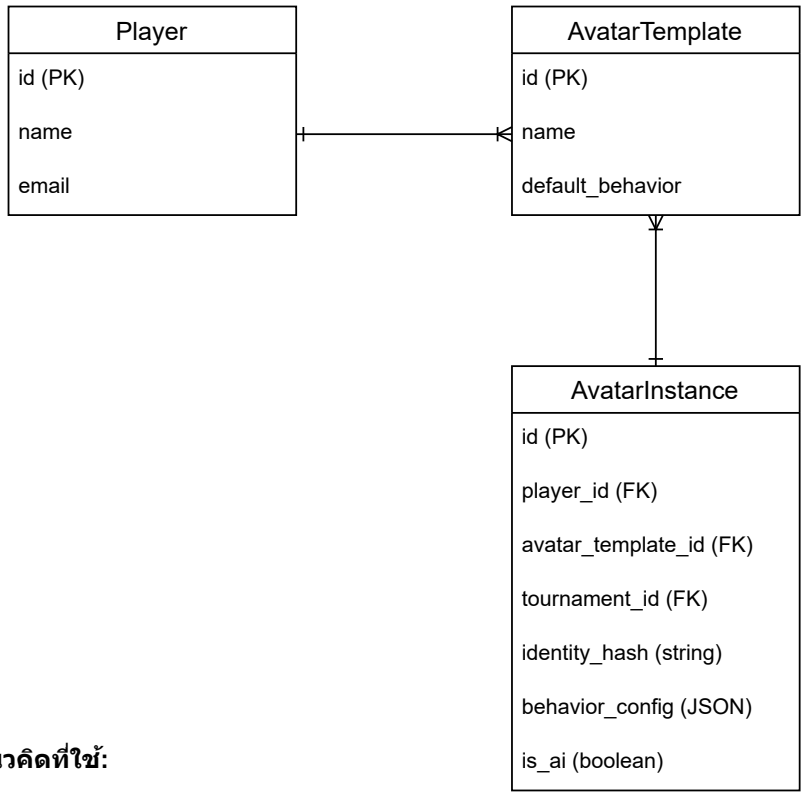
Module 1: Player / Avatar



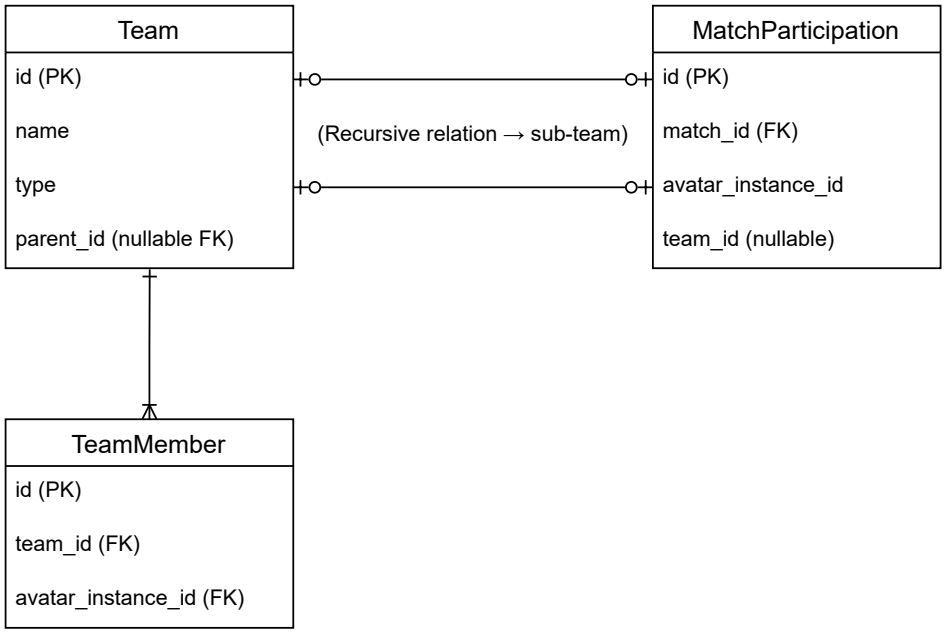
💬 แนวคิดที่ใช้:

Entity	อธิบาย
`Player`	- ผู้ใช้งานระบบ (1 คน)
`AvatarTemplate`	- เหมือนแบบฟอร์ม/โมเดลของ Avatar ที่ Player เลือกหรือฝึกไว้
`AvatarInstance`	- แต่ละ Tournament จะมี Avatar เป็น *Instance* ใหม่ (สามารถมี behavior ต่างกัน)
`identity_hash`	- ใช้ distinguish ตัว Avatar ในแต่ละ Tournament
`behavior_config`	- JSON ที่เก็บค่าการตั้งค่า AI, Skill, Parameter ต่าง ๆ
is_ai	- ใช้สำหรับ Identify ว่า Avatar ตัวนั้นๆเป็น Player หรือ Ai ในขณะนั้นๆ เพื่อ Watching

✅ เหตุผลที่ออกแบบแบบนี้:

- แยก `AvatarTemplate` ออกมา → ทำให้เราสามารถ “re-use” แบบ Avatar ได้
- มี `AvatarInstance` ผูกกับ Tournament(tournament_id) ทำให้ชื่อซ้ำได้, พฤติกรรมเปลี่ยนได้
- เก็บ config เป็น JSON → รองรับ behavior ที่เปลี่ยนตลอดเวลาโดยไม่ต้องแก้ schema
- มี is_ai เพื่อนำออกมานอกสุดง่ายต่อการ watching และเปลี่ยนแปลงตลอดเวลา เช่น
  เกมส์ csgo : ถ้าหาก player หลุดการเชื่อมต่อกับ Avatar ข้อมูล player ที่เก็บไว้ก็อาจจะยังใช้อยู่
  และ avatar ก็ยังคงทำงานได้โดยการเปลี่ยน is_ai เป็น true เพื่อให้ AI เข้ามาทำงานแทน player
  รวมถึงตรวจสอบการส่อง bahavior สำหรับ AI ที่มีการเปลี่ยนแปลงไปได้ตลอดเวลา

Module 2: Team / Sub-Team / Participation



💬 แนวคิดที่ใช้:

Entity	อธิบาย
`Team`	- หน่วยหลักที่ใช้รวม AvatarInstance
`parent_id`	- ใช้สำหรับ Sub-team (Team เป็น tree/graph ได้)
`TeamMember`	- ระบุสมาชิกในแต่ละทีม (สามารถเปลี่ยนตลอดเวลา)
`MatchParticipation`	- ใครบ้างที่เข้าร่วม Match นี้ และในฐานะอะไร
`team_id` ใน MatchParticipation	- ทำให้ Avatar คนเดียวกันสามารถเข้าร่วมแมตช์ในทีมที่ต่างกันได้

