Preliminary Report

Integrity Constraints

Primary Key Constraints

- Customer Profile has a unique username, therefore it has a Primary key constraint on the username attribute field
- Market Account has a unique Market Id, therefore it has a Primary key constraint on the mid attribute field
- Stock Account has a unique Stock Id, therefore it has a Primary key constraint on the sid attribute field
- Stock has a unique three-letter Symbol, therefore it has a Primary key constraint on the Symbol attribute field
- Actor/Director has a unique id, therefore it has a Primary key constraint on the id attribute field
    - Since the Actor and Director entities are an ISA of the Actor/Director entity, they both also share id as a Primary Key
- Transactions has a Transaction ID, therefore it has a Primary key constraint on the tid attribute field
    - Since the Accrue-Interest, Deposit, Withdraw, Can_Canel, and Cancel entities are an ISA of the Transaction entity, they both also share tid as a Primary Key
        - Since the Buy and Sell entities are an ISA of the Can_Cancel entity, they both also share tid as a Primary Key
- Movie has a unique title and production year, therefore it have Primary key constraints on the title and prod year attribute field

Relational Key Constraints

- Stock Account has username and as a Not Null Foreign Key through the many to one relationship between the two entities, since every Customer Profile can own multiple Stock Accounts. It also has a symbol as a Not Null Foreign Key because for every stock that is bought or traded a new Stock Account is created.
- Market Account shares username as a Not Null Foreign Key through a one to one relationship, since every Customer Profile can only own 1 Market Account. Conversely Customer Profile also has mid as a Not Null Foreign Key through a one to one relationship.
- Stock and Actor/Director have a one-to-one relationship, therefore symbol is a Not Null Foreign Key in Actor/Director and id is a Not Null Foreign Key in Stock
- There is a many to many relationship between Actor and Director entities with Movies as there exists at least 1 actor/director for the movie and they can be the same person.

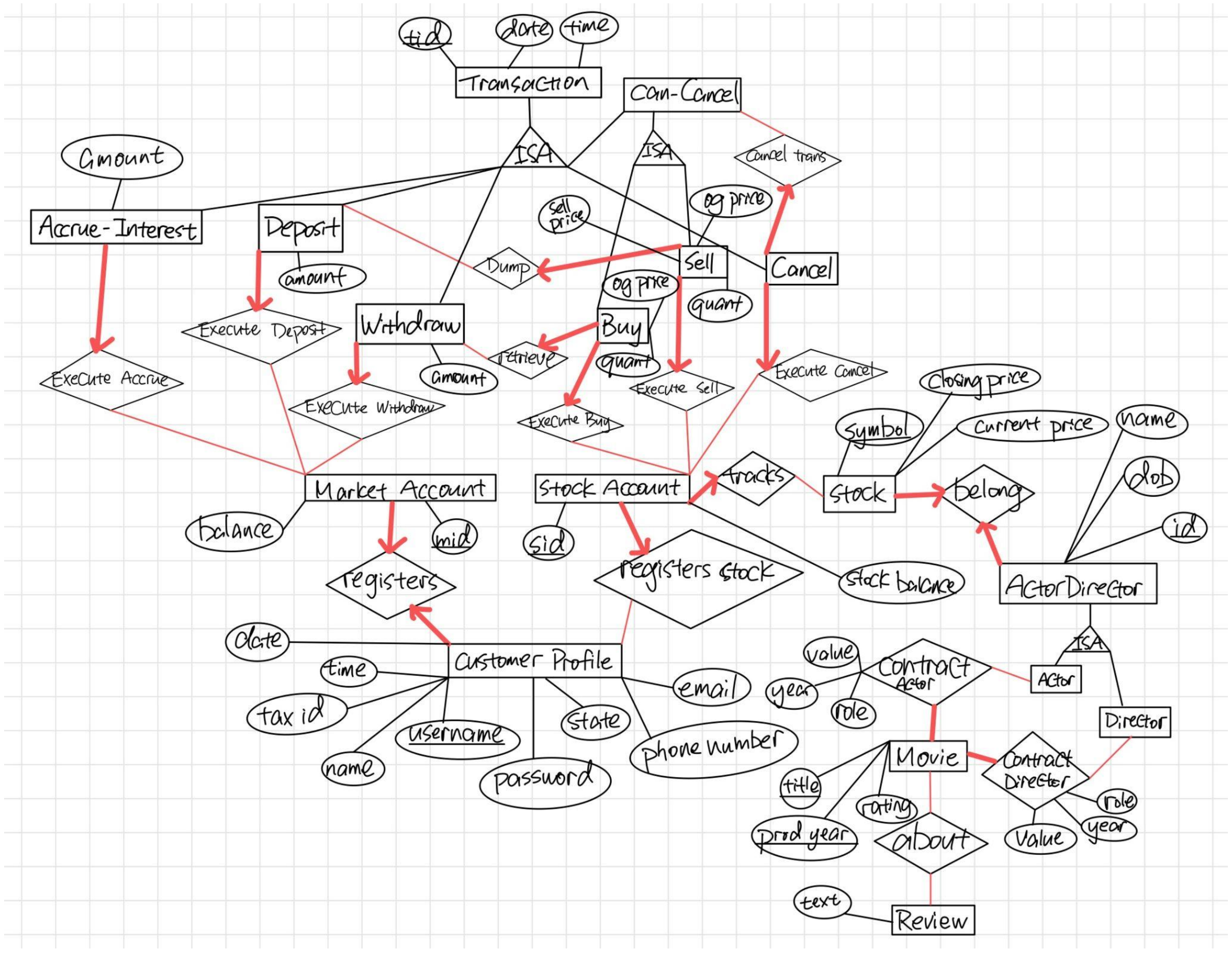
Therefore Actor and Director can have title and prod year as Not Null Foreign Key and Movie has id as a Not Null Foreign Key

