Freie Universität Berlin, Institut für Informatik
Datenbanksysteme
Sommersemester 2014

Dokumentation des Arbeitsablaufs

KickAss

Mitglieder

David Bohn Fabian Reimeier Luca Keidel

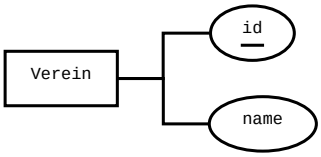
https://github.com/dbohn/KickAss

Inhaltsverzeichnis

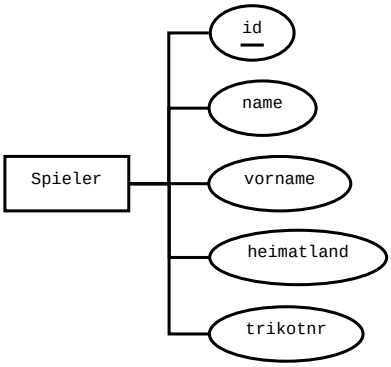
1 Entwicklung des Schemas

1.1 Grundkonzept

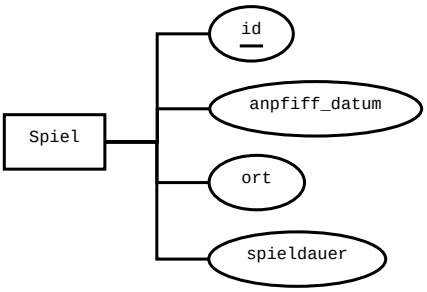
Unser erstes Konzept bestand hauptsächlich aus den Entitäten `Verein`, `Spieler`, `Spiel` und `Liga`. Da wir zu diesem Zeitpunkt noch keine Informationen über die Daten hatten, die wir zu Verfügung haben würden, haben wir den Entitäten alle grundlegenden und uns wichtig erscheinenden Attribute zugeordnet. Somit haben wir zunächst folgende Entitäten konstruiert:
Verein:



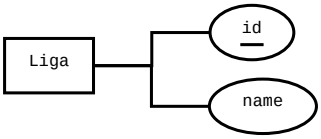
Spieler:



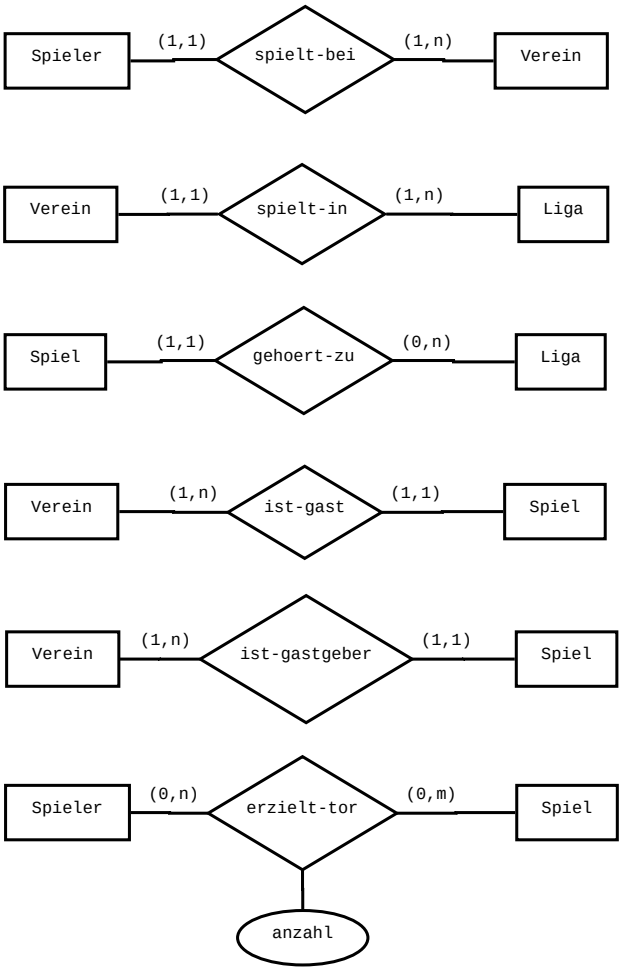
Spiel:



Liga:

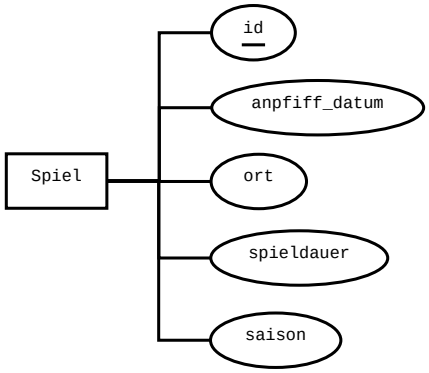


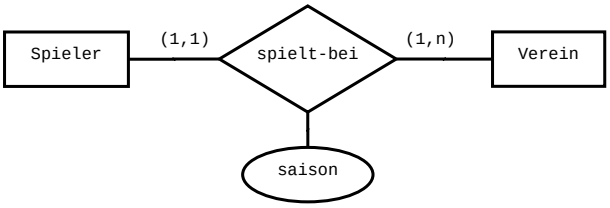
Wir haben des Weiteren angenommen, dass in einem Verein mindestens ein Spieler Mitglied sein muss und ein Spieler bei genau einem Verein spielt. Der Verein spielt in einer bestimmten Liga, in der mehrere Spiele stattfinden. Ein Verein kann in diesen Spielen entweder Gast oder Gastgeber sein und seine Spieler können während der Spiele eine bestimmte Anzahl an Toren erzielen. Daraus ergaben sich folgende Relationen mit Kardinalitäten in umgekehrter Chen-Notation. Die Attribute der Entitäten werden aus Übersichtlichkeitsgründen nicht gezeigt.

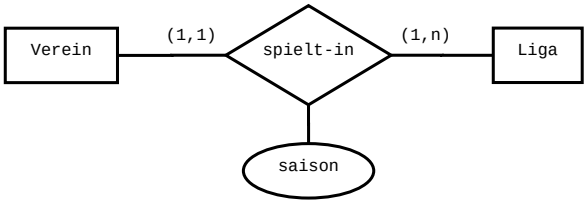


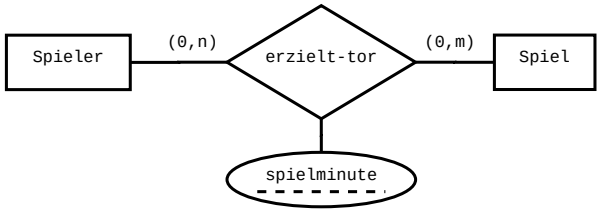
1.2 Einführung von Saison

Da unsere Anwendung saisonspezifische Antworten geben soll, mussten wir unsere Entität `Spiel` und einige unserer Relationen um `saison`- Attribute erweitern. So kann ein Spieler in unterschiedlichen Saisons bei unterschiedlichen Vereinen spielen und ein Verein in unterschiedlichen Ligen (schließlich ist es möglich, dass Vereine ab- oder aufsteigen). Desweiteren haben wir eine Änderung an der `schiesst-tor` Relation vorgenommen. Da wir die Anzahl der Tore eines Spielers auch durch die Anzahl der Einträge in der Relation bestimmen können, brauchen wir dieses Attribut nicht. Wir ersetzen es aber mit dem Attribut `spielminute`. Da ein Spieler in einem Spiel mehrere Tore schießen kann, ist die `spielminute` ein Teilschlüssel.







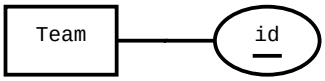


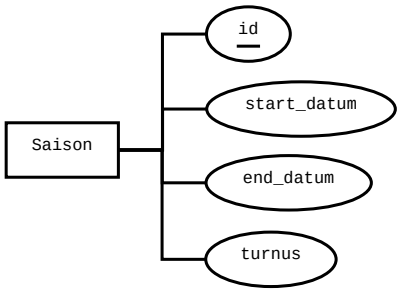
Offensichtlich offenbart sich aber hier schon ein Problem: Das Speichern der Saison als Attribut führt zu Redundanz. Dies beheben wir, indem wir die Saison als eine neue Entität auslagern.

