DNA

Project – Phase – 3

Patanjali B Revanth G Advaith Malladi

Balaji Suneetha.

Team – 28.

Converting to Relational Model:

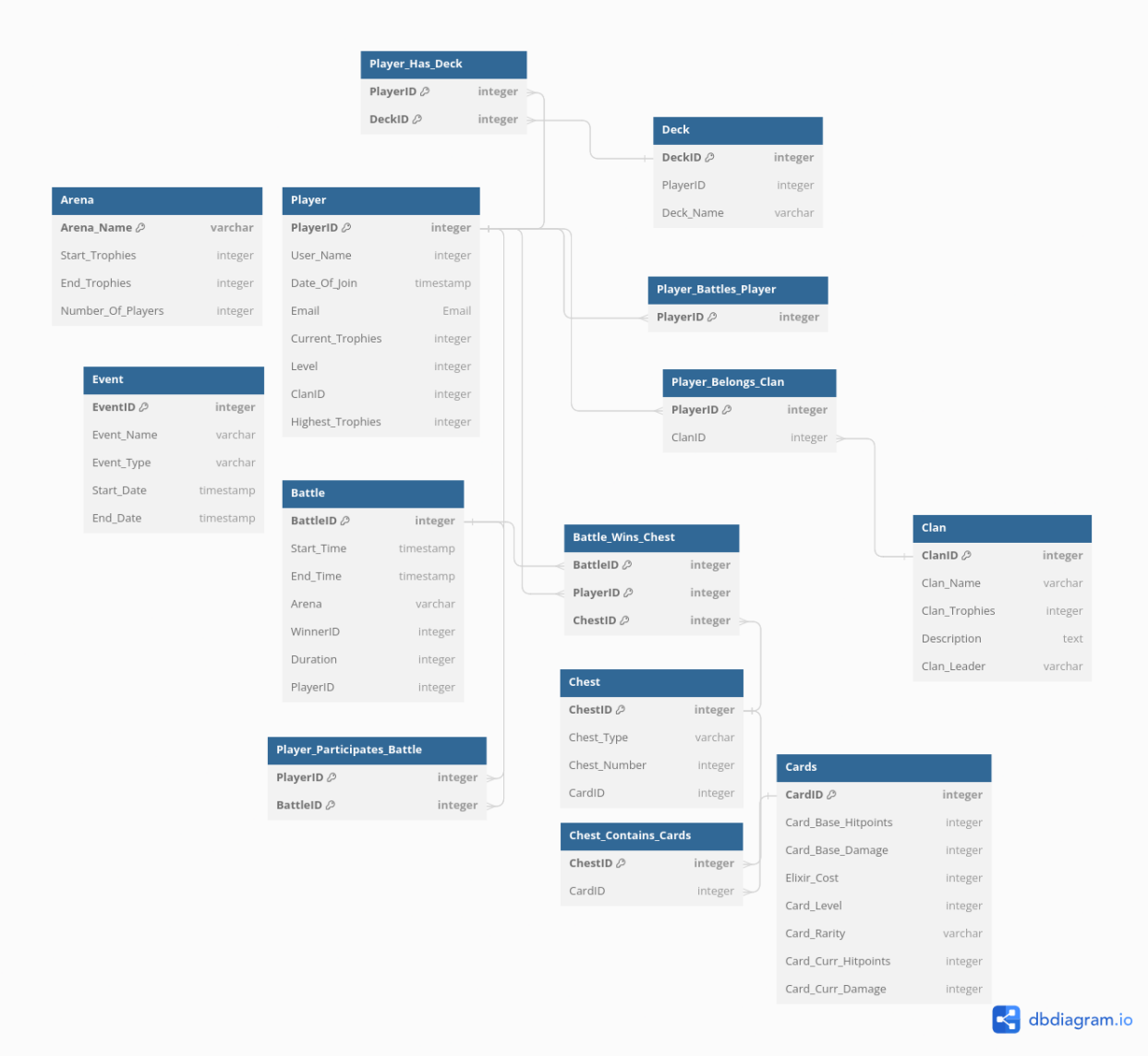
To convert an ER diagram into the Relational Model, we first converted each of the entities into Tables, clearly marking out the Key Attributes (Having a key sign in the diagram) and also marking out the non-prime attributes in the tables.

Next, we created and populated the relationships also as tables with carefully tracing back to the participating entities via connections. Next, we again demarcated the Primary Key attributes in each of these relationship tables also.

Redundancy reduction:

The steps we took to reduce redundancy in the relational model are, while creating and populating relationship tables, we did not put all the entities as a primary key, but if just one primary key was sufficient to identify the relationship, then only that was chosen as the primary key.