- Accrue-Interest, Deposit, and Withdraw has a many to one relationship with Market Account and therefore Accrue-Interest, Deposit, and Withdraw has mid as a Not Null Foreign Key
- Buy and Sell has a many to one relationship with Deposit and Withdraw, respectively, therefore it will have tid as a Not Null Foreign Key. Moreover Buy and Sell also has a many to one relationship with Stock Account, therefore it will have sid as a Not Null Foreign Key
- Cancel has a many to one relationship with Can-Cancel, therefore it will have tid as a Not Null Foreign Key. Moreover Cancel also has a many to one relationship with Stock Account, therefore it will have sid as a Not Null Foreign Key

ER Diagram

Relational Schema

Customer_Profile (<u>username</u>: **str**, password: **str**, state: **str**, email: **str**, phone_number: **int**, name: **str**, time: **float**, tax id: **int**, date: **str**, mid: **int**)
Market_Account (<u>mid</u>: **int**, balance: **float**, username: **str**)
Stock_Account (<u>sid</u>: **int**, username: **str**, symbol: **str**)
Stock (<u>symbol</u>: **str**, closing_price: **float**, current_price: **float,** id: **int**)
Actor/Director (<u>id</u>: **int**, dob: **str**, name: **string,** symbol: **str**)



Actor (id: **int**)
Director (id: **int**)
Movie (title: **str**, prod_year: **int**, rating: **float**)
Review ()
Transaction (tid: **int**, date: **str**, time: **float**)
Accrue-Interest (tid: **int**, amount: **float**, mid: **int**)
Deposit (tid: **int**, amount: **float**, mid: **int**)
Withdraw (tid: **int**, amount: **float**, mid: **int**)
Can Cancel (tid: **int**)
Buy (tid: **int**, og_price: **float**, quant: **float**, tid: **int**, sid: **int**)
Sell (tid: **int**, og_price: **float**, sell_price: float, quant: **float**, tid: **int**, sid: **int**)
Cancel (tid: **int**, sid: **int**, tid: **int**)
Contract Actor (title: **str**, prod_year: **int**, id: **int,** role: **str**, year: **int**, value: **float**)
Contract Director (title: **str**, prod_year: **int**, id: **int,** role: **str**, year: **int**, value: **float**)
About (title: **str**, prod_year: **int**)

Incorporated Integrity Constraints

Primary Key Constraints:

- Customer Profile: Primary Key Constraint: username
- Market Account: Primary Key Constraint: mid
- Stock Account: Primary Key Constraint: sid
- Stock: Primary Key Constraint: symbol
- Actor/Director: Primary Key Constraint: id
    - Actor and Director (ISA of Actor/Director): Primary Key Constraint: id (shared)
- Transactions: Primary Key Constraint: tid
    - Accrue-Interest, Deposit, Withdraw, Can_Cancel, Buy, Sell (ISA of Transaction): Primary Key Constraint: tid (shared)
- Movie: Primary Key Constraint: title, prod_year