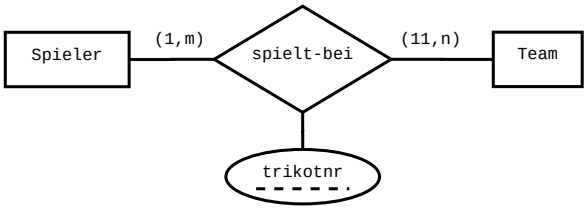
1.3 Einführung von Teams und Saison als Entität

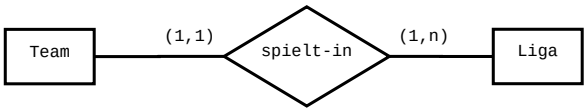
Die Modellierung von Saisons als dedizierte Entität würde zu einer Relation zwischen drei Entitäten führen (Spieler spielen während einer Saison bei einer Mannschaft). Aus diesem Grund haben wir noch eine weitere Entität eingeführt, `Team`. Ein Team besteht aus mindestens 11 Spielern. Des Weiteren haben wir die Entität `Verein` um die Attribute `ort` und `heimatstadion` erweitert. Für die Saison haben wir eine eigene Entität mit den Attributen *id*, `start_datum`, `end_datum` und `turnus` hinzugefügt, um mehr Aussagen treffen zu können. Die zuvor eingefügten `saison`-Attribute haben wir wieder entfernt.

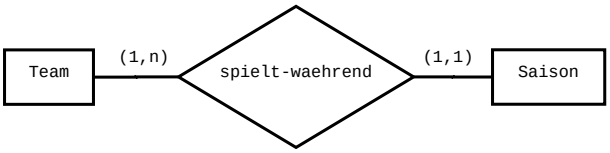
Ein Spieler spielt nun also bei einem Team, welches wiederum während einer bestimmten Saison in einer bestimmten Liga für einen bestimmten Verein spielt. Da Spieler in unterschiedlichen Teams eines Vereins die gleiche Trikotnummer tragen können, verschieben wir das Attribut `trikotnr` zur Relation `spielt-bei` , wo es einen Teilschlüssel darstellt. Außerdem finden alle Spiele finden in einer bestimmten Saison statt:

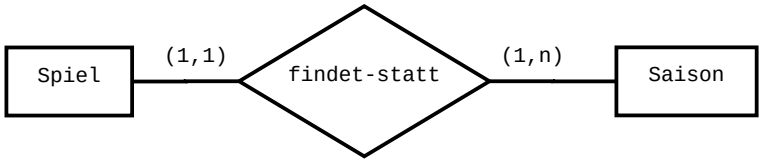






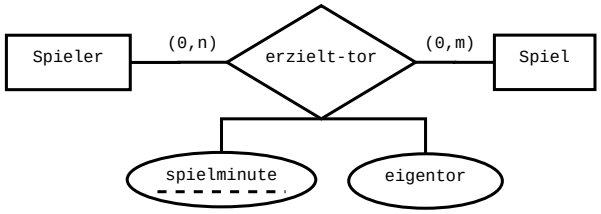






1.4 Eigentore

Im bisherigen Model war es nicht möglich Eigentore darzustellen. Deswegen haben wir der Relation `erzielt-tor` ein Attribut `eigentor` gegeben.



1.5 Anpassung an die Bundesliga Datenbank der Universität Bayreuth

Die Uni Bayreuth bietet eine Datenbank mit folgendem Relationsschema:

Verein(V_ID: int, Name: varchar, Liga: int)

Spiel(Spiel_ID: int, Spieltag: int, Datum: date, Uhrzeit: time, Heim: int, Gast: int, Tore_Heim: int, Tore_Gast: int)

Spieler(Spieler_ID: int, Vereins_ID: int, Trikot_Nr: int, Tore: int, Spieler_Name: varchar, Land: varchar, Spiele: int, Tore: int, Vorlage: int)

Liga(Liga_Nr: int, Verband: varchar, Erstaustragung: date, Meister: int, Rekordspieler: varchar, Spiele_Rekordspieler: int)[1]

Da wir bisher noch keine genaue Informationen zu den Daten, die unsere Anwendung als Grundlage nutzen sollte, besaßen, mussten wir unser Schema deshalb an die konkrete Struktur anpassen.

