University of British Columbia, Vancouver
Department of Computer Science

CPSC 304 Project Cover Page

Milestone #:2

Date: July 21, 2024

Group Number: 51

Name	Student Number	CS Alias (Userid)	Preferred E-mail Address
Perry Zhu	32653826	t9n60	perryz@students.ubc.ca
Yuchen Gu	37280534	i6m6h	guyuchen999@gmail.com
Ch Muhammad Daud Virk	26838482	e1h8m	daudvirk@student.ubc.ca

By typing our names and student numbers in the above table, we certify that the work in the attached assignment was performed solely by those whose names and student IDs are included above. (In the case of Project Milestone 0, the main purpose of this page is for you to let us know your e-mail address, and then let us assign you to a TA for your project supervisor.)

In addition, we indicate that we are fully aware of the rules and consequences of plagiarism, as set forth by the Department of Computer Science and the University of British Columbia

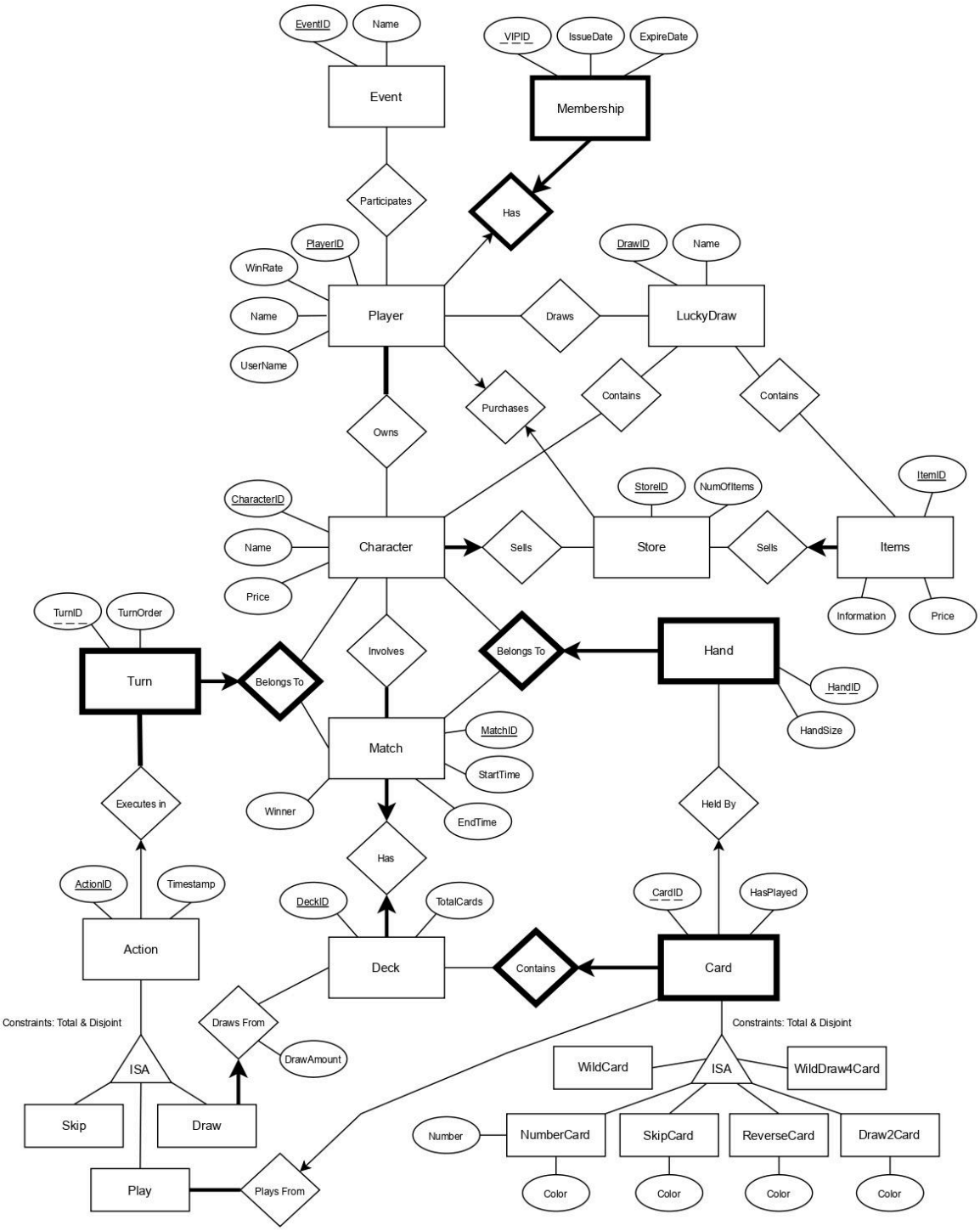
Brief Summary:

This project attempts to manage the internal State of an UNO Game Application. The “States” that are captured include the Game State(s) of a virtual Game of UNO (who holds what card, whose turn is it et cetera.) and information about Players who make use of the application e.g. the virtual items and characters linked to their UNO app account, their statistics et cetera.

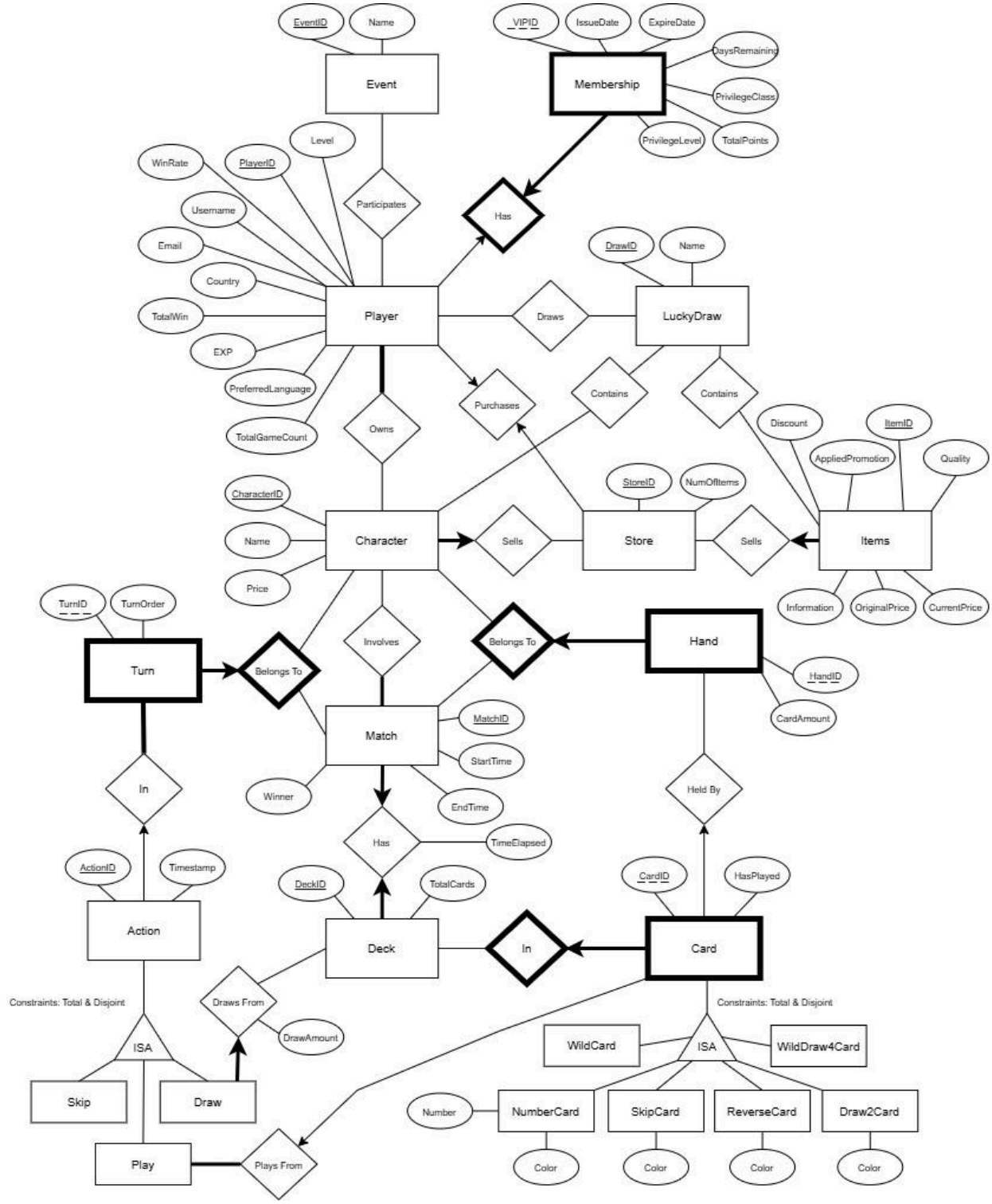
Changes on the diagram:

1. Added attributes to Player: TotalWin, TotalGameCount, EXP, Level, PreferredLanguage, Country, Email.
2. Added attributes to Membership: DaysRemaining, PrivilegeLevel, TotalPoints, PrivilegeClass.
3. Added attributes to Items: Quality, CurrentPrice, Discount, AppliedPromotion.
4. Changed attribute name in Items: Price → OriginalPrice.
5. Changed attribute name in Hand: HandSize → CardAmount.
6. Changed attribute name in Player: Name → Username.
7. Changed Relationship Set Name “Executes in” → In (b/w Entity Sets “Turn” and “Action”).
8. Changed Relationship Set Name “Contains” → In (b/w Entity Sets “Turn” and “Action”).

