0   출금  8000             0000
2021-12-03 03:52:10.0   입금  20000            0000
2021-12-03 15:50:17.0   입금  7500             0001
```

```
< 거래 내역 조회 >
 0. Return to previous menu
 1. 계좌 번호(Account ID)로 검색하기
 2. User ID로 검색하기
Input: 2
거래 내역을 검색할 User ID(8자리 수) 입력: 00000002

Account ID: 000-0000-0001
-------------------------------------------------------
TimeStamp               Type Amount           BranchID
-------------------------------------------------------
2021-12-03 15:52:42.0   입금  20000            0001
2021-12-03 15:53:19.0   출금  5400             0001

Account ID: 000-0000-0006
-------------------------------------------------------
TimeStamp               Type Amount           BranchID
-------------------------------------------------------
2021-12-03 02:12:45.0   입금  4500             0002
```

⑨ DBManagerMode(): DB Manager가 로그인시 수행할 수 있는 기능을 보여줍니다. 주로 Administrator나 User는 수정 및 삭제가 불가능한 영역까지 접근하여 수정하는 기능을 제공합니다. 사용에 유의해야합니다.

```
----------DB Manager MODE----------
 0. Return to previous menu
 1. Account Transaction 삭제하기
 2. Account Transaction 수정하기
 3. Bank Branch 삭제 또는 재활성화
 4. DB Manager 추가하기
 5. DB Manager 삭제하기
 6. DB Manager 정보 수정하기
 7. DB Manager 정보 조회하기
 8. 전체 지점 조회(과거 지점 포함)
-----------------------------------
 Input:
```

A. Account transaction 수정 및 삭제 – update된 결과가 계좌에 반영됩니다.

```
 <Account Transaction 삭제하기>
  Account ID로 계좌 입출금 내역을 조회한 뒤, 원하는 기록을 삭제할 수 있습니다.
  1. 입출금 내역을 검색할 계좌의 AccountID(000-0000-0000) 입력: 000-0000-0004
    000-0000-0004의 입출금 내역은 다음과 같습니다.
TimeStamp               Type Amount           BranchID
2021-12-03 03:51:30.0   입금  10000            0000
2021-12-03 03:51:50.0   출금  8000             0000
2021-12-03 03:52:10.0   입금  20000            0000
2021-12-03 15:50:17.0   입금  7500             0001

  2. 삭제할 기록의 TimeStamp를 입력해주세요(0000-00-00 00:00:00): 2021-12-03 03:51:50
  기록 삭제를 완료하였습니다.
```

B. Bank Branch 비활성화 또는 재활성화 – Bank Branch를 비활성화로 처리하는 이유는 계좌 거래 내역의 Branch ID가 계속 유지되기 때문입니다(실제 은행에서 발급받은 계좌를 참고함). 다른 ID로 덮어씌워지면 안되는 정보이기 때문에 비활성화(ManagerID = null)로 처리합니다. 이 경우 해당 Branch에서 근무

하던 모든 Administrator는 해고됩니다. 특정 지점의 은행이 영업을 종료하는 경우 수행합니다.

```
< Bank Branch 관리>
 1. Bank Branch 삭제
 2. Bank Branch 재활성화
Input: 2

< Bank Branch 재활성화 >
 재활성화할 Bank Branch ID를 입력(4자리 수): 0001
 해당하는 ID의 Branch가 존재하지 않거나 이미 활성화 되어있습니다. 이전 메뉴로 돌아갑니다.
```

C. DB Manager 추가/삭제/수정/조회

```
 < DB Manager 정보 조회 >
  1. 내 정보 보기
  2. 전체 DB Manager 조회하기
 Input: 2
ManagerID Name               phoneNum      Address                    BirthDate
00000001  Garam Jung          010-4656-5645 Seoul Mapogu               1979-09-03
00000002  Yuri Lee            010-5472-0210 Seoul Mapogu               1996-05-05
```

D. 전체 지점 조회(비활성화 지점 포함)

```
 <전체 지점 조회>
BranchID Location               ManagerID isValid
0000     online -               00000001  true
0001     Seoul Junggu           00000010  true
0002     Busan Sasanggu         00000005  true
0003     Busan Gijanggun        11221122  true
```

⑩ userMainMenu(): User가 수행할 수 있는 기능을 보여줍니다.

