

# ass2\_1\_wiki\_곽주리\_2022056262

## 과제 목표

ER diagram & Relational schema를 설계한다.

db가 연결된 주식 Home Trading System을 만들기 위해 요구 사항을 분석하여 ER diagram을 설계한다. design한 ER diagram을 구현하기 위해 relational schema로 변환한다.

## Entity Set

db를 설계할 때 가장 중요하게 생각해야 하는 것은 data에 접근할 때 드는 overhead를 줄이는 방법이다. table에 큰 관계가 없는 data를 묶으면 한 data를 찾을 때 불필요한 data들을 다 읽어야 하기 때문에 하나의 entity set에는 최대한 같이 접근될 것 같은 속성들만 넣으려고 하였다.

실제 주식 거래 사이트가 돌아가는 방식을 기반으로 db를 구축하면 좋을 것 같아서 실제 증권 사이트 정보를 참고하였다.

draw.io라는 tool을 사용하여 ER diagram을 그렸다.

### userInfo

**userInfo에는 모든 사용자의 정보가 담겨 있다.**

사용자는 많은 곳에 사용될 것 같아서 최대한 간단한 구성을 가질 수 있도록 하였다. 로그인을 해야 본인 거래 내역, 포트폴리오 등 정보를 알 수 있기 때문에 id, password, state (login여부)를 넣었다.

원래 balance를 portfolio entity set에 넣어, 주식 계좌처럼 사용하려고 했으나 입출금할 때마다 불필요한 주식 data까지 읽어야 하기 때문에 overhead가 너무 커질 것 같아서 해당 속성을 userInfo에 넣었다.

또한 회원가입은 했으나 주식을 구매하기 전인 사용자가 예수금을 입금하면, portfolio에 balance를 제외한 속성이 다 null로 설정되어야 하기 때문에 balance를 userInfo에 넣었다.

한 사람은 최대 1개의 계좌만 가질 수 있다고 가정했기 때문에 balance를 포함한 모든 속성은 single value이다.

userInfo	
PK	<u>userID</u>
	password
	state
	balance

```
userInfo(userID, password, state, balance)
```

```
PK=userID
```

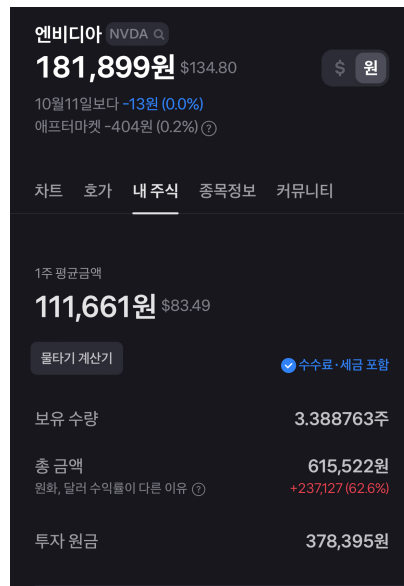
모든 사용자의 정보가 담겨 있기 때문에, 사용자ID로 사용자를 분간해야 한다. 사용자ID는 중복될 수 없다.

### Portfolio

**portfolio에는 모든 사용자가 보유하고 있는 모든 주식의 정보가 담겨 있다.**

이 entity set이 주식 계좌 역할을 할 것이다.

예수금은 userInfo에 저장되어 있지만 내 주식 정보는 portfolio에 저장되어 있다.



실제 증권사에서 보여주는 정보를 토대로 수정하여 schema를 정의하였다.

종목명, 보유 수량, 매수가, 평가손익, 등락률, 투자 원금, 총 금액...

추가로 거래 가능 수량을 portfolio에 저장하기로 결정했다. 보유 주식 현황을 보여주는 entity set에 가장 적합한 정보라고 생각했기 때문이다. 보유 수량 - 매도 수량으로 거래 가능 수량을 계산할 수 있는데, 호가창보다는 portfolio entity set이 더 적합하다.

현재가도 portfolio보다는 stock entity set에 들어가야 주식의 정보를 나타낸다는 취지에 맞다고 생각해, portfolio entity set에는 추가하지 않았다.

한 사용자가 한 주식에 대해 여러 번 거래했을 수도 있기 때문에 거래 내역을 모두 알고 있어야 한다. 그런데 portfolio의 primary key인 userID, stockID를 합치면 transaction entity set에서 해당 주식&사용자의 거래 내역을 다 찾아볼 수 있기 때문에 굳이 multi-value 속성을 포함할 필요는 없다. 특히 모든 거래 내역을 보유 주식 정보 화면에서 보여줄 필요가 없기 때문에 필요한 경우에만 userID와 stockID를 사용해서 transaction entity set에서 필터링해서 가져오면 될 것이다.

Portfolio(userID, stockID, 보유 수량, 매수가, 평가손익, 등락률, 거래 가능 수량)

한 사용자가 여러 개의 주식 종목을 소유할 수도 있기 때문에 userID만 primary key로 가질 수는 없다. 따라서 {userID, stockID}가 primary key이다.

PK={userID, stockID}

총 금액이나 투자 원금등은 화면상에 보여줄 때 빼고는 다른 entity set과 상호작용할 때 쓰일 일도 없고, 정의한 속성들에 의해 결정될 수 있기 때문에 포함하지 않았다.

Portfolio	
PK	<u>userID</u> <u>stockID</u>
	quantity avgPrice chgPercent evalPL tradQty

보유 수량: quantity

평균단가: avgPrice

등락률: chgPercent

평가손익: evalPL (evaluation profit/loss)

거래 가능 수량: tradQty

#### transaction

후후에 transaction이라는 이름이 DB에서 보장하는 최소한의 atomic한 일의 단위기 때문에 혼동을 야기할 수 있다 판단하여 transactionHistory로 변경하였습니다.

거래 내역을 저장하는 entity set이다. 거래 내역을 portfolio entity set에 넣는 대신 별도의 entity set을 정의한 이유는 다음과 같다. portfolio는 user의 통계를 내릴 때 사용되는 정보가 대다수이고 transaction은 말 그대로 주식들의 거래 내역이기 때문에 이 정보를 기반으로 주식 종목 통계를 내릴 수 있다.

또한 portfolio는 한 사용자와 한 주식에 대해 수행했던 모든 거래의 평균이 저장된다.

즉 portfolio와 transaction을 합친다면, entity set의 크기가 매우 커질 뿐더러 둘이 표현하고자 하는 내용이 다르기 때문에 주식 종목 통계를 내릴 때마다 불필요한 data들을 계속 필터링해야 한다는 단점이 있다.

이 entity set 역시 실제 증권사에서 보여주는 정보를 토대로 속성을 정의하였다.

종목명, 매수가/매도가, 매수/매도type, 거래 수량, 거래 날짜, 거래 시간, 주문유형...

