Subrahmanya Gaonkar

2022UG000072

DBMS

Prof Dr Kiran D C

**PUBG Mobile Match-Making Database**

**Problem Definition-**

Matchmaking is a critical component of the gaming experience in PUBG Mobile, ensuring that players are matched with opponents of similar skill levels and preferences to create balanced and enjoyable matches. Details are considered during the matchmaking process in PUBG Mobile, highlighting the key player details necessary for creating fair and engaging gameplay experiences. Designing an enhanced matchmaking system for PUBG Mobile that considers Account level, F/D ratio (finish per death), region, merit, previous match performance, language preference, tier, MMR, season rank and rating, badges etc... and dynamic adjustments to create balanced and enjoyable matches tailored to players' skill levels and preferences.

**Requirements-**

1. Account Level:

Requirement: The database should store each player's account level.

The account level represents a player's overall experience and progress within the game. It indicates the time and effort invested by the player and can be used as a factor in matchmaking to pair players of similar experience levels together.

1. F/D Ratio (Finish per Death):

Requirement: The database should track each player's F/D ratio.

The F/D ratio reflects a player's effectiveness in the game by measuring the number of matches a player wins or reaches a high-ranking position compared to the number of times they are eliminated. It helps assess a player's skill level and can be considered in matchmaking to create balanced matches.

1. Region:

Requirement: The database should record the region of each player. Matching players based on their geographical regions helps reduce network latency and ensures smoother gameplay experiences. The database needs to store this information to facilitate region-based matchmaking.

1. Merit:

Requirement: The database should maintain a record of each player's merit points. Merit points are awarded or deducted based on a player's behavior and performance in matches. They serve as a measure of a player's sportsmanship and reliability. Considering merit points in matchmaking helps pair players with similar conduct and gameplay attitudes.

1. Previous Match Performance:

Requirement: The database should store data on each player's performance in previous matches. Analyzing a player's performance in previous matches, including factors such as kills, damage dealt, survival time, and placement, helps determine their skill level and matchmaking suitability.

1. Language Preference:

Requirement: The database should include each player's language preference. Language preference ensures effective communication among teammates during matches. Storing language preferences allows the matchmaking system to prioritize matching players who speak the same language, enhancing coordination and teamwork.

1. Tier and MMR (Matchmaking Rating):

Requirement: The database should maintain records of each player's tier and MMR. Tiers and MMR reflect a player's skill level and competitive ranking. Incorporating tier and MMR data in matchmaking ensures that players are matched with opponents of similar skill levels, resulting in balanced and competitive matches.

1. Season Rank and Rating:

Requirement: The database should track each player's season rank and rating. Season rank and rating provide insights into a player's performance and progression within a specific season. Considering season rank and rating in matchmaking helps ensure fair matchups and promotes competitive gameplay.

1. Badges:

Requirement: The database should store information about the badges earned by each player. Explanation: Badges signify various achievements and accomplishments attained by players. Including badge data in matchmaking allows for additional criteria to be considered when pairing players, promoting fairness and recognition of players' accomplishments.

**Entities, Attributes, and types of values it can have:**

1. Player (Owner entity)

Attributes: Pid, In-game name, email, account level (Derived attribute), merit, clan name, evo level, achievement points, title (multi valued attribute), avatar (multi-valued attribute), Likes, popularity.

1. Select relation: The "Select" relation between "Player Account" and "Game Mode" entities represents the choices made by players regarding their gameplay preferences. It indicates whether players have opted for certain settings related to the game mode they want to play.
2. Game mode:

Attributes: Game mode id (primary key), Mode, Sub mode, Map, Event status, Player type, Perspective, Auto matching

Values of Attributes:

* Mode:
* Ranked
* Unranked
* Classic mode: Livik after math or Nusa or Livik or Erangel
* Map:
* TDM:
* Warehouse
* Hangar
* Arena training
* Warehouse
* Hangar
* Gun game
* Hangar
* Library
* Ultimate Arena
* Erangel
* Livik
* Royale Arena
* Assault
* Erangel
* Livik
* Sanhok
* Santorini
* 8 v 8 training
* 8 v 8 TDM
* Domination
* Assault
* Others
* VS AI: Easy or Normal or Hard
* Payload
* Quick match
* Sniper training
* War
* Sub mode:
* Classic
* Arena
* Sub mode- Classic - Maps:
* Erangel
* Sanhok
* Livik
* Nusa
* Miramar
* Vikendi
* Karakin
* Event status:
* Event name
* Player type
* Solo
* Duo
* Squad
* Perspective
* TPP
* FPP
* Auto-Matching
* Yes
* No

1. Matching relation:

The "Matching" relation indicates which game mode was selected for each match. It allows us to determine the characteristics of the match, such as the mode (e.g., Ranked, Unranked, Arena), sub-mode, map, and event status. This relation helps in organizing and categorizing matches based on their respective game modes, facilitating efficient data retrieval and analysis related to match statistics and player performance within each mode.

1. Match:

Attributes: Match id, Ranking, Rating (Derived attribute), Duration (Derived attribute), Date (compound attribute – Date, Month, Year), BP, RP, Grade (Derived attribute), In time, out time,

1. Generates relation:

The "Generates" relation between the "Match" entity and the "Player Stats" entity signifies that each match played by a player results in the generation or update of statistical data stored in the "Player Stats" entity.

1. Player Stats: (Weak entity):

Attributes: Pid (Partial key), Season (multi-valued attribute), Matches played.

1. Server:

Server name, province, city.

1. Has relation:

"has" relation signifies that each instance of the "Server" entity may be associated with one or more instances of the "Player Stats" entity. This association implies that the statistical data related to player performance (stored in "Player Stats") is linked to or associated with the specific server where the gameplay occurred (represented by the "Server" entity). For example, a particular match played on a specific server would generate statistical data (such as kills, damage dealt, survival time, etc.) for each player participating in that match. These statistics would be captured in the "Player Stats" entity and associated with the respective server through the "has" relation.

1. Plays on:

The "Plays On" relation between the "Server" and "Player Account" entities represents the association between player accounts and the game servers they play on. This relation signifies that each player's account is associated with a specific game server where they participate in matches or gameplay sessions.

**Functional Dependency:**

1. Player (Owner entity):
   * Functional Dependencies:
     + Pid 🡪 In-game name, email, account level, merit, clan name, evo level, achievement points, title, avatar, Likes, popularity
     + Achievement points, Evo level 🡪 Account level
   * The player's unique identifier (Pid) determines all other attributes associated with the player. Each player's in-game name, email, account level, merit, clan name, etc., are uniquely identified by their Pid.
   * But the combination of Achievement points and Evo level determines the Account level. For example, specific achievements and evolution progress may contribute to an overall account level in the system.
2. Game mode:
   * Functional Dependencies:
     + Game mode id -> Mode, Sub mode, Map, Event status, Player type, Perspective, Auto matching
     + Mode, Sub mode 🡪 Map
   * Each game mode ID uniquely determines the mode, sub-mode, map, event status, player type, perspective, and auto-matching settings associated with that game mode.
   * given a specific combination of Mode and Sub-mode, there is a unique associated Map. This functional dependency implies that the choice of game mode and its subcategory determines the specific map to be used in that game mode.

3. Match:

* + Functional Dependencies:
  + Match id 🡪 Ranking, Rating, Duration, Date, BP, RP, Grade, In time, out time,
    - In time, out time 🡪Duration
  + Each match ID uniquely determines all other attributes associated with that match, including ranking, rating, duration, date, BP (Battle Points), RP (Royale Points), grade, in-time, out-time, and game mode ID.
  + This functional dependency indicates that the values of In time and Out time together determine the Duration of the match. In other words, the duration of a match can be calculated based on the difference between the In-time and Out-time values.

1. Player Stats (Weak entity):

* Functional Dependencies:
* (Pid, Season [Multi valued attribute]) 🡪 Matches played,
* The combination of player ID and season uniquely determines the matches played, game mode ID, and match ID associated with that player's statistics for that season.

1. Server:

* Server:
* Functional Dependencies:
* City🡪Province, Server Name
* Here City alone uniquely determines the Province and Server name, serving as the primary key for the Server entity.