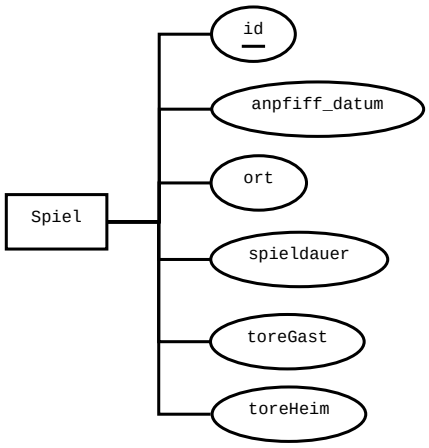
Probleme:

- fehlende Informationen in unserem ER-Diagramm:

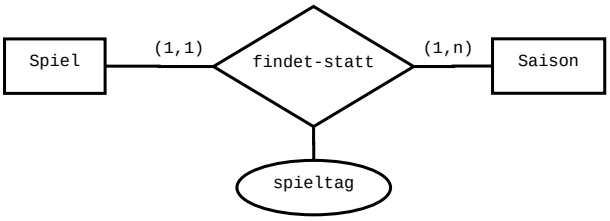
[1] Quelle: http://dbup2date.uni-bayreuth.de/bundesliga.html

  - Spieltage
  - Heimtore/Gasttore als Attribut
  - Vorlage (nicht benötigt)
  - Verband (nicht benötigt)
  - Uhrzeit (nicht benötigt)
- mit den Daten der Datenbank nicht darstellbar:
  - Turnus
  - Eigentore
  - Vorname getrennt vom Nachnamen eines Spielers
  - Tore eines Spielers pro Spiel
  - Spielminute der Tore
  - Spieldauer
  - mehrere Teams pro Verein (nur ein Team pro Verein gegeben)

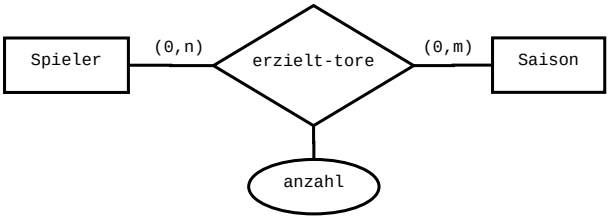
Wir haben zunächst der Entität `Spiel` alle fehlenden Attribute gegeben.



Um den Spieltag darzustellen haben wir das Attribut `spieltag` an die Relation `findet-statt` gehängt.



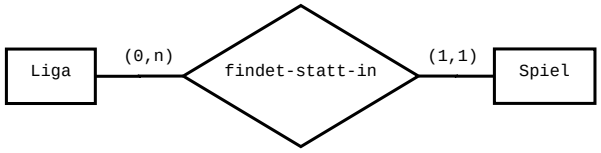
Da die Datenbank nur Daten über die Toranzahl eines Spielers pro Saison bietet, nicht aber pro Spiel, konnten wir die Relation `erzielt-tor` mit ihrem Attribut `eigentor` löschen und durch folgende ersetzen.

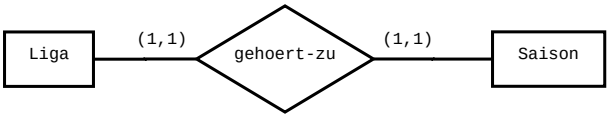


Das Attribut `vorname` der Entität `Spieler` genauso wie das `turnus` Attribut der `Saison` wurde gelöscht.

1.6 Unterstützung mehrerer Saisons

Um mehrere Saisons zu unterstützen haben wir die `gehört-zu` Relation in `findet-statt-in` umbenannt und eine neue Relation `gehoert-zu` zwischen `Liga` und `Saison` eingefügt.





