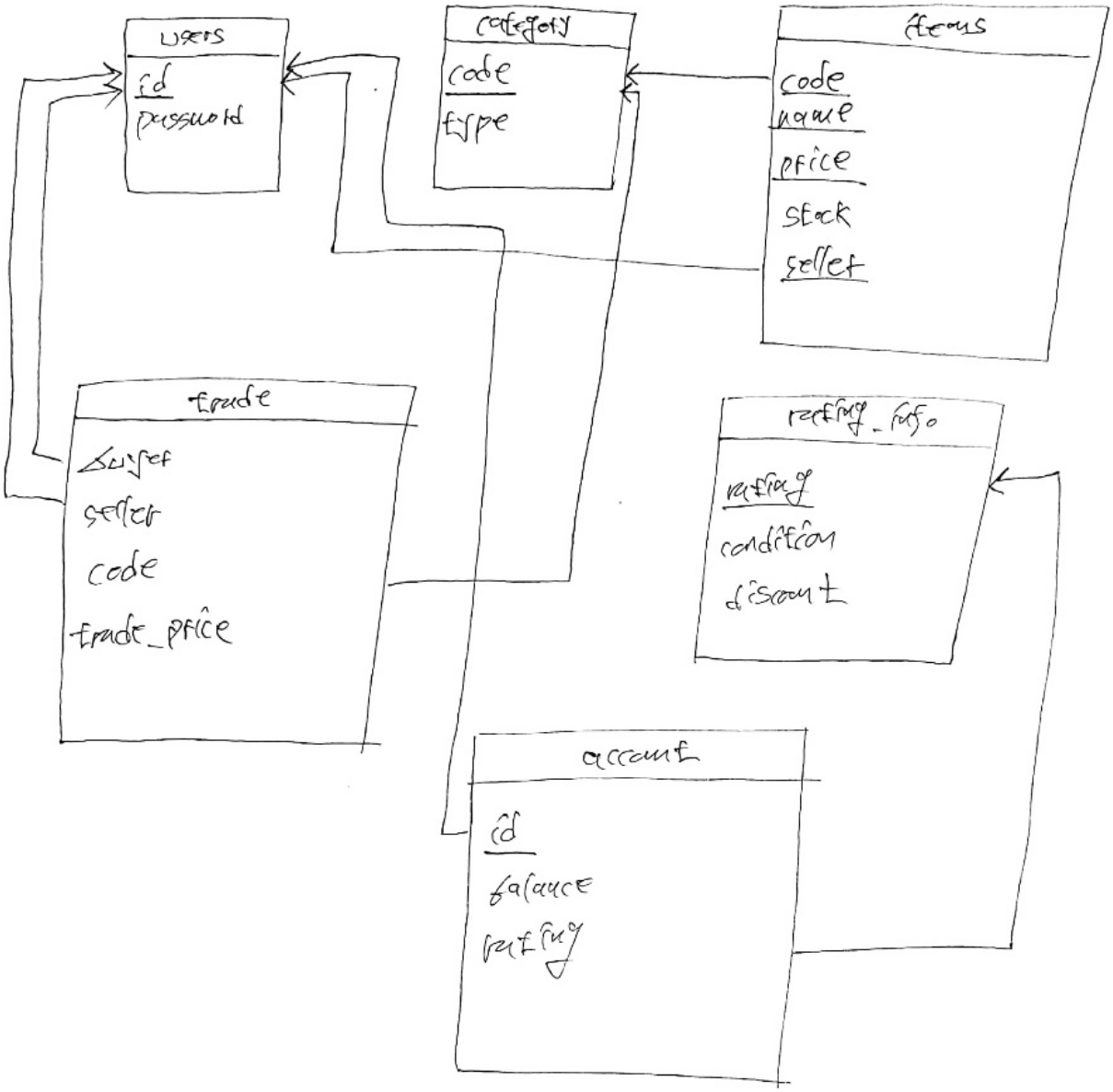
schema diagram



- `items` 테이블의 제약조건 변경

```
alter table items add constraint items_stock_check check(stock >= 0);
```