**Candidate Keys & Primary Keys:**

1. Player:

* Candidate Key(s): {Pid}, {In game name}, {email}
* Primary Key: Pid

1. Game mode:

* Candidate Key(s): {Game mode id}
* Primary Key: Game mode id

1. Match:

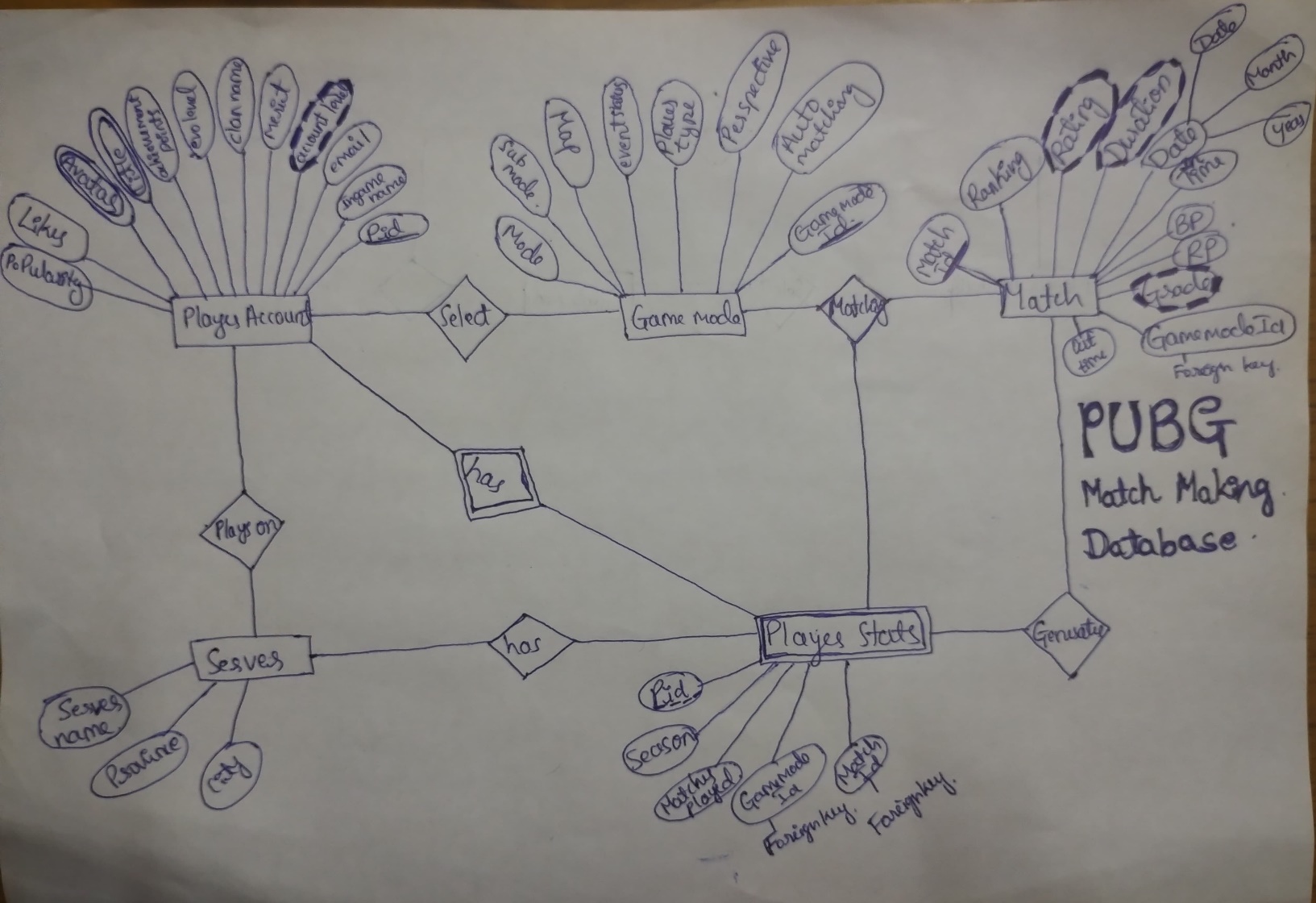
* Candidate Key(s): {Match id}
* Primary Key: Match ID

1. Player Stats:

* Candidate Key(s): {(Pid, Season)}
* Primary Key: (Pid, Season)

1. Server:

* Candidate Key(s): {(City)}
* Primary Key: (City)



**Normalization:**

1. Player Account table:

* **1NF:**

To address the issue of multivalued attributes (title and avatar) in the Player table and ensure compliance with the 1NF (First Normal Form), I decomposed the table to avoid redundancy and anomalies. Here's how I did it:

* Original Player account table:

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Pid | In  game  name | Email | Account  Level | Merit | Clan  name | Evo  level | Achievement  points | Title | Avatar | Likes | Popularity |
|  |  |  |  |  |  |  |  |  |  |  |  |

Decomposed Player table:

Multivalued attributes (Title and Avatar) have been removed from this table to adhere to the 1NF.

* Player (Primary key: Pid)

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Pid | In  game  name | Email | Account  Level | Merit | Clan  name | Evo  level | Achievement  points | Likes | Popularity |
|  |  |  |  |  |  |  |  |  |  |

* Player Title table (primary key: Pid, Title):

This table contains the relationship between player IDs (Pid) and their respective titles. The composite primary key (Pid, Title) ensures uniqueness for each player's title

|  |  |
| --- | --- |
| Pid | Title |
|  |  |

* Player Avatar (primary key: Pid, Avatar):

This table contains the relationship between player IDs (Pid) and their respective avatars.

The composite primary key (Pid, Avatar) ensures uniqueness for each player's avatar.

|  |  |
| --- | --- |
| Pid | Avatar |
|  |  |

* **2NF:**

each non-key attribute is fully functionally dependent on the primary key, without any partial dependencies.

Player table:

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Pid | In  game  name | Email | Account  Level | Merit | Clan  name | Evo  level | Achievement  points | Likes | Popularity |
|  |  |  |  |  |  |  |  |  |  |

* The "Account level" attribute is dependent on both "Achievement points" and "Evo level," violating the 2NF because it is not directly related to the primary key (Pid).
* To adhere to the 2NF, we need to decompose the table to ensure that each non-key attribute is dependent only on the primary key.

Decomposed Tables to Achieve 2NF:

* Player (Primary Key: Pid):

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Pid | In  game  name | Email | Account  Level | Merit | Clan  name | Likes | Popularity |
|  |  |  |  |  |  |  |  |

* Player Achievements (Primary Key: Pid):

|  |  |
| --- | --- |
| Pid | Achievement points |
|  |  |

* Player Evolevel (Primary Key: Pid):

|  |  |
| --- | --- |
| Pid | Evo level |
|  |  |

* I maintained the "Account level" attribute in the original Player table.
* The Player table now contains all attributes directly related to the primary key (Pid), including the "Account level" attribute.
* **3NF:**

To achieve 3NF, we need to ensure that all non-key attributes are independent of each other within each table.

* The "Account level" is determined by other attributes like "Achievement points" or "Evo level,” but I separated them from the original table where only the account level is remaining now in the "Player" table.

So, I didn’t separate the account level from the table.

* Here “Account level" attribute is functionally dependent solely on the primary key (Pid) and not on any other attributes in the table, so it remains in the original "Player" table.
* Player (Primary Key: Pid):

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Pid | In  game  name | Email | Account  Level | Merit | Clan  name | Likes | Popularity |
|  |  |  |  |  |  |  |  |

* Player Achievements (Primary Key: Pid):

|  |  |
| --- | --- |
| Pid | Achievement points |
|  |  |

* Player Evolevel (Primary Key: Pid):

|  |  |
| --- | --- |
| Pid | Evo level |
|  |  |

* **BCNF:**

To check for the applicability of the Boyce-Codd Normal Form (BCNF) and decompose the tables accordingly, I need to analyze the functional dependencies and ensure that each determinant in a table is a candidate key. So, I began by examining the original tables:

* The "Player" table's primary key is Pid, and there are no composite candidate keys.
* The "Player Achievements" and "Player Evolevel" tables also have Pid as the primary key.
* Since each table has a single candidate key, we need to ensure that all non-key attributes in each table are functionally dependent only on the candidate key.