Foreign Key Constraints:

- Customer Profile: Foreign Key Constraint: None
- Market Account:
    - Foreign Key Constraint: username (references Customer Profile)
    - Foreign Key Constraint: mid (references Customer Profile)
- Stock Account:
    - Foreign Key Constraint: username (references Customer Profile)
    - Foreign Key Constraint: symbol (references Stock)
- Stock and Actor/Director:
    - Foreign Key Constraint: symbol (references Stock) in Actor/Director
    - Foreign Key Constraint: id (references Actor/Director) in Stock

- Actor and Director (ISA of Actor/Director) and Movie:
  - Foreign Key Constraint: title, prod_year (references Movie) in Actor and Director
- Accrue-Interest, Deposit, Withdraw:
  - Foreign Key Constraint: mid (references Market Account)
- Buy and Sell:
  - Foreign Key Constraint: tid (references Deposit and Withdraw)
  - Foreign Key Constraint: sid (references Stock Account)
- Cancel:
  - Foreign Key Constraint: tid (references Can_Cancel)
  - Foreign Key Constraint: sid (references Stock Account)

Additional Suggestions:

Unique Constraints:

- Consider enforcing uniqueness on the email and tax id fields in the Customer Profile entity.
- Consider enforcing uniqueness on the combination of title and prod_year in the Movie entity.

Additional Foreign Key Constraints:

- Add foreign key constraints in Contract Actor and Contract Director entities referring to the id attribute in Actor/Director.
- If Review and About entities are related to movies, add foreign key constraints referencing the title and prod_year in the Movie entity.

Addressing Integrity Constraint Violations

Handling Violations of Primary Key Constraints:

- Customer Profile, Market Account, Stock Account, Stock, Actor/Director, Actor, Director, Transactions, Accrue-Interest, Deposit, Withdraw, Can_Cancel, Buy, Sell, Movie:
  - Violation Handling:
    - Prevent insertion of duplicate keys: The database system should reject any attempt to insert a record with a primary key that already exists in the table.
    - Generate an error or exception: Notify the application or user about the attempted violation, allowing them to correct the issue.

Handling Violations of Foreign Key Constraints:

- Stock Account, Market Account, Stock, Actor/Director, Actor, Director, Accrue-Interest, Deposit, Withdraw, Buy, Sell, Cancel:
  - Violation Handling:

        - Reject operations violating foreign key constraints: If an attempt is made to insert a record with a foreign key that doesn't match an existing primary key, the operation should be rejected.
        - Cascade actions: cascading actions can be configured to automatically delete or update related records when a primary key is modified.
        - Not Null actions: the referenced Foreign keys should not be null and cannot be deleted through the reference

Other Considerations:

- Unique Constraint Violation (Customer Profile, Movie):
    - Violation Handling:
        - Generate an error or exception: Notify the application or user about the attempted violation, allowing them to correct the issue.
        - Implement a retry mechanism: In some cases, it might be appropriate to allow the user to correct the duplication issue and retry the operation.
- Foreign Key Constraint in Contract Actor, Contract Director, Review, About:
    - Violation Handling:
        - Reject operations with invalid references: Ensure that foreign key constraints are maintained, and reject operations attempting to reference non-existent primary keys.
        - Cascade actions: May choose to cascade actions (e.g., delete or update related records) or reject the operation.
- Handling General Data Quality Issues:
    - Violation Handling:
        - Implement validation checks: Before attempting to insert or update records, implement validation checks in the application layer to ensure data consistency.
        - Provide informative error messages: When a violation occurs, the system should provide clear and informative error messages to guide users or developers in correcting the issue.