총 구매 금액은 구매 수량, 매수가를 기본 속성으로 지정할 거라서 이 두 속성으로 계산할 수 있기 때문에 포함하지 않았다. 만약 총 구매 금액을 자주 사용한다면, 별도의 속성으로 관리하는 게 좋겠지만 거래 내역을 보여줄 때만 사용하고 다른 곳에는 사용하지 않을 것 같아서 추가하지 않았다.

orderBook entity set에 요청들어온 모든 주문의 내역이 저장된다.  
어떤 주문에 의해 체결된 거래인지 구분하기 위해서 orderID 속성을 추가하였다.

transaction(transactionID, userID, stockID, orderID, 매수가/매도가, 매수/매도, 거래 수량, 지정가/시장가, 거래 일시)

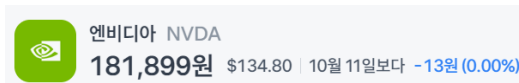
3

매수가/매도가: price  
 매수/매도: orderType  
 거래 수량: quantity  
 지정가/시장가: priceType  
 거래일시: time

## stock

**stock에는 모든 주식의 정보가 담겨 있다.**

usreInfo와 유사한 느낌으로 최대한 간결하게 구성하려고 하였다.



종목명, 현재가, 전일 종가 대비 등락률을 속성으로 설정했다.

주식의 종목명은 중복을 허용하지 않기 때문에 stockID를 primary key로 지정했다. 그림의 경우 stockID가 NVDA이다.

+전일 종가 대비 등락률을 알기 위해선 전일 종가를 알아야 한다. 따라서 장이 닫힐 때마다 종가를 기록하기 위해 종가 속성을 추가하였다.

R=(stockID, 현재가, 전일 종가, 전일 종가 대비 등락률)

PK=stockID

stock	
PK	<u>stockID</u>
	curPrice
	prevClose
	chgPercent

현재가: curPrice

전일 종가: prevClose

전일 종가 대비 등락률: chgPercent

## orderBook

**orderBook은 사용자가 요청한 주문 내역이 저장된다. (대기 상태, 체결 상태, 취소 상태 모두)**

5단계 호가창을 의미하는 entity set이다. transaction에 성사가 된 거래를 저장한다면, orderBook엔 요청이 들어왔던 모든 내역을 저장한다.

매도잔량	매도호가	매수호가	매수잔량
76,162	71,600		
149,766	71,500		
52,492	71,400		
103,980	71,300		
31,533	71,200		
		71,100	38,214
		71,000	135,915
		70,900	121,999
		70,800	97,175
		70,700	27,729
731,244	잔량합계		829,349

주문을 구분할 수 있는 orderID를 primary key로 지정한다. 그 외에 어떤 사용자가 어떤 주식을 거래 요청했는 지 알아야 하므로 userID, stockID도 필요하고, 그림에서 볼 수 있듯 주문 가격, 주문 수량도 알아야 한다. 추가사항으로 동일한 가격, 유형으로 거래 요청이 들어오면 주문 시간 순서대로 체결해야 하기 때문에 주문 날짜, 시간도 저장한다.

매수 잔량, 매도 잔량은 주문 수량을 다 더하면 되고, 화면상에 띄울 때를 제외하고는 사용되지 않을 것 같아서 속성에 추가하지 않았다.

orderBook(orderID, userID, stockID, 주문상태, 매수가/매도가, 매수 or 매도, 주문 수량, 지정가 or 시장가, 주문일시)

PK=orderId

schema만 보면 orderBook과 transaction은 완전히 동일한 entity set으로 볼 수도 있다.

하지만 거래일시와 주문일시는 차이가 있고, 주문을 요청한 수량이 전부 체결될 수 없다면, 가능한 수량만 먼저 체결하기 때문에 수량도 차이가 있고... 마찬가지로 시장가로 구매 요청 시 가격도 차이가 있을 수 있다.










나타내는 정보도 조금 다르고 data에도 차이가 있어서 별도의 entity set을 정의하는 게 올바르다.

orderBook	
PK	<u>orderId</u>
	stockID
	userID
	state
	price
	orderType
	quantity
	priceType
	time

## statistics

statistics에는 모든 주식에 대한 통계 data가 담겨 있다.

통계는 특정 기간 동안의 거래량, 특정 기간 동안의 거래대금, 전일 증가 대비 급상승, 급하락을 기준으로 순위를 나타낼 것이다. 통계 구분 종목은 실제 증권사를 참고하여 결정하였다.

	거래대금	거래량	인기	급상승	급하락
1		<b>YANG</b>			
		5,148원 +12.4%			
2		<b>라니 테라퓨틱스 홀딩스</b>			
		4,796원 +11.3%			
3		<b>리볼브</b>			
		37,828원 +8.5%			
4		<b>편웨어</b>			
		6,909원 +8.5%			
5		<b>뉴스케일파워</b>			
		20,391원 +7.9%			
6		<b>아메리카스 골드 앤 실버</b>			
		602원 +7.6%			
7		<b>MM텍</b>			
		659원 +7.4%			
8		<b>펠로톤</b>			
		7,858원 +7.4%			
9		<b>오클로</b>			
		14,565원 +7.0%			

PK는 stockID로 지정하였다.

이 entity set의 경우 모든 속성이 다른 entity set에 의해 결정된다.

거래량은 transaction에서 해당 stockID의 거래량을 다 합하면 되고, 거래대금은 transaction에서 해당 stockID의 거래량\*거래금액을 계산하면 된다. 주가 상승률은 stock entity set의 전일 종가 대비 등락률을 그대로 가져오면 된다.

이 entity set의 모든 속성은 single value이다.

PK=stockID

statistics(stockID, 거래량, 거래대금, 주가 등락률)

Statistics	
PK	stockID
	quantity
	tradPrice
	chg

추후에 다시 생각해보니, 통계장은 늘 실시간 정보를 띄워줘야 하기 때문에, 결국 통계장을 띄울 때마다 transaction, stock table을 다 훑어야 한다. 어차피 거래량, 거래대금, 주가 등락률 모두 join 없이 하나의 entity set에서 필터링만 걸어주면 되는 거라, 별도의 entity set에 data를 저장하는 게 overhead가 더 클 것이라 판단해서 최적화를 위해 구현할 때 해당 entity set은 제거하였다.

ER diagram은 실세계를 modeling하는 것이어서 통계치 자료가 존재한다는 사실을 나타내기 위해 entity set을 별도로 그려주는 게 적합하다고 판단하여 ER diagram에선 남겨두었다.

## Relationship set

각 entity set 사이의 관계를 표현할 수 있도록 relationship을 정의하였다.

실제로 두 entity set data 사이 유기성이 없고 일시적인 관계 혹은 간접적인 관계만 있다 해도 실세계를 modeling하는 거라서 relationship을 다 정의해주었다.

relational model로 변경할 때는 그런 relationship은 모두 제거될 것이다.

### holds (userInfo ↔ portfolio)

사용자가 보유한 주식 정보를 나타내기 위해서 이 관계가 필요하다. userInfo가 보유하고 있는 주식이기 때문에 relationship의 이름은 holds로 지었다.