Decomposed Tables to Achieve BCNF:

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Pid | In  game  name | Email | Merit | Clan  name | Likes | Popularity |
|  |  |  |  |  |  |  |

* I decomposed the "Player" table further to create a separate table, "Account Level," to store the "Account level" attribute.
* This ensures that all non-key attributes in each table are functionally dependent only on the candidate key (Pid).
* Each table now satisfies the Boyce-Codd Normal Form (BCNF) by ensuring that each determinant is a candidate key.
* Account Level (Primary Key: Pid):

|  |  |
| --- | --- |
| Pid | Account Level |
|  |  |

* Achievement points (Primary Key: Pid):

|  |  |
| --- | --- |
| Pid | Achievement points |
|  |  |

* Evo Level (Primary Key: Pid):

|  |  |
| --- | --- |
| Pid | Evo Level |
|  |  |

* So, after applying normalization up to BCNF the decomposed table are:

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Pid | In  game  name | Email | Merit | Clan  name | Likes | Popularity |
|  |  |  |  |  |  |  |

|  |  |
| --- | --- |
| Pid | Title |
|  |  |

|  |  |
| --- | --- |
| Pid | Avatar |
|  |  |

|  |  |
| --- | --- |
| Pid | Account Level |
|  |  |

|  |  |
| --- | --- |
| Pid | Achievement points |
|  |  |

|  |  |
| --- | --- |
| Pid | Evo Level |
|  |  |

1. Game Mode table:

**1NF:**

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Game  Mode id | Mode | Sub mode | Map | Event status | Player type | Perspective | Auto matching |
|  |  |  |  |  |  |  |  |

* Each cell in the Game Mode table contains a single value, adhering to the requirement of 1NF.

**2NF:**

Ensures that each non-key attribute is dependent on the primary key.

* Functional Dependencies:
* Game mode id 🡪 Mode, Sub mode, Map, Event status, Player type, Perspective, Auto matching
* Mode, Sub mode 🡪 Map
* The combination of Mode and Sub-mode uniquely determines the Map associated with that combination.

Decomposition to Achieve 2NF:

* Game mode (Primary Key: Game mode id):

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Game  Mode id | Mode | Sub mode | Event status | Player type | Perspective | Auto matching |
|  |  |  |  |  |  |  |

* Mode Submode Map (Primary Key: Mode, Sub mode, Map):

|  |  |  |
| --- | --- | --- |
| Mode | Sub mode | Map |
|  |  |  |

* I decomposed the Game mode table into two separate tables to achieve 2NF.
* The Game mode table now contains attributes directly related to the primary key (Game mode id).
* The [Mode Submode Map] table stores the combination of Mode, Sub mode, and Map, which are dependent on each other and serve as the primary key.
* By decomposing the table in this manner, I ensured that each non-key attribute is dependent only on the primary key, adhering to the requirements of 2NF.

**3NF:**

Ensures that all non-key attributes are independent of each other.

* The Game mode id serves as the primary key, and it uniquely determines Mode, Sub mode, Map, Event status, Player type, Perspective, and Auto matching.
* Additionally, the combination of Mode and Sub mode uniquely determines the Map associated with that combination.
* Here, each non-key attribute is fully functionally dependent on the primary key, and there are no transitive dependencies among the non-key attributes.
* Therefore, there is no need for further decomposition to achieve 3NF. The table remains as:

Game mode (Primary Key: Game mode id):

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Game  Mode id | Mode | Sub mode | Event status | Player type | Perspective | Auto matching |
|  |  |  |  |  |  |  |

|  |  |  |
| --- | --- | --- |
| Mode | Sub mode | Map |
|  |  |  |

* The Game mode table satisfies the requirements of the Third Normal Form (3NF) as each non-key attribute is directly related to the primary key (Game mode id), and there are no transitive dependencies among the non-key attributes.
* Therefore, no further decomposition is needed to achieve 3NF.

**BCNF:**

* Ensures that each determinant in the table is a candidate key.

Functional Dependencies:

Game mode id -> Mode, Sub mode, Map, Event status, Player type, Perspective, Auto matching

Mode, Sub mode -> Map

* In this scenario, the primary key (Game mode id) uniquely determines all other attributes in the table, satisfying the requirements of BCNF. Therefore, there are no partial dependencies, and all attributes are fully functionally dependent on the primary key.

Decomposition to Achieve BCNF:

* Since the Game mode table already satisfies the requirements of BCNF, there is no need for further decomposition. The table remains as:

Game mode (Primary Key: Game mode id):

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Game  Mode id | Pid | Mode | Sub mode | Event status | Player type | Perspective | Auto matching |
|  |  |  |  |  |  |  |  |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Game  Mode id | Pid | Mode | Sub mode | Map |
|  |  |  |  |  |

* Mode and Sub mode together determine Map, and since there are no other attributes in the table, the determinant {Mode, Sub mode} is also a candidate key.
* Since the determinant is a candidate key and there are no partial dependencies, the table is in Boyce-Codd Normal Form (BCNF).
* So, after applying normalization up to BCNF the decomposed table are:

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Game  Mode id | Mode | Sub mode | Event status | Player type | Perspective | Auto matching |
|  |  |  |  |  |  |  |

|  |  |  |
| --- | --- | --- |
| Mode | Sub mode | Map |
|  |  |  |

1. Match Table:

**1NF:**

* ensure that each table cell contains only a single value, and each column has a unique name.

Match (Primary Key: Match id):

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Match id | Ranking | Rating | Duration | Date | BP | RP | Grade | In time | Out time |
|  |  |  |  |  |  |  |  |  |  |

* Each attribute appears to contain a single value in each cell, and each column has a unique name, ensuring that the table satisfies the requirements of 1NF.
* Therefore, there is no need to decompose the table further to achieve 1NF.

**2NF:**

* Eliminates redundant data by ensuring that each non-key attribute is dependent on the entire primary key, rather than on a portion of it.

Functional Dependencies:

* Match id 🡪 Ranking, Rating, Duration, Date, BP, RP, Grade, In time, out time
* In time, out time 🡪 Duration
* The attribute Duration is functionally dependent on the combination of in time and Out time, as the duration of a match can be calculated based on the difference between these two timestamps.
* By decomposing the table in this manner, we ensure that each non-key attribute is dependent only on the primary key or a combination of attributes that constitute a candidate key, adhering to the requirements of 2NF.

Match (Primary Key: Match id):

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Match id | Ranking | Rating | Duration | Date | BP | RP | Grade | In time | Out time |
|  |  |  |  |  |  |  |  |  |  |

MatchDuration (Primary Key: Match id, In time, out time):

|  |  |  |  |
| --- | --- | --- | --- |
| Match id | In time | Out time | Duration |
|  |  |  |  |

**3NF:**

* Ensures that all non-key attributes are independent of each other.

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Match id | Ranking | Rating | Duration | Date | BP | RP | Grade | In time | Out time |
|  |  |  |  |  |  |  |  |  |  |

Match in out table: (primary key: Match id)

|  |  |  |
| --- | --- | --- |
| Match id | In time | Out time |
|  |  |  |

MatchDuration (Primary Key: Match id):

|  |  |
| --- | --- |
| Match id | Duration |
|  |  |

* The Match table contains attributes directly related to the primary key (Match id), excluding Duration, In time, and Out time.
* The Match in out table stores the combination of Match id, In time, and Out time, which are timestamps related to the duration of each match.
* The MatchDuration table stores the Duration attribute, which is calculated based on the difference between In time and Out time for each match.

**BCNF:**

Ensure that each determinant in the table is a candidate key.

The candidate key of the Match table is Match id, as it uniquely identifies each tuple in the table.

* The primary key, Match ID, determines all other attributes in the table, including Ranking, Rating, Duration, Date, BP, RP, Grade, In time, and Out time. Therefore, Match ID is a candidate key.
* The combination of In-time and Out-time determines the Duration attribute. However, In time and Out time are not candidate keys by themselves, as they do not uniquely identify each tuple in the table.

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Match id | pid | Ranking | Rating | Duration | Date | BP | RP | Grade | In time | Out time |
|  |  |  |  |  |  |  |  |  |  |  |

|  |  |  |  |
| --- | --- | --- | --- |
| Match id | pid | In time | Out time |
|  |  |  |  |

|  |  |  |
| --- | --- | --- |
| Match id | pid | Duration |
|  |  |  |

* There are no non-trivial functional dependencies where a non-key attribute depends on a subset of the candidate key.