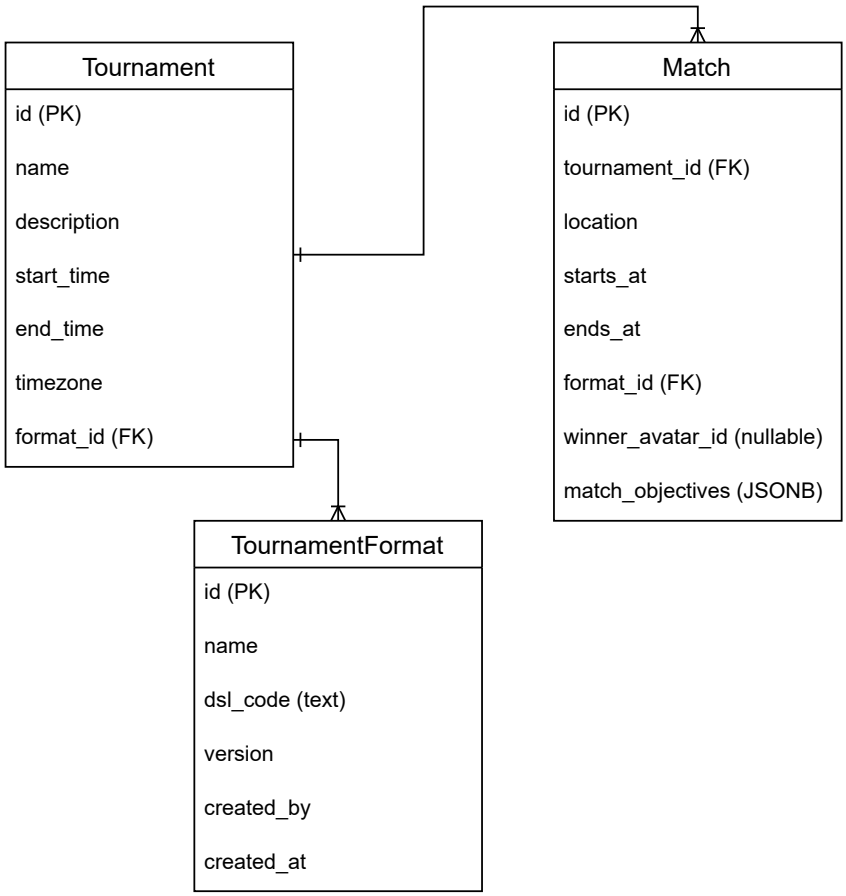
✅ เหตุผลการออกแบบ:

- ใช้ **Recursive FK (`parent_id`)** → รองรับ Sub-team ไม่จำกัดชั้น (หรือเราจะจำกัดชั้นไว้ก็ได้)
- แยก `MatchParticipation` ออก → รองรับ Avatar ที่เปลี่ยนทีมได้แบบ dynamic
- เก็บ Role แบบ Text/Enum → ยืดหยุ่นต่อการเปลี่ยนแปลง เช่น Tank / Sniper / Support

🚫 จุดที่ “ยอมแพ้” อย่างมีเหตุผล:

- **ไม่ทำ Many-to-Many Relationship ตรงกลางเชื่อม ระหว่าง Avatar กับ Team** เพราะ:
    - จะซับซ้อนเกินไปเวลาซ้อน Sub-team + Match Participation
    - เลือกใช้ TeamMember → ชัดเจนกว่า

Module 3: Tournament / Match / Game Format



💬 แนวคิด:

Entity	อธิบาย
`TournamentFormat`	- เก็บ DSL ที่ผู้ใช้กำหนด — version ได้
`Tournament`	- อิง format, เก็บเวลาแบบ timezone-aware
`Match`	- อิง Tournament แต่ override format ได้เอง
`match_objectives`	- เก็บเงื่อนไขพิเศษแบบ custom เช่น “Kill X คน”, “Stealth Time”
`winner_avatar_id`	- nullable → รองรับกรณี “ไม่มีผู้ชนะ” ได้

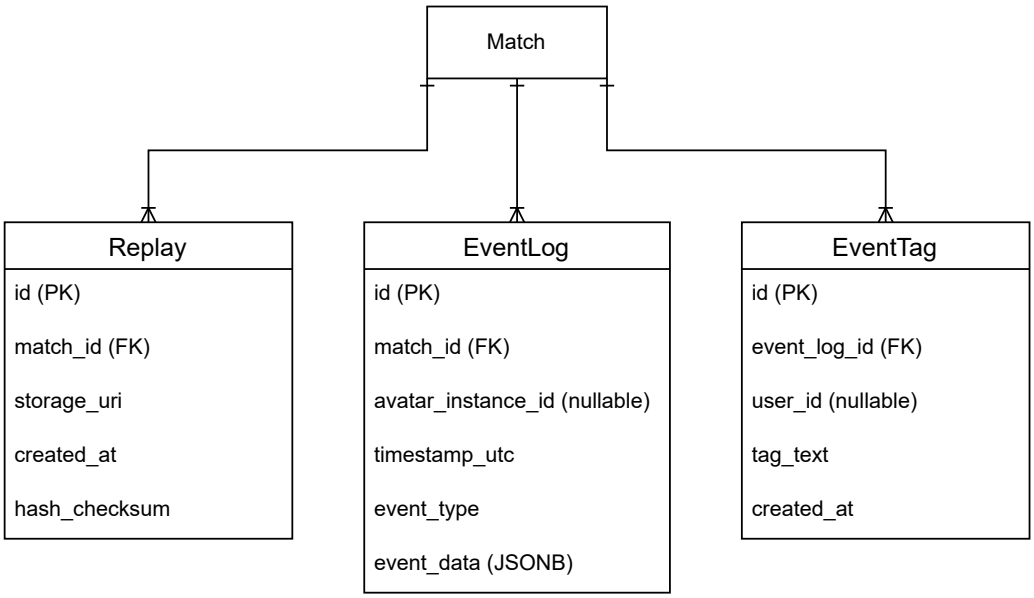
✅ เหตุผลการออกแบบ:

- แยก DSL ออกมาใน `TournamentFormat` → เพื่อรองรับ version และผู้ใช้ที่เขียน logic ได้เอง
- ใช้ `match_objectives` เป็น JSON → รองรับ logic ที่เปลี่ยนตามเกมหรือเป้าหมายเฉพาะ
- รองรับ match พร้อมกันหลาย location ด้วย `starts_at`, `timezone`, `location` → ใช้ร่วมกับ Timezone-aware indexing

🚫 จุดที่ “ยอมแพ้” อย่างมีเหตุผล:

- ไม่ normalize `match_objectives` เป็นตาราง → เพราะ logic มันแตกต่างกันมาก (game A vs game B)
    - ใช้ JSONB พร้อม validation ทาง backend แทน
- Tournament format ถูกออกแบบแบบ Metadata-driven ผ่าน DSL และมี version ซึ่ง mapping กับ tournament แต่ override ได้ในระดับ match ช่วยให้รองรับ game format ที่ custom ได้แบบไม่เปลี่ยน schema

Module 4: Replay / Event Log / Tagging



💡 แนวคิดที่ใช้:

Entity	อธิบาย
`Replay`	- เก็บ URI ของ replay (บน IPFS / Arweave ฯลฯ) พร้อม hash เพื่อการอ้างอิงแบบ Immutable
`EventLog`	- เก็บ event แบบ granular → ทุก action ที่เกิดใน match นั้น
`EventTag`	- ผู้ชมสามารถ tag event ในระหว่าง live หรือย้อนหลังได้

