WIKI

이름: 문준영

학과: 컴퓨터소프트웨어학부

학번: 2022006135

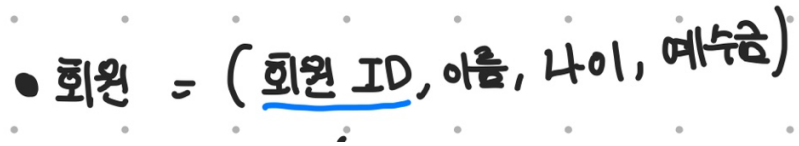
E-R model

- Abstract

Simple HTS의 구현을 위해 먼저 entity를 어떻게 정할 지 의문이었다. 우선적으로 가장 먼저 떠오른 entity로는 회원과 종목이었다. 주식이라는 것은 결국 회원이 주식을 서로 사고 팔고 하는 과정에서 변동이 일어나는 것이기 때문이다. 하지만 entity를 두 개만 두자니 회원과 종목 사이에 너무나 많은 relationship이 발생하게 되어 각 entity 또는 relationship에 넣을 attribute가 복잡해져 다른 방법을 생각해 보게 되었다. Conceptual design을 수행하는 과정에서 기능을 고려해 어떠한 entity와 relationship을 만들지가 가장 중요한 요소로 생각되었다. 때문에 다음과 같이 기능을 중점에 두고 entity와 relationship을 구현하게 되었다.

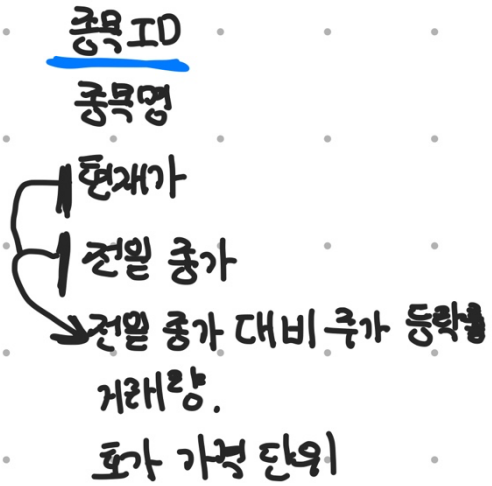
- E-R Model 설계 과정

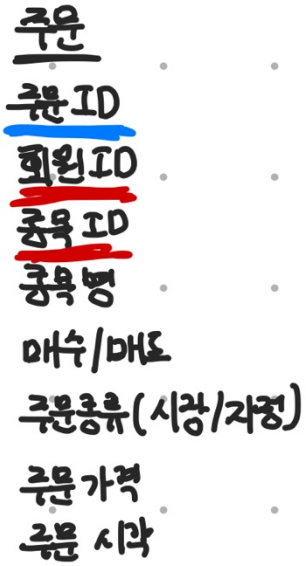
먼저, 본 과제에서 구현해야하는 가장 기본적인 것으로는 "내 정보"가 있다. 내 정보를 표현하기 위해 user(회원)라는 entity를 만들었다.



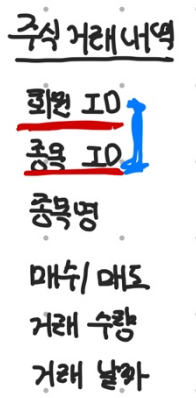
User 모두가 ID라는 primary key를 통해 구분되고 이들이 가진 예수금을 표시하여 주식 거래에서 사용가능한 돈을 확인 할 수 있도록 entity를 설계하였다. 입금을 하면 예수금이 추가되고 출금을 하면 예수금이 감소하는 기능을 생각하며 예수금 attribute를 추가하였다.

주식 정보의 확인을 위해 stock(종목)이라는 entity를 만들었다. 이 entity는 종목 ID를 primary key로 가지며 종목에 대한 정보를 제공한다. 현재가와 전일 종가를 확인하기 위해 해당 attribute를 추가하였다. 또한 전일 종가 대비 주가 등락률을 현재가와 전일 종가를 통해 후에 계산할 수 있다. (하지만 이는 derived attribute이기에 후에 구현 부분에서 고려될 것이므로 실제 그린 E-R diagram에는 포함시키지 않았다). 호가 가격 단위는 호가창에서 호가의 가격이 한 칸당 얼마의 가격 단위를 가지고 증가/감소 하는지를 나타내기 위해 attribute로 추가되었다. (예를 들어, 현대차는 500원 단위의 호가창이 형성되어 있고 삼성전자는 100원 단위의 호가창이 형성되어 있다.)