1. Server name table:

**1NF:**

Functional Dependencies:

* City🡪Province, Server name
* City uniquely determines the server, serving as the primary key for the server entity.
* The Server table satisfies the requirements of the First Normal Form (1NF) because each table cell contains only a single value, and each column has a unique name.
* The functional dependencies provided determine the attributes of the server entity, and the combination of Server name, Province, and City serves as the primary key for the server entity.

|  |  |  |  |
| --- | --- | --- | --- |
| City | Province | Pid | Server name |
|  |  |  |  |
|  |  |  |  |

**2NF:**

Since City is the primary key, in this case, there are no partial dependencies because both Province and Server name are fully dependent on the entire primary key (City).

Therefore, the original table already satisfies the requirements of the Second Normal Form (2NF) as each non-key attribute (Province and Server name) is directly related to the entire primary key (City) without any partial dependencies.

|  |  |  |  |
| --- | --- | --- | --- |
| City | Pid | Province | Server name |
|  |  |  |  |

* The Server table satisfies the requirements of the Second Normal Form (2NF) because each non-key attribute (Province, City) is functionally dependent on the entire primary key (Server name, Province, City).
* There are no partial dependencies, as each non-key attribute is directly related to the entire primary key.

**3NF:**

* Province is dependent on City.
* Server name is also dependent on City.

Since both Province and Server name are directly dependent on the primary key (City), and there are no transitive dependencies between non-key attributes, the table is already in Third Normal Form (3NF).

|  |  |  |  |
| --- | --- | --- | --- |
| City | Pid | Province | Server name |
|  |  |  |  |

**BCNF:**

* Province is dependent on the City.
* Server name is also dependent on the City.

Since both the Province and Server name are directly dependent on the candidate key (City), and there are no other non-trivial dependencies, the table already satisfies the Boyce-Codd Normal Form (BCNF).

1. Player stats (weak entity):

Functional dependency:

(Pid, Season [ Multi valued attribute]) 🡪Matches played,

* The combination of player id and season uniquely determines the matches played associated with that player's statistics for that season.

**1NF:**

* To decompose the Player Stats table to handle the multi-valued attribute (Season) and achieve First Normal Form (1NF), we'll create a new table to represent the multi-valued attribute separately.

|  |  |  |  |
| --- | --- | --- | --- |
| Player id | Season | Tier | Matches played |
|  |  |  |  |

* Decomposed table:

**Player Stats Table (Primary Key: Pid):**

|  |  |  |
| --- | --- | --- |
| Player id | Tier | Season |
|  |  |  |

**Season Table (Primary Key: Pid, Season):**

|  |  |  |
| --- | --- | --- |
| Player id | Season | Matches played |
|  |  |  |

**2NF:**

Ensures that each non-key attribute is directly related to the entire primary key without any partial dependencies.

There are no partial dependencies because Matches played depends on the entire composite primary key (Pid, Season).

|  |  |  |
| --- | --- | --- |
| Player id | Season | Tier |
|  |  |  |

|  |  |  |
| --- | --- | --- |
| Player id | Season | Matches played |
|  |  |  |

**3NF:**

1. Player Stats Table (3NF):
   * The Player Stats Table only contains the composite primary key Pid, and there are no non-key attributes to evaluate, ensuring that there are no transitive dependencies.
2. Season Table (3NF):
   * The matches played attribute is directly related to the composite primary key (Pid, Season), and there are no other non-key attributes in the table, ensuring that each column is directly related to the primary key and there are no transitive dependencies.

**Player Stats Table (Primary Key: Pid):**

|  |  |  |
| --- | --- | --- |
| Player id | Season | Tier |
|  |  |  |

**Season Table (Primary Key: Pid, Season):**

|  |  |  |
| --- | --- | --- |
| Player id | Season | Matches played |
|  |  |  |

**BCNF:**

1. **Player Stats Table (BCNF):**
   * Since the Player Stats Table only contains the composite primary key Pid, and there are no non-key attributes to evaluate, it trivially adheres to BCNF.
2. **Season Table (BCNF):**
   * The matches played attribute is directly related to the composite primary key (Pid, Season), and there are no other non-key attributes in the table. Therefore, each non-key attribute is dependent only on the candidate key (Pid, Season), ensuring that the table adheres to BCNF.

**Player Stats Table (Primary Key: Pid):**

|  |  |  |
| --- | --- | --- |
| Player id | Season | Tier |
|  |  |  |

**Season Table (Primary Key: Pid, Season):**

|  |  |  |
| --- | --- | --- |
| Player id | Season | Matches played |
|  |  |  |

**Query:**

**Creating table:**

Create the players table

CREATE TABLE player (

pid INT PRIMARY KEY,

ingamename VARCHAR(255),

email VARCHAR(255),

merit INT,

clanname VARCHAR(255),

likes INT,

popularity INT,

city VARCHAR(255)

);

Insert the provided values into the player table:

INSERT INTO players (pid, ingamename, email, merit, clanname, likes, popularity, city)

VALUES

(20, 'Player20', 'player20@example.com', 100, 'Clan20', 2400, 34550, 'Kavaratti'),

(19, 'Player19', 'player19@example.com', 90, 'Clan19', 1400, 7000, 'Indore'),

(18, 'Player18', 'player18@example.com', 100, 'Clan18', 2700, 4550, 'Thiruvanantapuram'),

(17, 'Player17', 'player17@example.com', 95, 'Clan17', 1700, 4560, 'Hubli and Dharwad'),

(16, 'Player16', 'player16@example.com', 100, 'Clan16', 3000, 9000, 'Ranchi'),

(15, 'Player15', 'player15@example.com', 100, 'Clan15', 2300, 8900, 'Srinagar'),

(14, 'Player14', 'player14@example.com', 100, 'Clan14', 1300, 2300, 'Shimla'),

(13, 'Player13', 'player13@example.com', 90, 'Clan13', 2600, 3450, 'Gurgaon'),

(12, 'Player12', 'player12@example.com', 85, 'Clan12', 850, 3450, 'Ahmedabad'),

(11, 'Player11', 'player11@example.com', 85, 'Clan11', 850, 4540, 'Goa Velha'),

(10, 'Player10', 'player10@example.com', 90, 'Clan10', 900, 8900, 'NDMC'),

(9, 'Player9', 'player9@example.com', 95, 'Clan9', 950, 4550, 'Daman'),

(8, 'Player8', 'player8@example.com', 100, 'Clan8', 1000, 34000, 'Chandigarh'),

(7, 'Player7', 'player7@example.com', 85, 'Clan7', 850, 4550, 'Raipur'),

(6, 'Player6', 'player6@example.com', 90, 'Clan6', 900, 1300, 'Patna'),

(5, 'Player5', 'player5@example.com', 85, 'Clan5', 850, 1250, 'Guwahati'),

(4, 'Player4', 'player4@example.com', 95, 'Clan4', 950, 950, 'Itanagar'),

(3, 'Player3', 'player3@example.com', 85, 'Clan3', 850, 1050, 'Bangalore'),

(2, 'Player2', 'player2@example.com', 90, 'Clan2', 900, 1000, 'Vishakapatnam'),

(1, 'Player1', 'player1@example.com', 85, 'Clan1', 850, 950, 'Port Blair');

Create the table

CREATE TABLE player\_achievement\_points(

pid INT PRIMARY KEY,

achievement\_points INT

);

Insert the values

INSERT INTO player\_achievement\_points (pid, achievement\_points)

VALUES

(1, 3565),

(2, 2900),

(3, 5580),

(4, 6000),

(5, 1540),

(6, 3475),

(7, 2385),

(8, 1225),

(9, 1300),

(10, 4400),

(11, 3460),

(12, 2395),

(13, 4005),

(14, 5040),

(15, 6700),

(16, 1900),

(17, 805),

(18, 6100),

(19, 3325),

(20, 2115);

create table player\_title(

pid int primary key not null,

title varchar(100));

insert into player\_title(pid,title)

