Entities

1. Users (userID, email (UNIQUE), userPassword, userBalance, userSignedUpAt)
    - Assumptions:
        - A user record represents a person using the app
        - `userEmail` must be unique so that no two users share the same login
        - `userBalance` is initialized with the amount the user deposits at sign-up
        - `userSignedUpAt` is automatically recorded at registration
        - A user's data persists even if they leave all groups
    - Why this is an entity:
        - `Users` are the main actors in the system and interact with multiple parts of the app (trading, groups, watchlists)
        - They exist independently of other entities and are referenced by many relationships
2. Groups (groupID, groupName, groupBalance):
    - Assumptions:
        - A group record represents a trading group mutually formed by users
        - A user can make multiple groups
        - `groupName` need not be unique
        - `groupBalance` is initialized with the collective balances of all users' individual balances
        - `groupCreatedAt` is automatically recorded when a group is made
        - `groupCreatedBy` references the user who created the group
    - Why this is an entity:
        - Groups exist independently of users and can include multiple users
        - Storing them separately avoids repeating group information for every user and supports group features like shared portfolios or discussions
3. Tickers (symbol, name, assetType):
    - Assumptions:
        - A ticker record represents a tradable symbol (e.g., "AAPL", "BTC", etc.)
        - `symbol` uniquely identifies each ticker
        - `assetType` classifies the ticker into common types (e.g., stocks, crypto, etc.)
    - Why this is an entity:

        - Tickers are referenced across transactions, news, and price data
        - Keeping them as their own entity avoids redundancy and provides a single source of truth for all tradable instruments

4. Price-Bars (symbol, time, open, high, low, close, volume, source):
    - Assumptions:
        - A price bar record represents market data for a specific ticker at a specific time
        - `symbol` and `time` combined uniquely identifies a record
        - `source` shows whether data is real or simulated
        - Market prices change frequently, leading to many records per ticker
    - Why this is an entity:
        - Price data forms a repeating time series for each ticker
        - Storing it separately allows the system to efficiently manage large amounts of market data without duplicating ticker information
5. News-Articles (articleID, publishedAt, source, title, URL, sentiment, impactTags):
    - Assumptions:
        - A news article record represents an news article
        - `source`, `title`, and `URL` describe where the news article came from and what it is about
        - `sentiment` and `impactTags` are optional fields describing potential market effects
        - A single news article can mention multiple tickers
    - Why this is an entity:
        - News articles have their own metadata and aren't tied to a single ticker or user
        - Keeping them separate allows one news article to be linked to many tickers and easily reused or analyzed
6. Transactions (transactionID, accountID, ticker, time, side, quantity, price, kind, status, requestedBy, approvedBy):
    - Assumptions:
        - A transaction record represents a single trade event
        - `transactionID` uniquely identifies each transaction
        - `accountID` links the trade to the account the trade originated from
        - Each trade involves one ticker and is requested by one user, as denoted with `requestedBy`
        - Some transactions may require approval from another user, as denoted with `approvedBy`
        - Transactions cannot be edited after its creation; in other words, they're treated as permanent records
    - Why this is an entity:

        - Transactions are the core log of trading activity
        - Since multiple accounts and users can each have many trades, storing them as their own entity allows for accurate tracking, auditing, and analysis

Relationships

1. Users ↔ Groups via Group-Memberships(role, joinedAt)
    - Cardinality: M:N (optional participation)
        - Users can belong to many groups
        - Groups can have many users
2. Users ↔ Groups via Created-By(createdAt, createdBy)
    - Cardinality: 1:M
        - Each group is created by exactly one user
        - A user can create many groups
3. Users ↔ Tickers via User-Watchlist
    - Cardinality: M:N (optional participation)
        - Each user can follow many tickers
        - Each ticker can be followed by many users
4. Users ↔ News-Articles via Users-News-Feed
    - Cardinality: M:N (optional participation)
        - Tracks which articles a user has seen/read
5. Users ↔ Transactions via User-Transactions-Requests(requestedBy)
    - Cardinality: 1:M
        - Each transaction request involves exactly one user
        - A user can request many transactions
6. Users ↔ Transactions via User-Transactions-Approvals(approvedBy)
    - Cardinality: 0..1:M
        - Each transaction can have either zero or one approver
        - Each user can approve many transactions
7. Groups ↔ Transactions via Groups-Transactions
    - Cardinality: 1:M
        - Each transaction involves exactly one group
        - A group can have many transactions
8. Tickers ↔ Transactions via Involves
    - Cardinality: 1:M
        - Each transaction involves exactly one ticker
        - A ticker can appear in many transactions
9. Tickers ↔ Price-Bars via HasPriceData
    - Cardinality: 1:M
        - Each ticker has many price bars

10. Tickers ↔ News-Articles via Mentions
    - Cardinality: M:N (optional participation)
        - An article can mention many tickers
        - A ticker can appear in many articles

Normalizing

Fortunately enough, our schema is already in 3NF, seeing as how all non-key attributes depend on the whole key and there are no transitive dependencies

Relational Schema

1. Users(userID:INT [PK], userEmail:VARCHAR(255) UNIQUE, userPassword:VARCHAR(255), userBalance:DECIMAL(12,2), userSignedUpAt:DATETIME)
2. Grps(groupID:INT [PK], groupName:VARCHAR(255), groupBalance:DECIMAL(12,2), groupCreatedAt:DATETIME, groupCreatedBy:INT [FK to Users.userID])
3. Group-Memberships(userID: INT [FK to Users.userID],groupID: INT [FK to Groups.groupID], role: VARCHAR(50), joinedAt: DATETIME, PRIMARY KEY (userID, groupID))
4. Tickers(symbol: VARCHAR(10) [PK], name: VARCHAR(255), assetType: VARCHAR(50))
5. Price-Bars(ticker: VARCHAR(10) [FK to Tickers.symbol], time: DATETIME, open: DECIMAL(12,2), high: DECIMAL(12,2), low: DECIMAL(12,2), close: DECIMAL(12,2), volume: BIGINT, source: VARCHAR(20), PRIMARY KEY (ticker, time))
6. News-Articles(articleID: INT [PK], publishedAt: DATETIME, source: VARCHAR(100), title: VARCHAR(255), URL: VARCHAR(500), sentiment: VARCHAR(20), impactTags: VARCHAR(255))
7. Users-News-Feed(userID: INT [FK to Users.userID], articleID: INT [FK to News-Articles.articleID], seenAt: DATETIME, isRead: BOOLEAN, PRIMARY KEY (userID, articleID))
8. User-Watchlist(userID: INT [FK to Users.userID], ticker: VARCHAR(10) [FK to Tickers.symbol], addedAt: DATETIME, PRIMARY KEY (userID, ticker))
9. Transactions(transactionID: INT [PK], accountID: INT, groupID: INT [FK to Groups.groupID], ticker: VARCHAR(10) [FK to Tickers.symbol], time: DATETIME, side: VARCHAR(10), quantity: DECIMAL(12,4), price: DECIMAL(12,2), kind: VARCHAR(20), status: VARCHAR(20), requestedBy: INT [FK to Users.userID], approvedBy: INT [FK to Users.userID])