구현 함수와 SQL

main()

- 메인 페이지

users()

- admin의 경우 모든 user의 정보를 볼 수 있다.

```
select * from users
```

trades()

- admin의 경우 모든 trade의 정보를 볼 수 있다.

```
select * from trade
```

logout()

- 로그인한 유저는 로그아웃할 수 있다.

item_get()

- 아이템을 추가하고자 하는 경우 데이터 입력 폼을 받을 수 있다.

```
select * from category
```

item_add()

- 입력폼을 통해서 아이템을 추가할 수 있다.
- 존재하지 않는 code의 경우 예외가 발생한다.
- 요구사항대로 기존에 존재하는 item의 경우 재고만 더해주고 그게 아니라면 새로운 item을 추가한다.

```
select code from category
```

```

```

```
select * from items where code = '{}' and name = '{}' and price = '{}' and
seller = '{}'".format(code, name, price, id)
```

```
insert into items values ('{}', '{}', {}, {}, '{}')".format(code, name,
price, stock, id)
```

item_buy()

- item을 구매하고자 하는 유저는 구매 입력 폼을 받을 수 있다.

```
select balance, rating from account where id = '{}'".format(buyer)
```

item_buying()

- 입력한 수량에 따른 분기 처리를 한다.
- 재고 또는 잔고가 부족한 경우 예외 처리한다.
- confirm을 통해 구매 의사를 재확인한다.

```
select discount from account natural join rating_info where id
='{}'".format(buyer)
```

confirm()

- 구매 확정의 경우 관련된 로직을 수행한다.
- 잔고를 변경하고, 잔고에 따른 유저의 등급을 변경하고, trade 테이블에 관련된 데이터를 삽입하고, 재고를 변경하고 재고가 0인 경우 items 테이블에서 해당 row을 삭제한다.
- 위와 같은 로직을 위해 기존 items 테이블 제약조건을 변경한다.

```
alter table items add constraint items_stock_check check(stock >= 0)
```

```
update account set balance = balance + {} where id
='{}'".format(total_price, seller)
```

```
update account set balance = balance - {} where id
='{}'".format(final_price, buyer)
```

```
select * from rating_info
```

```
select balance from account where id ='{}'".format(buyer)
```

```
select balance from account where id ='{}'".format(seller)
```

```
update account set rating = '{}' where id ='{}'".format(info[0], buyer)
```

```
update account set rating = '{}' where id ='{}'".format(info[0], seller)
```

```
insert into trade values('{}', '{}', '{}', {})".format(buyer, seller,
code, total_price)
```

```
update items set stock = stock - {} where code ='{}' and name ='{}' and
price ={} and seller ='{}'".format(amount, code, name, price, seller)
```

```
delete from items where stock = 0
```

re_turn()

- 메인 페이지로 돌아간다.

cancel()

- 동작 과정 중 `cancel` 버튼이 있는 경우 로그인 성공 페이지로 다시 이동한다.
- 이를 구현하기 위해 `redirect`을 이용한다.

register()

- 리다이렉트된 경우를 위해 `GET`을 이용한 분기 처리를 한다.
- 회원가입이라면 회원가입 로직을 수행하고 로그인이라면 로그인 로직을 수행한다.
- 아이디 중복 여부, 아이디 존재 여부, 비밀번호 일치 여부 등을 검사한다.

```
SELECT id FROM users where id like '{}';".format(id)
```

```
INSERT INTO users VALUES('{}', '{}');".format(id, password)
```

```
INSERT INTO account VALUES('{}', '{}', '{}');".format(id, 10000,
'beginner')
```

```
SELECT password FROM users where id like '{}';".format(id)
```

loginSuccess(id)