VALUES

(1, 'Well-Liked'),

(2, 'Collector'),

(3, 'Overachiever'),

(4, 'Star Trainer'),

(5, 'Perseverance'),

(6, 'Perfectionist'),

(7, 'Deadeye'),

(8, 'Warhorse'),

(9, 'Synergy Titles'),

(10, 'Dependable'),

(11, 'Cycle season tier Titles'),

(12, 'Weapon Master'),

(13, 'Maxed Out'),

(14, 'Pacifist'),

(15, 'Chicken Master'),

(16, 'Glass Cannon/Commando'),

(17, 'Unique Destiny'),

(18, 'Mythic Fashion'),

(19, 'On a mission'),

(20, 'Veteran');

create table player\_avatar(pid int primary key not null, avatar varchar(100));

insert into player\_avatar(pid, avatar)

values

(1, 'Newbie'),

(2, 'Berserker'),

(3, 'Sharpshooter'),

(4, 'Rookie'),

(5, 'Lone Wolf'),

(6, 'Speed Demon'),

(7, 'RP Avatar (M7)'),

(8, 'RP Avatar (M6)'),

(9, 'RP Avatar (M1)'),

(10, 'RP Avatar (M5)'),

(11, 'Domination'),

(12, 'Healer'),

(13, 'Elite Squad'),

(14, '100 Finishes'),

(15, 'Dynamic Duo'),

(16, 'Top 10 Regular'),

(17, 'Chicken Lover'),

(18, 'Trendy Social Butterfly'),

(19, 'Conqueror'),

(20, 'Grenadier'),

(21, 'Sniper');

create table Account\_Level (

pid int primary key,

account\_Level int

);

insert into Account\_Level (Pid, Account\_Level)

values

(1, 31),

(2, 43),

(3, 55),

(4, 69),

(5, 72),

(6, 40),

(7, 58),

(8, 61),

(9, 73),

(10, 29),

(11, 83),

(12, 28),

(13, 47),

(14, 64),

(15, 67),

(16, 37),

(17, 43),

(18, 53),

(19, 72),

(20, 78);

create table player\_evo\_level (

pid int primary key,

evo\_level int

);

insert into player\_evo\_level(Pid, evo\_level)

values

(1, 34),

(2, 70),

(3, 61),

(4, 55),

(5, 43),

(6, 43),

(7, 80),

(8, 31),

(9, 74),

(10, 89),

(11, 100),

(12, 67),

(13, 58),

(14, 90),

(15, 83),

(16, 47),

(17, 71),

(18, 47),

(19, 30),

(20, 42);

create table game\_mode (

game\_mode\_id int not null,

pid int not null,

primary key (game\_mode\_id,pid),

mode varchar(255),

sub\_mode varchar(255),

event\_status varchar(255),

player\_type varchar(255),

perspective varchar(255),

auto\_matching varchar(255)

);

insert into game\_mode(game\_mode\_id,pid,mode, sub\_mode,event\_status, player\_type, perspective, auto\_matching)

values

(101,1,'Ranked', 'Classic', 'skyhigh spectacle', 'Solo', 'TPP', 'Yes'),

(101,2,'Ranked', 'Classic','skyhigh spectacle', 'Solo', 'TPP', 'Yes'),

(103,3,'Ranked', 'Classic','no event', 'Duo', 'FPP', 'No'),

(104,4,'Unranked', 'Arena', 'no event', 'Squad', 'FPP', 'Yes'),

(103,5,'Ranked', 'Classic', 'no event', 'Duo', 'FPP', 'No'),

(103,6,'Ranked', 'Classic', 'no event', 'Duo', 'FPP', 'No'),

(104,7,'Unranked', 'Arena', 'no event', 'Squad', 'FPP', 'Yes'),

(102,8,'Unranked', 'Arena', 'no event', 'Solo', 'TPP', 'Yes'),

(104,9,'Unranked', 'Arena', 'no event', 'Squad', 'FPP', 'Yes'),

(102,10,'Unranked', 'Arena', 'no event', 'Solo', 'TPP', 'Yes'),

(111,11,'Ranked', 'Classic', 'skyhigh spectacle', 'Duo', 'FPP', 'Yes'),

(111,12,'Ranked', 'Classic', 'skyhigh spectacle', 'Duo', 'FPP', 'Yes'),

(113,13,'Ranked', 'Classic','skyhigh spectacle', 'Solo', 'TPP', 'Yes'),

(114,14,'Unranked', 'Classic','no event', 'Duo', 'FPP', 'Yes'),

(113,15,'Ranked', 'Classic','skyhigh spectacle', 'Solo', 'TPP', 'Yes'),

(116,16,'Unranked', 'Arena', 'no event', 'Squad', 'FPP', 'Yes'),

(113,17,'Ranked', 'Classic','skyhigh spectacle', 'Solo', 'TPP', 'Yes'),

(116,18,'Unranked', 'Arena','no event', 'Squad', 'FPP', 'Yes'),

(113,19,'Ranked', 'Classis','skyhigh spectacle', 'Solo', 'TPP', 'Yes'),

(120,20,'Ranked', 'Arena', 'no event', 'Duo', 'FPP', 'Yes');

create table mode\_submode(

game\_mode\_id int,

pid int,

mode varchar(50),

sub\_mode varchar(50),

map varchar(50),

primary key(game\_mode\_id,pid,mode,sub\_mode)

);

insert into mode\_submode(game\_mode\_id,pid,mode,sub\_mode,map)

values

(101,1,'Ranked', 'Classic', 'Livik'),

(101,2,'Ranked', 'Classic','Livik'),

(103,3,'Ranked', 'Classic','Nusa'),

(104,4,'Unranked', 'Arena', 'TDM'),

(103,5,'Ranked', 'Classic', 'Nusa'),

(103,6,'Ranked', 'Classic', 'Nusa'),

(104,7,'Unranked', 'Arena', 'TDM'),

(102,8,'Unranked', 'Arena', 'Gun game'),

(104,9,'Unranked', 'Arena', 'TDM'),

(102,10,'Unranked', 'Arena', 'Gun game'),

(111,11,'Ranked', 'Classic', 'Livik'),

(111,12,'Ranked', 'Classic', 'Livik'),

(113,13,'Ranked', 'Classic', 'Sanhok'),

(114,14,'Unranked', 'Classic', 'Assault'),

(113,15,'Ranked', 'Classic', 'Sanhok'),

(116,16,'Unranked', 'Arena', '8 V 8 TDM'),

(113,17,'Ranked', 'Classic', 'Sanhok'),

(116,18,'Unranked', 'Arena', '8 v 8 TDM'),

(113,19,'Ranked', 'Classis', 'Sanhok'),

(120,20,'Ranked', 'Arena', 'Domination');

create table matchmake(

match\_id int,

pid int ,

primary key(match\_id,pid),

ranking INT,

rating INT,

in\_time datetime,

out\_time datetime,

duration int,

bp INT,

rp INT,

grade VARCHAR(255));

insert into matchmake(match\_id,pid,ranking,rating,in\_time, out\_time, duration,bp,rp,grade)

values

(101,1,25,32,'2022-03-01 10:00:00', '2022-03-01 10:30:00', 30, 400, 30, 'SSS'),

(101,2,1,99,'2022-03-01 10:00:00', '2022-03-01 10:30:00', 30, 300, 200,'AAA'),

(103,3,23,11,'2022-03-01 11:00:00', '2022-03-01 11:30:00', 30, 100, 400, 'BB'),

(104,4,1,98,'2022-03-01 12:00:00', '2022-03-01 12:30:00', 30, 145, 45, 'AAA+'),

(103,5,12,85,'2022-03-01 11:00:00', '2022-03-01 11:30:00', 30, 33, 200, 'DDD'),

(103,6,50,2,'2022-03-01 11:00:00', '2022-03-01 11:30:00', 30, 298, 500, 'BBB'),

(104,7,25,13,'2022-03-01 12:00:00', '2022-03-01 12:30:00', 30, 700, 190, 'CCC'),

(102,8,1,99,'2022-03-01 13:00:00', '2022-03-01 13:30:00', 30, 500, 385, 'EEE'),

(104,9,2,97,'2022-03-01 12:00:00', '2022-03-01 12:30:00', 30, 890, 200, 'A'),

(102,10,3,94,'2022-03-01 13:00:00', '2022-03-01 13:30:00', 30, 560, 200, 'AA'),

(111,11,5,97,'2022-03-01 14:00:00', '2022-03-01 14:30:00', 30, 456, 50, 'C'),

(111,12,6,98,'2022-03-01 14:00:00', '2022-03-01 14:30:00', 30, 500, 100, 'DD'),

(113,13,3,99,'2022-03-01 15:00:00', '2022-03-01 15:30:00', 30, 439, 0, 'AAA'),

(114,14,7,90,'2022-03-01 16:00:00', '2022-03-01 16:30:00', 30, 320, 250, 'A+'),

(113,15,1,96,'2022-03-01 15:00:00', '2022-03-01 15:30:00', 30, 321, 0, 'SS'),

(116,16,1,99,'2022-03-01 17:00:00', '2022-03-01 17:30:00', 30, 439, 0, 'SSS'),

(113,17,4,89,'2022-03-01 15:00:00', '2022-03-01 15:30:00', 30, 321, 0, 'AAA'),

(116,18,3,94,'2022-03-01 17:00:00', '2022-03-01 17:30:00', 30, 459, 0, 'D'),

(113,19,100,0,'2022-03-01 15:00:00', '2022-03-01 15:30:00', 30, 65, 10, 'SSS+'),

(120,20,50,0,'2022-03-01 18:00:00', '2022-03-01 18:30:00', 30, 0, 20, 'AAA');

