데이터베이스 ESQL과제

* 고급언어 Python –

2315028 인공지능공학부 김성현

1. Progress Report

2024.11.09

PM 10:00 – 1:00 주제설정 및 데이터 생성, 연결

2024.11.10

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

2024.11.13~15   
코드 보완 (오류 수정, 기능 추가)

2024. 11. 16

PM 10:00-11:00 보고서 작성

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

2024. 11 . 23

PM 8:00 – 9:00 보고서 수정

1. 주제

**눈송이 은행원 프로그램**

: 눈송이 은행에서 일하는 직원이 사용하는 프로그램을 구현해보았습니다.

고객의 계좌 관리부터 지점관리까지 모든 것이 가능한 프로그램을 구현하였습니다.

1. 함수
2. 데이터베이스 생성함수 : create\_db()
3. def create\_schema():
4. schema\_creation\_query = """
5. CREATE TABLE 사용자 (
6. 사용자\_ID INT PRIMARY KEY,
7. 이름 VARCHAR(50) NOT NULL,
8. 성별 CHAR(1) CHECK (성별 IN ('M','F')),
9. 연락처 VARCHAR(15) NOT NULL,
10. 주민번호 CHAR(15) UNIQUE NOT NULL
11. );
12. CREATE TABLE 계좌 (
13. 계좌번호 INT PRIMARY KEY,
14. 사용자\_ID INT,
15. 계좌유형 VARCHAR(20) CHECK(계좌유형 IN ('저축','당좌','정기예금')),
16. 잔액 DECIMAL(15, 2) DEFAULT 0,
17. FOREIGN KEY (사용자\_ID) REFERENCES 사용자(사용자\_ID)
18. );
19. CREATE TABLE 거래내역 (
20. 거래\_ID INT PRIMARY KEY,
21. 계좌번호 INT,
22. 거래일시 TIMESTAMP DEFAULT CURRENT\_TIMESTAMP,
23. 거래유형 VARCHAR(10) CHECK (거래유형 IN ('입금','출금')),
24. 거래금액 DECIMAL(15, 2) REFERENCES 계좌(계좌번호)
25. );
26. CREATE TABLE 지점 (
27. 지점\_ID INT PRIMARY KEY,
28. 위치 VARCHAR(100) NOT NULL,
29. 지점장\_이름 VARCHAR(50) NOT NULL,
30. 직원수 INT CHECK (직원수 >= 0)
31. );
32. """
33. *try*:
34. connection = connect\_db()
35. *if* connection is None:
36. print("데이터베이스와 연결이 안됩니다.")
37. cursor = connection.cursor()
38. cursor.execute(schema\_creation\_query, *multi*=True)
39. connection.commit()
40. print("데이터베이스가 성공적으로 만들어졌습니다.")
41. *except* mysql.connector.Error *as* e:
42. *if* "already exists" in str(e):
43. print("\n 데이터베이스가 이미 존재합니다.")
44. *else*:
45. print(f"Unexpected error occurred: {e}")
46. *finally*:
47. *if* connection.is\_connected():
48. cursor.close()
49. connection.close()

기능 : 새로운 데이터 베이스를 생성하는 함수입니다.

반환값 : 데이터베이스와 연동여부에 따라 메세지가 출력됩니다.

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

1. 데이터베이스 연결함수 : connect\_db()
2. def connect\_db():
3. *return* mysql.connector.connect(
4. *host*="localhost",
5. *user*="root",
6. *password*="khyun0303",
7. *database*="ESQL"
8. )

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

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

2. 사용자 정보 조회 함수 : get\_user\_info(user\_id)

def get\_user\_info(*user\_id*):

db = connect\_db()

cursor = db.cursor()

query = """

SELECT 사용자.이름, 사용자.성별, 사용자.연락처, 계좌.계좌번호

FROM 사용자

JOIN 계좌 ON 사용자.사용자\_ID = 계좌.사용자\_ID

WHERE 사용자.사용자\_ID = %s;

"""

print("\n---------------SQL Code Test---------------\n", query)

print("---------------------------------------------\n")

cursor.execute(query, (*user\_id*,))

result = cursor.fetchall()

db.close()

*return* result

cursor.execute(query, (*user\_id*,))

result = cursor.fetchall()

db.close()

*return* result

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

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

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

1. 사용자 거래내역 조회 함수 : get\_account\_history(account\_number)
2. def get\_account\_history(*account\_number*):
3. db = connect\_db()
4. cursor = db.cursor()
5. query = """
6. SELECT 사용자.이름, 거래내역.거래일시, 거래내역.거래유형, 거래내역.거래금액, 지점.위치
7. FROM 거래내역
8. JOIN 계좌 ON 거래내역.계좌번호 = 계좌.계좌번호
9. JOIN 사용자 ON 계좌.사용자\_ID = 사용자.사용자\_ID
10. JOIN 지점 ON 사용자.지점\_ID = 지점.지점\_ID
11. WHERE 계좌.계좌번호 = %s;
12. """
13. print("\n---------------SQL Code Test---------------\n", query)
14. print("---------------------------------------------\n")
15. cursor.execute(query, (*account\_number*,))
16. result = cursor.fetchall()
17. db.close()
18. *return* result

• **기능**: 특정 계좌 번호를 통해 거래 내역과 지점 위치를 조회하는 함수입니다.