한 사용자는 여러 개의 주식을 가질 수 있고, 한 사용자가 하나의 주식에 대해 갖고 있는 정보는 tuple 하나이다. stockID, userID로 해당 정보를 구분할 수 있다. portfolio에는 모든 사용자가 보유하고 있는 모든 주식의 정보가 담겨 있다. ⇒ **one-to-many**

모든 보유 주식 정보는 당연히 임의의 사용자가 갖고 있는 주식의 정보이기 때문에

⇒

**userInfo → portfolio는 total participation**이다.

사용자는 회원 가입은 했지만 아직 주식을 보유하지 않았을 수도 있기 때문에

⇒

**portfolio → userInfo는 partial participation**이다.

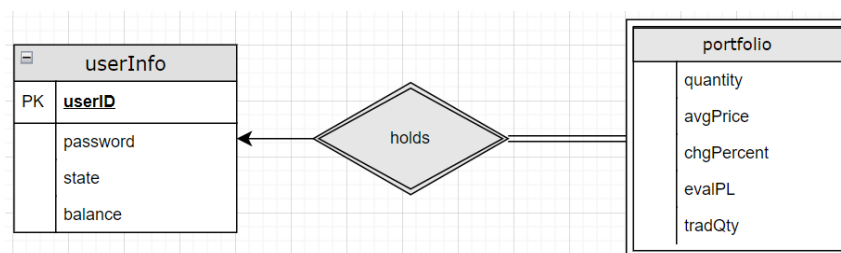
portfolio의 primary key 집합 중 하나의 요소인 userID는 userInfo의 primary key이기 때문에 data에 중복이 있다. 따라서 redundancy문제를 해결하기 위해 portfolio에 있는 userID를 삭제하였다.

이렇게 삭제하면 portfolio는 primary key를 자체적으로 가지지 못 해서

**weak entity set**이 된다.

따라서 **holds는 identifying relationship**이다.

portfolio의 PK 집합 중 나머지 하나 stockID가 없는 것도 위와 같은 이유이다.



### records (userInfo ↔ transaction)

사용자가 어떠한 주식에 대해서 거래했던 모든 내역이 저장된다. 실제로 사용자가 주문하는 건 orderBook이고, transaction에는 거래했던 내역이 기록되는 거라서 relationship 이름을 records로 설정하였다.

한 사용자는 여러 개의 거래 내역을 가질 수 있고, 하나의 거래 내역은 한 사용자에게 국한된다.

⇒

**one-to-many** 관계이다.

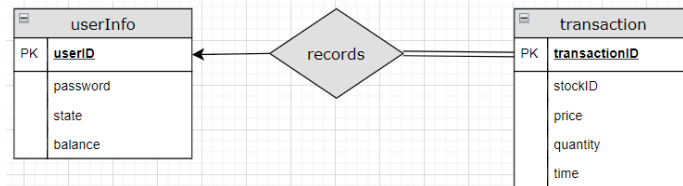
또한 모든 거래 내역은 임의의 사용자가 수행한 거래임으로

⇒

**transaction→userInfo**는 **total participation**이어야 한다.

사용자는 거래하지 않은 경우도 있으니 ⇒ **userInfo→transaction**은 **partial participation**.

transaction은 원래 속성으로 userID를 가지지만, userInfo와 관계를 갖고 있기 때문에 해당 data를 제거하였다. 하지만 userID를 제거해도, transaction은 자체적인 primary key를 가질 수 있기 때문에 weak entity set이 되진 않는다.



### orders (userInfo↔orderBook)

한 사용자가 거래가 체결되기 전에, 먼저 주문을 요청하는 상황을 표현하는 relationship이다.

transaction에 거래 내역이 가기 전에, 거래를 요청하고, 거래가 체결되면 해당 내역이 transaction으로 넘어간다. 주문을 요청하는 상황이라서 이름을 orders로 지정했다.

한 사용자는 여러 주문을 할 수 있고, 하나의 주문은 한 사용자에게만 해당하기 때문에

⇒

**one-to-many** 관계이다.

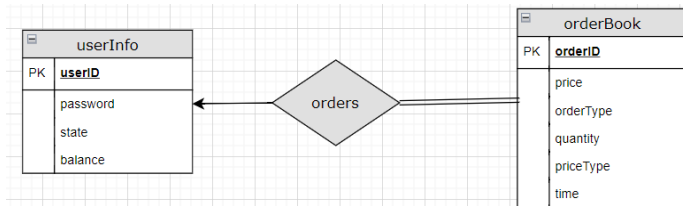
모든 주문은 무조건 임의의 사용자에게 의해 이뤄져야 하기 때문에

⇒

**orderBook→userInfo**는 **total participation**이다.

주문하지 않은 사용자가 있을 수 있으니 ⇒ **userInfo→orderBook**은 **partial participation**이다.

orderBook에 userID라는 속성이 포함되지만, 중복 문제를 방지하기 위해 orderBook에서 제거하였고, 이를 제거하여도 orderBook은 자체적인 primary key를 가져서 strong entity set이다.



### updates (portfolio↔transaction)

모든 거래 내역은 통합되어 portfolio에 저장된다. 즉 portfolio는 사용자가 보유한 주식의 정보를 나타내고 transaction은 사용자가 거래한 주식의 정보를 나타낸다. 한 사용자는 하나의 주식에 대해 여러 번의 거래를 체결할 수 있다. transaction 여러 개가 통합되어 portfolio에 반영된다. transaction에 tuple이 추가될 때마다 portfolio를 업데이트해주어야 해서 relationship 이름을 updates로 설정하였다.

하나의 portfolio는 여러 개의 transaction을 가질 수 있다. 다만 하나의 transaction은 오직 하나의 portfolio에만 반영되기 때문에

⇒

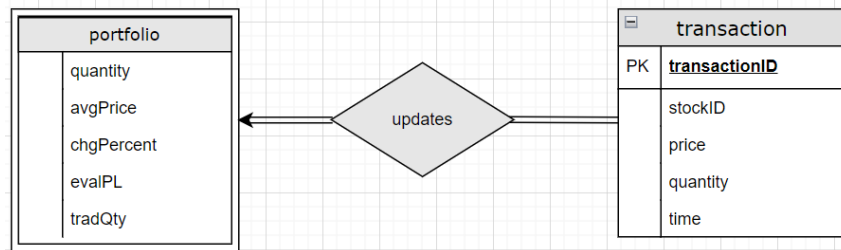
**one-to-many** 관계이다.

모든 portfolio는 transaction에 의해 만들어지고, 모든 transaction은 모든 portfolio에 반영되기 때문에 ⇒ **둘 다 total participation**이다.

둘 사이에 중복되는 data는 거의 없기 때문에 제거할 속성은 따로 없다.

transaction에서 userID, stockID가 모두 동일한 tuple을 모두 찾아서 평균을 낸 값이 portfolio에 반영되는 형식이기 때문에 한 사람이 한 주식에 대해 딱 한 번의 거래만 한 경우에만 data에 중복이 생긴다.

원래 stockID, userID가 중복인데, portfolio의 경우 userInfo, stock entity set에서 각각 userID, stockID가 제거되었고 transaction 역시 userInfo와의 관계에서 userID를 제거했기 때문에 이 둘 사이에 중복은 없을 것이다. (실제 구현 시엔 중복이 생기지만 ER digram 구성 시에는 안 생긴다.)