create table matchtime(match\_id int,pid int,primary key(match\_id,pid), in\_time datetime,out\_time datetime);

insert into matchtime(match\_id,pid,in\_time,out\_time)

values

(101, 1, '2022-03-01 10:00:00', '2022-03-01 10:30:00'),

(101, 2, '2022-03-01 10:00:00', '2022-03-01 10:30:00'),

(103, 3, '2022-03-01 11:00:00', '2022-03-01 11:30:00'),

(104, 4, '2022-03-01 12:00:00', '2022-03-01 12:30:00'),

(103, 5, '2022-03-01 11:00:00', '2022-03-01 11:30:00'),

(103, 6, '2022-03-01 11:00:00', '2022-03-01 11:30:00'),

(104, 7, '2022-03-01 12:00:00', '2022-03-01 12:30:00'),

(102, 8, '2022-03-01 13:00:00', '2022-03-01 13:30:00'),

(104, 9, '2022-03-01 12:00:00', '2022-03-01 12:30:00'),

(102, 10, '2022-03-01 13:00:00', '2022-03-01 13:30:00'),

(111, 11, '2022-03-01 14:00:00', '2022-03-01 14:30:00'),

(111, 12, '2022-03-01 14:00:00', '2022-03-01 14:30:00'),

(113, 13, '2022-03-01 15:00:00', '2022-03-01 15:30:00'),

(114, 14, '2022-03-01 16:00:00', '2022-03-01 16:30:00'),

(113, 15, '2022-03-01 15:00:00', '2022-03-01 15:30:00'),

(116, 16, '2022-03-01 17:00:00', '2022-03-01 17:30:00'),

(113, 17, '2022-03-01 15:00:00', '2022-03-01 15:30:00'),

(116, 18, '2022-03-01 17:00:00', '2022-03-01 17:30:00'),

(113, 19, '2022-03-01 15:00:00', '2022-03-01 15:30:00'),

(120, 20, '2022-03-01 18:00:00', '2022-03-01 18:30:00');

create table match\_duration(match\_id int,pid int,primary key(match\_id,pid),duration int);

insert into match\_duration(match\_id,pid,duration)

values

(101, 1, 30),

(101, 2, 30),

(103, 3, 30),

(104, 4, 30),

(103, 5, 30),

(103, 6, 30),

(104, 7, 30),

(102, 8, 30),

(104, 9, 30),

(102, 10, 30),

(111, 11, 30),

(111, 12, 30),

(113, 13, 30),

(114, 14, 30),

(113, 15, 30),

(116, 16, 30),

(113, 17, 30),

(116, 18, 30),

(113, 19, 30),

(120, 20, 30);

create table player\_stats1(pid int primary key not null,season char(20));

insert into player\_stats1(pid,season)values

(1, 'C5S16'),

(2,'C5S16'),

(3, 'C5S16'),

(4, 'C5S16'),

(5, 'C5S16'),

(6, 'C5S16'),

(7, 'C5S16'),

(8, 'C5S16'),

(9, 'C5S16'),

(10, 'C5S16'),

(11, 'C5S16'),

(12, 'C5S16'),

(13, 'C5S16'),

(14, 'C5S16'),

(15, 'C5S16'),

(16,'C5S16'),

(17, 'C5S16'),

(18, 'C5S16'),

(19, 'C5S16'),

(20, 'C5S16');

create table player\_stats2(pid int primary key not null,season char(20),tier varchar(30),matches\_played int);

insert into player\_stats2(pid,season,tier,matches\_played)values

(1, 'C5S16', 'Silver 1', 1500),

(2, 'C5S16', 'Silver 1', 2500),

(3, 'C5S16', 'Platinum 5', 3000),

(4, 'C5S16', 'Diamond 1', 4500),

(5, 'C5S16', 'Platinum 5', 5500),

(6, 'C5S16', 'Platinum 5', 6000),

(7, 'C5S16', 'Diamond 1', 8000),

(8, 'C5S16', 'Ace Dominator 5', 8500),

(9, 'C5S16', 'Diamond 1', 10000),

(10, 'C5S16', 'Ace Dominator 5', 1800),

(11, 'C5S16', 'Gold 4', 2700),

(12, 'C5S16', 'Gold 4', 3200),

(13, 'C5S16', 'Diamond 1', 4800),

(14, 'C5S16', 'Conqueror', 5800),

(15, 'C5S16', 'Diamond 1', 6300),

(16, 'C5S16', 'Acemaster 5', 8200),

(17, 'C5S16', 'Diamond 1', 8700),

(18, 'C5S16', 'Acemaster 5', 10500),

(19, 'C5S16', 'Diamond 1', 2900),

(20, 'C5S16', 'Conqueror', 3400);

Create the server table

CREATE TABLE server (

city VARCHAR(255) primary key not null,

province VARCHAR(255),

country VARCHAR(255),

pid INT);

INSERT INTO server (city, province, country, pid)

VALUES

('Port Blair', 'Andaman and Nicobar Islands', 'India', 1),

('Vishakapatnam', 'Andhra Pradesh', 'India', 2),

('Bangalore', 'Karnataka', 'India', 3),

('Itanagar', 'Arunachal Pradesh', 'India', 4),

('Guwahati', 'Assam', 'India', 5),

('Patna', 'Bihar', 'India', 6),

('Raipur', 'Chattisgarh', 'India', 7),

('Chandigarh', 'Chandigarh', 'India', 8),

('Daman', 'Dadra and Nagar haveli and Daman and Diu', 'India', 9),

('NDMC', 'Delhi', 'India', 10),

('Goa Velha', 'Goa', 'India', 11),

('Ahmedabad', 'Gujarat', 'India', 12),

('Gurgaon', 'Haryana', 'India', 13),

('Shimla', 'Himachal Pradesh', 'India', 14),

('Srinagar', 'Jammu and Kashmir', 'India', 15),

('Ranchi', 'Jharkhand', 'India', 16),

('Hubli and Dharwad', 'Karnataka', 'India', 17),

('Thiruvanantapuram', 'Kerala', 'India', 18),

('Indore', 'Madhya Pradesh', 'India', 19),

('Kavaratti', 'Lakshadweep', 'India', 20);

**Trigger 1:** This trigger is executed whenever there is an update operation on the player table on any row. Whenever other players give likes after every match old. likes should be updated but with that for each like given 5 popularities should be increased.

create trigger calculate\_popularity

before update on player

for each row

set new. popularity = old. popularity + ((new. likes - old. likes) \* 5);

**Trigger 2:** This trigger is executed whenever there is an update operation on the account\_Level table on any row. Whenever there is 5 level increase in evo\_level column, 1 account level should be increased.

DELIMITER //

create trigger increase\_account\_level

after update on player\_evo\_level

for each row

begin

if new.evo\_level - old.evo\_level >= 5 then

update Account\_Level

set account\_Level = account\_Level + 1

where pid = new.pid;

end if;

end; //

DELIMITER;

**Trigger 3:** This trigger is executed whenever there is an insert operation on matchmake table on any row. Whenever there is a new match\_id created in matchmake table for a particular pid, for 1 match\_id created matches\_played will be increased by 1 in player\_stats2 table.