You will find the ORIGINAL (from Milestone 1) diagram on the following page and the UPDATED diagram on the page after that.


EventID
Name
Event
VIPID
IssueDate
ExpireDate
Membership
Participates
Has
PlayerID
WinRate
Name
UserName
Player
Draws
DrawID
Name
LuckyDraw
Owns
Purchases
Contains
Contains
CharacterID
Name
Price
Character
Sells
StoreID
NumOfItems
Store
Sells
ItemID
Items
Information
Price
TurnID
TurnOrder
Turn
Belongs To
Involves
Belongs To
Hand
HandID
HandSize
Match
MatchID
StartTime
EndTime
Winner
Executes in
Held By
Has
ActionID
Timestamp
Action
DeckID
TotalCards
Deck
Contains
CardID
HasPlayed
Card
Constraints: Total & Disjoint
ISA
Skip
Draw
Draws From
DrawAmount
Play
Plays From
Constraints: Total & Disjoint
WildCard
ISA
WildDraw4Card
Number
NumberCard
SkipCard
ReverseCard
Draw2Card
Color
Color
Color
Color

Original ^


EventID
Name
Event
VIPID
IssueDate
ExpireDate
DaysRemaining
Membership
PrivilegeClass
PrivilegeLevel
TotalPoints
Level
WinRate
PlayerID
Participates
Has
Username
Email
Country
TotalWin
EXP
PreferredLanguage
TotalGameCount
Player
DrawID
Name
Draws
LuckyDraw
Purchases
Contains
Contains
Owns
Discount
ItemID
AppliedPromotion
Quality
StoreID
NumOfItems
CharacterID
Name
Price
Character
Sells
Store
Sells
Items
Information
OriginalPrice
CurrentPrice
TurnID
TurnOrder
Turn
Belongs To
Involves
Belongs To
Hand
HandID
CardAmount
MatchID
StartTime
EndTime
Winner
Match
In
Held By
Has
TimeElapsed
ActionID
Timestamp
DeckID
TotalCards
CardID
HasPlayed
Action
Deck
In
Card
Constraints: Total & Disjoint
Draws From
DrawAmount
Constraints: Total & Disjoint
ISA
WildCard
ISA
WildDraw4Card
Skip
Draw
Number
NumberCard
SkipCard
ReverseCard
Draw2Card
Play
Plays From
Color
Color
Color
Color

Updated ^

Note: It is implied that Candidate Keys would have the "same" Functional Dependencies as the Primary Key (except that the candidate key would be substituted with the primary key in the statement. Please refer to Player's schema for an example. Other examples would follow the same format.).

Note: The functional dependencies enumerated (separatedly) are those that entail the non-primary and non-candidate keys (for the PK and CK, the FDs are listed explicitly). They are used for decomposition. It is never the case that a foreign key is used for decomposition as specified in the milestone document, though if a functional dependency exists this is listed.

Note: Functional Dependencies do not include the trivial cases e.g. A → A (as mentioned inside the milestone).

Note: Primary Keys and Foreign keys are specified using the notation in class:
> **bold** for foreign key
> <u>underlined</u> for primary keys
> **<u>both</u>** when both.
> same for composite keys

Note: We normalize to BCNF. As indicated in an email with the Project Mentor the FDs aren't explicitly shown for the derived tables. PK FD et cetera is intended to reiterate the PK; the same applies to others.

Note: Implicit FDs e.g. A → B, and B → C resulting in A → C are omitted unless their inclusion allows for decomposition, or their inclusion is considered important.

Note: To build on the above, derivable FDs are also not included e.g. A → B, C → D, resulting in AC → BD.
Note: The ISA Entities follow this Naming Convention: SubclassSuperclass e.g. SkipAction etc.

Note: Note that ALL implications of the decomposition are intentional, and this is a design choice. When a table has to decomposed it is: the constraints mentioned at the top hold in the final tables that are used. The applicability of the FDs are mentioned as well (on the final tables).
To build on the above all the tables adhere to BCNF: when they don't they are split until they do.

Note: The entirety of the included ER diagram (the updated one) has been implemented in both the schema and using SQL. The entity sets and relationship sets are combined as prescribed in the lecture, and the naming is intended to reflect that, as much as possible.
Notes:
> There was nothing that was to be changed in our Miletone 1 ER diagram, as per the mentor's indication.
> N/A can mean one of many things: trivial, not applicable et cetera (as regards the FDs). Though this has been made as clear as possible, it is requested that minor oversights be forgiven.

> Weak Entities are indicated via their rather Unique Primary Keys: to name them from the ER diagram: Card, Hand, Turn, and Membership. This is primarily indicated by the usage of a foreign key in their primary key. Total Participation for the Weak Entity is implied.

Player(PlayerID: NUMBER, WinRate: FLOAT, Username: VARCHAR(255), Email: VARCHAR(255), TotalWin: NUMBER, TotalGameCount: NUMBER, EXP: NUMBER, Level: NUMBER, PreferredLanguage: VARCHAR(255), Country: VARCHAR(255))

Constraints: **[ALL OF THESE APPLY TO THE FINAL (AFTER DECOMPOSITION) TABLES AS WELL]**
Email must not be NULL and it must be UNIQUE.
Username must not be NULL and it must be UNIQUE.
TotalWin must not be NULL.
TotalGameCount must not be NULL.
WinRate must not be NULL.
EXP must not be NULL.
[EXP must be less than or equal to 10,000.]
Level must not be NULL.
Country must not be NULL.
PreferredLanguage must not be NULL.

Primary Key Functional Dependency:
PlayerID → WinRate, Username, Email, TotalWin, TotalGameCount, EXP, Level, PreferredLanguage, Country

Candidate Key(s):
Email → PlayerID, WinRate, Username, TotalWin, TotalGameCount, EXP, Level, PreferredLanguage, Country
Username → PlayerID, WinRate, Email, TotalWin, TotalGameCount, EXP, Level, PreferredLanguage, Country

Foreign Key Functional Dependency:
N/A

Functional Dependencies:
Email → Username
Username → Email
TotalWin, TotalGameCount → WinRate
EXP → Level
Country → PreferredLanguage

Normalizing to BCNF:

1. Decompose with respect to Username → Email
    a. PlayerUsernameAndEmail(Username: VARCHAR(255), Email: VARCHAR(255))
    b. Remaining1(PlayerID: NUMBER, WinRate: FLOAT, **Username: VARCHAR**(255), TotalWin: NUMBER, TotalGameCount: NUMBER, EXP: NUMBER, Level: NUMBER, PreferredLanguage: VARCHAR(255), Country: VARCHAR(255))
        i. PK FD: PlayerID → WinRate, Username, TotalWin, TotalGameCount, EXP, Level, PreferredLanguage, Country.
        ii. FK: Username
        iii. CK FD: Username
2. Decompose with respect to TotalWin, TotalGameCount → WinRate
    a. PlayerGameStatistics(TotalWin: NUMBER, TotalGameCount: NUMBER, WinRate: FLOAT)
    b. Remaining2(PlayerID: NUMBER, **Username: VARCHAR**(255), **TotalWin: NUMBER**, **TotalGameCount: NUMBER**, EXP: NUMBER, Level: NUMBER, PreferredLanguage: VARCHAR(255), Country: VARCHAR(255))
        i. PK FD: PlayerID →Username, TotalWin, TotalGameCount, EXP, Level, PreferredLanguage, Country.
        ii. FK: Username, TotalWin, and TotalGameCount
        iii. CK FD: Username → PlayerID, TotalWin, TotalGameCount, EXP, Level, PreferredLanguage, Country.
3. Decompose with respect to EXP → Level
    a. PlayerLevel(EXP: NUMBER, Level: NUMBER)
    b. Remaining3(PlayerID: NUMBER, **Username: VARCHAR**(255), **TotalWin: NUMBER**, **TotalGameCount: NUMBER**, **EXP: NUMBER**, PreferredLanguage: VARCHAR(255), Country: VARCHAR(255))
        i. PK FD: PlayerID →Username, TotalWin, TotalGameCount, EXP, PreferredLanguage, Country.
        ii. FK: Username, TotalWin, EXP, and TotalGameCount,
        iii. CK FD: Username → PlayerID, TotalWin, TotalGameCount, EXP, PreferredLanguage, Country.
4. Decompose with respect to Country → PreferredLanguage
    a. PlayerLanguage(Country: VARCHAR(255), PreferredLanguage: VARCHAR(255))
    b. Remaining4(PlayerID: NUMBER, **Username: VARCHAR**(255), **TotalWin: NUMBER**, **TotalGameCount: NUMBER**, **EXP: NUMBER**, **Country: VARCHAR**(255))
        i. PK FD: PlayerID →Username, TotalWin, TotalGameCount, EXP, Country.
        ii. FK: Username, TotalWin, EXP, and TotalGameCount, Country,
        iii. CK FD: Username → PlayerID, TotalWin, TotalGameCount, EXP, Country.
5. Final decomposed tables
    a. PlayerUsernameAndEmail(Username: VARCHAR(255), Email: VARCHAR(255))
        i. PK FD: Username → Email.
        ii. No FK.
        iii. Note that Email is (still) a CK, in this table with CK FD: Email → Username.
    b. PlayerGameStatistics(TotalWin: NUMBER, TotalGameCount: NUMBER, WinRate: FLOAT)