• **쿼리 설명**: 거래내역, 계좌, 사용자, 지점 테이블을 JOIN하여 계좌번호에 맞는 거래일시, 거래유형, 거래금액, 지점 위치 등을 가져옵니다.

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

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

1. 계좌 잔액 조회 함수 (출금) : get\_account\_balance(account\_number)
2. def get\_account\_balance(*account\_number*):
3. db = connect\_db()
4. cursor = db.cursor()
5. query = "SELECT 잔액 FROM 계좌 WHERE 계좌번호 = %s;"
6. print("\n---------------SQL Code Test---------------\n", query)
7. print("---------------------------------------------\n")
8. cursor.execute(query, (*account\_number*,))
9. result = cursor.fetchone()
10. db.close()
11. *return* result[0] *if* result *else* None

• **기능**: 특정 계좌의 잔액을 조회하는 함수로, 출금 시 잔액을 확인하기 위해 사용됩니다.

• **쿼리 설명**: 계좌 테이블에서 계좌번호에 맞는 잔액을 조회합니다.

• **출력**: 조회된 잔액을 반환하여 출금 시 금액이 충분한지 확인할 수 있게 합니다.

5. 계좌 잔액 업데이트 함수(입금) : update\_account(account\_number, amount, transaction\_type)

def update\_account(*account\_number*, *amount*, *transaction\_type*):

db = connect\_db()

cursor = db.cursor()

*if* *transaction\_type* == "입금":

update\_query = "UPDATE 계좌 SET 잔액 = 잔액 + %s WHERE 계좌번호 = %s;"

print("\n---------------SQL Code Test---------------\n", update\_query)

print("---------------------------------------------\n")

*elif* *transaction\_type* == "출금":

update\_query = "UPDATE 계좌 SET 잔액 = 잔액 - %s WHERE 계좌번호 = %s;"

print("\n---------------SQL Code Test---------------\n", update\_query)

print("---------------------------------------------\n")

cursor.execute(update\_query, (*amount*, *account\_number*))

query = "SELECT COALESCE(MAX(거래\_ID), 2000) + 1 AS next\_transaction\_id FROM 거래내역;"

print("\n----------------SQL Code Test--------------\n", query)

print("---------------------------------------------\n")

cursor.execute(query)

transaction\_id\_result = cursor.fetchone()

*if* transaction\_id\_result and transaction\_id\_result[0] is not None:

transaction\_id = transaction\_id\_result[0]

*else*:

transaction\_id = 2001

insert\_query = """

INSERT INTO 거래내역 (거래\_ID, 계좌번호, 거래일시, 거래유형, 거래금액)

VALUES (%s, %s, %s, %s, %s);

"""

print("\n----------------SQL Code Test--------------\n, insert\_query")

print("---------------------------------------------\n")

cursor.execute(insert\_query, (transaction\_id, *account\_number*, datetime.now(), *transaction\_type*, *amount*))

db.commit()

db.close()

print(f"{*transaction\_type*} 완료: {*amount*}원이 계좌에 반영되었습니다.")

• **기능**: 입금 또는 출금 시 계좌 잔액을 업데이트하는 함수입니다.

• **조건**: transaction\_type이 입금인지 출금인지에 따라 쿼리를 다르게 설정합니다.

입금: 잔액을 증가시키는 쿼리 실행

출금: 잔액을 감소시키는 쿼리 실행

• **출력**: SQL 쿼리 실행 후 잔액이 업데이트됩니다.

• **오류 처리:**

출금 시 잔액이 부족하면 출금을 차단하고 오류 메시지를 출력합니다.

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

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

6. 신규 사용자 및 계좌 생성 함수 : create\_user\_account(user\_data, account\_data):

def create\_user\_account(*user\_data*, *account\_data*):

db = connect\_db()

cursor = db.cursor()

user\_query = "INSERT INTO 사용자 (사용자\_ID, 이름, 성별, 연락처, 주민번호, 지점\_ID) VALUES (%s, %s, %s, %s, %s, %s);"

print("\n---------------SQL Code Test---------------\n", user\_query)

print("---------------------------------------------\n")

cursor.execute(user\_query, *user\_data*)

account\_query = "INSERT INTO 계좌 (계좌번호, 사용자\_ID, 계좌유형, 잔액, 지점\_ID) VALUES (%s, %s, %s, %s, %s);"

print("\n---------------SQL Code Test---------------\n", account\_query)

print("---------------------------------------------\n")

cursor.execute(account\_query, *account\_data*)

db.commit()

db.close()

print("새로운 사용자 및 계좌가 성공적으로 등록되었습니다!")