DELIMITER //

CREATE TRIGGER update\_matches\_played

AFTER INSERT ON matchmake

FOR EACH ROW

BEGIN

UPDATE player\_stats2

SET matches\_played = matches\_played + 1

WHERE pid = NEW.pid;

END; //

DELIMITER;

**View 1:** I created a view – ‘player\_match\_view’ which displays match\_id of players who played same match, pid of players who played same match, ingamename of players who played same match, displays only players who has likes>=800 and merit>=85 and who played with auto\_matching and perspective is FPP and account\_Level>20 and achievement\_points>999 and evo\_level>30 and season is C5S16 and matches\_played>10002

create view player\_match\_view as

select

m.match\_id, m.pid, p.ingamename, p.likes, p.merit, g.auto\_matching, g.perspective, al.account\_Level, pap.achievement\_points, pel.evo\_level, ps2.season, ps2.matches\_played

from matchmake m

join player p on m.pid = p.pid

join game\_mode g on m. match\_id= g. game\_mode\_id and m.pid = g.pid

join Account\_Level al on m.pid = al.pid

join player\_achievement\_points pap on m.pid = pap.pid

join player\_evo\_level pel on m.pid = pel.pid

join player\_stats2 ps2 on m.pid = ps2.pid

where p.likes >= 800

and p. merit >= 85

and g. auto\_matching = 'Yes'

and g. perspective = 'FPP'

and al. account\_Level > 20

and pap. achievement\_points > 999

and pel.evo\_level > 30

and ps2.season like 'C5S16'

and ps2.matches\_played > 1000;

**Function:** The function get\_player\_level takes pid as a parameter and returns the string Beginner player, if total achievement points is <2000. Returns string Intermediate player if total achievement points >= 2000 and <3000. Returns string Advanced player if total achievement points >= 3000 and < 5000. Returns string Expert player If achievement points > 5000.

CREATE DEFINER=`root`@`localhost` FUNCTION `get\_player\_level` (p\_pid INT) RETURNS varchar (20) CHARSET utf8mb4

DETERMINISTIC

BEGIN

DECLARE total\_achievement\_points INT;

DECLARE player\_level VARCHAR(20);

SELECT SUM(achievement\_points)

INTO total\_achievement\_points

FROM player\_achievement\_points

WHERE pid = p\_pid;

IF total\_achievement\_points < 2000 THEN

SET player\_level = 'Beginner Player';

ELSEIF total\_achievement\_points >= 2000 AND total\_achievement\_points < 3000 THEN

SET player\_level = 'Intermediate Player';

ELSEIF total\_achievement\_points >= 3000 AND total\_achievement\_points < 5000 THEN

SET player\_level = 'Advanced Player';

ELSE

SET player\_level = 'Expert Player';

END IF;

RETURN player\_level;

END  
  
**Input**: select pubg.get\_player\_level (12);

**Output**: Intermediate Player

**Procedure:** This procedure named `top\_players\_by\_likes\_and\_popularity\_per\_match` calculates average likes and average popularity per match for a player and the output will be ordered in decreasing order of average\_likes\_per\_match and decreasing order of average\_popularity\_per\_match hence it will be ‘top\_players\_by\_likes\_and\_popularity\_per\_match’

CREATE DEFINER=`root`@`localhost` PROCEDURE `top\_players\_by\_likes\_and\_popularity\_per\_match`()

BEGIN

SELECT

p.pid,

p.ingamename,

AVG(p.likes / ps.matches\_played) AS average\_likes\_per\_match,

AVG(p.popularity / ps.matches\_played) AS average\_popularity\_per\_match

FROM

player p

JOIN player\_stats2 ps ON p.pid = ps.pid

GROUP BY

p.pid

ORDER BY

average\_likes\_per\_match DESC,

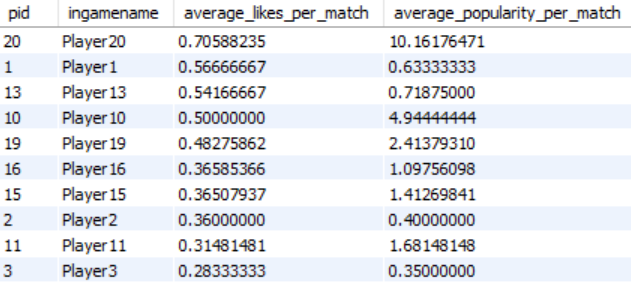
average\_popularity\_per\_match DESC

LIMIT 10;

END

**Input:** call pubg.top\_players\_by\_likes\_and\_popularity\_per\_match();

**Output:**

**Query 1:** This query gives output of pid, ingamename, game\_mode\_id for the players who are playing in same match. The match is

mode = 'Ranked', sub\_mode = 'Classic', map = 'Livik', event\_status = 'skyhigh spectacle', player\_type = 'Solo', perspective = 'TPP', auto\_matching = 'Yes'

SELECT DISTINCT p.pid, p. ingamename, gm.game\_mode\_id

FROM player p

JOIN game\_mode gm ON p.pid = gm.pid

JOIN mode\_submode ms ON gm.game\_mode\_id = ms.game\_mode\_id

WHERE gm.mode = 'Ranked'

AND gm.sub\_mode = 'Classic'

AND ms.map = 'Livik'

AND gm.event\_status = 'skyhigh spectacle'

AND gm.player\_type = 'Solo'

AND gm.perspective = 'TPP'

AND gm.auto\_matching = 'Yes'

AND gm.game\_mode\_id IN (

SELECT DISTINCT gm.game\_mode\_id

FROM game\_mode gm

JOIN mode\_submode ms ON gm.game\_mode\_id = ms.game\_mode\_id

WHERE gm.mode = 'Ranked'

AND gm.sub\_mode = 'Classic'

AND ms.map = 'Livik'

AND gm.event\_status = 'skyhigh spectacle'

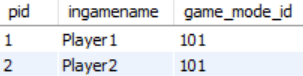
AND gm.player\_type = 'Solo'

AND gm.perspective = 'TPP'

AND gm.auto\_matching = 'Yes'

);

**Output:**

****

**Query 2:** this query displaysthe pid, ingamename, clanname, tier, avatar, city of the player who played in Diamond or Platinum or Ace tier.

SELECT p.pid, p.ingamename,p.clanname, ps2.tier,pa.avatar,s.city

FROM player p

JOIN player\_stats2 ps2 ON p.pid = ps2.pid

join server s on p.pid=s.pid

join player\_avatar pa on p.pid=pa.pid

WHERE p.pid IN (

SELECT gm.pid

FROM game\_mode gm

WHERE gm.pid IN (

SELECT ps2.pid

FROM player\_stats2 ps2

WHERE ps2.tier like 'Diamond%' or ps2.tier like 'Platinum%' or ps2.tier like 'Ace%'

)

);

**Output:**

****

**Query 3:** This query retrieves the pid, ingamename, tier, and popularity from the player and player\_stats2 tables for players who meet the following conditions: Their tier in the player\_stats2 table is similar to 'Diamond%'

Their popularity in the player\_stats2 table is greater than 1000.

SELECT p.pid, p.ingamename, ps2.tier, p.popularity

FROM player p

JOIN player\_stats2 ps2 ON p.pid = ps2.pid

WHERE ps2.tier LIKE 'Diamond%'

AND p.popularity > 1000

INTERSECT

SELECT p.pid, p.ingamename, ps2.tier, p.popularity

FROM player p

JOIN player\_stats2 ps2 ON p.pid = ps2.pid

WHERE ps2.tier LIKE 'Diamond%'

AND p.popularity > 1000;



**Query 4:** This query calculates the average duration, total battle points (BP), and total rank points (RP) for each player who has participated in ranked classic matches. It filters out players who have played at least one match and presents the results ordered by total BP in descending order.

SELECT p.pid, p.ingamename,

AVG(m.duration) AS avg\_duration,

SUM(m.bp) AS total\_bp,

SUM(m.rp) AS total\_rp

FROM player p

JOIN matchmake m ON p.pid = m.pid

JOIN game\_mode g ON m.match\_id = g.game\_mode\_id AND p.pid = g.pid

WHERE g.mode = 'Ranked' AND g.sub\_mode = 'Classic' AND p.pid IN (

SELECT pid

FROM (

SELECT pid, COUNT(\*) AS matches\_played

FROM matchmake

GROUP BY pid

HAVING COUNT(\*) >= 1

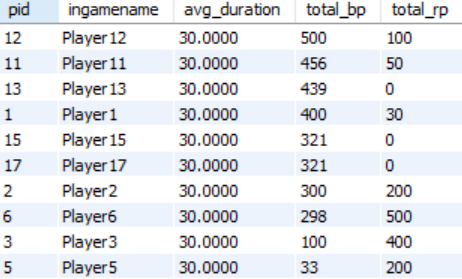
)as sub

)

GROUP BY p.pid

ORDER BY total\_bp DESC;

**Output:**

****

**Query 5:** This query retrieves details of players who achieved the title 'Chicken Master', joining their player ID, in-game name, account level, evolution level, and achievement points. It first filters players who participated in at least one ranked classic match. Then, it sorts the results based on account level, evolution level, and achievement points in descending order, limiting the output to the top 20 players.