Below is the diagram for Relational Model.



Converting to 1NF

Now that we have a Relational Model, we need to convert this into a 1NF form.

First Normal Form (1NF) primarily deals with eliminating repeating groups and ensuring that each attribute contains atomic values.

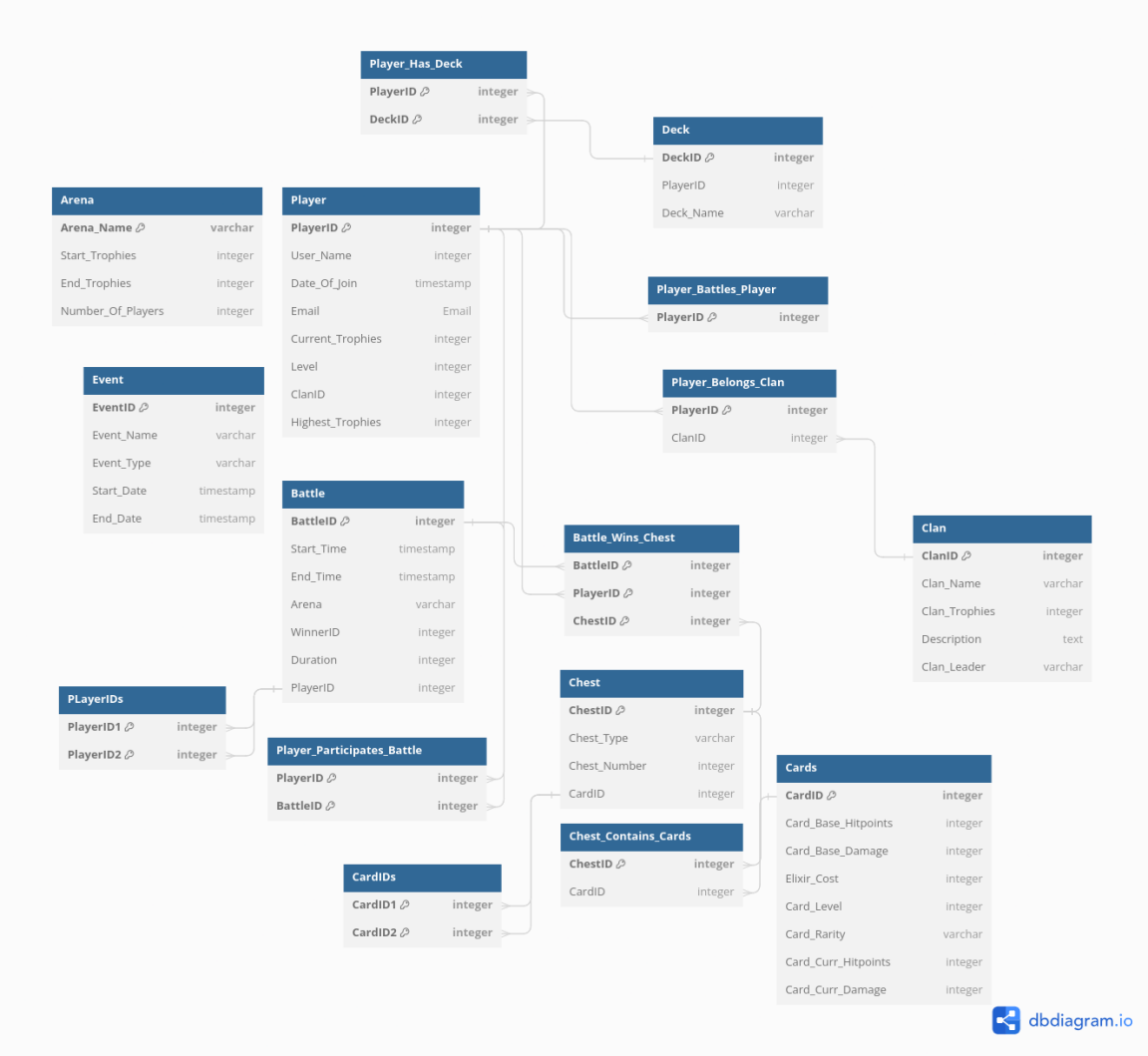
To reduce to 1NF we have identified several Multi-valued Attributes to break it down into different tables like:

The CardID(attribute) in the Chest (Entity) is a Multi-valued Attribute, so it has been broken down into a different table and populated accordingly.

The PlayerID(attribute) of the Battle (entity) is also a Multi-valued Attribute, so it has been broken down into a different table and populated accordingly.

Since there were no composite attributes to any of the entities that was not something which we needed to take care of.

Below is the diagram for 1NF.