•**기능**: 신규 사용자와 계좌를 생성하는 함수입니다.

•**쿼리 설명**:

첫 번째 쿼리: 사용자 테이블에 새로운 사용자 정보를 추가합니다.

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

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

7. 지점별 사용자 조회 함수 : get\_branch\_accounts(branch\_id)

def get\_branch\_accounts(*branch\_id*):

db = connect\_db()

cursor = db.cursor()

query = """

SELECT 지점.위치, 지점.지점장\_이름, 지점.직원수, COUNT(사용자.사용자\_ID) AS 사용자\_수

FROM 지점

JOIN 사용자 ON 사용자.지점\_ID = 지점.지점\_ID

WHERE 지점.지점\_ID = %s

"""

print("\n---------------SQL Code Test---------------\n", query)

print("---------------------------------------------\n")

cursor.execute(query, (*branch\_id*,))

result = cursor.fetchall()

db.close()

*return* result

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

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

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

1. 사용자 출력화면 함수 : display\_menu()
2. def display\_menu():
3. print("\n<<<<<<<<<< 눈송이 은행에 오신걸 환영합니다!! >>>>>>>>>>\n")
4. print("1. 사용자 정보 조회\n")
5. print("2. 거래내역 조회\n")
6. print("3. 입출금\n")
7. print("4. 신규 가입 및 계좌 개설\n")
8. print("5. 은행 지점 조회\n")
9. print("6. 종료\n")
10. choice = input("원하시는 번호를 입력해주세요 : ")
11. *return* choice

• **기능**: 사용자에게 은행 메뉴를 보여주는 함수입니다.

• **입력**: 사용자가 원하는 기능에 해당하는 번호를 입력받습니다.

• **출력**: 입력된 번호에 따라 적절한 함수를 호출하고 그 결과를 화면에 출력합니다.

동작

*while* True:

create\_schema()

choice = display\_menu()

*if* choice == "1":

user\_id = input("사용자 ID를 입력하세요 : ")

result = get\_user\_info(user\_id)

*if* result:

*for* row *in* result:

print(f"이름 : '{row[0]}'")

print(f"성별 : '{row[1]}'")

print(f"전화번호 : '{row[2]}'")

print(f"계좌번호 : '{row[3]}'")

*else*:

print("입력하신 사용자아이디가 올바르지 않습니다.")

*elif* choice == "2":

account\_number = input("계좌번호를 입력하세요 : ")

result = get\_account\_history(account\_number)

*if* result:

*for* row *in* result:

print(f"이름 : '{row[0]}'")

print(f"거래일시 : ({row[1].year}년, {row[1].month}월, {row[1].day}일, {row[1].hour}시:{row[1].minute}분)")

print(f"거래유형 : '{row[2]}'")

print(f"거래금액 : '{row[3]}'")

print(f"지점위치 : '{row[4]}'")

*else*:

print("입력하신 계좌번호가 올바르지 않습니다.")

*elif* choice == "3":

account\_number = input("계좌번호를 입력하세요 : ")

transaction\_type = input("입금 / 출금 : ")

*if* transaction\_type in ["입금", "출금"]:

amount = int(input(f"{transaction\_type}할 금액을 입력하세요 : "))

*if* transaction\_type == "출금":

balance = get\_account\_balance(account\_number)

*if* balance is not None and balance >= amount:

update\_account(account\_number, amount, transaction\_type)

print(f"{amount}원이 출금되었습니다. 잔액: {balance - amount}원")

*else*:

print("잔액이 부족합니다.")

*else*:

update\_account(account\_number, amount, transaction\_type)

print(f"{amount}원이 입금되었습니다.")

*else*:

print("잘못입력하셨습니다. (입금/출금) 정확히 입력해주세요.")

*elif* choice == "4":

user\_data = (

input("사용자 ID: "),

input("이름: "),

input("성별(M/F): "),

input("연락처(010-0000-0000): "),

input("주민번호(900101-1234567): "),

input("지점 번호 : ")

)

account\_data = (

input("계좌번호(4자리): "),

user\_data[0],

input("계좌유형(저축/당좌/정기예금): "),

input("초기 잔액: "),

user\_data[5]

)

create\_user\_account(user\_data, account\_data)

*elif* choice == "5":

branch\_id = input("지점 ID를 입력하세요 : ")

result = get\_branch\_accounts(branch\_id)

*if* result: *# 결과가 존재하는지 확인*

*for* row *in* result:

print(f"지점 위치 : '{row[0]}'")

print(f"지점장 이름 : '{row[1]}'")

print(f"직원 수 : '{row[2]}'")

print(f"사용자 수 : '{row[3]}'")

*else*:

print("입력하신 지점은 존재하지 않습니다.")