i. PK FD: TotalWin, TotalGameCount → WinRate.
ii. No FK.
iii. No CK.

c. PlayerLevel(EXP: NUMBER, Level: NUMBER)
i. PK FD: EXP → Level.
ii. No FK.
iii. No CK.

d. PlayerLanguage(Country: VARCHAR(255), PreferredLanguage: VARCHAR(255))
i. PK FD Country → PreferredLanguage.
ii. No FK.
iii. No CK.

e. Player(PlayerID: NUMBER, **Username: VARCHAR**(255), **TotalWin: NUMBER**, **TotalGameCount: NUMBER**, **EXP: NUMBER**, **Country: VARCHAR**(255))
i. PK FD: PlayerID → Username, TotalWin, TotalGameCount, EXP, Country.
ii. All attributes other than PlayerID are FKs.
iii. Note that Username is (still) a CK in this table i.e. Username → PlayerID, TotalWin, TotalGameCount, EXP, Country.

Note that all original FDs are maintained (in the final decomposed tables), here.

MembershipInPlayer(**PlayerID: NUMBER**, VIPID: NUMBER, IssueDate: DATE, DaysRemaining: NUMBER, ExpireDate: DATE, PrivilegeLevel: NUMBER, TotalPoints: NUMBER, PrivilegeClass: VARCHAR(255))

Constraints: **[UPHELD IN THE FINAL TABLES (THE DECOMPOSITIONS, AS WELL.]**
IssueDate cannot be NULL.
DaysRemaining cannot be NULL.
ExpireDate cannot be NULL.
PrivilegeLevel cannot be NULL.
PrivilegeClass cannot be NULL.
TotalPoints cannot be NULL.
PlayerID is UNIQUE (and not NULL).

Primary Key Functional Dependency:
PlayerID, VIPID → IssueDate, DaysRemaining, ExpireDate, PrivilegeLevel, TotalPoints, PrivilegeClass

Foreign Key Functional Dependency:
PlayerID by itself is insufficient to uniquely identify ANYTHING in this Entity set. It has to be used as indicated above.

Candidate Key(s):

N/A.

Functional Dependencies:
IssueDate, DaysRemaining → ExpireDate
TotalPoint → PrivilegeLevel
PrivilegeLevel → PrivilegeClass

Normalizing to BCNF:

1. Decompose with respect to IssueDate, DaysRemaining → ExpireDate
    a. MembershipExpireDate(IssueDate: DATE, DaysRemaining: NUMBER, ExpireDate: DATE)
    b. Remaining1(**PlayerID: NUMBER**, VIPID: NUMBER, **IssueDate: DATE**, **DaysRemaining: NUMBER**, PrivilegeLevel: NUMBER, TotalPoints: NUMBER, PrivilegeClass: VARCHAR(255))
        i. PK FD: PlayerID, VIPID→ IssueDate, DaysRemaining, PrivilegeLevel, TotalPoints, PrivilegeClass.
        ii. No CK.
        iii. FKs: PlayerID, IssueDate, DaysRemaining,.
2. Decompose with respect to PrivilegeLevel → PrivilegeClass
    a. MembershipPrivilegeClass(PrivilegeLevel: NUMBER, PrivilegeClass: VARCHAR(255))
    b. Remaining2(**PlayerID: NUMBER**, VIPID: NUMBER, **IssueDate: DATE**, **DaysRemaining: NUMBER**, **PrivilegeLevel: NUMBER**, TotalPoints: NUMBER)
        i. PK FD: PlayerID, VIPID→ IssueDate, DaysRemaining, PrivilegeLevel, TotalPoints.
        ii. No CK.
        iii. FKs: PlayerID, IssueDate, DaysRemaining, PrivilegeLevel.
3. Decompose with respect to TotalPoint → PrivilegeLevel
    a. PrivilegeLevelPoint(TotalPoint: NUMBER, **PrivilegeLevel: NUMBER**)
    b. Remaining3(**PlayerID: NUMBER**, VIPID: NUMBER, **IssueDate: DATE**, **DaysRemaining: NUMBER**, **TotalPoints: NUMBER**)
        i. PK FD: PlayerID, VIPID→ IssueDate, DaysRemaining, TotalPoints.
        ii. No CK.
        iii. FKs: PlayerID, IssueDate, DaysRemaining, TotalPoints.
4. Final decomposed tables
    a. MembershipExpireDate(IssueDate: DATE, DaysRemaining: NUMBER, ExpireDate: DATE)
        i. PK FD: IssueDate, DaysRemaining → ExpireDate
        ii. No CK.
        iii. No FK.
    b. MembershipPrivilegeClass(PrivilegeLevel: NUMBER, PrivilegeClass: VARCHAR(255))
        i. PK FD: PrivilegeLevel → PrivilegeClass
        ii. No CK.
        iii. No FK.
    c. PrivilegeLevelPoint(TotalPoint: NUMBER, **PrivilegeLevel: NUMBER**)

i. PK FD: TotalPoint → PrivilegeLevel
ii. No CK.
iii. FK: PrivilegeLevel.

d. MembershipInPlayer(**PlayerID: NUMBER**, VIPID: NUMBER, **IssueDate: DATE**, **DaysRemaining: NUMBER**, **TotalPoints: NUMBER**)
   i. PK FD: PlayerID, VIPID→ IssueDate, DaysRemaining, PrivilegeLevel, TotalPoints.
   ii. No CK.
   iii. FKs: PlayerID, IssueDate, DaysRemaining, PrivilegeLevel, TotalPoints.

Note: All FDs are upheld in the final tables.
Note: "IssueDate, ExpireDate→ DaysRemaining" IS NOT a FD, as you cannot determine the Days Remaining if you don't know the current Date.

Event(EventID: NUMBER, Name: VARCHAR(255))

Constraints:
Name is not NULL.
Primary Key Functional Dependency:
EventID → Name

Foreign Key FD:
N/A.

Candidate Key(s):
None.
[Name is not a CK, as different events may have the same name.]

Functional Dependencies:
None that are non-trivial or not covered above.

PlayerParticipatesEvent(**EventID: NUMBER**, **PlayerID: NUMBER**)

Constraints:
N/A

Primary Key Functional Dependency:
Only Trivial.

Foreign Key FD:
N/A. (The foreign keys have to be used together to get the composite primary key which gives a trivial relationship.)

Candidate Key(s):
None.

Functional Dependencies:
None that are non-trivial or not covered above.

Character(CharacterID: NUMBER, Name: VARCHAR(255), Price: NUMBER, **StoreID: NUMBER**)

Constraints:
StoreID cannot be NULL. This is for the total participation constraint. This applies to the relevant Table, as well.
Name cannot be NULL.
Price cannot be NULL.

Primary Key Functional Dependency:
CharacterID → Name, Price, StoreID

Foreign Key FD:
N/A

Candidate Key(s):
None.
[Name is NOT a CK, as some items may have the same names.]

Functional Dependencies:
None that are non-trivial or not covered above.

PlayerOwnsCharacter(**CharacterID: NUMBER**, **PlayerID: NUMBER**)

Constraints:
N/A

Primary Key Functional Dependency:
Only Trivial.

Foreign Key FD:
N/A. (FKs have to be used together to get the composite PK which has the trivial FD mentioned above.)

Candidate Key(s):

None.

Functional Dependencies:
None that are non-trivial or not covered above.