2 Implementierung

2.1 Technik

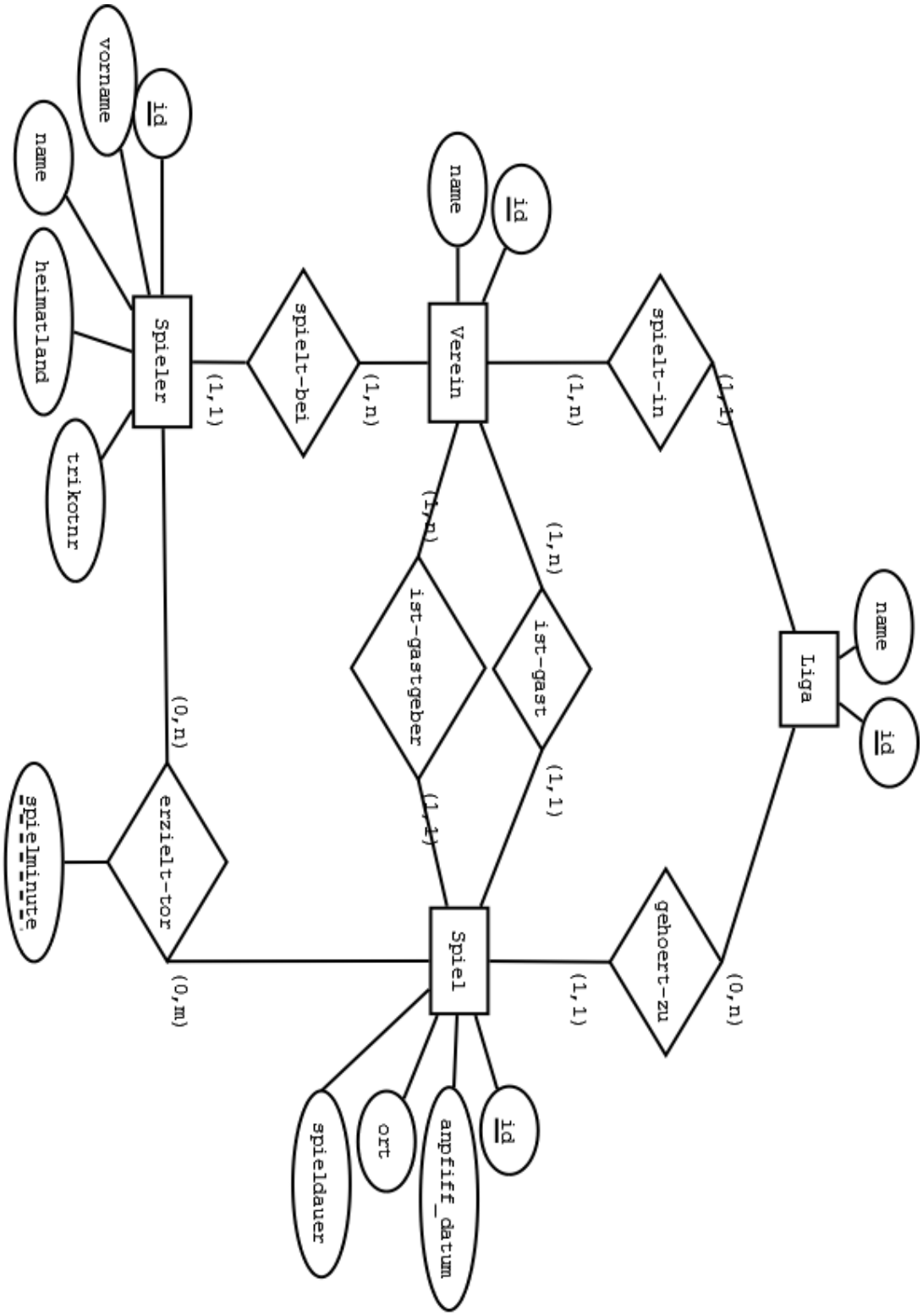
Wir haben uns für die Verwendung von PHP für den primären Datenbankzugriff entschieden. Als Datenbank kommt PostgreSQL zum Einsatz. Für die Weboberfläche verwenden wir (neben den offensichtlichen Technologien wie HTML und CSS) AngularJS. Für den Zugriff auf die Datenbank in PHP verwenden wir *PHP Data Objects* (PDO), die vereinfacht ausgedrückt den eigentlichen Treiber für die Datenbank kapseln und eine einheitliche Schnittstelle für verschiedene DBMS bieten. Außerdem bietet PDO noch die Möglichkeit Prepared Statements zu verwenden.