*elif* choice == "6":

print(">>>>>>>>>> 프로그램을 종료합니다! <<<<<<<<<<")

*break*

*else*:

print("잘못된 입력입니다. 다시 입력해주세요.")

1. **데이터베이스 생성 및 사용자 입력**

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

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

**1. 사용자 정보 조회 (Option 1)**

사용자 ID를 입력받아 get\_user\_info(user\_id) 함수를 호출합니다.

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

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

**2. 계좌 거래 내역 조회 (Option 2)**

계좌번호를 입력받아 get\_account\_history(account\_number) 함수를 호출합니다.

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

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

**3. 입출금 처리 (Option 3)**

계좌번호와 거래 유형(“입금” 또는 “출금”)을 입력받습니다.

출금의 경우, 먼저 get\_account\_balance(account\_number)로 잔액을 확인합니다. 잔액이 부족하면 출금이 불가하며, 충분할 경우 출금 처리를 수행합니다.

입금의 경우 잔액 확인 없이 바로 update\_account(account\_number, amount, transaction\_type) 함수로 처리됩니다.

결과에 따라 입출금 내역 및 잔액을 출력합니다.

**4. 새 사용자 및 계좌 생성 (Option 4)**

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

create\_user\_account(user\_data, account\_data)를 호출하여 데이터베이스에 사용자 및 계좌 정보를 저장합니다.

**5. 지점 정보 조회 (Option 5)**

지점 ID를 입력받아 get\_branch\_accounts(branch\_id)를 호출합니다.

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

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

**6. 프로그램 종료 (Option 6)**

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

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

1. 출력 결과

텍스트, 스크린샷, 폰트이(가) 표시된 사진

자동 생성된 설명

이미 데이터베이스가 존재하기에 데이터베이스가 존재한다는 문구가 출력됩니다.

사용자 화면이 동작되며, 서비스의 이름과 번호가 출력됩니다.

원하는 서비스의 번호를 입력받습니다.

텍스트, 스크린샷, 폰트이(가) 표시된 사진

자동 생성된 설명

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

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

텍스트, 스크린샷, 폰트이(가) 표시된 사진

자동 생성된 설명

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

SQL CODE TEST가 진행되고 나서, 계좌번호의 주인인 사용자의 이름, 거래가 이루어진 시간, 거래유형, 거래 금액, 은행 지점 위치가 출력됩니다. 계좌의 거래내역을 조회할 수 있습니다.

텍스트, 스크린샷, 폰트, 디자인이(가) 표시된 사진

자동 생성된 설명

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

SQL CODE TEST가 진행되고나서, 입금이 완료되었다는 메세지가 뜨게 됩니다.

텍스트, 스크린샷, 폰트이(가) 표시된 사진

자동 생성된 설명

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

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

텍스트, 스크린샷, 폰트이(가) 표시된 사진

자동 생성된 설명

4번 신규가입 및 계좌개설 서비스를 선택했을 경우입니다.

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

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

아래 사진은 신규가입 및 계좌재설 하기 전과 후의 사진입니다.

텍스트, 스크린샷, 소프트웨어이(가) 표시된 사진

자동 생성된 설명텍스트, 스크린샷, 메뉴이(가) 표시된 사진

자동 생성된 설명

(신규 사용자 개설 전과 후)

텍스트, 스크린샷, 소프트웨어이(가) 표시된 사진

자동 생성된 설명텍스트, 스크린샷, 소프트웨어, 멀티미디어 소프트웨어이(가) 표시된 사진

자동 생성된 설명

(신규 계좌 개설 전과 후)

텍스트, 스크린샷, 폰트이(가) 표시된 사진

자동 생성된 설명

5번 은행지점 조회 서비스를 입력했을 경우입니다.

지점 아이디를 입력받습니다.

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

텍스트, 스크린샷, 폰트, 디자인이(가) 표시된 사진

자동 생성된 설명

6번 종료를 입력했을 경우입니다.

프로그램을 종료합니다의 메세지가 뜨면서 프로그램이 종료됩니다.

1. Cross check

(시나리오) 웹 언어에서 바뀐 입출금 내역 확인

텍스트, 스크린샷, 폰트, 로고이(가) 표시된 사진

자동 생성된 설명 텍스트, 스크린샷, 폰트, 번호이(가) 표시된 사진

자동 생성된 설명 텍스트, 스크린샷, 폰트이(가) 표시된 사진

자동 생성된 설명