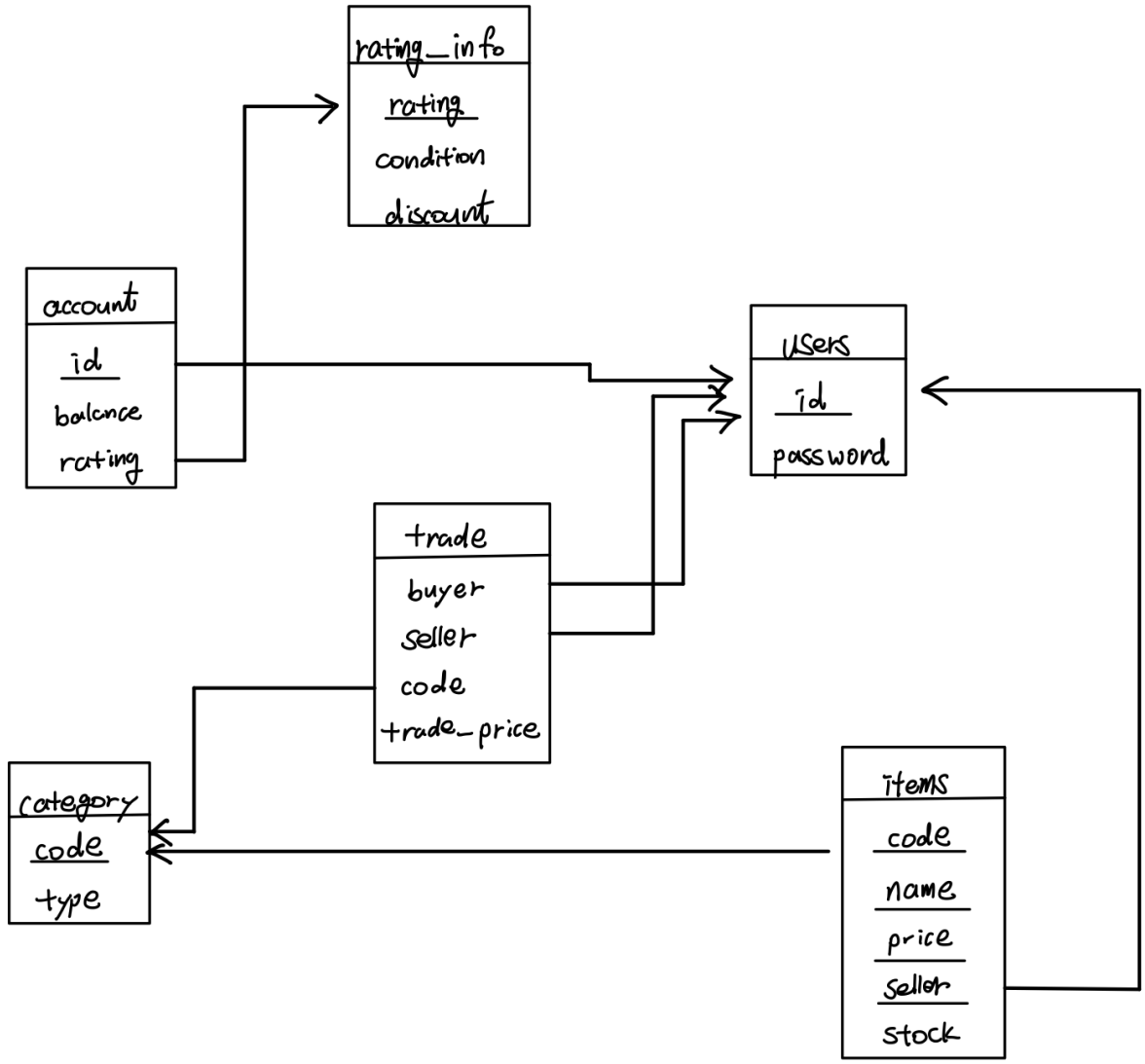
1. Schema diagram



2. 추가기능

1. 유저가 직접 자신의 쇼핑몰 캐시를 충전/인출할 수 있도록 하는 기능

2. 혹시 seller 가 물건을 add 할 때 카테고리를 잘못 입력한 경우, 쇼핑몰을 관리하는 Admin 이 잘못 입력된 카테고리를 수정할 수 있도록 하는 기능