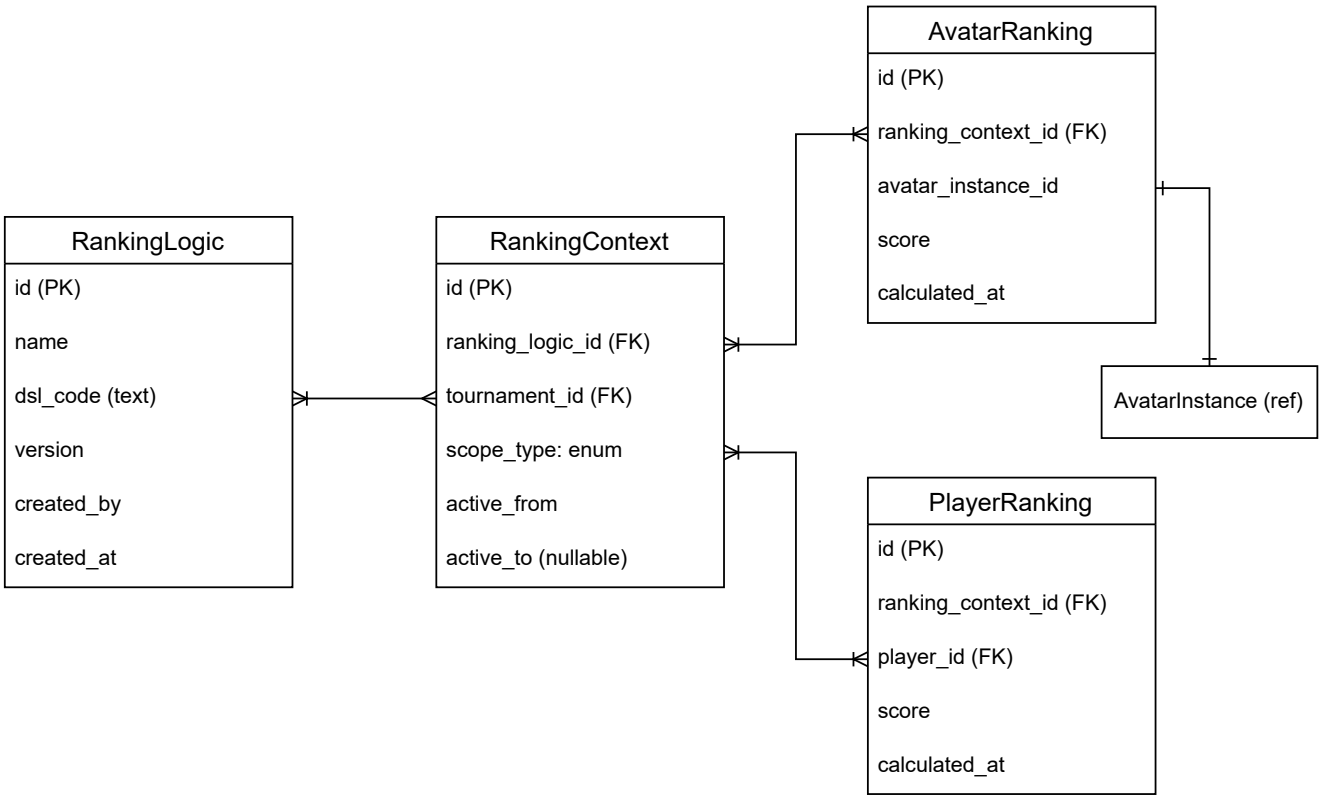
✅ เหตุผลการออกแบบ:

- การแยก `EventLog` และ `Replay` ออกจากกัน → ช่วยให้ mapping / query ได้ง่าย โดยใช้ `timestamp`
- เก็บ `event_data` แบบ JSON → เพื่อรองรับ event type ใหม่ในอนาคตโดยไม่แก้ schema
- `EventTag` ช่วยเพิ่มความ “มีชีวิต” ให้กับ replay (อารมณ์แบบ YouTube comments on timestamp)

🚫 จุดที่ยอมแพ้:

- เรา **ไม่เก็บทุก snapshot ของระบบทุกวินาที** เพราะมันใหญ่เกินไป ทางแก้คือใช้ **temporal log + ETL สร้าง snapshot แบบ periodic** (เช่น ทุก 10 วินาที)

Module 5: Ranking System



💬 แนวคิด:

Entity	อธิบาย
`RankingLogic`	- เก็บ DSL logic ที่ผู้ใช้สร้าง (versioned)
`RankingContext`	- เป็น instance ของ logic + scope → ใช้ใน Tournament ไหน? ช่วงเวลาไหน?
`PlayerRanking` / `AvatarRanking`	- เก็บคะแนนแต่ละรอบที่ calculate มาแล้ว

✅ เหตุผลการออกแบบ:

- แยก `RankingLogic` ออกต่างหาก → เพื่อให้รองรับ DSL versioned และ track ได้แม้ logic ถูกลบ
- มี `RankingContext` → ช่วยให้เรารู้ว่า logic ไหน active ตอนไหน
- การ recalculation สามารถทำได้แบบ selective โดยไล่ query ว่า context ไหนอยู่ในช่วงเวลาไหน
- แยกตารางสำหรับ Player และ Avatar เพื่อความยืดหยุ่น (บางเกมมีแต่ avatar, บางเกมอิง player โดยตรง)

🚫 จุดที่ยอมแพ้:

- เรา **ไม่บันทึกผลการคำนวณทุกครั้งแบบ real-time ทั้งหมด** เพราะมันใหญ่เกินไป ใช้ cron-based recalculation + cache + audit log แทน