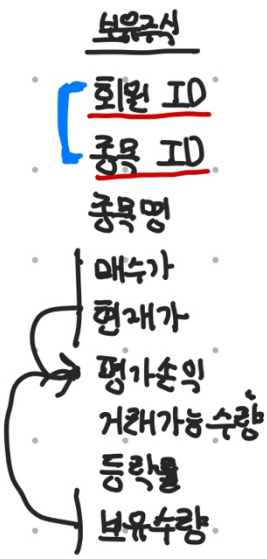




처음에는 종목과 회원사이의 relationship으로 "주문(order)"이라는 relationship을 만들려고 했지만 이를 하나의 entity로 만드는 것이 더 좋을 것 같아 이를 entity로 만들었다. 이 entity는 회원이 어떤 종목에 대해 주문을 넣는 것을 생각하며 만들었다. 먼저 order entity는 order ID를 primary key로 가지면서 user id(회원 id)와 stock id(종목 id)를 foreign key로 가진다. 매수인지 매도인지와 시장가로 거래를 할 것인지 아니면 지정가로 거래를 할 것인지는 1과 0으로 구분하도록 할 것이기에 매수/매도, 주문종류 attribute를 추가하였다. 주문 가격과 주문 시각을 알기 위해 해당 attribute도 추가하였다.



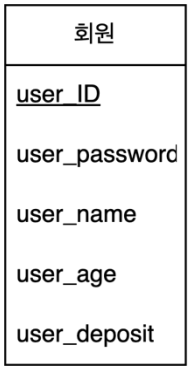
주식 거래 내역을 확인하기 위해 "stock_trading_log(주식 거래 내역)" entity를 추가하였다. 이는 문자 그대로 주식의 거래, 즉, 주문이 체결된 것들을 모아놓은 entity다. 후에 구현 시, 주문 체결이 일어나면 주식 거래 내역에 집어넣을 것이다. 여기서 정말 많은 고민을 했다. 주문과 주식 거래 내역을 주문 entity에 체결/미체결 이라는 attribute를 추가하여 두 entity를 하나로 합칠 지를 고민했다. 이렇게 하면 분명 redundancy는 줄일 수 있지만 후에 구현하고 실사용을 할 때 overhead가 매우 커질 것으로 예상되어 이 둘을 분리하였다.



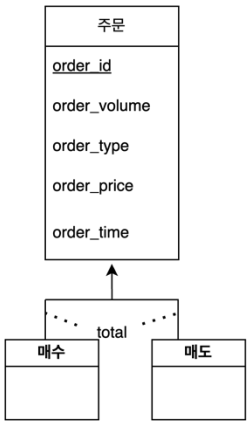
마지막 entity로는 "holding_stock(보유 주식)"이 있다. 구현할 기능 중 보유 주식과 관련된 기능이 많았기에 해당 entity를 만들었다. 하지만 해당 entity에는 많은 derived attribute가 있기에 실제 E-R model 구현에서는 제거 된 것이 많다. 이 entity는 user id(회원 id)와 stock id(종목 id)를 primary key이자 foreign key로 가진다. 때문에 E-R diagram에서는 weak entity set으로 변환될 것이다.

- E-R Diagram

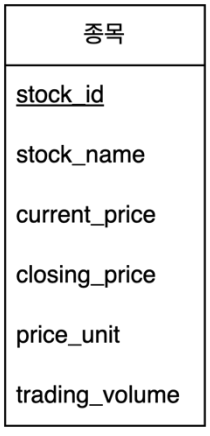
본격적으로 E-R diagram을 살펴보도록 하자. 위에서 설명한 entity를 기반으로 다양한 relationship으로 연결하였다. 설계 과정보다 새로운 attribute가 추가되기도 삭제되기도 했다.



첫 번째 entity는 user(회원) entity이다. 앞서 설계한 entity와 매우 유사하지만 회원 가입이라는 기능을 생각하여 user_password를 attribute로 추가하였다.