#### completes ( transaction ↔ orderBook )

주문이 체결되면 거래가 성사되는 상황을 그리는 relationship이다.

주문 대기, 취소 상태인 내역은 orderBook에 남아 있고, 거래가 체결되는 경우 transaction에 data가 넘어간다. 즉 transaction에 있는 모든 data는 orderBook에서 주문 상태가 complete인 data에서 넘어온 것이다. ⇒ **transaction → orderBook은 total participation.**

주문 상태가 대기거나 취소인 경우는 transaction에 저장 안 되기 때문에

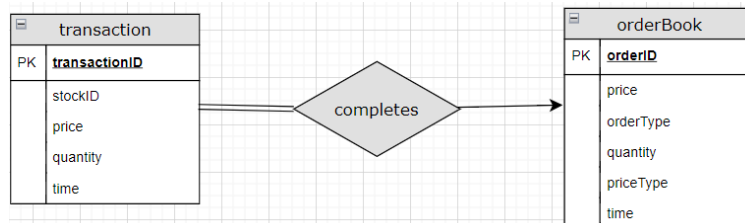
⇒

**orderBook → transaction은 partial participation.**

거래 일시, 주문 일시, 거래 수량 등에 차이가 있을 수 있기 때문에 data 자체가 동일한 것은 아니다. 100주를 주문했는데, 50주만 체결되는 경우등...

시장가로 거래할 때, 50주는 100만원, 50주는 98만원에 부분 거래될 수도 있다. 이 경우 order 하나에 두 개의 transaction이 나오기 때문에 **many-to-one** 관계를 갖는다.

orderId, orderType, priceType은 일치하기 때문에 제거하였고 quantity, price는 일전에 언급했던 상황에선 data가 다르기 때문에 유지하였다.



#### tracks ( portfolio ↔ stock )

사용자가 보유한 주식의 정보를 나타내기 위해선, 현재 주식의 정보를 알아야 한다. 즉 사용자가 보유한 주식과 주식 종목 사이의 관계를 나타내는 relationship이다. portfolio의 주식이 stock entity set의 주식 정보를 tracking한다는 의미로 tracks로 이름을 지었다.

portfolio의 primary key는 (stockID, userID)이다. stock entity set과 stockID가 중복되어서 해당 속성을 제거하였다. 제거하면서 portfolio는 자체적으로 primary key를 갖지 못 하게 되었기 때문에

**weak entity set**이 되었다. 따라서 **tracks relationship이 identifying relationship**이 된다. portfolio의 나머지 primary key인 userID는 user entity set과의 관계에 의해 제거된 상태이다.

사용자가 보유한 주식은 모두 stock entity set에 있는 주식들 중 하나여야 한다.

⇒

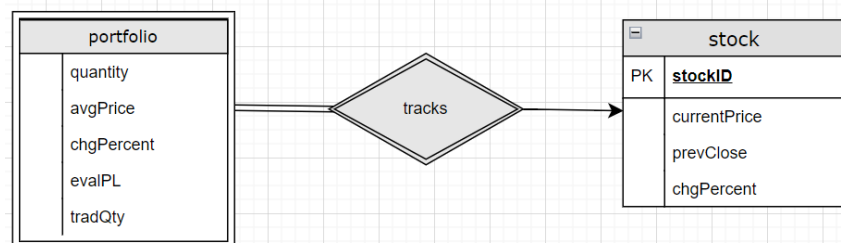
**portfolio → stock은 total participation.**

반면 사용자 중 그 누구도 보유하지 않은 주식이 있을 수도 있기 때문에

⇒

**stock → portfolio는 partial participation**이다.

또한 하나의 주식은 여러 사용자에 의해 보유될 수 있지만, 사용자id, stockid에 의해 정의된 주식 정보는 오직 하나의 주식에만 해당될 수 있기 때문에 **many-to-one** 관계이다.



#### requests ( orderBook ↔ stock )



사용자가 주식을 주문하면, 주식 거래소에서 사용자가 주문한 주식의 정보를 알고 있어야 한다.

현재가를 알아야 거래 체결 여부를 정할 수 있으므로... 따라서 그런 관계를 나타내는 relationship이다. 주문이 들어오면 주식 종목에 정보를 요청한다는 의미로 requests라는 이름을 지었다.

모든 주문은 주식 종목에 있는 주식들 중 하나로 들어와야 하기 때문에

⇒

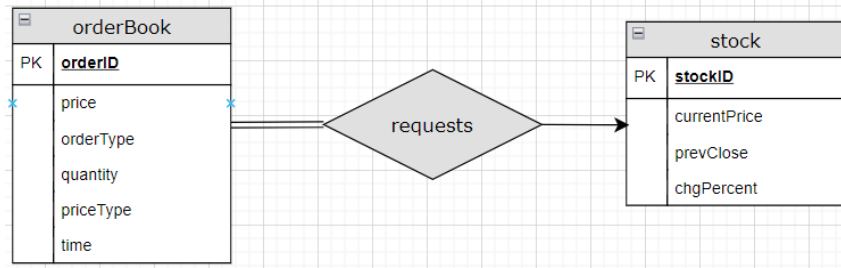
**orderBook→ stock은 total participation.**

주문이 안 들어온 주식도 있기 때문에⇒ **stock→orderBook은 partial participation**이다.

마찬가지로 하나의 주문은 하나의 주식에만 해당될 수 있고, 하나의 주식은 여러 개의 주문에 해당될 수 있기 때문에 **many-to-one** 관계이다.

orderBook과 stock entity set은 stockID라는 중복 속성이 있기 때문에 orderBook entity set에서 stockID를 제거하였고, 이 속성을 제거하여도 orderBook entity set은 자체적인 primary key를 가질 수 있기 때문에 여전히 strong entity set이다.

price는 지정가, 시장가에 따라 다르기 때문에 stock의 currentPrice와 동일하지 않아서 제거하지 않았다.



#### calculates (transaction ↔ statistics )

거래 내역을 기반으로 각 주식의 거래량을 계산하여 통계를 내린다. 따라서 relationship의 이름은 calculates로 지정하였다.

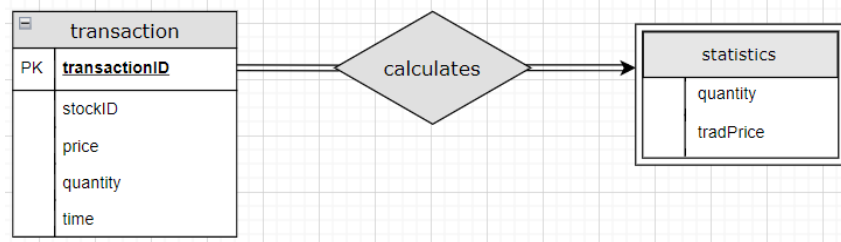
모든 거래 내역은 통계를 위해 쓰인다. 또한 모든 통계는 거래 내역에 의해 계산된다.