Functional Architecture

Represents a customer profile with information such as username, password, state, email, phone number, and account details. Methods include updating the profile and retrieving the account balance.

```
class Customer_Profile {
        attributes: username(str), password(str), state(str), email(str), phone_number(int),
        name(str), time(float), tax_id(int), date(str), mid(int)
        def update_profile(self, new_data: dict) -> bool
        def get_username(self) -> str
        def get_password(self) -> str
        def get_state(self) -> str
        def get_email(self) -> str
        def get_phone_number(self) -> int
        def get_name(self) -> str
        def get_time(self) -> float
        def get_tax_id(self) -> int
        def get_date(self) -> str
        def get_mid(self) -> int
        def get_MarketAccount(self) -> Market_Account
}
```

Represents a market account with attributes like account ID, balance, and associated username.

```
class Market_Account{
        attributes: mid(int), balance(float), username(str)
        def get_mid(self) -> int
        def get_balance(self) -> float
        def get_username(self) -> str
        def set_balance(self, float x) -> void
}
```

Represents a stock trading account with attributes such as stock ID, username, and associated stock symbol.

```
class Stock_Account{
        attributes: sid(int), username(str), symbol(str)
        def get_sid(self) -> int
        def get_symbol(self) -> str
        def get_username(self) -> str
        def get_stock(self) -> Stock
        def get_CustomerProfile(self) -> CustomerProfile
}
```

Represents information about a stock, including its symbol, closing price, current price, and ID. Methods include retrieving stock information.

```
class Stock{
        attributes: symbol(str), closing_price(float), current_price(float), id(int)
        def get_symbol(self) -> str
        def get_current_price(self) -> float
        def get_closing_price(self) -> float
        def get_id(self) -> int
        def get_associated_ActorDirector(self) -> ActorDirector
        def set_current_price(self, float new_price) -> void
        def set_closing_price(self, float new_price) -> void

}
```

Represents a common entity for both actors and directors with attributes such as ID, date of birth, name, and a symbol. Methods include retrieving details about the actor or director.

```
class ActorDirector{
        attributes: id(int), dob(str), name(str), symbol(str)
        def get_id(self) -> int
        def get_dob(self) -> str
        def get_name(self) -> str
        def get_symbol(self) -> int
        def get_Stock(self) -> Stock
}
```

Represents an actor with an ID.

```
class Actor{
        attributes: id (int)
        def get_id(self) -> int
}
```

Represents a director with an ID.

```
class Director{
        attributes: id (int)
        def get_id(self) -> int
}
```

Represents a movie with attributes like title, production year, and rating. Methods include retrieving details about the movie.

```
class Movie{
        attributes: title (str), prod_year (int), rating (float)
        def get_title(self) -> str
        def get_prod_year(self) -> int
        def get_rating(self) -> float
        def getActor/Director(self) -> Actor/Director
}
```

Represents a review to a movie.

```
class Review{
        attributes:
        def get_text(self) -> str
}
```

Represents a generic financial transaction with attributes such as transaction ID, date, and time. Methods include retrieving transaction details.

```
class Transaction{
        attributes: tid (int), date (str), time (float)
        def get_tid(self) -> int
        def get_date(self) -> str
        def get_time(self) -> float
}
```

Represents an interest accrual transaction with attributes like transaction ID, amount, and account ID. Methods include calculating accrued interest

```
class Accrue-Interest{
        attributes: tid (int), amount (float), mid (int)
        def get_tid(self) -> int
        def get_amount(self) -> float
        def get_mid(self) -> int
        def calculate_interest(self) -> float
        def get_MarketAccount(self) -> Market_Account
}
```

Represents a deposit transaction with attributes like transaction ID, amount, and account ID. Methods include processing the deposit.

```
class Deposit{
        attributes: tid (int), amount (float), mid (int)
        def get_tid(self) -> int
        def get_amount(self) -> float
        def get_mid(self) -> int
        def deposit(self, amount) -> bool
        def get_MarketAccount(self) -> Market_Account
}
```

Represents a withdrawal transaction with attributes like transaction ID, amount, and account ID. Methods include processing the withdrawal.

```
class Withdraw{
        attributes: tid (int), amount (float), mid (int)
        def get_tid(self) -> int
        def get_amount(self) -> float
        def get_mid(self) -> int
        def withdraw(self, amount) -> bool
        def get_MarketAccount(self) -> Market_Account
}
```