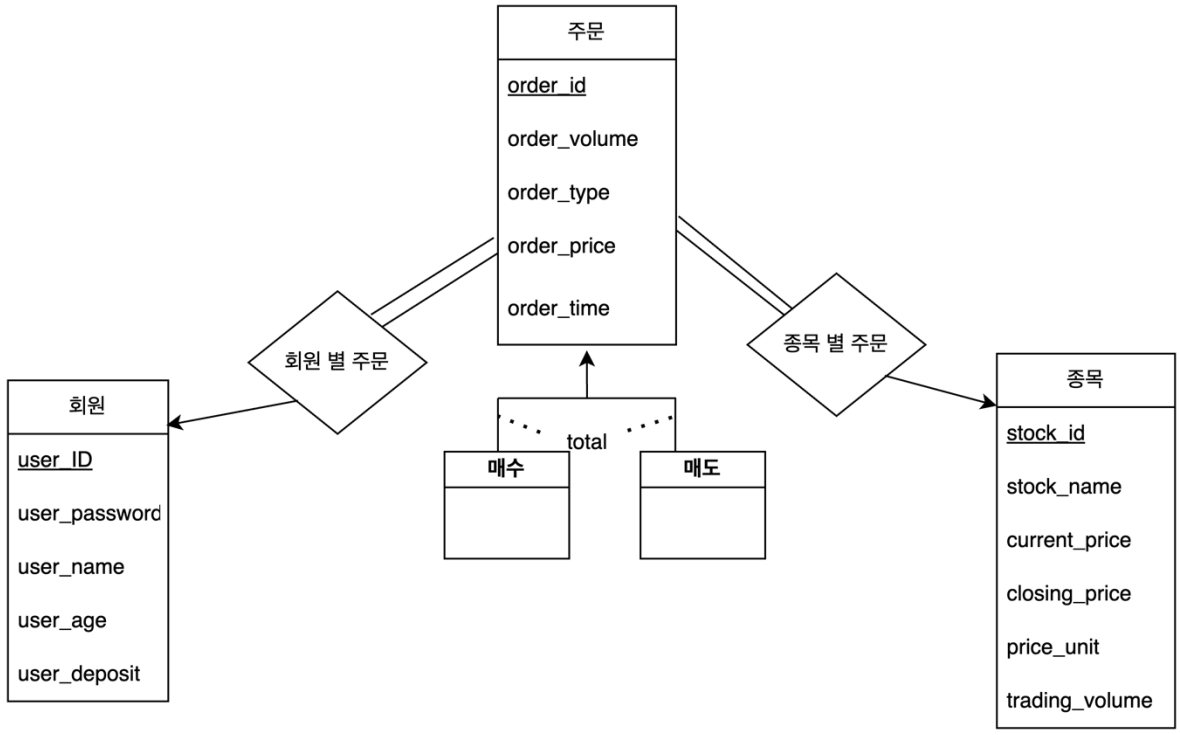


다음으로는 order(주문) entity이다. 앞서 설계한 entity에서 user_id와 stock_id는 redundancy를 제거하기 위해 삭제되었다. 또한 매수와 매도를 나누기 위해 order entity를 specialization 하였다. 후에 relational model에서 order relation을 생성하지 않고 매수, 매도 relation으로 order 전체를 표현 할 것이다.