⇒

**둘 다 total 관계**이다.

하나의 주식에 대해 여러 개의 거래 내역이 있다면, 그 내역을 하나로 모아 계산하기 때문에 **many-to-one** 관계이다.

statistics는 stockID가 primary key인데 그 속성은 statistics와 stock 관계에 의해 제거된 상태이다. transaction과 statistics entity set 사이에는 사실 중복 data가 많이 존재하진 않는다. 하나의 주식에 대해 일정 기간 동안 딱 하나의 거래 내역만 존재한다면 거래 수량, 거래량이 중복되지만, 그 외의 경우엔 각기 다른 값을 갖는다.



#### derivedFrom (statistics ↔ stock)

통계창에서 보여주는 주가 상승률은 전일 종가 대비 등락률 기준으로 통계를 내릴 것이기 때문에 stock entity set의 전일 종가 대비 등락률을 그대로 가져오면 된다. data를 계산하지 않고 바로 갖고 오기 때문에 relationship 이름을 derivedFrom으로 지었다.

주식 종목에 있는 모든 주식은 통계 대상이고, 모든 통계는 주식 종목에 있는 주식으로부터 나온다.

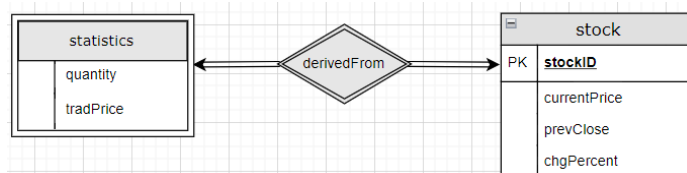
⇒

**둘 다 total participation**이다.

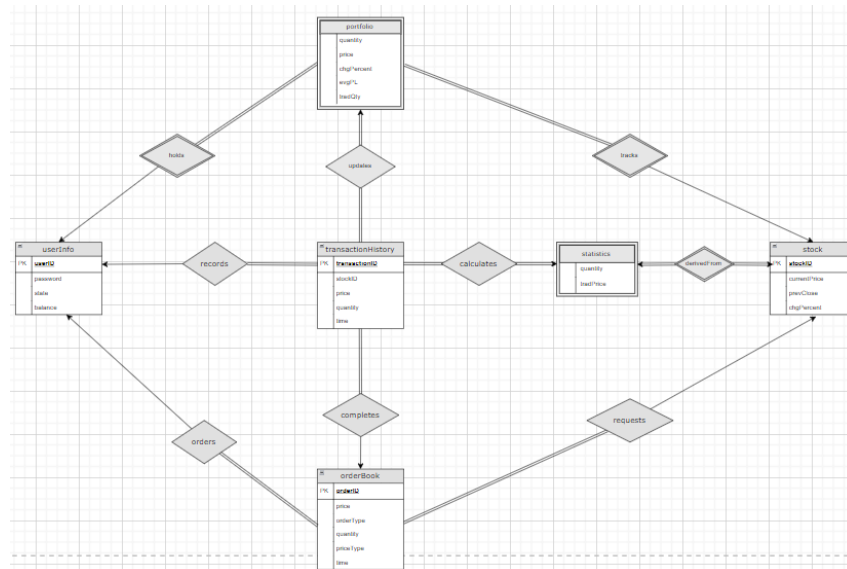
또한 하나의 주식은 하나의 종가 대비 등락률을 가지고, 하나의 종가 대비 등락률 역시 하나의 주식에만 해당되기 때문에 ⇒

**one-to-one** 관계이다.

stock entity set과 statistics는 동일한 stockID, 전일 종가 대비 등락률을 갖고 있고 data 역시 완전히 동일하기 때문에 statistics set에서 제거해주었다. 이 때 statistics의 primary key가 stockID이므로 statistics는 weak entity set, derive는 identifying relationship이 된다.



## 최종 ER diagram



주식 거래 사이트를 디자인한 ER diagram이다.

## Relational Schema

ER diagram을 relational schema로 바꿀 때, one-to-many관계이면서 many가 total인 경우, one-to-one 관계인 경우에는 relationship을 위한 별도의 table을 만들어줄 필요가 없다. (key를 옮기는 등의 추가 행위는 필요하지만)

ER diagram을 보면 모든 관계가 one-to-one이거나 one-to-many이자 many가 total이기 때문에, relationship을 위한 별도 table은 만들어줄 필요가 없다.

따라서 entity set을 위한 table만 구성하면 된다.

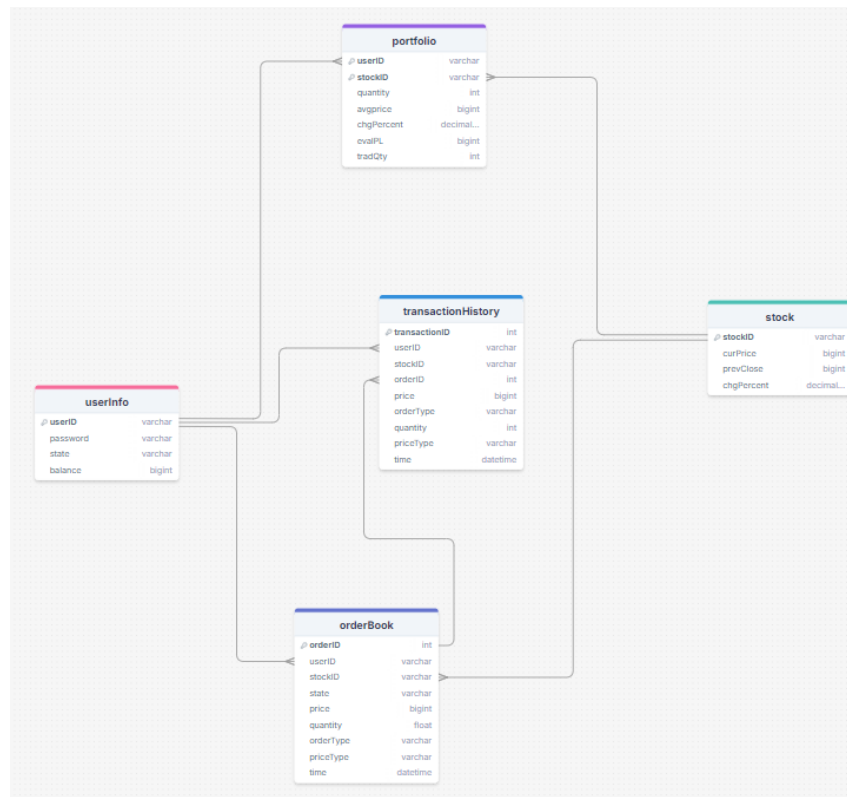
또한 모든 entity set에는 multi-value가 없기 때문에 multi-value를 위한 별도의 table 역시 구성할 필요가 없다.

완성된 relational schema diagram이다.

statistics entity set은 앞서 언급했던 것처럼 통계창을 띄울 때마다 계속해서 실시간으로 transaction, stock table을 훑어야 하기 때문에 별도의 entity set을 만들어서 계산하고 저장하는 게 효율적이다.

따라서 statistics entity set은 최적화를 위해 삭제하고, 필요할 때는 transaction, stock entity set에서 필요한 data만 골라 내어 보여주기로 하였다.