[NOTE: Total Participation of Player was not captured in DDL. As prescribed in the milestone document: we understand that these constraints cannot be modelled right now and will model them at a later time. THIS IS A REFERENCE COMMENT. IT APPLIES INSIDE THE DDL COMPONENT.]

CharacterInvolvesMatch(**CharacterID: NUMBER**, **MatchID: NUMBER**)

Constraints:
N/A

Primary Key Functional Dependency:
Only Trivial

Foreign Key FD:
N/A. (FKs have to be used together to get the composite PK which has the trivial FD mentioned above.)

Candidate Key(s):
None.

Functional Dependencies:
None that are non-trivial or not covered above.

[NOTE: Total Participation of Match was not captured in DDL. As prescribed in the milestone document: we understand that these constraints cannot be modelled right now and will model them at a later time. THIS IS A REFERENCE COMMENT. IT APPLIES INSIDE THE DDL COMPONENT.]

LuckyDraw(DrawID: NUMBER, Name: VARCHAR(255))

Constraints:
N/A.

Primary Key Functional Dependency:
DrawID → Name.

Foreign Key FD:
N/A.

Candidate Key(s):
None.
[Name is NOT a CK as it can be NULL, and names can be shared b/w LuckyDraws.]

Functional Dependencies:
None that are non-trivial or not covered above.

PlayerDrawsLuckyDraw(**DrawID: NUMBER**, **PlayerID: NUMBER**)

Constraints:
N/A.

Primary Key Functional Dependency:
Only Trivial.

Foreign Key FD:
N/A. (FKs have to be used together to get the composite PK which has the trivial FD mentioned above.)

Candidate Key(s):
None.

Functional Dependencies:
None that are non-trivial or not covered above.

LuckyDrawContainsCharacter(**DrawID: NUMBER**, **CharacterID: NUMBER**)

Constraints:
N/A

Primary Key Functional Dependency:
Only Trivial.

Foreign Key FD:
N/A. (FKs have to be used together to get the composite PK which has the trivial FD mentioned above.)

Candidate Key(s):
None.

Functional Dependencies:
None that are non-trivial or not covered above.

LuckyDrawContainsItem(**DrawID: NUMBER**, **ItemID: NUMBER**)

Constraints:
N/A

Primary Key Functional Dependency:
Only Trivial.

Foreign Key FD:
N/A. (FKs have to be used together to get the composite PK which has the trivial FD mentioned above.)

Candidate Key(s):
None.

Functional Dependencies:
None that are non-trivial or not covered above.

Store(StoreID: NUMBER, NumOfItems: NUMBER, **PlayerID: NUMBER**)

Constraints:
PlayerID must be not NULL and UNIQUE.

Primary Key Functional Dependency:
StoreID → NumOfItems, PlayerID.

Foreign Key FD:
PlayerID → StoreID
Which implies:
PlayerID → StoreID, NumOfItems.
However, the above only holds for tuples with non-null values for the attribute, PlayerID.

Candidate Key(s):
PlayerID → StoreID, NumOfItems.

Functional Dependencies:
None that are non-trivial or not covered above.

Items(ItemID: NUMBER, **StoreID: NUMBER**, Information: VARCHAR(255), OriginalPrice: NUMBER, Quality: VARCHAR(255), Discount: NUMBER, CurrentPrice: NUMBER, AppliedPromotion: VARCHAR(255))

Constraints [**UPHELD IN FINAL TABLES, AS WELL.**]:
StoreID must NOT be NULL. This is for the Total Participation Constraint and this applies to the table as well.
Information must NOT be NULL.
OriginalPrice must NOT be NULL.
Quality must NOT be NULL.
Discount must NOT be NULL.
CurrentPrice must NOT be NULL.
AppliedPromotion must NOT be NULL.

Primary Key Functional Dependency:
ItemID → StoreID, Information, OriginalPrice, Quality, Discount, CurrentPrice, AppliedPromotion

Foreign Key Functional Dependencies:
N/A
[StoreID does not have any FDs, owing to the many to one relationship (one Store can have many Items)]

Candidate Key(s):
None.
[Information is NOT a candidate key, as some may be equivalent for different instances.]

Functional Dependencies:
Quality → OriginalPrice
AppliedPromotion→ Discount
OriginalPrice, Discount → CurrentPrice

Note: For the RemainingX (where X is a number) the PKs and the FDs are indicated using the notation in class. There are no CKs in any of the RemainingX tables, and the PK and FKs are as specified.

Normalizing to BCNF:

1. Decompose with respect to Quality → OriginalPrice
    a. ItemOriginalPrice(Quality: VARCHAR(255), OriginalPrice: NUMBER)
    b. Remaining1(ItemID: NUMBER, **StoreID: NUMBER**, Information: VARCHAR(255), **Quality: VARCHAR**(255), Discount: NUMBER, CurrentPrice: NUMBER, AppliedPromotion: VARCHAR(255))
2. Decompose with respect to AppliedPromotion → Discount
    a. ItemDiscount(AppliedPromotion: VARCHAR(255), Discount: NUMBER)

    b. Remaining2(ItemID: NUMBER, **StoreID: NUMBER**, Information: VARCHAR(255), **Quality: VARCHAR**(255), CurrentPrice: NUMBER, **AppliedPromotion: VARCHAR**(255))
3. Final decomposed tables
    a. ItemOriginalPrice(Quality: VARCHAR(255), OriginalPrice: NUMBER)
        i. PK FD: Quality → Original Price.
        ii. No CKs.
        iii. No FKs.
    b. ItemDiscount(AppliedPromotion: VARCHAR(255), Discount:NUMBER)
        i. PK FD: AppliedPromotion → Discount.
        ii. No CKs.
        iii. No FKs.
    c. Items(ItemID: NUMBER, **StoreID: NUMBER**, Information: VARCHAR(255), **Quality: VARCHAR**(255), CurrentPrice: NUMBER, **AppliedPromotion: VARCHAR**(255))
        i. PK FD: ItemID → StoreID, Information, Quality, CurrentPrice, AppliedPromotion.
        ii. No CKs.
        iii. FKs: StoreID, Quality, and AppliedPromotion.

Note that the FD OriginalPrice, Discount → CurrentPrice is “broken” by normalization to BCNF.

TurnInCharacterAndMatch(TurnID: NUMBER, **CharacterID: NUMBER**, **MatchID: NUMBER**, TurnOrder: VARCHAR(255))

Constraints:
TurnOrder cannot be NULL.

Primary Key Functional Dependency:
TurnID, CharacterID, MatchID → TurnOrder.

Foreign Key FD:
N/A(FKs by themselves cannot do anything).

Candidate Key(s):
None.

Functional Dependencies:
None that are non-trivial or not covered above.

ActionInTurn(ActionID: NUMBER, **TurnID: NUMBER, CharacterID: NUMBER**, **MatchID: NUMBER**, TimeStamp: TIMESTAMP)

Constraints:
TimeStamp cannot be NULL.

Primary Key Functional Dependency:
ActionID, TurnID, CharacterID, MatchID → TimeStamp.

Foreign Key FD:
N/A (FKs by themselves do not determine anything.)

Candidate Key(s):
None.

Functional Dependencies:
None that are non-trivial or not covered above.

SkipAction(**ActionID: NUMBER**, **TurnID: NUMBER, CharacterID: NUMBER**, **MatchID: NUMBER**)

Constraints:
N/A.

Primary Key Functional Dependency:
Only Trivial.

Foreign Key FD:
N/A (Same as above: all foreign keys must be used together to have a PK.)

Candidate Key(s):
None.

Functional Dependencies:
None that are non-trivial or not covered above.

[Is an ISA relationship implemented via creating a table for the parent Entity, and tables for the child entities' sets.]

DrawActionFromDeck(**ActionID: NUMBER**, **TurnID: NUMBER, CharacterID: NUMBER**, **MatchID: NUMBER, DeckID: NUMBER**, DrawAmount: NUMBER)

Constraints:
DrawAmount cannot be NULL.

Primary Key Functional Dependency:
ActionID, TurnID, CharacterID, MatchID, DeckID → DrawAmount.

Foreign Key FD:
N/A. (Same as above: all foreign keys must be used together to have a PK.)

Candidate Key(s):
None.

Functional Dependencies:
None that are non-trivial or not covered above.

[Is an ISA relationship implemented via creating a table for the parent Entity, and tables for the child entities' sets.]

PlayAction(**ActionID: NUMBER**, **TurnID: NUMBER, CharacterID: NUMBER**, **MatchID: NUMBER**)

Primary Key Functional Dependency:
Only Trivial.

Foreign Key FD:
N/A (Same as above: all foreign keys must be used together to have a PK.)

Candidate Key(s):
None.

Functional Dependencies:
None that are non-trivial or not covered above.