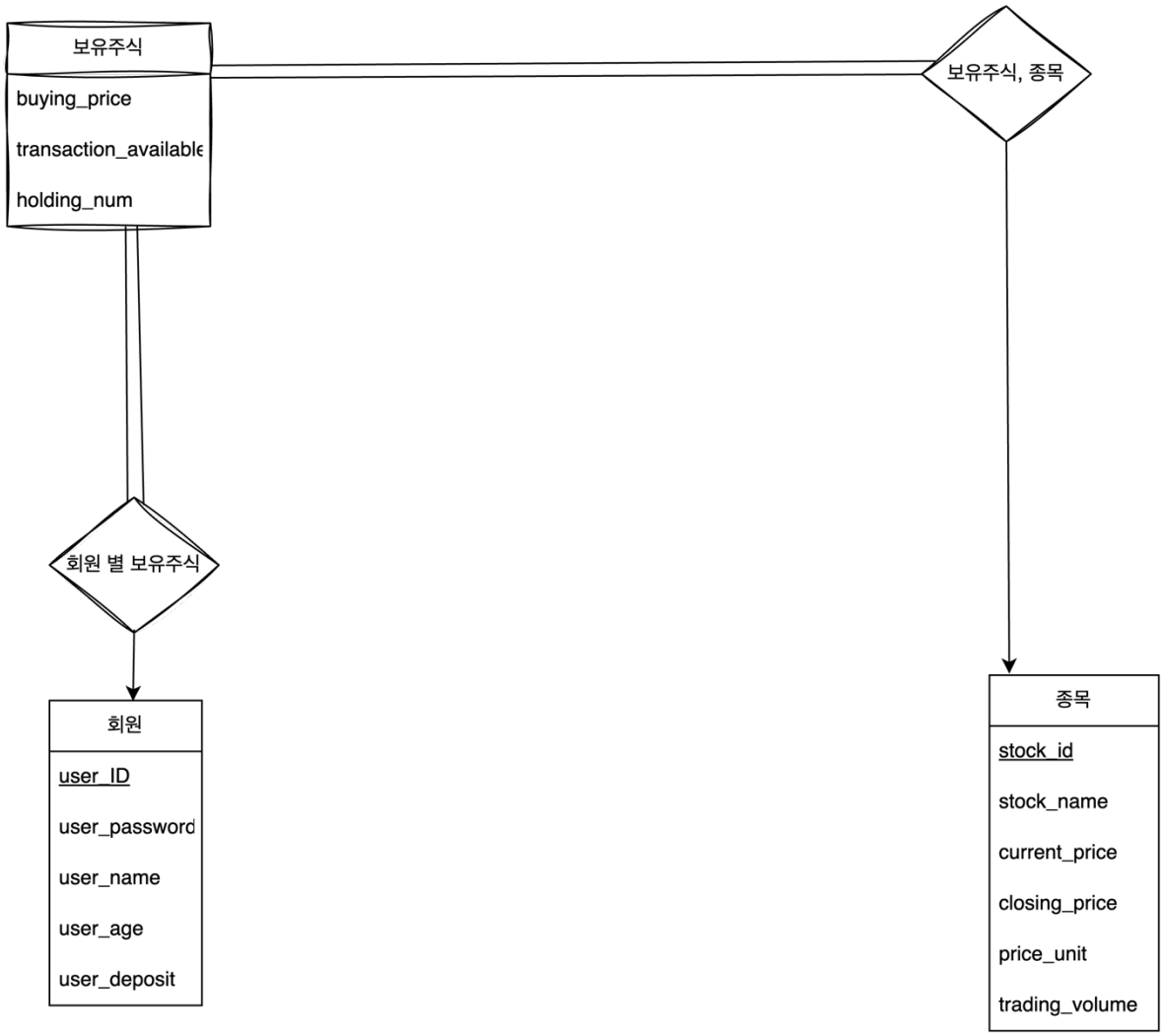


다음 entity는 stock(종목) entity이다. 앞서 설계한 종목 entity와 매우 유사하지만 전일 종가 대비 주가 등락률이 derived attribute이기에 삭제하였다. 후에 주식 통계를 위해 trading_volume(거래량) attribute를 추가하였다.

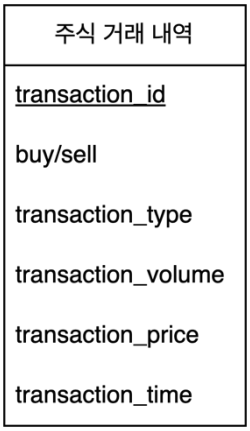
이것이 user, order, stock entities간의 relationship이다.



다음 entity로는 "holding_stock"(보유 주식)이 있다. 앞서 설계에서 먼저 평가 손익과 등락률은 결국 derived attribute에 속해 삭제하였다. 또한, 중복되는 stock entity와 중복되는 attribute인 stock_name 역시 삭제되었다. 이 entity는 primary key로 원래는 user_id와 stock_id를 가졌는데, 이 entity 자체가 user entity와 stock entity와 관계를 맺고 있기에 redundancy 제거를 위해 holding_stock entity를 weak entity set으로 만들었다.