SELECT p.pid, p.ingamename, al.account\_Level, pel.evo\_level, pap.achievement\_points

FROM player p

JOIN Account\_Level al ON p.pid = al.pid

JOIN player\_evo\_level pel ON p.pid = pel.pid

JOIN player\_achievement\_points pap ON p.pid = pap.pid

JOIN player\_title pt ON p.pid = pt.pid

JOIN (

SELECT m.pid

FROM matchmake m

JOIN game\_mode g ON m.match\_id = g.game\_mode\_id AND m.pid = g.pid

WHERE g.mode = 'Ranked' AND g.sub\_mode = 'Classic'

GROUP BY m.pid

HAVING COUNT(\*) >= 1

) subquery ON p.pid = subquery.pid

WHERE pt.title = 'Chicken Master'

ORDER BY al.account\_Level DESC, pel.evo\_level DESC, pap.achievement\_points DESC

LIMIT 20;



**Query 6:** This query retrieves player details along with their account level from 'Account\_Level' table, joining with 'game\_mode' and 'server' tables based on specified conditions. It selects players who have played at least one game in 'Unranked' mode with 'Arena' sub-mode, having an account level between 20 and 90, and are located in Kerala. The results are grouped by player ID and filtered to include only those with at least one distinct game mode ID.

SELECT p.pid, p.ingamename, s.city, s.province, s.server\_name,al.account\_Level

FROM player p

JOIN game\_mode g ON p.pid = g.pid

JOIN Account\_Level al ON p.pid = al.pid

JOIN server s ON p.city = s.city

WHERE g.mode = 'Unranked' AND g.sub\_mode = 'Arena'

AND al.account\_Level BETWEEN 20 AND 90

AND s.province = 'Kerala'

GROUP BY p.pid

HAVING COUNT(DISTINCT g.game\_mode\_id) >= 1;

**Query 7:**

This query retrieves match details along with player information and titles earned by players who participated in ranked classic mode matches. It filters players who have titles starting with 'Well', 'Collector', 'Over', 'Persever', or 'Perfect' and have achieved more than 1000 achievement points. The results are sorted by player ratings in descending order and match durations in ascending order.

SELECT m.match\_id, m.pid, p.ingamename, m.ranking, m.rating, m.duration, m.bp, m.rp, m.grade,pt.title

FROM matchmake m

JOIN player p ON m.pid = p.pid

JOIN player\_title pt ON p.pid = pt.pid

JOIN player\_achievement\_points pap ON p.pid = pap.pid

JOIN game\_mode g ON m.match\_id = g.game\_mode\_id AND m.pid = g.pid

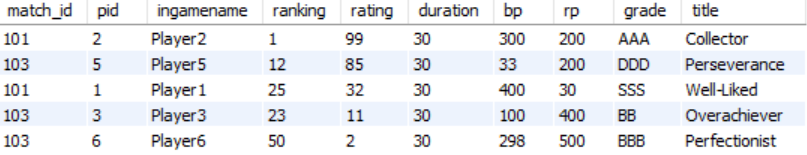
WHERE g.mode = 'Ranked' AND g.sub\_mode = 'Classic'

AND (pt.title LIKE 'Well%' OR pt.title LIKE 'Collector%' or pt.title like 'Over%' or pt.title like 'Persever%' or pt.title like 'perfect%')

AND pap.achievement\_points > 1000

ORDER BY m.rating DESC, m.duration ASC;

**Output:**

****

**Query 8:** This query retrieves the player ID, in-game name, and merit score from the player table. It also calculates the total number of matches played by each player in the 'Ranked' mode with 'Classic' sub-mode using a correlated subquery. The total\_matches column displays the count of such matches for each player. The results are sorted based on the merit score in descending order, and only the top 10 players are returned.

SELECT p.pid, p.ingamename, p.merit,

(SELECT COUNT(\*)

FROM matchmake m

WHERE m.pid = p.pid AND

EXISTS (SELECT 1

FROM game\_mode g

WHERE m.match\_id = g.game\_mode\_id AND

m.pid = g.pid AND

g.mode = 'Ranked' AND

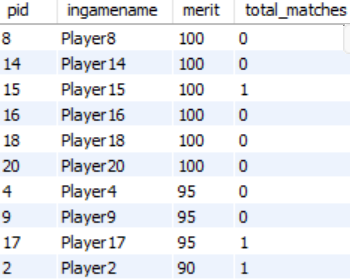
g.sub\_mode = 'Classic')) AS total\_matches

FROM player p

ORDER BY p.merit DESC

LIMIT 10;

**Output:**

****

**Query 9:** This query retrieves the player ID, in-game name, and the maximum ranking, rating, and grade achieved in matches. It joins tables to gather match data, ensures the player's account level is at least 20, and filters matches based on game mode, player type, and sub-mode. The results are grouped by player ID, and players must have participated in at least one match. The final output is sorted by the best ranking in ascending order, followed by the best rating in descending order, and finally by the best grade in ascending order.

SELECT p.pid, p.ingamename,

MAX(m.ranking) AS best\_ranking,

MAX(m.rating) AS best\_rating,

MAX(m.grade) AS best\_grade

FROM player p

JOIN matchmake m ON p.pid = m.pid

JOIN game\_mode g ON m.match\_id = g.game\_mode\_id AND p.pid = g.pid

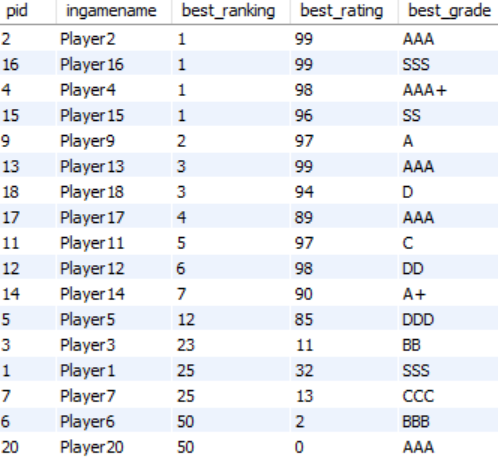
JOIN Account\_Level al ON p.pid = al.pid

WHERE g.player\_type like 'Squad' AND al.account\_Level >=20 AND g.mode = 'Unranked' or g.mode='Ranked' AND g.sub\_mode = 'Arena' or g.sub\_mode='Classic'

GROUP BY p.pid

HAVING COUNT(DISTINCT m.match\_id) >= 1

ORDER BY best\_ranking ASC, best\_rating DESC, best\_grade ASC;



**Query 10:** This queryreturnsthe player details who played in a same match

DELIMITER //

CREATE FUNCTION get\_player\_match\_details (match\_id INT)

RETURNS TABLE (

match\_id INT,

pid INT,

ingamename VARCHAR(255),

email VARCHAR(255),

merit INT,

clanname VARCHAR(255),

likes INT,

popularity INT,

achievement\_points INT,

evo\_level INT,

account\_level INT,

ranking INT,

rating INT,

duration INT,

bp INT,

rp INT,

grade VARCHAR(255),

in\_time DATETIME,

out\_time DATETIME

)

DETERMINISTIC

BEGIN

RETURN QUERY

SELECT

m.match\_id,

p.pid,

p.ingamename,

p.email,

p.merit,

p.clanname,

p.likes,

p.popularity,

pap.achievement\_points,

pel.evo\_level,

al.account\_level,

mm.ranking,

mm.rating,

md.duration,

mm.bp,

mm.rp,

mm.grade,

mt.in\_time,

mt.out\_time

FROM

matchmake m

JOIN player p ON m.pid = p.pid

JOIN player\_achievement\_points pap ON p.pid = pap.pid

JOIN player\_evo\_level pel ON p.pid = pel.pid

JOIN Account\_Level al ON p.pid = al.pid

JOIN matchmake mm ON m.match\_id = mm.match\_id AND m.pid = mm.pid

JOIN matchtime mt ON m.match\_id = mt.match\_id AND m.pid = mt.pid

JOIN match\_duration md ON m.match\_id = md.match\_id AND m.pid = md.pid

WHERE

m.match\_id = match\_id;

END //

DELIMITER ;