Data Architecture, System & Migration

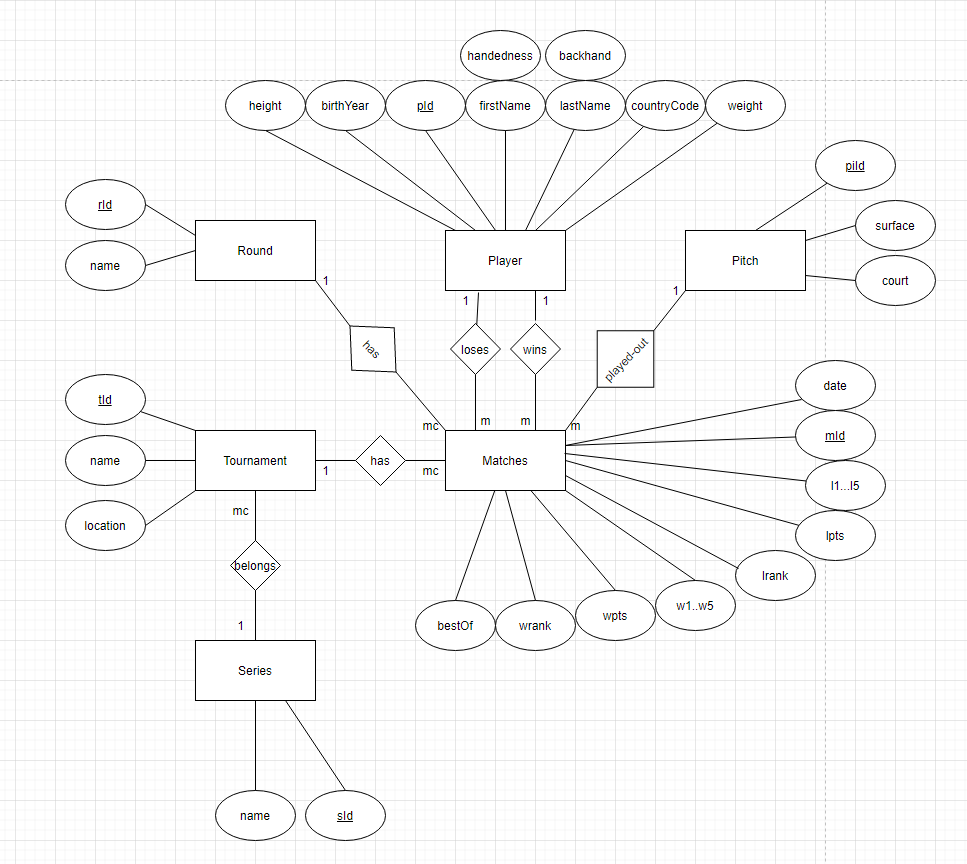
1. Build the system based on a clear system architecture

Um unsere Daten sauber und schnell auswerten zu können, war es uns wichtig die

1. Defince conceptual data model & database schema

Um mit unseren Daten sauber und schnell arbeiten zu können, war es uns wichtig die Informationen sinnvoll zu trennen. Eine Trennung von gewissen Informationen vermeidet Redundanten wie zum Beispiel beim Spielername. Pro match wollen wir über eine Forgeign Key einen Spieler referenzieren und nicht sein Name in der Match Tabelle verwalten. Diese Informationen nicht zu trennen ist schlecht für die Performance und wäre bei Änderungen sehr mühsam. Bei Änderungen müsste man ansonsten bei jedem Match der Name anpassen, durch die sinnvolle Trennung nur an einem Ort und nur einmal.

Mit diesem Gedanke erarbeiteten wir folgendes initial ER-Modell:



ER-Model 1.0.0 Beschrieb.

Mit diesem ER-Modell wird gezeigt, wie man die Grundlegenden Daten in Tabellen gliedert. Die Tabellen die wir dann für die Berechnungen der Simulation verwenden, waren hier noch nicht definiert.

Der Zentrale Punkt welches die meisten Beziehungen involvieren die „Matches“-Tabelle. Matches hat pro Match(Tupel) 2 referenzierte Spieler, eine referenzierte Runde, ein referenzierten Pitch, referenziertes Turnier und sein Attribute.   
Da hier gewisse Attributname unklar seien könnten, sind hier Definitionen.

|  |  |
| --- | --- |
| Attribute | Definition |
| Wrank | Den Weltrang des Gewinners beim Zeitpunkt von diesem Match |
| Lrank | Den Weltrang des Verlierers beim Zeitpunkt von diesem Match |
| Wpts | Die Punkte des Gewinners in der Weltrangliste |
| Lpts | Die Punkte des Verlierers in der Weltrangliste |
| w1 (bis w5) | Die Punkte des Gewinners pro Set |
| l1 (bis l5) | Die Punkte des Verlierers pro Set |
| Date | Datum des Spieltages |

Der dazugehöriger SQL-Code findet man in create-0.1.0.sql.

1. Migrating existing data into the database

Hier ist der Migrationsprozess in 3 einfachen Schritten aufgeteilt.

SCHRITT 1 CSV Import

Wie im Kapitel 1C definiert, entschieden wir die CVS von xxx zu verwenden.  
Die Herausforderung hier war, diese Daten in die einzelnen Tabellen aufteilen zu können.  
Um dies zu verwirklichen haben wir das csv importiert mit SQL workbench.  
Dank diesem Import wurde die folgende Tabelle erzeugt

\*pic\*

SCHRITT 2 Aufteilen mittels SQL

Diese Tabelle hat immer noch das Problem, dass viele Informationen redundant sind, sehr fest gekoppelt sind und nicht unserem Schema übereinstimmen. Mittels SQL wurde dieses Problem jedoch gelöst. Das Script migrate-temp-tables.sql befüllt unsere Tabellen mittels der generierten Tabelle.

SCHRITT 3 Kontrolle und Cleanup

Als letzter Schritt sollte kurz kontrolliert werden, ob das Befüllen funktioniert hat. Alles kann man hier nicht testen, jedoch ein paar Datensätze zu kontrollieren ist hier empfohlen.  
Falls gewünscht kann die vom Import generierten Tabelle hier gelöst werden

Zu beachten:

Da diese csv pro Jahr aufgeteilt sind, muss dieser Prozess für jedes Jahr wiederholt werden.

1. Analyse the system architecture for its security protection

Da die Zielgruppen Websieten, Wettbüros, Videospiel Entwickler und ähnliche sind. Wird der Zugriff am über eine REST-Schnittstelle ermöglicht. Der Endbenutzer wird daher nie direkt mit der Datenbank arbeiten, dies soll sicherstellen, dass der Endbenutzer nicht die Daten böswillig manipulieren kann.

Weitere Gedanken die bezügliche Sicherheit gemacht worden sind:

* Da es nicht um klassifizierte Daten handelt, müssen wir uns nur auf die Integrität, Konsistenz und Verfügbarkeit fokussieren.
* Um die Integrität zu gewährleisten, dürfen nur Administratoren Datensätze ändern. Über die REST-Schnittstelle können die Daten nicht manipuliert werden.
* Um sicherzustellen das wir die Daten nicht verlieren, führen wir Backups der Daten durch.

Glossar (ER = Entity-Relationship )