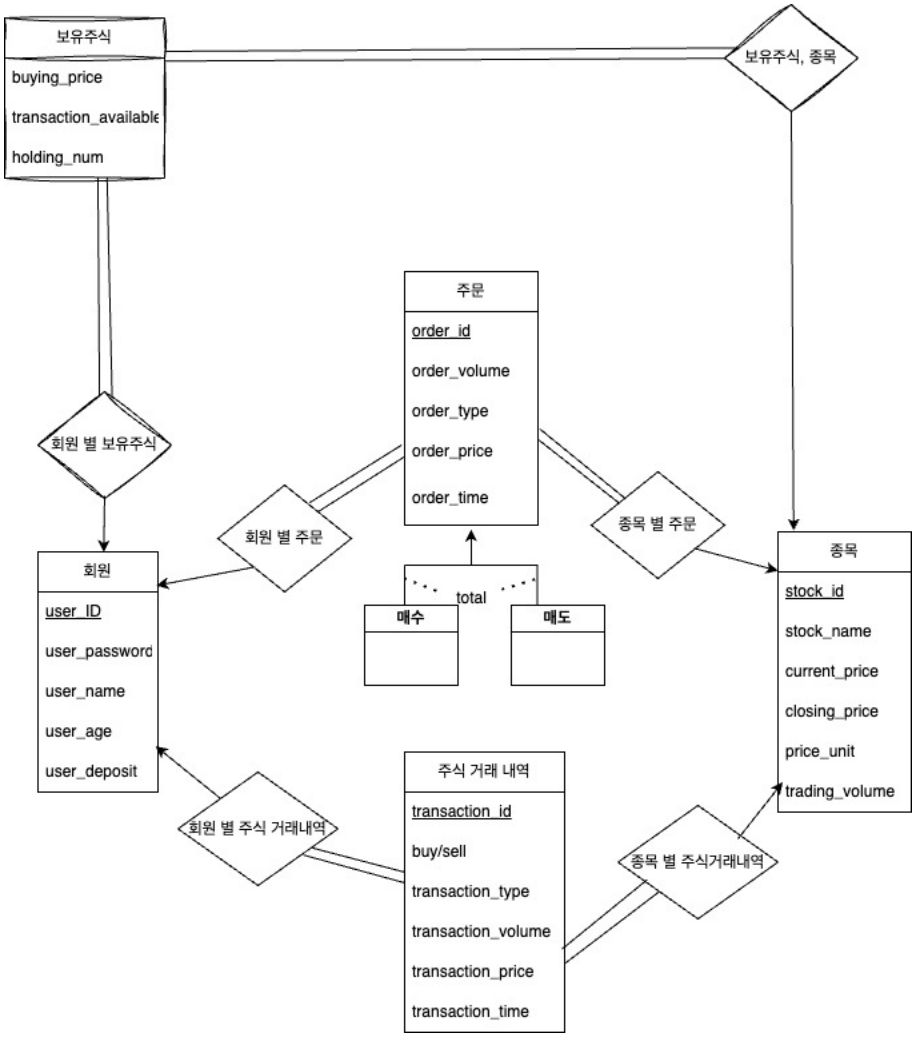


이것이 user, stock, holding_stock entities가 이루는 relationship이다. Holding_stock entity set은 user와 stock 모두를 identifying entity set으로 가지고 있다.