3. 구현한 함수들에 대한 간단한 설명 (app.py 파일에 나온 순으로 정렬함)

category_buyer_seller_itemlist()

login에 성공하면 '거래 사이트의 현재 정보'가 나와야 한다. 그러기 위해서 popular_category, best_buyer, best_seller 정보가 필요하다. 또한, 현재 item들에 대한 정보도 나와야 한다. 그래서 items_list 정보도 필요하다. 그래서 위의 정보들을 매번 sql로 꺼내서 사용하기 번거롭기 때문에 위의 정보들을 담은 tuple을 통채로 return하는 함수를 만들었다.
이 함수에 사용된 sql은 다음과 같다.

"select type from category, trade where category.code = trade.code group by type order by count(type) desc"

"select buyer from trade group by buyer order by sum(trade_price) desc"

"select seller from trade group by seller order by sum(trade_price) desc"

"select * from items"

quotation(string)

sql의 where문을 사용하여 비교할 때 파이썬의 .format()으로 그냥 string을 넣으면 sql 안에서는 따옴표가 사라지는 현상을 발견했다. 그래서 .format으로 sting을 넣을 때 quotation 함수로 따옴표를 붙여서 넣기 위해 quotation 함수를 만들었다.

balance_rating_admin(ID)

매번 다른 html로 옮겨다닐 때마다 각 ID에 대한 balance, rating, admin인지 아닌지에 대한 bool 값을 반복해서 값을 가져오는 것이 불편했다. 그래서 위의 정보가 필요할 때, ID 값만 있으면 (balance, rating, admin) 튜플을 통채로 return하는 함수를 만들었다.
이 함수에 사용된 sql은 다음과 같다.

"select * from account where id = {}".format(quotation(ID))

render_template_login_success(ID)

add 작업을 끝낸 뒤, buy 작업을 끝낸 뒤 등 다시 login 직후의 상황으로 돌아가야 할 경우가 많았다. 그래서 위에서 만든 balance_rating_admin 함수와 category_buyer_seller_itemlist 함수를 호출하여 정보들을 가져와서 다시 login_success.html 을 render_template 해주는 함수를 따로 만들었다.

rating_discount(balance)

buy 작업을 할 때, rating에 맞는 할인율을 알아야 한다. 때문에, rating을 결정하는 balance의 값을 넣으면 rating과 할인율을 return하는 함수를 만들었다.
원래 실제 쇼핑몰들에서는 주로 sum(trade_price)를 기준으로 rating을 결정한다. 하지만, 뒤에서 구현할 추가 기능이 사용자가 쇼핑몰 캐시를 충전/인출할 수 있도록 하는 기능인데, 이 충전/인출

에 연동되어 바로바로 rating이 변하는 것을 구현해 보고 싶었다. 그래서, 처음에는 rating의 기준 값으로 사용할 sum(trade_price) 값을 “select sum(trade_price) from trade where buyer = {}”.format(quotation(ID)) 코드로 구했지만, 추가 기능을 구현한 뒤, rating의 기준값을 balance로 하기로 정했기 때문에 함수의 인자로 balance값을 받도록 수정하였다.
그리고, 각 balance 값 커트라인에 대한 rating의 정보를 rating_info 에서 가져오기 위해서는 “select * from rating_info” 코드를 사용했다.

match_balance_rating(ID, balance)

buy 작업 후, 또는 충전/인출 작업 후 등 현재 account에 저장되어 있는 balance값과 rating 값이 올바르게 매치되지 않은 경우가 생길 수 있다. 이러한 가능성이 존재하는 모든 곳에 반복적으로 같은 코드를 작성해야 함을 발견하여 이 기능을 수행하는 함수를 따로 만들었다.
이 함수에서 사용된 sql 코드는 다음과 같다.