```
-------------USER MODE-------------
------Banking System(for user)------
 0. Return to previous menu
 1. 계좌에 입금하기
 2. 계좌로부터 출금하기
 3. 내 계좌 조회
 4. 내 계좌 개설
 5. 내 계좌 삭제
 6. 내 계좌 정보 수정
-------------------------------------
 Input:
```

⑪ userDeposit(): 계좌번호를 입력받고, 유효하다면, 해당 계좌에 input 금액을 입금합니다.

```
-----------------------------------
 Input: 1
-----------------------------------
 입금할 계좌 번호 입력(000-0000-0000): 000-0000-0006
 입금할 금액 입력: 20
 현재 계좌 잔액: 4520
```

⑫ userWithdraw(): 계좌번호를 입력받고, 유효하다면, 계좌 비밀번호를 받고 올바른 경우 input 금액만큼 출금합니다. 단, 마이너스 통장이 아닌 경우 0원 이하로 잔액을 남길 수 없습니다.

```
-----------------------------------
 Input: 2
-----------------------------------
 출금할 계좌 번호 입력(000-0000-0000): 000-0000-0006
 계좌 비밀번호 입력(4자리 수): 3573
 출금할 금액 입력: 20
 현재 계좌 잔액: 4500
```

⑬ userShowAccounts(): userID로 로그인한 뒤, 내 계좌 정보를 보거나 내 계좌의 거래 내역을 볼 수 있습니다.

```
-----------------------------------
 Input: 3
-----------------------------------

 <User Login>
  고객님의 User ID(8자리 수)를 입력해주세요: 00000001
  안녕하세요, Lee Younghee님.

 <내 계좌 조회>
  1. 내 계좌 정보 조회
  2. 내 계좌 거래 내역 조회
 Input: 1


AccountID     Balance      Overdraft     Branch                          OpeningDate
-------------------------------------------------------------------------------------
000-0000-0004 29500        possible      online -지점                     2021-12-01
000-0000-0008 0            possible      Busan Sasanggu지점               2021-12-01
```

```
 <내 계좌 조회>
  1. 내 계좌 정보 조회
  2. 내 계좌 거래 내역 조회
 Input: 2


AccountID     Balance      Overdraft     Branch                          OpeningDate
-------------------------------------------------------------------------------------
000-0000-0004 29500        possible      online -지점                     2021-12-01
000-0000-0008 0            possible      Busan Sasanggu지점               2021-12-01


 거래 내역을 검색할 Account ID(000-0000-0000) 입력: 000-0000-0002
  고객님의 계좌 중에서는 해당 Account가 존재하지 않습니다.
```

⑭ userMakeNewAccounts(): userID로 로그인한 뒤, 내 계좌를 새롭게 개설할 수 있습니다. 마이너스 통장 여부와 비밀번호를 입력하여 개설할 수 있습니다. 계좌 번호는 코드에 의해 자동적으로 수행됩니다.

```
-------------------------------------
 Input: 4
-------------------------------------

 <User Login>
  고객님의 User ID(8자리 수)를 입력해주세요: 00000001
  안녕하세요, Lee Younghee님.

 <내 계좌 개설>
  새로운 계좌 개설을 위해 다음의 정보를 입력해주십시오.
  1. 개설할 통장의 비밀 번호 입력(4자리 수): 5656
  2. 마이너스 통장으로 개설하시겠습니까? (Y/N): Y
고객님의 Account가 새롭게 개설 되었습니다. 계좌 번호: 000-0000-0011
```

⑮ userDeleteAccount(): userID로 로그인한 뒤, 내 계좌를 삭제할 수 있습니다. 본인 소유의 계좌 번호와 해당 계좌의 Password를 올바르게 입력하여 계좌를 삭제할 수 있습니다. 해당 계좌에 남아있던 금액은 반환됩니다.

```
-------------------------------------
 Input: 5
-------------------------------------

 <User Login>
  고객님의 User ID(8자리 수)를 입력해주세요: 00000001
  안녕하세요, Lee Younghee님.

 <내 계좌 삭제>
  현재 고객님께서 소유한 계좌는 다음과 같습니다.


