1. **Submission located within the doc folder**

See doc/BurgerCryptoUML.pdf

2. **Having a complete and correct ER/UML diagram**

See doc/BurgerCryptoUML.pdf

3. **The ER/UML diagram has 5 or more entities**

See doc/BurgerCryptoUML.pdf

4. **Database Schema is normalized and the process of normalization is clearly stated**

Comparing 3NF and BCNF, we finally choose to make our tables satisfy BCNF as much as possible. BCNF can greatly reduce data redundancy and is helpful to make data strictly matched and reliable. Although this project is for course use so far, we take it seriously and follow the best practices. And such redundancy reduction can save storage perfectly.
Also, we evaluate the shortcomings of using BCNF. One of them is the low query performance caused by over-optimization. However, after going through our design, there is no such need. All the tables so far seem decent and good at functioning.
****

User Table
- Primary Key: UserID
- Fields: UserName, Password, Email, JoinTime, Balance, BurgerCoins
- Analysis:
  - All fields are atomic, satisfying 1NF.
  - All non-primary key fields (UserName, Password, Email, JoinTime, Balance, BurgerCoins) are directly dependent on the primary key UserID, satisfying 2NF.
  - No non-primary key fields depend on other non-primary key fields, satisfying 3NF.

BalanceHistory Table
- Primary Key: BalanceHisID
- Foreign Key: UserID references User.UserID
- Analysis:
  - All fields are atomic, satisfying 1NF.
  - All non-primary key fields (UserID, Date, Amount) are directly dependent on the primary key BalanceHisID, satisfying 2NF.
  - No non-primary key fields depend on other non-primary key fields, satisfying 3NF.

EducationalContent Table
- Primary Key: ContentID
- Foreign Key: UserID references User.UserID
- Analysis:
  - All fields are atomic, satisfying 1NF.
  - All non-primary key fields (UserID, Title, ThumbsUpNum, Description, Content) are directly dependent on the primary key ContentID, satisfying 2NF.
  - No non-primary key fields depend on other non-primary key fields, satisfying 3NF.

InvestmentAdvice Table
- Primary Key: AdviceID
- Foreign Key: PortfolioID references Portfolio.PortfolioID

- Analysis:
    - All fields are atomic, satisfying 1NF.
    - All non-primary key fields (PortfolioID, Content, Time) are directly dependent on the primary key AdviceID, satisfying 2NF.
    - No non-primary key fields depend on other non-primary key fields, satisfying 3NF.

Portfolio Table

- Primary Key: PortfolioID
- Foreign Key: UserID references User.UserID
- Analysis:
    - The table has only two fields, with UserID as a foreign key directly dependent on the primary key PortfolioID, satisfying 1NF, 2NF, and 3NF.

CryptoAsset Table

- Primary Key: AssetID
- Analysis:
    - All fields are atomic, satisfying 1NF.
    - All non-primary key fields (CryptoName, SymbolURL) are directly dependent on the primary key AssetID, satisfying 2NF.
    - No non-primary key fields depend on other non-primary key fields, satisfying 3NF.

Trade Table

- Primary Key: TradeID
- Foreign Keys: PortfolioID references Portfolio.PortfolioID, AssetID references CryptoAsset.AssetID
- Analysis:
    - All fields are atomic, satisfying 1NF.
    - All non-primary key fields (PortfolioID, AssetID, TradeType, Quantity, TradePrice, TradeTime) are directly dependent on the primary key TradeID, satisfying 2NF.
    - No non-primary key fields depend on other non-primary key fields, satisfying 3NF.

PriceHistory Table

- Primary Key: PriceHisID
- Foreign Key: AssetID references CryptoAsset.AssetID
- Analysis:
    - All fields are atomic, satisfying 1NF.
    - All non-primary key fields (AssetID, Time, Price) are directly dependent on the primary key PriceHisID, satisfying 2NF.
    - No non-primary key fields depend on other non-primary key fields, satisfying 3NF.

PortfolioCryptoAsset Table

- Composite Primary Key: PortfolioID, AssetID
- Analysis:
    - All fields are atomic, satisfying 1NF.
    - All non-primary key fields (Quantity, Value) are directly dependent on the composite primary key (PortfolioID, AssetID) with no partial or transitive dependencies, satisfying 2NF and 3NF.

Non-primary key fields in each table are directly dependent on the primary key without any observed transitive dependencies.

User, BalanceHistory, EducationalContent, InvestmentAdvice, CryptoAsset, Trade, PriceHistory Tables

Each non-primary key attribute is directly dependent on the primary key. In these tables, the primary keys are superkeys, meeting the BCNF requirement.

Portfolio Table
This table also has simple dependencies, where UserID as a foreign key depends on the primary key PortfolioID. Since there are no non-primary key attributes determining other attributes, this table also satisfies BCNF.