Represents a transaction cancellation indicator with a transaction ID. Methods include checking if the transaction can be canceled.

```
class Can_Cancel{
        attributes: tid (int)
        def get_tid(self) -> int
}
```
Represents a stock purchase transaction with attributes like transaction ID, original price, quantity, and stock ID. Methods include executing the buy order.
```
class Buy{
        attributes: tid (int), og_price (float), quant (float), tid (int), sid (int)
        def get_tid(self) -> int
        def get_og_price(self) -> float
        def get_quant(self) -> float
        def get_tid(self) -> int
        def get_sid(self) -> int
        def execute_buy(self, quant, stock) -> bool
        def get_Withdraw(self) -> Withdraw
        def get_StockAccount(self) -> Stock_Account
}
```
Represents a stock sale transaction with attributes like transaction ID, original price, quantity, and stock ID. Methods include executing the sell order of a specific stock.
```
class Sell{
        attributes: tid (int), og_price (float), sell_price (float), quant (float), tid (int), sid (int)
        def get_tid(self) -> int
        def get_og_price(self) -> float
        def get_sell_price(self) -> float
        def get_quant(self) -> float
        def get_tid(self) -> int
        def get_sid(self) -> int
        def execute_buy(self, quant, stock)() -> bool
        def get_Deposit(self) -> Deposit
        def get_StockAccount(self) -> Stock_Account
}
```
Represents a transaction cancellation request with attributes like transaction ID and stock ID. Methods include canceling the specified transaction.
```
class Cancel{
        attributes: tid (int), sid (int), tid (int)
        def get_tid(self) -> int
        def get_tid(self) -> int
        def get_sid(self) -> int
        def cancel_transaction(self, transaction) -> bool
        def get_StockAccount(self) -> Stock_Account
```

```
    def get_CanCancelTransaction(self) -> Can_Cancel
}
```

Represents a contract for an actor with attributes like movie title, production year, actor ID, role, contract year, and value. Methods include retrieving contract details.

```
class ContractActor{
    attributes: title (str), prod_year (int), id (int), role (str), year (int), value (float)
    def get_title(self) -> str
    def get_prod_year(self) -> int
    def get_id(self) -> int
    def get_role(self) -> str
    def get_year(self) -> int
    def get_value(self) -> float
    def get_Movie(self) -> Movie
    def get_Actor(self) -> Actor
}
```

Represents a contract for a director with attributes like movie title, production year, director ID, role, contract year, and value. Methods include retrieving contract details.

```
class ContractDirector{
    attributes: title (str), prod_year (int), id (int), role (str), year (int), value (float)
    def get_title(self) -> str
    def get_prod_year(self) -> int
    def get_id(self) -> int
    def get_role(self) -> str
    def get_year(self) -> int
    def get_value(self) -> float
    def get_Movie(self) -> Movie
    def get_Director(self) -> Director
}
```

Represents information about a movie, including its title and production year. Methods include retrieving details about the movie.

```
class About{
    attributes: title (str), prod_year (int)
    def get_title(self) -> str
    def get_prod_year(self) -> int
    def get_Movie(self) -> Movie
}
class ManagerInterface {
    def add_interest(String customer_username) -> void
    def monthly_statement(String customer_username) -> str
    def active_customers() -> List(Customer)
```

```
        def DTER() -> List(Customer)
        def customer_report(String customer_username) -> Tuple(List(MarketAccount,
        StockAccount))
        def del_transactions() -> void
}
class TraderInterface{
        attributes: customer_username(str)
        def register(String username, String name, String pw, String state, String email, String
        phone_number, String tax_id, String date) -> CustomerProfile
        def deposit(double amount) -> void
        def withdraw(double amount) -> void
        def buy(String stock_symbol, double amount) -> void
        def sell(String stock_symbol, double price, double amount) -> void
        def cancel(int tid) -> void
        def show_market_balance() -> float
        def show_stock_history() -> String
        def stock_info(String symbol) -> String
        def movie_info(String name, int year) -> String
        def top_movies(String date1, String date2) -> String
        def reviews(String, name, int year) -> String

}
```

Task Division

Members:

Johnson Chan
Richard Gao

Task Division:

Richard Gao

- Database Programming for Transactions and related entities
- Setting up Oracle and other backend associated tasks
- Draw ER Diagram and discuss constraints
- Create bi-weekly meeting schedules

Johnson Chan

- Design and create frontend interfaces (web app/GUI)
- Database Programming for Customer Profile and related entities
- Combine all answers into Preliminary report and discuss relevant constraints

- Attend bi-weekly meetings meetings