"update account set rating = {} where id = {}".format(quotation(real_rating), quotation(ID))
discount_price(original_price, rating)

login()

유저가 처음 login을 하거나 sign up을 할 수 있는 초기 화면인 login.html을 render_template하는 함수다.

re_turn()

유저가 logout등의 버튼을 눌러서 초기 화면으로 돌아가고자 할 때, login.html을 render_template하는 함수다.

login_action()

login과 signup 작업을 내부적으로 처리하는 함수다. login.html의 form에서 ID, pw, send 값을 가져와서, send가 login이면, (ID, pw)가 users 테이블에 존재하는지 검사하고, send가 signup이면 중복되는 ID가 있는지 검사한다.
이 함수에 사용된 sql은 다음과 같다.

users 테이블에 현재 저장되어 있는 모든 ID, pw 쌍을 가져오기 위해 "select ID, password as pw from users" 코드를 사용했다.

signup 작업에서 ID 중복검사 이후 users 테이블과 account 테이블에 새로운 계정을 추가하기 위해 "insert into users values('{}', '{}');".format(ID, pw) 와 "insert into account values('{}','{}','{}');".format(ID, 0, "beginner") 코드를 사용했다.

admin_function()

로그인된 ID가 admin인 경우에 Admin function 기능을 제공하기 위한 함수다. Admin이 users info 버튼을 누른 경우, "select * from users"로 모든 유저들에 대한 정보를 보여주고, Admin이 trades info 버튼을 누른 경우 "select * from trade"로 모든 거래내역에 대한 정보를 보여준다.

cash()

추가 기능을 구현하기 위한 함수다. 기존의 login_success.html에 충전/인출 버튼을 만든 뒤, 충전인지 인출인지의 정보와 user에 대한 정보들을 함께 넣어 cash.html을 render_template 하여 유저가 충전/인출 작업을 할 수 있게 한다.

cash_action()

cash.html에서 유저가 얼마나 돈을 충전/인출할지에 대한 정보를 받은 후, 만약 in_out이 “충전”이면, "update account set balance = balance + {} where id = {}".format(cash_amount, quotation(ID))을 실행하고, 만약 in_out이 “인출”이면, "update account set balance = balance - {} where id = {}".format(cash_amount, quotation(ID))을 실행하여 내부적으로 캐시 충전/인출 기능을 작동하게 한다.
그리고, 이렇게 충전/인출 작업에 의해 balance에 연동되는 rating이 변할 수 있기 때문에 위에서 구현된 match_balance_rating 함수를 호출하여 유저가 충전/인출을 함과 동시에 rating이 변하는 것을 확인할 수 있도록 한다.

add()

유저가 상품을 거래할 수 있도록 추가하는 기능을 구현하기 위해서, 유저가 Add 버튼을 누르면 add.html을 render_template 하는 함수다.

add_action()

유저가 add.html에서 item을 추가하면, 이를 내부적으로 sql에 연동시켜주기 위한 함수다. 만약 유저가 추가한 상품이 기존 상품과 같은 상품이면 "update items set stock = {};".format(new_stock) 코드로 재고를 업데이트 하고, 새로운 상품이면 "insert into items values('{}', '{}', '{}', '{}', '{}');".format(code, name, price, stock, ID) 코드로 상품을 추가한다.
만약 유효하지 않은 코드를 입력한 경우에는 item_code_fail.html를 렌더링한다.

buy()

유저가 상품을 거래할 수 있도록 추가하는 기능을 구현하기 위해서, 유저가 buy 버튼을 누르면 buy.html을 render_template 하는 함수다.

buy_action()