[Is an ISA relationship implemented via creating a table for the parent Entity, and tables for the child entities' sets]

HandInCharacterAndMatch(**CharacterID: NUMBER, MatchID: NUMBER,** HandID: NUMBER, CardAmount: VARCHAR(255))

Constraints:
CardAmount cannot be NULL.

Primary Key Functional Dependency:
CharacterID, MatchID, HandID → CardAmount.

Foreign Key FD:
N/A. (the foreign keys by themselves cannot determine anything.)

Candidate Key(s):
None.

Functional Dependencies:
None that are non-trivial or not covered above.

Match(MatchID: NUMBER, StartTime: TIMESTAMP, EndTime: TIMESTAMP, Winner: VARCHAR(255))

Constraints:
StartTime cannot be NULL.
EndTime cannot be NULL.

Primary Key Functional Dependency:
MatchID → StartTime, EndTime, Winner.

Foreign Key FD:
N/A

Candidate Key(s):
None.

Functional Dependencies:
None that are non-trivial or not covered above.

MatchHasDeck(**MatchID: NUMBER, DeckID: NUMBER,** TimeElapsed: FLOAT)

Constraints:

MatchID is UNIQUE.
DeckID is UNIQUE.

Primary Key Functional Dependency:
MatchID, DeckID → TimeElapsed.

Foreign Key FD:
N/A. (Both FKs together have the same FD as above)

Candidate Key(s):
None.

Functional Dependencies:
None that are non-trivial or not covered above.

Deck(DeckID: NUMBER, TotalCards: NUMBER)

Constraints:
TotalCards cannot be NULL.

Primary Key Functional Dependency:
DeckID → TotalCards.

Foreign Key FD:
N/A.

Candidate Key(s):
None.

Functional Dependencies:
None that are non-trivial or not covered above.

PlayActionPlaysFromCard(**CardID: NUMBER**, **ActionID: NUMBER**, **TurnID: NUMBER,**
CharacterID: NUMBER, **MatchID: NUMBER**, **DeckID: NUMBER**)

Constraints:
N/A.

Primary Key Functional Dependency:
Only Trivial.

Foreign Key FD:
Only Trivial (all Foreign Keys are used together to identify this)

Candidate Key(s):
None.

Functional Dependencies:
None that are non-trivial or not covered above.

[NOTE: Total Participation of PlayAction was not captured in DDL. As prescribed in the milestone document: we understand that these constraints cannot be modelled right now and will model them at a later time. THIS IS A REFERENCE COMMENT. IT APPLIES INSIDE THE DDL COMPONENT.]

CardHeldByHand(**CardID: NUMBER**, **DeckID: NUMBER**, **HandID: NUMBER, CharacterID: NUMBER**, **MatchID: NUMBER**)

Constraints:
N/A.

Primary Key Functional Dependency:
Only Trivial.

Foreign Key FD:
N/A. (FKs have to be used together to get the composite PK which has the trivial FD mentioned above.)

Candidate Key(s):
None.

Functional Dependencies:
None that are non-trivial or not covered above.

CardInDeck(**DeckID: NUMBER**, CardID: NUMBER**,** HasPlayed: NUMBER)

Constraints:
HasPlayed cannot be NULL.

Primary Key Functional Dependency:
DeckID, CardID → HasPlayed.

Foreign Key FD:
N/A. (DeckID by itself has no FD.)

Candidate Key(s):
None.

Functional Dependencies:
None that are non-trivial or not covered above.

WildCard(**DeckID: NUMBER**, **CardID: NUMBER**)

Constraints:
N/A.

Primary Key Functional Dependency:
Only Trivial.

Foreign Key FD:
N/A. (Same as above: the primary key is composed of the foreign keys when used together.)

Candidate Key(s):
None.

Functional Dependencies:
None that are non-trivial or not covered above.

[Is an ISA relationship implemented via creating a table for the parent Entity, and tables for the child entities' sets]

WildDraw4Card(**DeckID: NUMBER**, **CardID: NUMBER**)

Constraints:
N/A.

Primary Key Functional Dependency:
Only Trivial.

Foreign Key FD:
N/A. (Same as above: the primary key is composed of the foreign keys when used together.)

Candidate Key(s):

None.

Functional Dependencies:
None that are non-trivial or not covered above.

[Is an ISA relationship implemented via creating a table for the parent Entity, and tables for the child entities' sets]

NumberCard(**DeckID: NUMBER**, **CardID: NUMBER**, Number: NUMBER, Color: VARCHAR(255))

Constraints:
Number cannot be NULL.
Color cannot be NULL.

Primary Key Functional Dependency:
DeckID, CardID → Number, Color.

Foreign Key FD:
N/A. (Same as above: the primary key is composed of the foreign keys when used together.)

Candidate Key(s):
None.

Functional Dependencies:
None that are non-trivial or not covered above.

[Is an ISA relationship implemented via creating a table for the parent Entity, and tables for the child entities' sets]

SkipCard(**DeckID: NUMBER**, **CardID: NUMBER**, Color: VARCHAR(255))

Constraints:
Color cannot be NULL.

Primary Key Functional Dependency:
DeckID, CardID → Color.

Foreign Key FD:
N/A. (Same as above: the primary key is composed of the foreign keys when used together.)

Candidate Key(s):
None.

Functional Dependencies:
None that are non-trivial or not covered above.

[Is an ISA relationship implemented via creating a table for the parent Entity, and tables for the child entities' sets]

ReverseCard(**DeckID: NUMBER, CardID: NUMBER**, Color: VARCHAR(255))

Constraints:
Color cannot be NULL.

Primary Key Functional Dependency:
DeckID, CardID → Color.

Foreign Key FD:
N/A. (Same as above: the primary key is composed of the foreign keys when used together.)

Candidate Key(s):
None.

Functional Dependencies:
None that are non-trivial or not covered above.

[Is an ISA relationship implemented via creating a table for the parent Entity, and tables for the child entities' sets]

Draw2Card(**DeckID: NUMBER, CardID: NUMBER**, Color: VARCHAR(255))

Constraints:
Color cannot be NULL.

Primary Key Functional Dependency:
DeckID, CardID → Color.

Foreign Key FD:
N/A. (Same as above: the primary key is composed of the foreign keys when used together.)

Candidate Key(s):
None.

Functional Dependencies:
None that are non-trivial or not covered above.

[Is an ISA relationship implemented via creating a table for the parent Entity, and tables for the child entities' sets]

Extremely important regarding naming: As we are now writing SQL, we change our naming conventions:
> For ALL attributes (except EXP) we use shift from UpperCamelCase the_underscore_naming convention (snake_case) e.g. TotalGameCount becomes total_game_count.
> EXP is made more clear here and is changed to "experience_point."

Note that we are aware that Oracle does not support ON UPDATE CASCADE, so we need to write extra codes during implementation to maintain the foreign keys. As such logic pertaining to updating has been omitted. However, it should be noted that the logic would've been inserted in places where we have ON DELETE CASCADE e.g. for the maintenance of Items held in a store or The cards held inside a player's hand (although this DOES NOT ALWAYS apply).

Also, note that we are well aware that Primary Keys are supposed to be NOT NULL. We include these keywords (especially NOT NULL as this is best practice.)

It is the case that we make liberal usage of NOT NULL: this was a design choice for the most part. However, there are instances in which it is used to meet a constraint. There are other occasions too when NOT NULL and UNIQUE are mentioned together. This would be to uphold the fact that the attribute is a Candidate Key, or to enforce a constraint modelled in the schema or in the diagram. When a constraint e.g. Total Participation in many to many is not captured it is mentioned.

NOTE: ISA CONSTRAINTS DISJOINT and COMPLETE are acknowledged. We are unable to specify/represent them at present. (This refers to Action and Card Tables as appropriate.)

