FSL Database Structure & Design Summary (Version 1.0)

This document provides the **latest FSL database structure, relationships, key functionalities, and sample queries**.

🔹 Overview

The **FSL database** manages **player statistics, aliases, team memberships, championships, and team league performances**. It efficiently stores **cumulative statistics** while maintaining **historical team records** using JSON.

Key Features:

✔ **Tracks cumulative player statistics** across **divisions and races**.
✔ **Links players to multiple in-game aliases**.
✔ **Stores both current and historical team membership**.
✔ **Uses JSON fields for detailed championship history**.
✔ **Ensures data integrity with constraints on unique player statistics**.

📌 Database Schema

1️⃣ Players Table (Core Player Data)

Holds **player details**, including **current team and championship records**.

sql
CopyEdit

```
CREATE TABLE Players (
    Player_ID INT AUTO_INCREMENT PRIMARY KEY,
    Real_Name VARCHAR(255) NOT NULL UNIQUE,
    Team_ID INT DEFAULT NULL,
    Championship_Record JSON DEFAULT NULL,
```

```
    TeamLeague_Championship_Record JSON DEFAULT NULL,
    Teams_History JSON DEFAULT NULL,
    FOREIGN KEY (Team_ID) REFERENCES Teams(Team_ID) ON DELETE SET NULL
);
```

✔ Key Fields

- `Player_ID` → Unique identifier for each player.
- `Real_Name` → The **player's actual name or alias (if real name is unknown)**.
- `Team_ID` → The **player's current team** (links to `Teams` table).
- `Championship_Record` (JSON) → **Tracks personal championship achievements**.
- `TeamLeague_Championship_Record` (JSON) → **Tracks team championship achievements**.
- `Teams_History` (JSON) → **Tracks historical teams a player has been part of**.

✅ **Historical teams are stored in `Teams_History` JSON instead of a separate table.**

2️⃣ Player_Aliases Table (Tracking In-Game Names)

A **player can have multiple aliases** used in different games or tournaments.

sql
CopyEdit

```
CREATE TABLE Player_Aliases (
    Alias_ID INT AUTO_INCREMENT PRIMARY KEY,
    Player_ID INT NOT NULL,
    Alias_Name VARCHAR(255) NOT NULL UNIQUE,
    FOREIGN KEY (Player_ID) REFERENCES Players(Player_ID) ON DELETE
CASCADE
);
```

✔ Key Fields

- `Alias_ID` → Unique identifier for each alias.
- `Player_ID` → Links to the **real player**.
- `Alias_Name` → The **in-game name** used by the player.

✅ **Allows tracking of multiple in-game names per player.**

3️⃣ FSL_STATISTICS Table (Cumulative Player Performance)

Holds **cumulative statistics** for **each player-division-race combination**.

sql
CopyEdit

```
CREATE TABLE FSL_STATISTICS (
    Player_Record_ID INT AUTO_INCREMENT PRIMARY KEY,
    Player_ID INT NOT NULL,
    Alias_ID INT NOT NULL,
    Division VARCHAR(10) NOT NULL,
    Race CHAR(1) NOT NULL,
    MapsW INT DEFAULT 0,
    MapsL INT DEFAULT 0,
    SetsW INT DEFAULT 0,
    SetsL INT DEFAULT 0,
    FOREIGN KEY (Player_ID) REFERENCES Players(Player_ID) ON DELETE
CASCADE,
    FOREIGN KEY (Alias_ID) REFERENCES Player_Aliases(Alias_ID) ON
DELETE CASCADE,
    UNIQUE (Player_ID, Alias_ID, Division, Race)
);
```

✔ Key Fields

- `Player_Record_ID` → Unique ID for each stat entry.
- `Player_ID` → Links to `Players` table.
- `Alias_ID` → Links to `Player_Aliases` table.
- `Division` → The **competitive division** (`A`, `B`, `S`, etc.).
- `Race` → The **race played** (`T`, `P`, `Z` for Terran, Protoss, Zerg).
- `MapsW`, `MapsL`, `SetsW`, `SetsL` → **Cumulative** wins/losses for **maps and sets**.

✅ **Each player only has one stat record per `Division` & `Race` to prevent duplication.**

4️⃣ Teams Table (Organizes Team Data)

Holds **team details** and tracks **captains**.

sql
CopyEdit

```
CREATE TABLE Teams (
    Team_ID INT AUTO_INCREMENT PRIMARY KEY,
    Team_Name VARCHAR(255) NOT NULL UNIQUE,
    Captain_ID INT DEFAULT NULL,
    Co_Captain_ID INT DEFAULT NULL,
    FOREIGN KEY (Captain_ID) REFERENCES Players(Player_ID) ON DELETE
SET NULL,
    FOREIGN KEY (Co_Captain_ID) REFERENCES Players(Player_ID) ON
DELETE SET NULL
);
```

✔ Key Fields

- `Team_ID` → Unique identifier for each team.
- `Team_Name` → The **team's name**.
- `Captain_ID`, `Co_Captain_ID` → Identifies **team leaders**.

✅ **Current team is linked via `Players.Team_ID`, and historical teams are in `Players.Teams_History` JSON.**

📌 Sample Queries

1️⃣ Comprehensive Query (All Player Data)

sql
CopyEdit

```
SELECT
    p.Player_ID,
    p.Real_Name AS Player_Name,
```

```
    a.Alias_ID,
    a.Alias_Name,
    s.Division,
    s.Race,
    s.MapsW,
    s.MapsL,
    s.SetsW,
    s.SetsL,
    t.Team_ID AS Current_Team_ID,
    t.Team_Name AS Current_Team_Name,
    p.Championship_Record,
    p.TeamLeague_Championship_Record,
    p.Teams_History AS Past_Team_History
FROM Players p
LEFT JOIN Player_Aliases a ON p.Player_ID = a.Player_ID
LEFT JOIN FSL_STATISTICS s ON p.Player_ID = s.Player_ID AND a.Alias_ID
= s.Alias_ID
LEFT JOIN Teams t ON p.Team_ID = t.Team_ID
ORDER BY t.Team_Name, p.Real_Name, s.Division, s.Race;
```

✅ **Pulls all player stats, aliases, current team, and championship history.**

2️⃣ Find Players Without a Current Team

sql
CopyEdit

```
SELECT p.Player_ID, p.Real_Name
FROM Players p
WHERE p.Team_ID IS NULL;
```

✅ **Finds players who are unassigned.**

3️⃣ Find Players Without Statistics

sql
CopyEdit

```sql
SELECT p.Player_ID, p.Real_Name
FROM Players p
LEFT JOIN FSL_STATISTICS s ON p.Player_ID = s.Player_ID
WHERE s.Player_Record_ID IS NULL;
```

✅ **Finds players with no recorded stats.**

4️⃣ Get a Player's Full Championship & Team History

sql
CopyEdit

```sql
SELECT
    p.Real_Name AS Player_Name,
    p.Championship_Record,
    p.TeamLeague_Championship_Record,
    p.Teams_History
FROM Players p
WHERE p.Real_Name = 'DarkMenace';
```

✅ **Displays JSON history for championships and teams.**

📌 Summary of Final Changes

✔ **Removed `Team_History` table (replaced by `Teams_History` JSON field in `Players`).**
✔ **Added `Team_ID` to `Players` for active team tracking.**
✔ **Simplified team and player queries.**
✔ **Maintains historical tracking in JSON while ensuring clean, normalized relational data.**