AccountID     Balance      Overdraft     Branch                          OpeningDate
-----------------------------------------------------------------------------------------
000-0000-0004 29500        possible      online -지점                     2021-12-01
000-0000-0008 0            possible      Busan Sasanggu지점               2021-12-01
000-0000-0011 0            possible      Seoul Junggu지점                 2021-12-03


  삭제할 Account ID(000-0000-0000) 입력: 000-0000-0011

 삭제할 계좌 비밀번호 입력(4자리 수): 5656
 고객님의 Account가 삭제되었습니다.
```

⑯ userUpdateAccount(): userID로 로그인 한 뒤, 내 계좌 정보를 수정할 수 있습니다. Minus 여부와 비밀번호를 수정할 수 있습니다.

```
 <User Login>
  고객님의 User ID(8자리 수)를 입력해주세요: 00000001
  안녕하세요, Lee Younghee님.

 <내 계좌 정보 수정>
 0. Return to previous menu
 1. 마이너스 통장 여부 변경
 2. Password 변경
 Input: 1

  현재 소유하고 계신 계좌는 다음과 같습니다.

AccountID     Balance      Overdraft     Branch                          OpeningDate
-----------------------------------------------------------------------------------------
000-0000-0004 29500        possible      online -지점                     2021-12-01
000-0000-0008 0            possible      Busan Sasanggu지점               2021-12-01


  Minus 여부를 변경할 계좌 번호를 입력해주세요(000-0000-0000):000-0000-0008
  마이너스 통장 가능 여부를 변경하였습니다.
```

2) 사용되는 SQL문 명세

대표적으로 사용되는 SQL문에 대해 설명하였습니다.

A. find○○

```
//find 함수
public static void findUserByID(int inputUserID) throws SQLException {
    resultSet = statement.executeQuery( sql: "SELECT * FROM user WHERE UserID = " + inputUserID);
    //resultSet = statement.executeQuery("SELECT UserID, Fname, Lname, phoneNum, Ad_state, Ad_details, BirthDate FROM user WHERE UserID = " + inputUserID);
}

public static void findAdminByID(int inputAdminID) throws SQLException {
    //resultSet = statement.executeQuery("SELECT AdminID, Fname, Lname, phoneNum, Ad_state, Ad_details, BirthDate, AdBranchID FROM administrator WHERE AdminID = " + inputAdminID);
    resultSet = statement.executeQuery( sql: "SELECT * FROM administrator WHERE AdminID = " + inputAdminID);
}

public static void findDBManagerByID(int inputManagerID) throws SQLException {
    //resultSet = statement.executeQuery("SELECT AdminID, Fname, Lname, phoneNum, Ad_state, Ad_details, BirthDate, AdBranchID FROM administrator WHERE AdminID = " + inputAdminID);
    resultSet = statement.executeQuery( sql: "SELECT * FROM dbmanager WHERE DBManagerID = " + inputManagerID);
}

public static void findAccountByAccountID(String inputAccountID) throws SQLException {
    preparedStatement = connection.prepareStatement( sql: "SELECT AccountID, Balance, isMinus, AcBranchID, AcUserID, StartDate FROM account WHERE AccountID = ?");
    preparedStatement.setString( parameterIndex: 1, inputAccountID);
    resultSet = preparedStatement.executeQuery();
}


public static void findTransactionByAccountID(String inputAccountID) throws SQLException {
    preparedStatement = connection.prepareStatement( sql: "SELECT acTimeStamp, acType, amount, TBranchID, TAccountID FROM actransaction WHERE TAccountID = ? ORDER BY acTimeStamp");
    preparedStatement.setString( parameterIndex: 1, inputAccountID);
    resultSet = preparedStatement.executeQuery();
}


public static void findBranchByID(int inputBranchID) throws SQLException {
    resultSet = statement.executeQuery( sql: "SELECT BranchID, Lo_state, Lo_details, ManagerID FROM bankbranch WHERE BranchID = " + inputBranchID + " AND ManagerID is NOT NULL");
}
```

주어진 inputID를 바탕으로 해당하는 table에서 원하는 tuple을 SELECT합니다.

B. void showBranches(int inputBranchID)

```
public static void showBranches(int inputBranchID) {

    try {
        if (inputBranchID == -1) {
            resultSet = statement.executeQuery( sql: "SELECT BranchID, Lo_state, Lo_details, ManagerID, COUNT(BranchID) " +
                    "FROM bankbranch INNER JOIN administrator ON bankbranch.BranchID = administrator.AdBranchID " +
                    "WHERE ManagerID is NOT NULL GROUP BY BranchID ORDER BY BranchID");