유저가 buy.html에서 item을 구매하면, 이를 내부적으로 sql에 연동시켜주기 위한 함수다. 구매자의 rating에 대한 할인율은 위에서 구현한 함수들로부터 가져왔다. 재고와 구매하고자 하는 양, 구매자의 balance와 결제금액을 따져보고 오류가 있다면 각 오류들을 보여주는 html을 렌더링하고, 오류가 없는 주문이라면 이 주문 정보를 sql에 업데이트한다.
Item 테이블에 업데이트 하기 위해서, 만약 재고가 0이 된다면 "delete from items where code = {} and name = {} and price = {} and seller = {}".format(quotation(code), quotation(name), seller_price, quotation(seller))으로 item을 삭제하고, 구매 후에도 재고가 남아있게 된다면, cur.execute("update items set stock = {} where code = {} and name = {} and price = {} and seller = {}".format(new_stock, quotation(code), quotation(name), seller_price, quotation(seller))으로 재고를 업데이트 한다.

account 테이블에는 "update account set balance = {} where id = {}".format(buyer_new_balance, quotation(buyer))과 "update account set balance = {} where id = {}".format(seller_new_balance, quotation(seller)) 코드로 업데이트 한 후, 변화된 balance에 연동되어 rating 값을 맞춰주기 위해 match_balance_rating 함수를 호출한다.
trade 테이블에도 거래 정보를 적용하기 위해서, "insert into trade values({}, {}, {}, {})".format(quotation(buyer), quotation(seller), quotation(code), seller_price) 코드를 사용했다.

check_category()

두 번째 추가 기능인 Admin이 잘못된 카테고리를 수정할 수 있는 기능을 구현하기 위해서, amdin이 check category 버튼을 누르면 check_category.html을 render_template 하는 함수다.

check_category_action()

admin이 check_category.html에서 카테고리를 수정하면, 이를 내부적으로 sql에 연동시켜주기 위한 함수다. "update items set code = {} where code = {} and name = {} and price = {} and stock = {} and seller = {}".format(quotation(category), quotation(code), quotation(name), price, stock, quotation(seller)) 코드를 사용하여 카테고리를 업데이트 하도록 하였다. 만약 admin이 실수로 유효하지 않은 카테고리 코드를 입력한 경우, item_code_fail.html을 렌더링 하도록 하였다.

4. 사용한 html 파일들의 간단한 설명 (알파벳 순으로 정렬함)

add.html
유저가 상품을 추가할 수 있도록 하는 페이지이다.

buy_fail_balance.html
만약 유저가 자신의 balance보다 많은 결제금액으로 주문할 경우 오류가 있다고 알려주는 페이지이다.

buy_fail_stock.html
만약 유저가 현재 재고보다 많은 양을 주문하려고 할 때 오류가 있다고 알려주는 페이지이다.

buy.html
유저가 상품을 구매할 수 있도록 하는 페이지이다.

cash_out_fail.html
만약 유저가 현재 balance보다 많은 금액을 인출하려고 할 때 오류가 있다고 알려주는 페이지이다.

cash.html
유저가 쇼핑몰 캐시를 충전/인출 할 수 있도록 하는 페이지이다.

check_category.html
Admin이 잘못 입력된 카테고리를 수정할 수 있도록 하는 페이지이다.

ID_collision.html
만약 signup 과정에서 중복되는 ID로 회원가입을 하려 하는 경우 불가능하다고 알려주는 페이지이다.

Item_code_fail.html
만약 유저가 상품을 입력하거나 Admin이 카테고리를 수정할 때 유효하지 않은 카테고리 코드를 입력한 경우 오류가 있다고 알려주는 페이지이다.

login_fail.html
ID와 pw 쌍이 users 테이블에 존재하지 않는 경우 로그인에 실패했다고 알려주는 페이지이다.

login_success.html
로그인에 성공한 경우 보여지는 쇼핑몰의 메인 페이지이다.

login.html
유저가 로그인을 하거나 새로운 유저가 회원가입을 할 수 있도록 하는 페이지이다.

print_trades.html

Admin이 모든 거래내역을 확인할 수 있는 페이지이다.

print_users.html

Admin이 모든 유저 정보를 확인할 수 있는 페이지이다.