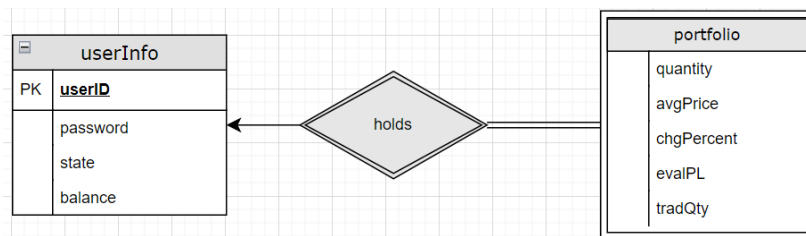
모든 relationship에서 one-to-many인 경우, one의 primary key를 many에 넣어주었다. 또한 weak entity set에는 identifying strong entity set의 primary key를 넣어주었다.



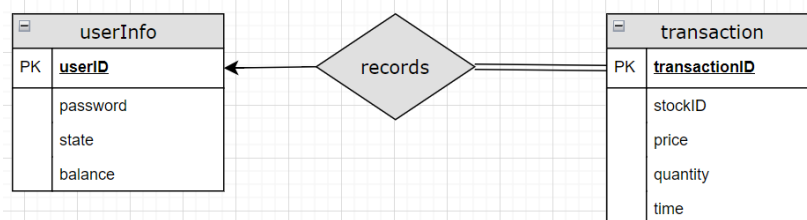
portfolio와 transaction 사이 관계가 없어 보이지만, 사실 userInfo, stock에 의해 다 연결되어 있기 때문에 유기성이 있음을 짐작할 수 있다.

- **userInfo**

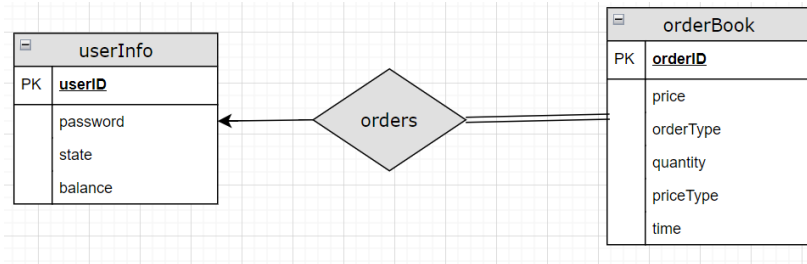
ER diagram에서의 기본 속성은 userID, password, state, balance



weak entity set 아니고 one-to-many 관계인데, userInfo가 one이라서 추가되는 속성이 없다.



weak entity set 아니고 one-to-many 관계인데, userInfo가 one이라서 추가되는 속성이 없다.



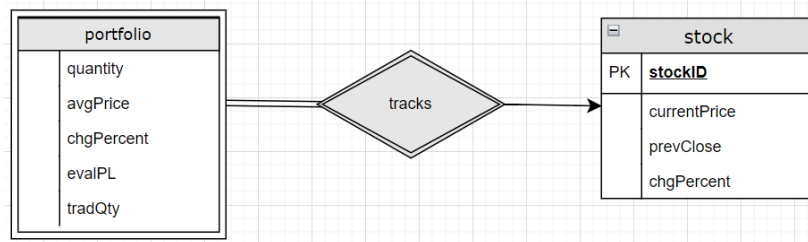
weak entity set 아니고 one-to-many 관계인데, userInfo가 one이어서 추가되는 속성이 없다.

$R=(userID, password, state, balance)$

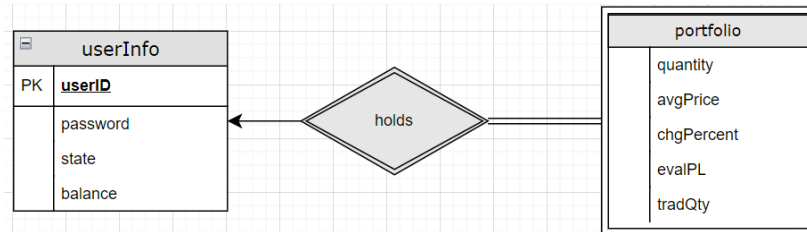
userID와 password는 사용자가 지정할 거고, 문자와 숫자가 섞여도 되기 때문에 자료형은 varchar,  
state는 on 혹은 off기 때문에 varchar,  
balance는 소수점 허용하지 않고 딱 떨어지게 잡을 거기 때문에 bigint  
primary key는 userID고, 모든 속성은 null이 되어선 안 된다.

#### • portfolio

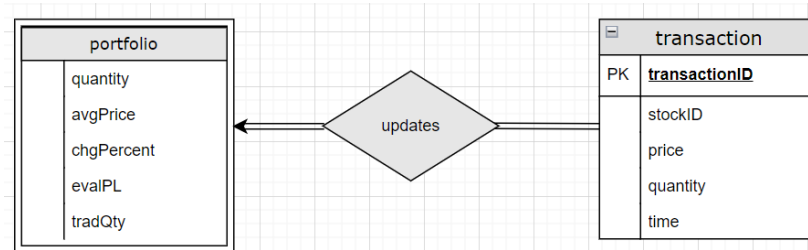
ER diagram에서의 기존 속성 quantity, price(avgPrice를 의미한다.), chgPercent, evalPL, tradQty



여기서 weak entity set, identifying strong entity set 관계이기 때문에 stockID 추가



여기서도 weak entity set, identifying strong entity set 관계이기 때문에 userID 추가



여기서는 weak entity set이지만 identifying strong entity set이 아니고, one-to-many 관계인데, portfolio가 one이기 때문에 portfolio relation에 추가되는 속성은 없다.

⇒  $R=(userID, stockID, quantity, avgPrice, chgPercent, evalPL, tradQty)$  이다

userID는 userInfo에서 갖고 오니까 varchar, stockID는 문자로 이루어지고 길이가 다 다르니까 varchar,  
quantity는 소수점 구매를 불가능하게 할 거라서 BIGINT,  
avgPrice는 돈은 모두 소수점 없이 딱 떨어지게 저장할 거라서 BIGINT,  
chgPercent는 소수점 밑으로 한 자리까지만 저장할 거고, 9999% 이상 급락하거나 급상승하지 않을 거라 판단하여 decimal(4,1)로 자료형을 설정하였다.

evalPL은 평가손익이라 BIGINT,  
tradQty 역시 소수점 구매 불가능하게 할 거라서 BIGINT로 설정하였다.

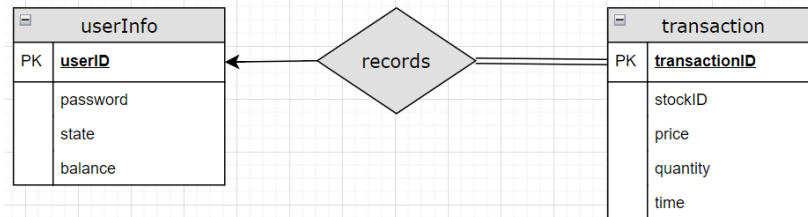
portfolio에는 당연히 사용자 명단에 있는 사람들, 주식 목록에 있는 주식들에 대한 정보만 담겨야 한다.

transaction에 있는 userID, stockID만 들어가야 하지만, transaction의 userID, stockID는 primary key가 아니기 때문에 application단에서 제약 조건을 걸어, 관리해야 한다.