            System.out.println("\nBranchID Location                ManagerID 사원 수");
            System.out.println("-------------------------------------------------");

            while (resultSet.next()) {
                String address = resultSet.getString( columnIndex: 2) + " " + resultSet.getString( columnIndex: 3);
                System.out.print(String.format("%04d", resultSet.getInt( columnIndex: 1)) + "      ");
                System.out.printf("%-24s ", address);
                System.out.println(String.format("%08d", resultSet.getInt( columnIndex: 4)) + "   " + resultSet.getInt( columnIndex: 5));
            }
        } else {
            resultSet = statement.executeQuery( sql: "SELECT BranchID, Lo_state, Lo_details, ManagerID, COUNT(BranchID) " +
                    "FROM bankbranch INNER JOIN administrator ON bankbranch.BranchID = administrator.AdBranchID " +
                    "WHERE ManagerID is NOT NULL AND BranchID = " + inputBranchID + " GROUP BY BranchID ORDER BY BranchID");
            if (resultSet.next()) {

                System.out.println("\nBranchID Location                ManagerID 사원 수");
                System.out.println("-------------------------------------------------");
                String address = resultSet.getString( columnIndex: 2) + " " + resultSet.getString( columnIndex: 3);
                System.out.print(String.format("%04d", resultSet.getInt( columnIndex: 1)) + "      ");
                System.out.printf("%-24s ", address);
                System.out.println(String.format("%08d", resultSet.getInt( columnIndex: 4)) + "   " + resultSet.getInt( columnIndex: 5));
            } else {
                System.out.println(" Branch가 존재하지 않습니다.");
            }
        }
    } catch (SQLException throwables) {
        System.out.println("DB 에러 발생");
    }
}
```

a) inputBranchID가 -1인 경우, 전체 Branch 목록과 Branch의 정보 및 근무하고 있는 사원 수를 출력합니다. 이를 위해 bankbranch와 administrator의 table을 join합니다. 유효한 Branch만 출력하기 위해서 Manager ID가 NULL이 아닌 tuple만 고릅니다. 이후 골라진 tuple을 BranchID에 따라 GROUP BY를 통해 group을 생성하게 되면, 근무 사원 수 만큼 동일한 BranchID를 갖는 tuple이 생기게 됩니다. 따라서 이를 COUNT()해주면 사원 수를 구할 수 있습니다. 마지막으로 BranchID 순으로 오름차순 정렬하여 이를 resultSet에 담아줍니다.

```
resultSet = statement.executeQuery( sql: "SELECT BranchID, Lo_state, Lo_details, ManagerID, COUNT(BranchID) " +
        "FROM bankbranch INNER JOIN administrator ON bankbranch.BranchID = administrator.AdBranchID " +
        "WHERE ManagerID is NOT NULL GROUP BY BranchID ORDER BY BranchID");
```

b) inputBranchID가 유효한 경우, a)와 비슷한 query를 실행하되 inputBranchID에 해당하는 Branch만 출력합니다.

```
resultSet = statement.executeQuery( sql: "SELECT BranchID, Lo_state, Lo_details, ManagerID, COUNT(BranchID) " +
        "FROM bankbranch INNER JOIN administrator ON bankbranch.BranchID = administrator.AdBranchID " +
        "WHERE ManagerID is NOT NULL AND BranchID = " + inputBranchID + " GROUP BY BranchID ORDER BY BranchID");
```

C. showAccountsByUserID(int inputUserID)

```
resultSet = statement.executeQuery( sql: "SELECT AccountID, Balance, isMinus, Lo_state, Lo_details, StartDate " +
        "FROM account, bankbranch " +
        "WHERE AcUserID = " + inputUserID + " AND AcBranchID = BranchID " +
        "ORDER BY StartDate");
```

UserID를 입력받아 해당 User가 소유한 Account의 정보를 출력합니다. 이때, 계좌를 개설한 Branch의 ID가 아닌 지점명을 출력해주기 위해 bankbranch와 join해주며, account 소유주 ID가 inputUserID와 같고 계좌 개설지점 ID가 join한 bankbranch table의 ID와 같은 경우 해당 tuple을 골라줍니다. 최종적으로는 이를 개설날짜순으로 오름차순 정렬해줍니다.

D. Administrator: User, Administrator, BankBranch 추가

a) User추가: User table에 tuple을 삽입하기 위한 정보를 입력받아 INSERT 문으로 삽입합니다.