PortfolioCryptoAsset Table
This table uses a composite primary key consisting of PortfolioID and AssetID. All non-primary key fields (Quantity, Value) are directly dependent on this composite primary key. There are no non-primary key attributes determining other attributes, so this table meets BCNF as well.

Based on the analysis, we can conclude that our database design appears to satisfy BCNF.

5. Having a complete and correctly translated relational schema

- User(UserID: INT [PK], UserName: VARCHAR(255), Password: VARCHAR(255), Email: VARCHAR(255), JoinTime: DATETIME, Balance: DECIMAL, BurgerCoins: INT)
- BalanceHistory(BalanceHisID: INT [PK], UserID: INT [FK to User.UserID], Date: DATETIME, Amount: DECIMAL)
- EducationalContent(ContentID: INT [PK], UserID: INT [FK to User.UserID], Title: VARCHAR(255), ThumbsUpNum: INT, Description: TEXT, Content: TEXT)
- InvestmentAdvice(AdviceID: INT [PK], PortfolioID: INT [FK to Portfolio.PortfolioID], Content: TEXT, Time: DATETIME)
- Portfolio(PortfolioID: INT [PK], UserID: INT [FK to User.UserID])
- CryptoAsset(AssetID: INT [PK], CryptoName: VARCHAR(255), SymbolURL: VARCHAR(255))
- Trade(TradeID: INT [PK], PortfolioID: INT [FK to Portfolio.PortfolioID], AssetID: INT [FK to CryptoAsset.AssetID], TradeType: VARCHAR(50), Quantity: DECIMAL, TradePrice: DECIMAL, TradeTime: DATETIME)
- PriceHistory(PriceHisID: INT [PK], AssetID: INT [FK to CryptoAsset.AssetID], Time: DATETIME, Price: DECIMAL)
- PortfolioCryptoAsset(PortfolioID: INT [FK to Portfolio.PortfolioID], AssetID: INT [FK to CryptoAsset.AssetID], Quantity: DECIMAL, Value: DECIMAL, [Composite PK: PortfolioID, AssetID])

6. Assumptions of the ER/UML diagram

- **User:** Represents the platform's users, including both investors and content creators. The inclusion of balance and BurgerCoins directly within the User entity simplifies financial transactions and rewards mechanisms, avoiding the need for a separate Wallet entity. This assumes that each user's financial interactions can be directly managed through their profile for efficiency.
- **BalanceHistory:** Tracks the historical changes in a user's balance, linked to the User directly. This entity assumes the importance of auditing and tracking financial transactions over time for transparency and user insight.
- **EducationalContent:** Encapsulates educational materials provided by users. Linking directly to the User entity assumes that every piece of content can be traced back to its creator, fostering

accountability and enabling features like content rating (ThumbsUpNum) and detailed descriptions for user engagement.

- **InvestmentAdvice**: Contains tailored investment advice tied to specific portfolios. This assumes a personalized approach to investment guidance, where advice is directly relevant to the configurations of a user's investment portfolio, ensuring customized support.
- **Portfolio:** Represents a collection of investments held by a user. The direct link to User underscores the assumption that each portfolio is unique to a user, facilitating personalized investment tracking and management.
- **CryptoAsset**: Defines the various cryptocurrencies available for trading. This entity's standalone nature assumes the need for a central repository of crypto assets that all users can access, trade, and analyze, separate from individual user actions or portfolios.
- **Trade:** Records transactions made within a portfolio. By associating trades with both Portfolio and CryptoAsset, it's assumed that tracking the specifics of each trade (type, quantity, price, and time) is crucial for investment strategy and performance review.
- **PriceHistory:** Keeps a record of the price changes for each crypto asset over time. This entity assumes the critical nature of historical price data for investment analysis and decision-making, independent of individual user actions.
- **PortfolioCryptoAsset:** A junction table that models the many-to-many relationship between Portfolios and CryptoAssets, detailing the quantity and value of each asset within a portfolio. This design assumes the need to manage and evaluate individual asset holdings within the broader context of a user's investment strategy, allowing for detailed tracking and valuation of assets over time.

7. A description of each relationship and its cardinality

- A User may have 0 to many BalanceHistory while a BalanceHistory belongs to one and only User.
- An Educational Content can be posted by exactly 1 User.
- A Portfolio may have 0 to many InvestmentAdvice while an InvestmentAdvice belongs to one and only Portfolio.
- A portfolio may hold 0 to many assets and one asset can be held by many portfolios.
- A crypto asset may have many price history records.
- A Portfolio may record 0-many Trades while a Trade belongs to one and only Portfolio.
- A user may have 0 to many Portfolio while a Portfolio belongs to one and only User.

8. Create a release with the correct tag for your submission and submit it on PrairieLearn

See Stage.2: https://github.com/cs411-alawini/sp24-cs411-team068-BurgerMaker/tree/stage.2

9. Fix all suggestions for the previous stages

See in Proposal.md.