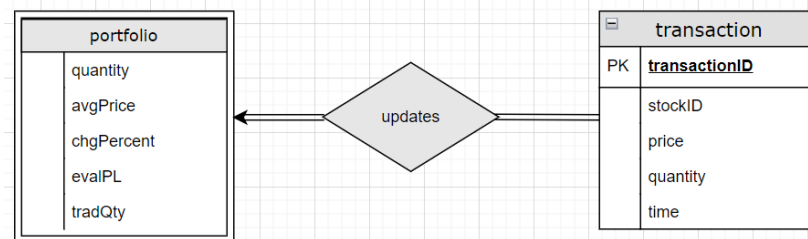
⇒ portfolio의 userID, stockID는 각각 userInfo, stock의 userID, stockID를 참조한다.

#### • transaction

ER diagram에서의 기존 속성은 transactionID, stockID, price, quantity, time이다.

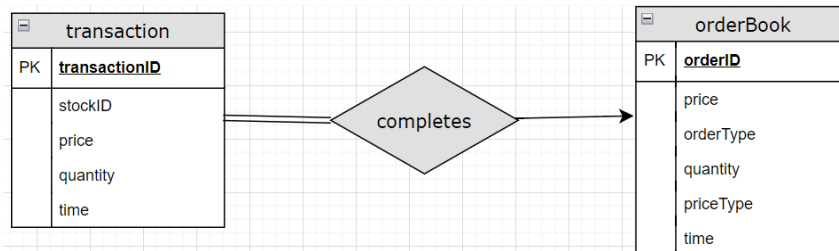


weak entity set 아니지만 one-to-many 관계이고, transaction이 many기 때문에 userID를 transaction에 추가한다.



weak entity set 아니지만 one-to-many 관계이고, transaction이 many기 때문에 portfolio의 primary key를 추가한다. portfolio의 primary key는 (stockID, userID)이다.

diagram에는 직관적으로 표현하기 위해 중복 참조를 허용했으나, 실제로 구현할 때는 portfolio가 userID를 이미 참조하고 있기 때문에



weak entity set 아니지만 one-to-many 관계이고, transaction이 many기 때문에 orderID를 transaction에 추가한다.

**transaction의 orderID는 orderBook의 orderID를 참조한다.**

원래 초기에 구상했던 transaction entity set의 속성에는 orderType과 priceType이 있었다.

그런데 relational schema로 변경하면서 생각해보니, 완벽하게 동일한 data라서 굳이 redundancy 만들지 않고, 필요할 때 orderBook에서 orderID로 matching해서 보는 게 overhead가 더 적을 것 같았다. 두 속성을 transaction에 넣으면 transaction에 접근할 때마다 너무 많은 column을 순회해야 하는데, 이 두 속성을 제거하고 필요할 때 orderBook에서 찾으면 overhead가 줄어든 것이다. 따라서 relational schema에는 transaction에 orderType과 priceType 속성이 없다.

같은 맥락으로 userID와 stockID도 불필요한 거 아닌가 생각할 수 있지만, userID와 stockID는 portfolio entity set과 소통할 때 중요하게 쓰이기 때문에 transaction내에 남겨두는 것이 적합하다. 예를 들어 A라는 사용자가 B 주식에 대해 거래했던 내역을 다 보고 싶다고 할 때, userID와 stockID만 있으면 transaction entity set에서 찾을 수 있는데, 해당 속성이 없다면 이 정보를 찾을 수 없다. 즉 userID와 stockID는 transaction과 관계를 맺는 entity set들의 primary key니까 제거해선 안 된다.

최종 속성 `R=(transactionID, userID, stockID, orderID, price, quantity, time)`

transactionID는 사용자가 지정하지 않고 임의로 정해주면 된다. unsigned int로 하고, 자동으로 증가하는 값을 갖도록 auto increment라는 조건을 추가해주면 된다.

userID, stockID, price, quantity는 모두 앞에서 언급했듯이 각각 BIGINT들로 설정한다.

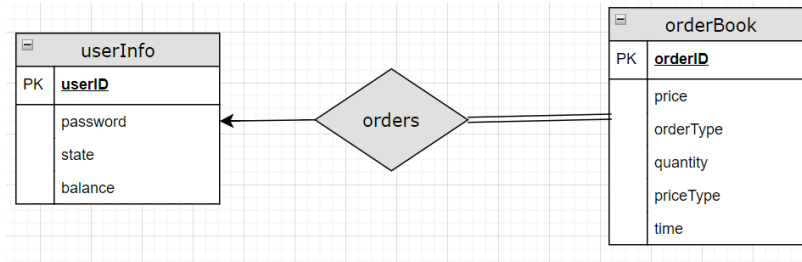
time은 날짜와 시간을 모두 적어야 하기 때문에 datetime이라는 자료형으로 지정한다.

transaction에 있는 userID, stockID는 orderBook에 있는 정보들만 담겨야 한다. 주문이 들어가고 체결이 된 거래만 저장해야 하기 때문이다. 허나 orderBook에선 해당 data들이 primary key가 아니기 때문에, 각각 userInfo, stock에 있는 primary key를 참조하게 한다. orderBook에 있는 data들로만 구성되었는지는 application단에서 검사해야 한다. 또한 transaction의 orderID는 당연히 orderBook에 있는 orderID로만 구성되어야 하기 때문에 orderBook의 orderID를 참조한다.

⇒ transaction의 userID, stockID, orderID는 각각 userInfo, stock, orderBook의 primary key를 참조한다.

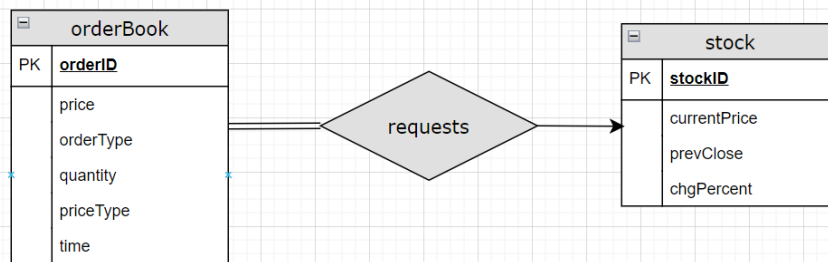
- orderBook

ER digram에서의 기본 속성은 orderID, price, orderType, quantity, priceType, time이다.