```
preparedStatement = connection.prepareStatement( sql: "INSERT INTO User(UserID, Fname, Lname, phoneNum, Ad_state, Ad_details, BirthDate) " +
        "values (?,?,?,?,?,?,?)");
preparedStatement.setInt( parameterIndex: 1, inputUserID);
preparedStatement.setString( parameterIndex: 2, inputFname);
preparedStatement.setString( parameterIndex: 3, inputLname);
preparedStatement.setString( parameterIndex: 4, inputphoneNum);
preparedStatement.setString( parameterIndex: 5, inputAd_state);
preparedStatement.setString( parameterIndex: 6, inputAd_details);

Date sqlDate = Date.valueOf(inputBirthDate);
preparedStatement.setDate( parameterIndex: 7, sqlDate);

preparedStatement.executeUpdate();
```

b) Administrator, BankBranch도 마찬가지로 동작합니다.

다만, BankBranch를 추가할 때, Administrator table에서는 근무처 ID에 대한 attribute를 갖지 않으므로 Manager의 근무 부서를 UPDATE문으로 함께 변경해줍니다.

```
preparedStatement = connection.prepareStatement( sql: "INSERT INTO bankBranch(BranchID, Lo_State, Lo_details, ManagerID) values (?,?,?,?)");
preparedStatement.setInt( parameterIndex: 1, inputBankBranchID);
preparedStatement.setString( parameterIndex: 2, inputLo_state);
preparedStatement.setString( parameterIndex: 3, inputLo_details);
preparedStatement.setInt( parameterIndex: 4, inputManagerID);
preparedStatement.executeUpdate();

preparedStatement = connection.prepareStatement( sql: "UPDATE administrator SET AdBranchID = ? WHERE AdminID = ?");
preparedStatement.setInt( parameterIndex: 1, inputBankBranchID);
preparedStatement.setInt( parameterIndex: 2, inputManagerID);
preparedStatement.executeUpdate();
```

E. Administrator: User, Administrator 삭제

```
statement.executeUpdate( sql: "DELETE FROM User WHERE UserID = " + inputUserID);
```

DELETE문으로 삭제해줍니다.

F. Administrator: User, Administrator, BankBranch 정보 수정

a) User수정: Administrator의 권한으로 수정할 수 있는 data를 UPDATE 문으로 수정합니다.

```
preparedStatement = connection.prepareStatement( sql: "UPDATE User SET Fname = ?, Lname = ?, phoneNum = ?, Ad_state = ?, Ad_details = ? WHERE UserID = ?");
preparedStatement.setString( parameterIndex: 1, inputFname);
preparedStatement.setString( parameterIndex: 2, inputLname);
preparedStatement.setString( parameterIndex: 3, inputphoneNum);
preparedStatement.setString( parameterIndex: 4, inputAd_state);
preparedStatement.setString( parameterIndex: 5, inputAd_details);
preparedStatement.setInt( parameterIndex: 6, inputUserID);

preparedStatement.executeUpdate();
```

b) Administrator, BankBranch도 마찬가지로 동작합니다.

G. Administrator: 계좌 입출금 내역 조회

actransaction 테이블과 account table을 join하여 해당 user가 소유한 account의 정보만을 보여줍니다. 이때, 이 정보를 AccountID로 정렬하고, 안에서는 TimeStamp 순으로 정렬합니다.

```
resultSet = statement.executeQuery( sql: "SELECT acTimeStamp, acType, amount, TBranchID, TAccountID " +
        "FROM actransaction, account " +
        "WHERE TAccountID = AccountID AND AcUserID = " + inputUserID + " " +
        "ORDER BY TAccountID, acTimeStamp");
```

H. Administrator: 적자 은행 조회

특정 branch가 소유한 금액이 음수라면 적자이므로, 해당되는 bankbranch를 골라주는 작업입니다. 먼저 account와 bankbranch를 join하고, 검사하려는 bankbranch가 소유한 account만을 남기기 위해 WHERE절에서 branch ID가 같은 것만 찾아주는 조건을 걸어줍니다. 동시에, 유효한 branch만을 찾기 위해서 ManagerID가 NULL이 아니라는 조건도 AND로 같이 걸어줍니다. 이렇게 해서 골라진 tuple을 BranchID로 묶기 위해서(account 자체는 섞여 있기 때문) GROUP BY로 묶어줍니다. 각 GROUP안에서 해당되는 account들의 Balance attribute들의 합이 0 미만이라면 해당 은행은 적자이므로, GROUP 내에서 조건을 걸어주기 위해 HAVING문을 이용하여 SUM(balance)가 0 미만인지 각 그룹별로 확인해줍니다.