These tables correspond to the schema above, however their order is unfortunately not maintained at times (wrt to the schema).

```
CREATE TABLE PlayerUsernameAndEmail (
        username VARCHAR(255) NOT NULL UNIQUE,
        email VARCHAR(255) NOT NULL UNIQUE,
        PRIMARY KEY (username)
);
```

```
CREATE TABLE PlayerGameStatistics (
        total_win NUMBER NOT NULL,
        total_game_count NUMBER NOT NULL,
        win_rate FLOAT NOT NULL,
        PRIMARY KEY (total_win, total_game_count)
);

CREATE TABLE PlayerLevel (
        experience_point NUMBER NOT NULL DEFAULT 0,
        level NUMBER NOT NULL DEFAULT 1,
        PRIMARY KEY (experience_point),
        CHECK (experience_point <= 10000)
);

CREATE TABLE PlayerLanguage (
        country VARCHAR(255) NOT NULL,
        preferred_language VARCHAR(255) NOT NULL,
        PRIMARY KEY (country)
);

CREATE TABLE Player (
        player_id NUMBER NOT NULL GENERATED BY DEFAULT AS IDENTITY,
        username VARCHAR(255) NOT NULL UNIQUE,
        total_win NUMBER NOT NULL,
        total_game_count NUMBER NOT NULL,
        experience_point NUMBER NOT NULL,
        country VARCHAR(255) NOT NULL,
        PRIMARY KEY (player_id),
        FOREIGN KEY (username) REFERENCES
                PlayerUsernameAndEmail(username)
                ON DELETE CASCADE,
        FOREIGN KEY (total_win, total_game_count) REFERENCES
                PlayerGameStatistics(total_win, total_game_count)
                ON DELETE CASCADE,
        FOREIGN KEY (experience_point) REFERENCES
                PlayerLevel(experience_point)
                ON DELETE CASCADE,
        FOREIGN KEY (country) REFERENCES
                PlayerLanguage(country)
                ON DELETE CASCADE
);

CREATE TABLE MembershipExpireDate (
```

```
        issue_date DATE  NOT NULL,
        days_remaining NUMBER NOT NULL,
        expire_date DATE NOT NULL,
        PRIMARY KEY (issue_date, days_remaining)
);

CREATE TABLE MembershipPrivilegeClass (
        privilege_level NUMBER NOT NULL,
        privilege_class VARCHAR(255) NOT NULL,
        PRIMARY KEY (privilege_level)
);

CREATE TABLE PrivilegeLevelPoint(
        total_points NUMBER NOT NULL,
        privilege_level NUMBER NOT NULL,
        PRIMARY KEY (total_points)
        FOREIGN KEY (privilege_level) REFERENCES
                MembershipPrivilegeClass(privilege_level)
                ON DELETE CASCADE
);

CREATE TABLE MembershipInPlayer (
        membership_id NUMBER NOT NULL GENERATED BY DEFAULT AS IDENTITY,
        player_id NUMBER NOT NULL UNIQUE,
        issue_date DATE  NOT NULL,
        days_remaining NUMBER NOT NULL,
        total_points NUMBER NOT NULL,
        PRIMARY KEY (membership_id),
        FOREIGN KEY (issue_date, days_remaining) REFERENCES
                MembershipExpireDate(issue_date, days_remaining)
                ON DELETE CASCADE,
        FOREIGN KEY (player_id) REFERENCES
                Player(player_id)
                ON DELETE CASCADE,
        FOREIGN KEY (total_points) REFERENCES
                PrivilegeLevelPoint(total_points)
                ON DELETE CASCADE
);

CREATE TABLE Event (
        event_id NUMBER NOT NULL GENERATED BY DEFAULT AS IDENTITY,
        name VARCHAR(255) NOT NULL,
        PRIMARY KEY (event_id)
);
```

```
CREATE TABLE LuckyDraw (
        draw_id NUMBER NOT NULL GENERATED BY DEFAULT AS IDENTITY,
        name VARCHAR(255),
        PRIMARY KEY (draw_id)
);

CREATE TABLE Store (
        store_id NUMBER NOT NULL GENERATED BY DEFAULT AS IDENTITY,
        player_id NUMBER NOT NULL UNIQUE,
        num_of_items NUMBER,
        PRIMARY KEY (store_id),
        FOREIGN KEY (player_id) REFERENCES Player(player_id)
                ON DELETE CASCADE
);

CREATE TABLE ItemOriginalPrice (
        quality VARCHAR(255) NOT NULL,
        original_price NUMBER NOT NULL,
        PRIMARY KEY (quality)
);

CREATE TABLE ItemDiscount (
        applied_promotion VARCHAR(255) NOT NULL,
        discount NUMBER NOT NULL,
        PRIMARY KEY (applied_promotion)
);

CREATE TABLE Item (
        item_id NUMBER NOT NULL GENERATED BY DEFAULT AS IDENTITY,
        store_id NUMBER NOT NULL,
        current_price NUMBER NOT NULL,
        information VARCHAR(255) NOT NULL,
        quality VARCHAR(255) NOT NULL,
        applied_promotion VARCHAR(255),
        PRIMARY KEY (item_id),
        FOREIGN KEY (quality) REFERENCES
                ItemOriginalPrice(quality)
                ON DELETE CASCADE,
        FOREIGN KEY (applied_promotion) REFERENCES
                ItemDiscount(applied_promotion)
                ON DELETE CASCADE,
        FOREIGN KEY (store_id) REFERENCES
                Store(store_id)
```

```
        ON DELETE CASCADE
);

CREATE TABLE Character (
    character_id NUMBER NOT NULL GENERATED BY DEFAULT AS IDENTITY,
    store_id NUMBER NOT NULL,
    name VARCHAR(255) NOT NULL,
    price NUMBER NOT NULL,
    PRIMARY KEY (character_id),
    FOREIGN KEY (store_id) REFERENCES Store(store_id)
        ON DELETE CASCADE
);

CREATE TABLE Match (
    match_id NUMBER NOT NULL GENERATED BY DEFAULT AS IDENTITY,
    winner VARCHAR(255),
    start_time TIMESTAMP DEFAULT CURRENT_TIMESTAMP NOT NULL,
    end_time TIMESTAMP DEFAULT CURRENT_TIMESTAMP NOT NULL,
    PRIMARY KEY (match_id)
);

CREATE TABLE Deck (
    deck_id NUMBER NOT NULL GENERATED BY DEFAULT AS IDENTITY,
    total_cards NUMBER NOT NULL,
    PRIMARY KEY (deck_id)
);

CREATE TABLE CardInDeck (
    card_id NUMBER NOT NULL,
    deck_id NUMBER NOT NULL,
    has_played NUMBER NOT NULL DEFAULT 0,
    PRIMARY KEY (card_id, deck_id),
    FOREIGN KEY (deck_id) REFERENCES Deck(deck_id)
        ON DELETE CASCADE
);

CREATE TABLE WildCard (
    card_id NUMBER NOT NULL,
    deck_id NUMBER NOT NULL,
    PRIMARY KEY (card_id, deck_id),
    FOREIGN KEY (card_id, deck_id) REFERENCES CardInDeck(card_id, deck_id)
        ON DELETE CASCADE
);
```

```
CREATE TABLE WildDraw4Card (
        card_id NUMBER NOT NULL,
        deck_id NUMBER NOT NULL,
        PRIMARY KEY (card_id, deck_id),
        FOREIGN KEY (card_id, deck_id) REFERENCES CardInDeck(card_id, deck_id)
                ON DELETE CASCADE
);

CREATE TABLE NumberCard (
        card_id NUMBER NOT NULL,
        deck_id NUMBER NOT NULL,
        number NUMBER NOT NULL,
        color VARCHAR2(255) NOT NULL,
        PRIMARY KEY (card_id, deck_id),
        FOREIGN KEY (card_id, deck_id) REFERENCES CardInDeck(card_id, deck_id)
                ON DELETE CASCADE
);

CREATE TABLE SkipCard (
        card_id NUMBER NOT NULL,
        deck_id NUMBER NOT NULL,
        color VARCHAR2(255) NOT NULL,
        PRIMARY KEY (card_id, deck_id),
        FOREIGN KEY (card_id, deck_id) REFERENCES CardInDeck(card_id, deck_id)
                ON DELETE CASCADE
);

CREATE TABLE ReverseCard (
        card_id NUMBER NOT NULL,
        deck_id NUMBER NOT NULL,
        color VARCHAR2(255) NOT NULL,
        PRIMARY KEY (card_id, deck_id),
        FOREIGN KEY (card_id, deck_id) REFERENCES CardInDeck(card_id, deck_id)
                ON DELETE CASCADE
);

CREATE TABLE Draw2Card (
        card_id NUMBER NOT NULL,