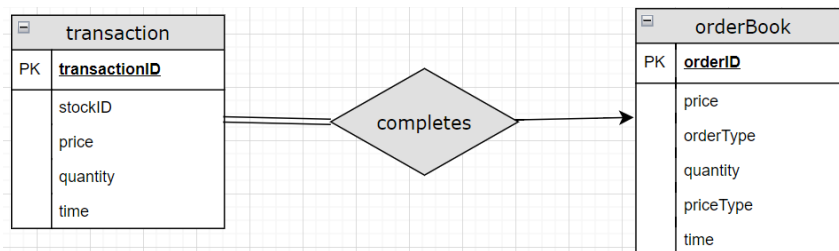
weak entity set은 아니지만 one-to-many 관계이고, orderBook이 many기 때문에 userInfo의 primary key를 추가한다. usreInfo의 primary key는 userID이다.

orderBook의 userID는 userInfo의 userID를 참조한다.



weak entity set 아니지만 one-to-many 관계이고, orderBook이 many기 때문에 stock의 primary key를 추가한다. stock의 primary key는 stockID이다.

orderBook의 stockID는 stock의 stockID를 참조한다.



weak entity set이 아니고, one-to-many 관계이지만 orderBook이 one이므로 추가할 속성이 없다.

따라서 최종 속성 R=(orderID, userID, stockID, price, orderType, quantity, priceType, time)

orderID, userID, stockID, price, quantity, time은 모두 앞에서 언급했던 방식으로 자료형을 지정해주면 된다.

orderType은 매수인지 매도인지를 의미하고, priceType은 지정가인지 시장가인지를 의미한다.

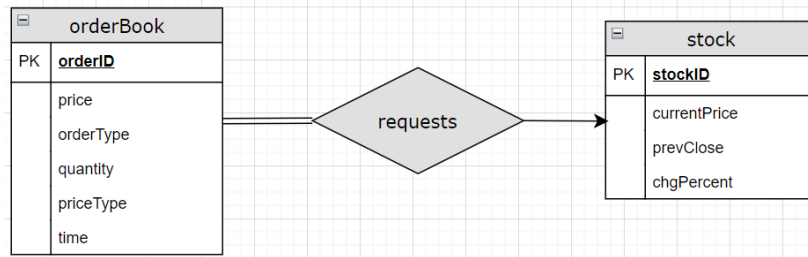
varchar로 정의하고, application 개발 시 enum을 사용해서 들어갈 수 있는 data들을 정하면 될 것이다.

orderBook에는 사용자 명단, 주식 목록에 있는 것들로만 저장될 수 있다. 당연히 회원가입한 사람만, 존재하는 주식에 대해서 주문을 할 수 있기 때문이다.

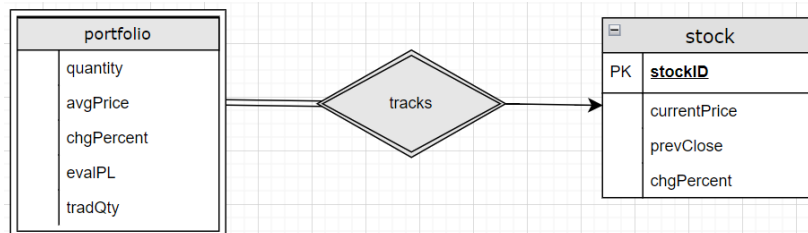
⇒ orderBook의 userID, stockID는 각각 userInfo, stock의 primary key를 참조한다.

- stock

ER diagram에서의 기본 속성은 stockID, currentPrice, prevClose, chgPercent이다.



weak entity set이 아니고, one-to-many 관계이지만 stock이 one이므로 추가할 속성이 없다.



weak entity set이 아니고, one-to-many 관계이지만 orderBook이 one이므로 추가할 속성이 없다.

따라서 최종 속성은 R=(stockID, currentPrice, prevClose, chgPercent)이다.

stockID, chgPercent는 앞에서 얘기했던 각각 varchar, decimal(4,1)이고

currentPrice와 prevClose는 모두 가격이므로 앞에서 얘기했듯이 둘 다 BIGINT 자료형이다.

## query 작성

과제 명세에 적혀 있듯이 database를 초기화하고, 특정 이름을 가진 database를 만들고, 해당 database에서 table들을 만들면 된다.

다음 코드를 query script 제일 초반에 추가하면 된다.

```

DROP DATABASE IF EXISTS DB_2022056262;
CREATE DATABASE DB_2022056262;
USE DB_2022056262;
  
```

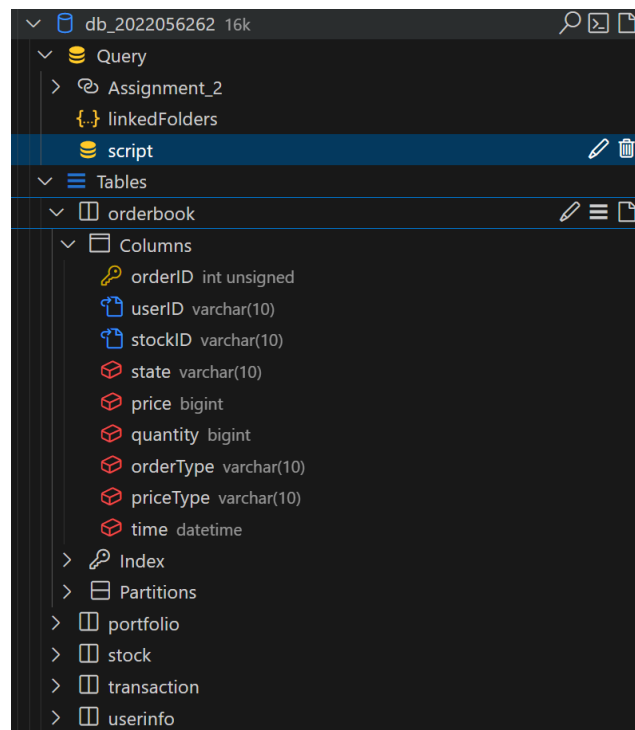
예시로 orderBook relation을 만드는 query만 갖고 왔다.

모든 속성은 공란일 수 없기 때문에 동일하게 NOT NULL 조건을 넣어주었다. orderBook의 primary는 orderID기 때문에 제약 조건으로 해당 사항도 넣어주었다. 또한 userID와 stockID는 각각 userInfo, stock relation의 primary key를 참조하는 것이기 때문에 제약조건으로 외래 키까지 설정해주었다.

```

CREATE TABLE `orderBook` (
  `orderID` INT UNSIGNED NOT NULL AUTO_INCREMENT,
  `userID` VARCHAR(10) NOT NULL,
  `stockID` VARCHAR(10) NOT NULL,
  `state` VARCHAR(10) NOT NULL,
  `price` BIGINT NOT NULL,
  `quantity` BIGINT NOT NULL,
  `orderType` VARCHAR(10) NOT NULL,
  `priceType` VARCHAR(10) NOT NULL,
  `time` DATETIME NOT NULL,
  PRIMARY KEY (`orderID`),
  FOREIGN KEY (`userID`) REFERENCES `userInfo` (`userID`),
  FOREIGN KEY (`stockID`) REFERENCES `stock` (`stockID`)
);
  
```

뒤 이어서 모든 relation을 만들어주고 실행시켜보면



이런 식으로 database, tables이 잘 생성된 것을 확인할 수 있다. orderBook table의 columns만 열어서 확인해보면, relational diagram에 그려진 모든 속성이 정상적으로 들어간 것을 확인할 수 있다.

primary key 역시 orderId로 설정되었고, userID와 stockID가 foreign key로 잘 설정되었다.