Converting to 2NF:

Second Normal Form (2NF) is a database normalization process that ensures all non-prime attributes are fully functionally dependent on the entire primary key, eliminating partial dependencies in a relational table.

n our database, we neither have multiple primary keys nor do any of the primary keys possess multiple attributes.

While 2NF builds upon 1NF, introducing the notion of addressing partial dependencies in composite primary keys, but in our current scenario, where the data structure is relatively straightforward and lacks composite primary keys, the distinctions between 1NF and 2NF may not be as pronounced. Therefore, in this particular case, we are concluding that 1NF is effectively equivalent to 2NF due to the absence of multi-attribute primary keys in our dataset.

Converting to 3NF:

Third Normal Form (3NF) is a database normalization level that ensures that there are no transitive dependencies, meaning that non-prime attributes are not indirectly dependent on the primary key through other non-prime attributes.

1. In the context of achieving Third Normal Form (3NF) in our database, we've encountered a scenario where a derived attribute, let's say "Duration," is determined by the difference between the "Start_Time" and "End_Time," and this duration is indirectly dependent on the primary key, which we'll refer to as "BattleID." To ensure that our database adheres to 3NF principles, we've opted to create a separate table. This new table is designed to store the raw data, including "BattleID," "Start_Time," and "End_Time." By doing so, we've effectively eliminated the transitive dependency of the derived attribute on the primary key.

2. We've also encountered a situation within the "Event" entity where a derived attribute, specifically "Duration," is calculated based on the difference between the "Start_Time" and "End_Time." Recognizing that this duration is indirectly dependent on the primary key, let's call it "EventID," via the temporal attributes, we have taken the initiative to enhance the normalization of our database. To address this transitive dependency, we've chosen to create a distinct table dedicated to the raw data associated with events. This new table encompasses "EventID," "Start_Time," and "End_Time," effectively separating the derived attribute "Duration" from the primary key.

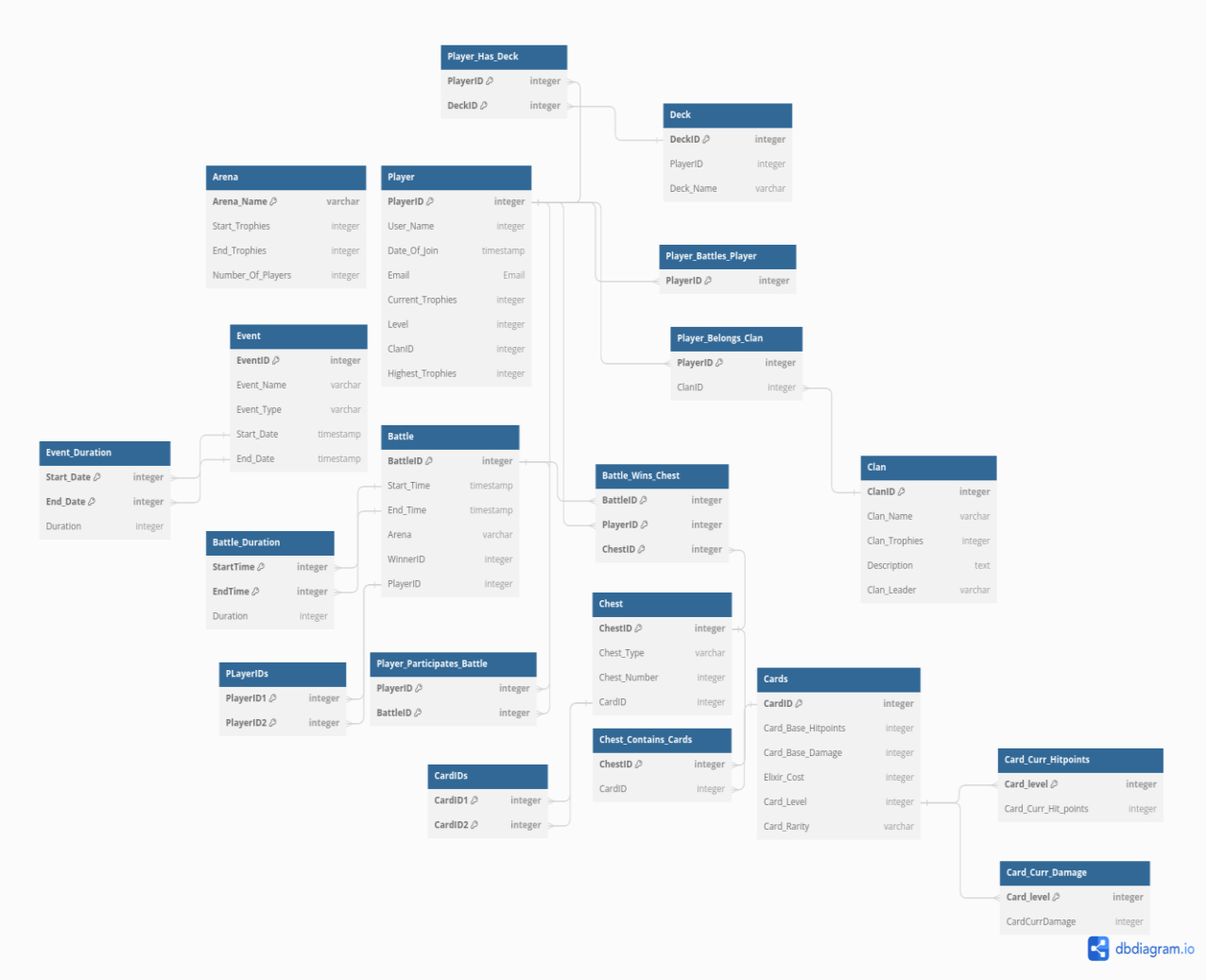
3. Calculating Card_Curr_Hitpoints:

   Recognizing that "Card_Curr_Hitpoints" is a derived attribute dependent on "Card_Level," a new table has been created specifically for raw data storage, separating the derived attribute from the primary key in the "Card" entity.

4. Calculating Card_Curr_Damage:

   Addressing the derived attribute "Card_Curr_Damage" in the "Card" entity, a dedicated table has been established with "Card_Level," facilitating the normalization process by eliminating transitive dependencies and ensuring 3NF compliance.

Below is the figure for 3NF



```
Table Player {
  PlayerID integer [PK]
  User_Name integer
  Date_Of_Join timestamp
  Email  Email
  Current_Trophies integer
  Level integer
  ClanID  integer
  Highest_Trophies integer

}

Table Clan
{
  ClanID integer [PK]
  Clan_Name varchar
  Clan_Trophies integer
  Description text
  Clan_Leader varchar
}

Table Battle
{
  BattleID integer [PK]
  Start_Time timestamp
  End_Time timestamp
  Arena varchar
  WinnerID  integer
  PlayerID integer
}

Table Cards
{
  CardID integer PK
  Card_Base_Hitpoints integer
  Card_Base_Damage integer
  Elixir_Cost integer
  Card_Level integer
  Card_Rarity varchar
}

Table Arena
{
Arena_Name varchar PK
Start_Trophies integer
End_Trophies integer
Number_Of_Players integer
}
Table Event
{
EventID integer PK
Event_Name varchar
Event_Type varchar
Start_Date timestamp
End_Date timestamp
}
Table Deck
{
  DeckID integer PK
  PlayerID integer
  Deck_Name varchar

}
Table Player_Participates_Battle
{
  PlayerID integer PK [ref: >  Player.PlayerID]
  BattleID integer PK [ref: > Battle.BattleID]
}
Table Player_Belongs_Clan
{
  PlayerID integer PK [ref: > Player.PlayerID]
  ClanID integer [ref: > Clan.ClanID]
}

Table Player_Has_Deck
{
  PlayerID integer PK [ref: > Player.PlayerID]
  DeckID integer PK [ref: > Deck.DeckID]
}
Table Chest_Contains_Cards
{
  ChestID integer PK [ref: > Chest.ChestID]
  CardID integer [ref: > Cards.CardID]
}
Table Player_Battles_Player
{
  PlayerID integer PK [ref: > Player.PlayerID]
}

Table Battle_Wins_Chest
{
  BattleID integer PK [ref: > Battle.BattleID]
  PlayerID integer PK [ref: > Player.PlayerID]
  ChestID integer PK [ref: > Chest.ChestID]

}
Table PLayerIDs
{
  PlayerID1 integer PK [ref: > Battle.PlayerID]
  PlayerID2 integer PK [ref: > Battle.PlayerID]
}

Table CardIDs
{
  CardID1 integer PK [ref: > Chest.CardID]
  CardID2 integer PK [ref: > Chest.CardID]
}
Table Battle_Duration
{
  StartTime integer PK [ref: > Battle.Start_Time]
  EndTime integer PK [ref: > Battle.End_Time]
  Duration integer
}

Table Card_Curr_Hitpoints
{
  Card_level integer PK [ref: > Cards.Card_Level]
  Card_Curr_Hit_points integer
}
Table Card_Curr_Damage
{
  Card_level integer PK [ref: > Cards.Card_Level]
  CardCurrDamage integer
}
Table Event_Duration
{
  Start_Date integer PK [ref: > Event.Start_Date]
  End_Date integer PK [ref: > Event.End_Date]
  Duration integer
```

The above is the code for this.