```
case 3:
    resultSet = statement.executeQuery( sql: "SELECT BranchID, Lo_state, Lo_details, SUM(Balance) " +
            "FROM bankbranch, account WHERE BranchID = AcBranchID AND ManagerID is NOT NULL " +
            "GROUP BY BranchID HAVING SUM(Balance) < 0 ORDER BY BranchID");
    if (resultSet.next()) {

        System.out.println("\nBranchID BranchName                                Money"); //공백32
        System.out.println("-----------------------------------------------------");

        do {
            System.out.print(String.format("%04d", resultSet.getInt( columnIndex: 1)));
            String address = resultSet.getString( columnIndex: 2) + " " + resultSet.getString( columnIndex: 3);
            System.out.printf("     %-38s ", address);
            System.out.println(resultSet.getLong( columnIndex: 4));
        } while (resultSet.next());
    } else {
        System.out.println("  해당하는 은행 지점이 없습니다.");
    }

    break;
```

I. 은행 지점을 새로 만들 때, primary key인 Branch ID를 자동으로 부여하기 위해 내림차순으로 정렬하여 순차적으로 만들어지고 있는 ID중 가장 큰 것에 +1을 하여 ID를 자동생성하는 code를 만들었습니다.

```
System.out.println("\n <은행 지점 추가>");

resultSet = statement.executeQuery( sql: "SELECT BranchID FROM bankbranch ORDER BY BranchID DESC");
```

J. User: 입금, 출금

a) 입금: CURRENT_TIMESTAMP에 일어난 입출금 기록을 actransaction 테이블에 추가합니다. 필요한 attribute 정보를 함께 INSERT 해줍니다. 또한, 대응되는 Account의 Balance 또한 UPDATE로 수정해줍니다.

```
//계좌 입금 기록 기록하기
preparedStatement = connection.prepareStatement( sql: "INSERT INTO actransaction(acTimeStamp, acType, amount, TBranchID, TAccountID) values(CURRENT_TIMESTAMP, 1, ?, ?, ?)");
preparedStatement.setInt( parameterIndex: 1, inputBalance);
preparedStatement.setInt( parameterIndex: 2, currentBranchID);
preparedStatement.setString( parameterIndex: 3, inputAccountID);
preparedStatement.executeUpdate();

//실제로 반영하기
preparedStatement = connection.prepareStatement( sql: "UPDATE account SET Balance = Balance + ? WHERE AccountID = ?");
preparedStatement.setInt( parameterIndex: 1, inputBalance);
preparedStatement.setString( parameterIndex: 2, inputAccountID);
preparedStatement.executeUpdate();
```

b) 출금: 비밀번호가 일치한다면, CURRENT_TIMESTAMP에 일어난 입출금 기록을 actransaction 테이블에 추가합니다. 필요한 attribute 정보를 함께 INSERT 해줍니다. 또한, 대응되는 Account의 Balance 또한 UPDATE로 수정해줍니다.

```
//계좌 출금 기록 기록하기
preparedStatement = connection.prepareStatement( sql: "INSERT INTO actransaction(acTimeStamp, acType, amount, TBranchID, TAccountID) values(CURRENT_TIMESTAMP , -1, ?, ?, ?)");
preparedStatement.setInt( parameterIndex: 1, inputBalance);
preparedStatement.setInt( parameterIndex: 2, currentBranchID);
preparedStatement.setString( parameterIndex: 3, inputAccountID);
preparedStatement.executeUpdate();

//실제로 반영하기
preparedStatement = connection.prepareStatement( sql: "UPDATE account SET Balance = Balance - ? WHERE AccountID = ?");
preparedStatement.setInt( parameterIndex: 1, inputBalance);
preparedStatement.setString( parameterIndex: 2, inputAccountID);
preparedStatement.executeUpdate();
```

K. User: 계좌 생성

a) DB의 용량이 초과되지 않았는지 살펴봅니다. account table의 tuple의 개수를 COUNT문을 통해 SELECT합니다.