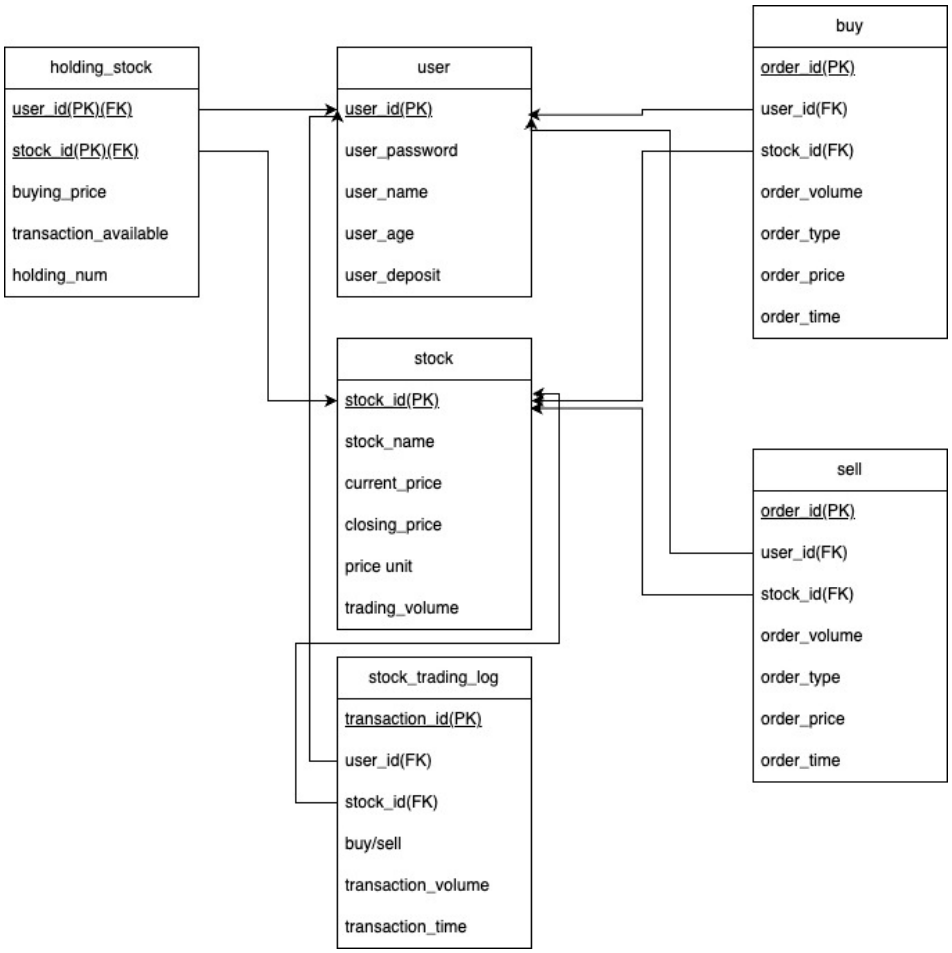
마지막 entity는 'stock_trading_log"(주식 거래 내역)이다. 이 역시 앞서 설계와 달리 user_id와 stock_id는 redundancy를 위해 제거하였다.

전체적인 E-R diagram은 다음과 같다.



- Relational Model

위의 E-R diagram을 relational model로 만들었다.



E-R diagram에서 모든 relationship이 one-to-many 관계를 이루며 many 쪽이 total relationship 이었기에 relationship을 따로 하나의 relation으로 만들지 않았다. E-R diagram에서의 primary key들은 PK로 표현하였고 밑줄을 그었다. 또한 foreign key(FK)는 참조하고 있는 relation에 연결해주었다.

- SQL 구현

```
CREATE TABLE user(
    user_id varchar(20) NOT NULL,
    user_name varchar(20) NOT NULL,
    user_password varchar(20) NOT NULL,
    user_age INT NOT NULL,
    user_deposit INT NOT NULL,
    PRIMARY KEY (user_id)
);
```

먼저 user relation을 sql로 구현하였다. User_id는 가변길이의 char형으로 설정하여 임의의 char string을 id로 설정할 수 있게 하였다. 또한, primary key로 user_id를 설정하여 user의 각각의 tuple을 구분할 수 있도록 하였다.

```
CREATE TABLE stock(
    stock_id INT AUTO_INCREMENT NOT NULL,
    stock_name varchar(20) NOT NULL,
    current_price INT,
    closing_price INT,
    price_unit INT,
    trading_volume INT NOT NULL,
    PRIMARY KEY (stock_id)
);
```

stock relation 역시 relational model에서 크게 달라진 점은 없다. 가장 두드러지는 부분은 auto_increment 부분이다. 이는 stock에 tuple이 하나 추가될 때마다 알아서 stock_id가 하나씩 증가하게 되는 것이다.

```
CREATE TABLE holding_stock(
    user_id varchar(20) NOT NULL,
    stock_id INT NOT NULL,
    buying_price INT NOT NULL,
    transaction_available INT NOT NULL,
    holding_num INT NOT NULL,
    PRIMARY KEY (user_id, stock_id),
    FOREIGN KEY (user_id) REFERENCES user(user_id) ON DELETE CASCADE,
    FOREIGN KEY (stock_id) REFERENCES stock(stock_id) ON DELETE CASCADE
);
```