        deck_id NUMBER NOT NULL,
        color VARCHAR2(255) NOT NULL,
        PRIMARY KEY (card_id, deck_id),
        FOREIGN KEY (card_id, deck_id) REFERENCES CardInDeck(card_id, deck_id)
                ON DELETE CASCADE
);
```

```
CREATE TABLE HandInCharacterAndMatch (
        hand_id NUMBER NOT NULL,
        character_id NUMBER NOT NULL,
        match_id  NUMBER NOT NULL,
        card_amount NUMBER NOT NULL,
        PRIMARY KEY (hand_id, character_id, match_id),
        FOREIGN KEY (character_id) REFERENCES Character(character_id)
                ON DELETE CASCADE,
        FOREIGN KEY (match_id) REFERENCES Match(match_id)
                ON DELETE CASCADE
);

CREATE TABLE TurnInCharacterAndMatch (
        turn_id NUMBER NOT NULL,
        character_id NUMBER NOT NULL,
        match_id  NUMBER NOT NULL,
        turn_order VARCHAR2(255) NOT NULL,
        PRIMARY KEY (turn_id, character_id, match_id),
        FOREIGN KEY (character_id) REFERENCES Character(character_id)
                ON DELETE CASCADE,
        FOREIGN KEY (match_id) REFERENCES Match(match_id)
                ON DELETE CASCADE
);

CREATE TABLE ActionInTurn (
        action_id NUMBER NOT NULL,
        turn_id NUMBER NOT NULL,
        character_id NUMBER NOT NULL,
        match_id  NUMBER NOT NULL,
        time_stamp TIMESTAMP DEFAULT CURRENT_TIMESTAMP NOT NULL,
        PRIMARY KEY (action_id, turn_id, character_id, match_id),
        FOREIGN KEY (turn_id, character_id, match_id) REFERENCES
                TurnInCharacterAndMatch(turn_id, character_id, match_id)
                ON DELETE CASCADE
);

CREATE TABLE SkipAction (
        action_id NUMBER NOT NULL,
        turn_id NUMBER NOT NULL,
        character_id NUMBER NOT NULL,
        match_id  NUMBER NOT NULL,
        PRIMARY KEY (action_id, turn_id, character_id, match_id),
        FOREIGN KEY (action_id, turn_id, character_id, match_id) REFERENCES
```

```
                ActionInTurn(action_id, turn_id, character_id, match_id)
                ON DELETE CASCADE
);

CREATE TABLE DrawActionFromDeck (
        action_id NUMBER NOT NULL,
        turn_id NUMBER NOT NULL,
        character_id NUMBER NOT NULL,
        match_id  NUMBER NOT NULL,
        draw_amount NUMBER NOT NULL,
        PRIMARY KEY (action_id, turn_id, character_id, match_id),
        FOREIGN KEY (action_id, turn_id, character_id, match_id) REFERENCES
                ActionInTurn(action_id, turn_id, character_id, match_id)
                ON DELETE CASCADE
);

CREATE TABLE PlayAction (
        action_id NUMBER NOT NULL,
        turn_id NUMBER NOT NULL,
        character_id NUMBER NOT NULL,
        match_id  NUMBER NOT NULL,
        PRIMARY KEY (action_id, turn_id, character_id, match_id),
        FOREIGN KEY (action_id, turn_id, character_id, match_id) REFERENCES
                ActionInTurn(action_id, turn_id, character_id, match_id)
                ON DELETE CASCADE
);

CREATE TABLE PlayerParticipatesEvent (
        player_id NUMBER NOT NULL,
        event_id NUMBER NOT NULL,
        PRIMARY KEY (player_id, event_id),
        FOREIGN KEY (player_id) REFERENCES Player(player_id)
                ON DELETE CASCADE,
        FOREIGN KEY (event_id) REFERENCES Event(event_id)
                ON DELETE CASCADE
);

CREATE TABLE PlayerOwnsCharacter (
        player_id NUMBER NOT NULL,
        character_id NUMBER NOT NULL,
        PRIMARY KEY (player_id, character_id),
        FOREIGN KEY (player_id) REFERENCES Player(player_id)
                ON DELETE CASCADE,
        FOREIGN KEY (character_id) REFERENCES Character(character_id)
```

```
        ON DELETE CASCADE
);
/*
NOTE: Total Participation of Player was not captured. As prescribed in the milestone document: we understand that these constraints cannot be modelled right now and will model them at a later time.
*/

CREATE TABLE PlayerDrawsLuckyDraw (
    player_id NUMBER NOT NULL,
    draw_id NUMBER NOT NULL,
    PRIMARY KEY (player_id, draw_id),
    FOREIGN KEY (player_id) REFERENCES Player(player_id)
        ON DELETE CASCADE,
    FOREIGN KEY (draw_id) REFERENCES LuckyDraw(draw_id)
        ON DELETE CASCADE
);

CREATE TABLE LuckyDrawContainsCharacter (
    draw_id NUMBER NOT NULL,
    character_id NUMBER NOT NULL,
    PRIMARY KEY (draw_id, character_id),
    FOREIGN KEY (draw_id) REFERENCES LuckyDraw(draw_id)
        ON DELETE CASCADE,
    FOREIGN KEY (character_id) REFERENCES Character(character_id)
        ON DELETE CASCADE
);

CREATE TABLE LuckyDrawContainsItem (
    draw_id NUMBER NOT NULL,
    item_id NUMBER NOT NULL,
    PRIMARY KEY (draw_id, item_id),
    FOREIGN KEY (draw_id) REFERENCES LuckyDraw(draw_id)
        ON DELETE CASCADE,
    FOREIGN KEY (item_id) REFERENCES Item(item_id)
        ON DELETE CASCADE
);

CREATE TABLE CharacterInvolvesMatch (
    character_id NUMBER NOT NULL,
    match_id NUMBER NOT NULL,
    PRIMARY KEY (character_id, match_id),
    FOREIGN KEY (character_id) REFERENCES Character(character_id)
        ON DELETE CASCADE,
    FOREIGN KEY (match_id) REFERENCES Match(match_id)
```

```
                ON DELETE CASCADE
);
/*
NOTE: Total Participation of Match was not captured in DDL. As prescribed in the milestone document:
we understand that these constraints cannot be modelled right now and will model them at a later time.
*/

CREATE TABLE MatchHasDeck (
        match_id NUMBER NOT NULL UNIQUE,
        deck_id NUMBER NOT NULL UNIQUE,
        time_elapsed FLOAT,
        PRIMARY KEY (match_id, deck_id),
        FOREIGN KEY (match_id) REFERENCES Match(match_id)
                ON DELETE CASCADE,
        FOREIGN KEY (deck_id) REFERENCES Deck(deck_id)
                ON DELETE CASCADE
);

CREATE TABLE CardHeldByHand (
        card_id NUMBER NOT NULL,
        deck_id NUMBER NOT NULL,
        hand_id NUMBER NOT NULL,
        character_id NUMBER NOT NULL,
        match_id  NUMBER NOT NULL,
        PRIMARY KEY (card_id, deck_id, hand_id, character_id, match_id),
        FOREIGN KEY (card_id, deck_id) REFERENCES
                CardInDeck(card_id, deck_id)
                ON DELETE CASCADE,
        FOREIGN KEY (hand_id, character_id, match_id) REFERENCES
                HandInCharacterAndMatch(hand_id, character_id, match_id)
                ON DELETE CASCADE
);

CREATE TABLE PlayActionPlaysFromCard (
        action_id NUMBER NOT NULL,
        turn_id NUMBER NOT NULL,
        character_id NUMBER NOT NULL,
        match_id  NUMBER NOT NULL,
        card_id NUMBER NOT NULL,
        deck_id NUMBER NOT NULL,
        PRIMARY KEY (action_id, turn_id, character_id, match_id, card_id, deck_id),
        FOREIGN KEY (action_id, turn_id, character_id, match_id) REFERENCES
                PlayAction(action_id, turn_id, character_id, match_id)
                ON DELETE CASCADE,
```

```
        FOREIGN KEY (card_id, deck_id) REFERENCES
                CardInDeck(card_id, deck_id)
                ON DELETE CASCADE
);
/*
NOTE: Total Participation of Player was not captured in DDL. As prescribed in the milestone document:
we understand that these constraints cannot be modelled right now and will model them at a later time.
*/
```