```
resultSet = statement.executeQuery( sql: "SELECT count(*) FROM account");

if (resultSet.next() && resultSet.getLong( columnIndex: 1) >= ((long) Math.pow(10, 11)) - 1) {
    System.out.println("DB 용량 초과: 이전 메뉴 선택 창으로 돌아갑니다.");
    return;
}
```

b) 계좌 번호를 코드 상에서 생성해내기 위해, sequential하게 생성된 계좌 번호 중 가장 큰 값을 account table내의 튜플들을 ORDER BY AccountID를 하여 알아냅니다.

```
resultSet = statement.executeQuery( sql: "SELECT AccountID FROM account ORDER BY AccountID DESC");
```

max가 아니라면, 계좌 번호를 바로 배분합니다. 만약 max까지 사용하였다면, 이전에 delete되어 빈 곳을 휴리스틱하게 찾아봅니다.

```
preparedStatement = connection.prepareStatement( sql: "SELECT AccountID FROM account WHERE accountID = ?");
preparedStatement.setString( parameterIndex: 1, newAccountID);
preparedStatement.executeQuery();

if (!resultSet.next()) break;
```

c) 계좌 번호 생성이 완료되었다면, 생성일 CURRENT_DATE를 포함하여 필요한 attribute 정보를 INSRET 문을 사용하여 Account 테이블에 새로운 튜플로 넣어줍니다.

```
//계좌 생성 SQL 실행
preparedStatement = connection.prepareStatement( sql: "INSERT INTO account(AccountID, Password, isMinus, AcBranchID, AcUserID, StartDate) " +
        "values (?, ?, ?, ?, ?, CURRENT_DATE )");

preparedStatement.setString( parameterIndex: 1, newAccountID);
preparedStatement.setString( parameterIndex: 2, inputPassword);
preparedStatement.setInt( parameterIndex: 3, inputIsMinus);
preparedStatement.setInt( parameterIndex: 4, currentBranchID);
preparedStatement.setInt( parameterIndex: 5, currentUserID);
preparedStatement.executeUpdate();
```

L. User: 계좌 삭제

본인 소유의 계좌가 맞고 비밀번호도 올바르게 입력했다면 account 테이블로부터 해당 튜플을 DELETE 문을 통해 삭제합니다. 이때, foreign key constraint에 의해 CASCADE로 설정되어있던 actransaction의 tuple도 모두 삭제되게 됩니다.

```
preparedStatement = connection.prepareStatement( sql: "DELETE FROM account WHERE AccountID = ?");
preparedStatement.setString( parameterIndex: 1, inputDeleteAccountID);
preparedStatement.executeUpdate();
```

M. User: 계좌 update

a) 마이너스 여부 변경: tuple의 attribute 중 isMinus를 반전시켜줍니다. 다만, 마이너스 통장에서 일반통장

으로 변환시킬 때는 계좌 잔액이 양수여야 합니다.

```
        //마이너스 통장 -> 일반 통장
        preparedStatement = connection.prepareStatement( sql: "UPDATE account SET isMinus = -1 WHERE AccountID = ?");
        preparedStatement.setString( parameterIndex: 1, inputAcocuntID);
        preparedStatement.executeUpdate();
    }
} else {
    //일반 통장 -> 마이너스 통장
    preparedStatement = connection.prepareStatement( sql: "UPDATE account SET isMinus = 1 WHERE AccountID = ?");
    preparedStatement.setString( parameterIndex: 1, inputAcocuntID);
    preparedStatement.executeUpdate();
}
```

b) 비밀번호 변경: 기존의 비밀번호를 올바르게 입력했고, 유효한 새로운 4자리 비밀번호를 입력했다면, 비밀번호가 UPDATE 문을 통해 변경됩니다.

```
preparedStatement = connection.prepareStatement( sql: "UPDATE Account SET Password = ? WHERE AccountID = ?");
preparedStatement.setString( parameterIndex: 1, newPW);
preparedStatement.setString( parameterIndex: 2, inputAcocuntID);
preparedStatement.executeUpdate();
```