Holding_stock relation 역시 relational model과 차이점이 크게 존재하지 않는다. 가장 두드러지는 점은 foreign key로 user_id와 stock_id를 활용하는데 만약 user가 사라지거나 stock이 사라질 때 (회원 탈퇴 / 주식 상장 폐지) 존재하는 holding_stock의 tuple 역시 삭제되게 "ON DELETE CASCADE" 옵션을 추가해주었다.

```
CREATE TABLE stock_trading_log(
    transaction_id INT AUTO_INCREMENT NOT NULL,
    user_id varchar(20) NOT NULL,
    stock_id INT NOT NULL,
    buy_sell INT NOT NULL,
    transaction_volume INT NOT NULL,
    transaction_time DATETIME NOT NULL,
    PRIMARY KEY (transaction_id),
    FOREIGN KEY (user_id) REFERENCES user(user_id) ON DELETE CASCADE,
    FOREIGN KEY (stock_id) REFERENCES stock(stock_id) ON DELETE CASCADE
);

CREATE TABLE sell(
    order_id INT AUTO_INCREMENT NOT NULL,
    user_id varchar(20) NOT NULL,
    stock_id INT NOT NULL,
    order_volume INT NOT NULL,
    order_type INT NOT NULL,
    order_price INT NOT NULL,
    order_time DATETIME NOT NULL,
    PRIMARY KEY (order_id),
    FOREIGN KEY (user_id) REFERENCES user(user_id) ON DELETE CASCADE,
    FOREIGN KEY (stock_id) REFERENCES stock(stock_id) ON DELETE CASCADE
);
CREATE TABLE buy(
    order_id INT AUTO_INCREMENT NOT NULL,
    user_id varchar(20) NOT NULL,
    stock_id INT NOT NULl,
    order_volume INT NOT NULL,
    order_type INT NOT NULL,
    order_price INT NOT NULL,
    order_time DATETIME NOT NULL,
    PRIMARY KEY (order_id),
    FOREIGN KEY (user_id) REFERENCES user(user_id) ON DELETE CASCADE,
    FOREIGN KEY (stock_id) REFERENCES stock(stock_id) ON DELETE CASCADE
);
```

Stock_trading_log, buy, sell relation은 모두 유사한 구조를 가졌다. 이 역시 "ON DELETE CASCADE" 옵션을 추가하여 회원 탈퇴나 주식 상장 폐지가 일어났을 때 거래 내역 삭제 및 주문 삭제를 하도록 하였다.

- 기능 별 구현 계획

지금까지 설계를 하였으니 각 기능별로 이를 어떻게 활용할 것인지에 대해 몇 가지 간단하게 알아보도록 하겠다.

1. 내 정보

예수금 표시: user relation의 deposit attribute를 이용

예수금 입금/출금: user relation의 deposit attribute를 변환하여 구현

보유 주식 현황: holding_stock relation에서 user_id를 통해 user가 보유하고 있는 주식을 확인

주식 거래 내역: stock_trading_log relation에서 user_id를 통해 user의 거래내역 확인

2. 주식 정보

종목 목록: stock relation을 활용

종목 검색 기능: stock relation을 활용

종목 별 정보: stock relation에서 stock_id와 stock_name을 이용해 구현

종목 별 거래내역: stock_trading_log에서 stock_id를 통해 종목 별 거래내역을 확인

종목 별 5단계 호가창: buy, sell relation을 통해 호가창 구현, stock relation의 price_unit attribute를 이용하여 호가창의 단위 결정

주식 통계: stock relation의 trading_volume을 이용하여 거래량 순위 통계 냄

3. 주식 거래

시장가 거래 기능: buy 또는 sell relation의 order_type relation을 0으로 설정하여 시장가 거래를 구현

지정가 거래 기능: buy 또는 sell relation의 order_type relation을 1로 설정하여 지정가 거래 구현