Note: Please note that we leverage syntactic sugar and thus for some Insertions, a value is auto-specified, meaning we may skip it in the tuple.

```
INSERT INTO PlayerUsernameAndEmail (username, email)
VALUES
    ('Handsome Programmer', 'handsome_programmer@student.ubc.ca'),
    ('Happy Professor', 'happy_professor@student.ubc.ca'),
    ('Poor Student', 'poor_student@student.ubc.ca'),
    ('Selfish Engineer', 'selfish_engineer@student.ubc.ca'),
    ('Dangerous Salesman', 'dangerous_salesman@student.ubc.ca');

INSERT INTO PlayerGameStatistics (total_win, total_game_count, win_rate)
VALUES
    (10, 20, 0.51),
    (15, 25, 0.63),
    (8, 10, 0.85),
    (12, 13, 0.92),
    (20, 27, 0.74);

INSERT INTO PlayerLevel (experience_point, level)
VALUES
    (3000, 3),
    (5000, 5),
    (8000, 8),
    (9000, 9),
    (10000, 10);

INSERT INTO PlayerLanguage (country, preferred_language)
VALUES
    ('USA', 'English'),
    ('Canada', 'English'),
    ('France', 'French'),
    ('China', 'Chinese'),
    ('Korea', 'Korean');

INSERT INTO Player (username, total_win, total_game_count, experience_point, country)
VALUES
    ('Handsome Programmer', 10, 20, 3000, 'Canada'),
    ('Happy Professor', 15, 25, 5000, 'France'),
    ('Poor Student', 8, 10, 8000, 'China'),
    ('Selfish Engineer', 12, 13, 9000, 'USA'),
    ('Dangerous Salesman', 20, 27, 10000, 'Korea');

INSERT INTO MembershipExpireDate (issue_date, days_remaining, expire_date)
```

```
VALUES
    (DATE '2024-01-01', 30, DATE '2024-01-31'),
    (DATE '2024-02-01', 28, DATE '2024-02-29'),
    (DATE '2024-03-01', 31, DATE '2024-03-31'),
    (DATE '2024-04-01', 30, DATE '2024-04-30'),
    (DATE '2024-05-01', 31, DATE '2024-05-31');

INSERT INTO MembershipPrivilegeClass (privilege_level, privilege_class)
VALUES
    (1, 'Bronze'),
    (2, 'Silver'),
    (3, 'Gold'),
    (4, 'Platinum'),
    (5, 'Diamond');

INSERT INTO PrivilegeLevelPoint (total_points, privilege_level)
VALUES
    (100, 1),
    (200, 2),
    (300, 3),
    (400, 4),
    (500, 5);

INSERT INTO MembershipInPlayer (player_id, issue_date, days_remaining, total_points)
VALUES
    (1, DATE '2024-01-01', 30, 100),
    (2, DATE '2024-02-01', 28, 200),
    (3, DATE '2024-03-01', 31, 300),
    (4, DATE '2024-04-01', 30, 400),
    (5, DATE '2024-05-01', 31, 500);

INSERT INTO Event (name)
VALUES
    ('Grand Tournament'),
    ('Weekly Challenge'),
    ('Holiday Special'),
    ('Friendship Match'),
    ('Ultimate Showdown');

INSERT INTO LuckyDraw (name)
VALUES
    ('New Year Draw'),
    ('Summer Surprise'),
    ('Winter Wonderland'),
```

```
        ('Spring Fling'),
        ('Autumn Adventure');

INSERT INTO Store (player_id, num_of_items)
VALUES
        (1, 10),
        (2, 15),
        (3, 8),
        (4, 12),
        (5, 20);

INSERT INTO ItemOriginalPrice (quality, original_price)
VALUES
        ('Common', 188),
        ('Uncommon', 288),
        ('Rare', 488),
        ('Epic', 888),
        ('Legendary', 1688);

INSERT INTO ItemDiscount (applied_promotion, discount)
VALUES
        ('New Year Sale', 35),
        ('Summer Sale', 20),
        ('Winter Sale', 15),
        ('Spring Sale', 25),
        ('Autumn Sale', 30);

INSERT INTO Item (store_id, current_price, information, quality, applied_promotion)
VALUES
        (1, 122, 'Red Card', 'Common', 'New Year Sale'),
        (2, 187, 'Blue Card', 'Uncommon', 'New Year Sale'),
        (3, 317, 'Green Card', 'Rare', 'New Year Sale'),
        (4, 577, 'Yellow Card', 'Epic', 'New Year Sale'),
        (5, 1097, 'Wild Card', 'Legendary', 'New Year Sale');

INSERT INTO Character (store_id, name, price)
VALUES
        (1, 'Yuchen', 1688),
        (2, 'Perry', 1688),
        (3, 'Daud', 1688),
        (4, 'Joy', 888),
        (5, 'Hazel', 488);

INSERT INTO Match (winner)
```

```
VALUES
        ('Handsome Programmer'),
        ('Happy Professor'),
        ('Poor Student'),
        ('Selfish Engineer'),
        ('Dangerous Salesman');

INSERT INTO Deck (total_cards)
VALUES
        (108),
        (108),
        (108),
        (108),
        (108);

INSERT INTO CardInDeck (card_id, deck_id, has_played)
VALUES
        (1, 1, 0),
        (2, 1, 0),
        (3, 1, 0),
        (4, 1, 0),
        (5, 1, 0),
        (6, 2, 0),
        (7, 2, 0),
        (8, 2, 0),
        (9, 2, 0),
        (10, 2, 0);

INSERT INTO WildCard (card_id, deck_id)
VALUES
        (1, 1),
        (2, 1),
        (3, 1),
        (4, 1),
        (5, 1);

INSERT INTO WildDraw4Card (card_id, deck_id)
VALUES
        (6, 2),
        (7, 2),
        (8, 2),
        (9, 2),
        (10, 2);
```

```
INSERT INTO NumberCard (card_id, deck_id, number, color)
VALUES
        (1, 1, 1, 'Red'),
        (2, 1, 2, 'Blue'),
        (3, 1, 3, 'Green'),
        (4, 1, 4, 'Yellow'),
        (5, 1, 5, 'Red');

INSERT INTO SkipCard (card_id, deck_id, color)
VALUES
        (6, 2, 'Red'),
        (7, 2, 'Blue'),
        (8, 2, 'Green'),
        (9, 2, 'Yellow'),
        (10, 2, 'Red');

INSERT INTO ReverseCard (card_id, deck_id, color)
VALUES
        (1, 1, 'Red'),
        (2, 1, 'Blue'),
        (3, 1, 'Green'),
        (4, 1, 'Yellow'),
        (5, 1, 'Red');

INSERT INTO Draw2Card (card_id, deck_id, color)
VALUES
        (6, 2, 'Red'),
        (7, 2, 'Blue'),
        (8, 2, 'Green'),
        (9, 2, 'Yellow'),
        (10, 2, 'Red');

INSERT INTO HandInCharacterAndMatch (hand_id, character_id, match_id, card_amount)
VALUES
        (1, 1, 1, 7),
        (2, 2, 2, 7),
        (3, 3, 3, 7),
        (4, 4, 4, 7),
        (5, 5, 5, 7);

INSERT INTO TurnInCharacterAndMatch (turn_id, character_id, match_id, turn_order)
VALUES
        (1, 1, 1, 'clockwise'),
        (2, 2, 2, 'clockwise'),
```

```
    (3, 3, 3, 'clockwise'),
    (4, 4, 4, 'counter clockwise'),
    (5, 5, 5, 'counter clockwise');

INSERT INTO ActionInTurn (action_id, turn_id, character_id, match_id)
VALUES
    (1, 1, 1, 1),
    (2, 2, 2, 2),
    (3, 3, 3, 3),
    (4, 4, 4, 4),
    (5, 5, 5, 5);

INSERT INTO SkipAction (action_id, turn_id, character_id, match_id)
VALUES
    (1, 1, 1, 1),
    (2, 2, 2, 2),
    (3, 3, 3, 3),
    (4, 4, 4, 4),
    (5, 5, 5, 5);

INSERT INTO DrawActionFromDeck (action_id, turn_id, character_id, match_id, draw_amount)
VALUES
    (1, 1, 1, 1, 2),
    (2, 2, 2, 2, 2),
    (3, 3, 3, 3, 2),
    (4, 4, 4, 4, 2),
    (5, 5, 5, 5, 2);


INSERT INTO PlayerParticipatesEvent (player_id, event_id)
VALUES
    (1, 1),
    (2, 1),
    (3, 1),
    (4, 1),
    (5, 1);

INSERT INTO PlayerOwnsCharacter (player_id, character_id)
VALUES
    (1, 1),
    (2, 2),
    (3, 3),
    (4, 4),
    (5, 5);
```

```
INSERT INTO PlayerDrawsLuckyDraw (player_id, draw_id)
VALUES
        (1, 1),
        (2, 1),
        (3, 1),
        (4, 1),
        (5, 1);

INSERT INTO LuckyDrawContainsCharacter (draw_id, character_id)
VALUES
        (1, 1),
        (1, 2),
        (1, 3),
        (1, 4),
        (1, 5);

INSERT INTO LuckyDrawContainsItem (draw_id, item_id)
VALUES
        (1, 1),
        (1, 2),
        (1, 3),
        (1, 4),
        (1, 5);

INSERT INTO CharacterInvolvesMatch (character_id, match_id)
VALUES
        (1, 1),
        (2, 2),
        (3, 3),
        (4, 4),
        (5, 5);

INSERT INTO MatchHasDeck (match_id, deck_id, time_elapsed)
VALUES
        (1, 1, 200),
        (2, 2, 300),
        (3, 3, 400),
        (4, 4, 250),
        (5, 5, 550);

INSERT INTO CardHeldByHand (card_id, deck_id, hand_id, character_id, match_id)
VALUES
        (1, 1, 1, 1, 1),
```

```
    (2, 1, 2, 2, 2),
    (3, 1, 3, 3, 3),
    (4, 1, 4, 4, 4),
    (5, 1, 5, 5, 5);

INSERT INTO PlayActionPlaysFromCard (action_id, turn_id, character_id, match_id, card_id, deck_id)
VALUES
    (1, 1, 1, 1, 1, 1),
    (2, 2, 2, 2, 2, 1),
    (3, 3, 3, 3, 3, 1),
    (4, 4, 4, 4, 4, 1),
    (5, 5, 5, 5, 5, 1);
```