- 로그인이 성공한 경우의 로직을 수행한다.
- `trade` 테이블이 비어있을 경우를 대비해 분기 처리를 한다.
- 거래 횟수가 가장 많은 카테고리를 고른다. 이때 여러 개의 카테고리가 있을 수 있으므로 리스트 형식으로 데이터를 `fetch`하고 각 카테고리 별 best seller, best buyer을 계산한다.

```
select max(count) from (select count(*) as count from trade group by code)
as result
```

```
select type from trade natural join category group by type having count(*)
= {};".format(maxCount[0])
```

```
select max(count) from (select count(*) as count from trade natural join
category where type = '{}' group by buyer) as
result".format(popular_category[0])
```

```
select buyer from trade natural join category where type = '{}' group by
buyer having count(*) = {}".format(popular_category[0], maxCount[0])
```

```
select max(count) from (select count(*) as count from trade natural join
category where type = '{}' group by seller) as
result".format(popular_category[0])
```

```
select seller from trade natural join category where type = '{}' group by
seller having count(*) = {}".format(popular_category[0], maxCount[0])
```

getItems(id)

- 로그인 이후 페이지에서 보여줄 아이템 목록을 가져온다.
- 본인이 등록한 아이템은 제외한다.

```
select * from items where seller <> '{}'".format(id)
```

getProfile(id)

- 로그인 이후 페이지에서 보여줄 개인 정보를 가져온다.

```
select id, balance, rating from account where id = '{}'".format(id)
```

추가로 구현한 기능

카테고리 추가

- `admin`은 카테고리를 추가할 수 있다.

category_get()

- 카테고리 입력 폼을 가져올 수 있다.

category_add()

- 카테고리를 추가할 수 있다.
- 이미 존재하는 code의 경우 예외 처리한다.

```
select * from category where code = '{}'".format(code)
```

```
insert into category values ('{}', '{}')".format(code, type)
```

탈퇴

- 로그인한 유저는 탈퇴를 할 수 있다.
- foreign key 제약조건이 있으므로 `users`의 `id`를 참조하고 있는 다른 테이블부터 삭제한다.

```
delete from items where seller = '{}'".format(id)
```

```
delete from account where id = '{}'".format(id)
```

```
delete from trade where buyer = '{}' or seller = '{}'".format(id, id)
```

```
delete from users where id = '{}'".format(id)
```

HTML 설명

add_fail.html

- 아이템 추가에 실패한 경우
- return 클릭시 입력 폼으로 다시 이동한다.

add_success.html

- 아이템 추가가 성공한 경우
- return 클릭 시 다시 로그인 이후 페이지로 이동한다.

buy_fail.html

- 아이템 구매가 실패한 경우
- return 클릭 시 다시 아이템 구매 입력 페이지로 이동한다.

buy_success.html

- 아이템 구매가 성공한 경우
- return 클릭 시 다시 로그인 이후 페이지로 이동한다.

category_add.html

- 카레고리 추가 입력 페이지

category_add_fail.html

- 카테고리 추가가 실패한 경우
- return 클릭 시 다시 카테고리 입력 페이지로 이동한다.

category_add_success.html

- 카테고리 추가가 성공한 경우

- `return` 클릭 시 다시 로그인 이후 페이지로 이동한다.

ID_collision.html

- 아이디가 중복되는 경우

item_add.html

- 아이템 추가 입력 페이지
- `cancel` 클릭 시 다시 로그인 이후 페이지로 이동한다.

item_buy.html

- 아이템 구매 입력 페이지
- `cancel` 클릭시 다시 로그인 이후 페이지로 이동한다.

item_buying.html

- 아이템 구매 확정 여부 페이지
- `cancel` 클릭 시 다시 로그인 이후 페이지로 이동한다.

login_fail.html

- 로그인에 실패한 경우
- `return` 클릭 시 메인 페이지로 이동한다.

login_success.html

- 로그인에 성공한 경우
- 거래 횟수가 높은 아이템, 등록된 아이템을 보여준다.
- 로그아웃, 탈퇴, 아이템 추가, 아이템 구매 링크를 제공한다.
- `admin`의 경우 유저 정보, 거래 정보, 카테고리 추가를 할 수 있는 링크를 제공한다.

main.html

- 메인 페이지

signup_success.html

- 회원가입이 성공한 경우
- `return` 클릭 시 메인 페이지로 이동한다.

trades.html

- 모든 거래 정보를 보여주는 페이지

users.html

- 모든 유저 정보를 보여주는 페이지