2.2 Architektur

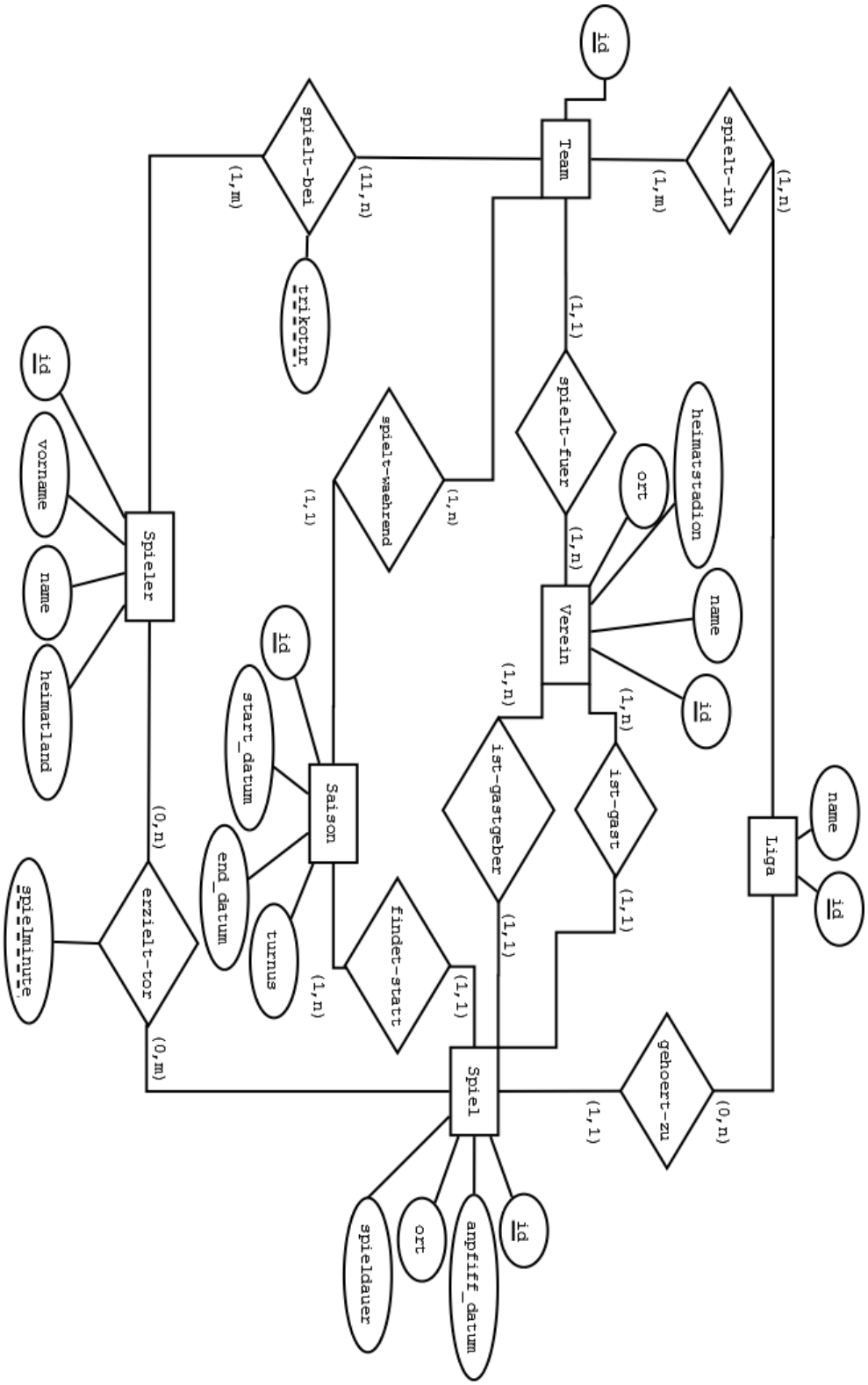
Wir haben uns für eine strikte Trennung von Weboberfläche und Zugriff auf die Datenbank entschieden. Konkret bedeutet dies, dass wir die Queries und die ganze Programmlogik zum Datenbankzugriff hinter einer API (Application Programming Interface) verborgen haben. Bei der API handelt es sich um eine REST-API, die die Daten im JSON-Format zurückliefert. Die Weboberfläche verwendet diese API und ruft die benötigten Datein mittels AJAX-Requests ab. Für eine genauere Dokumentation verweisen wir an dieser Stelle auf die Dokumentation der API oder unsere allgemeine technische Dokumentation.

3 Anhang

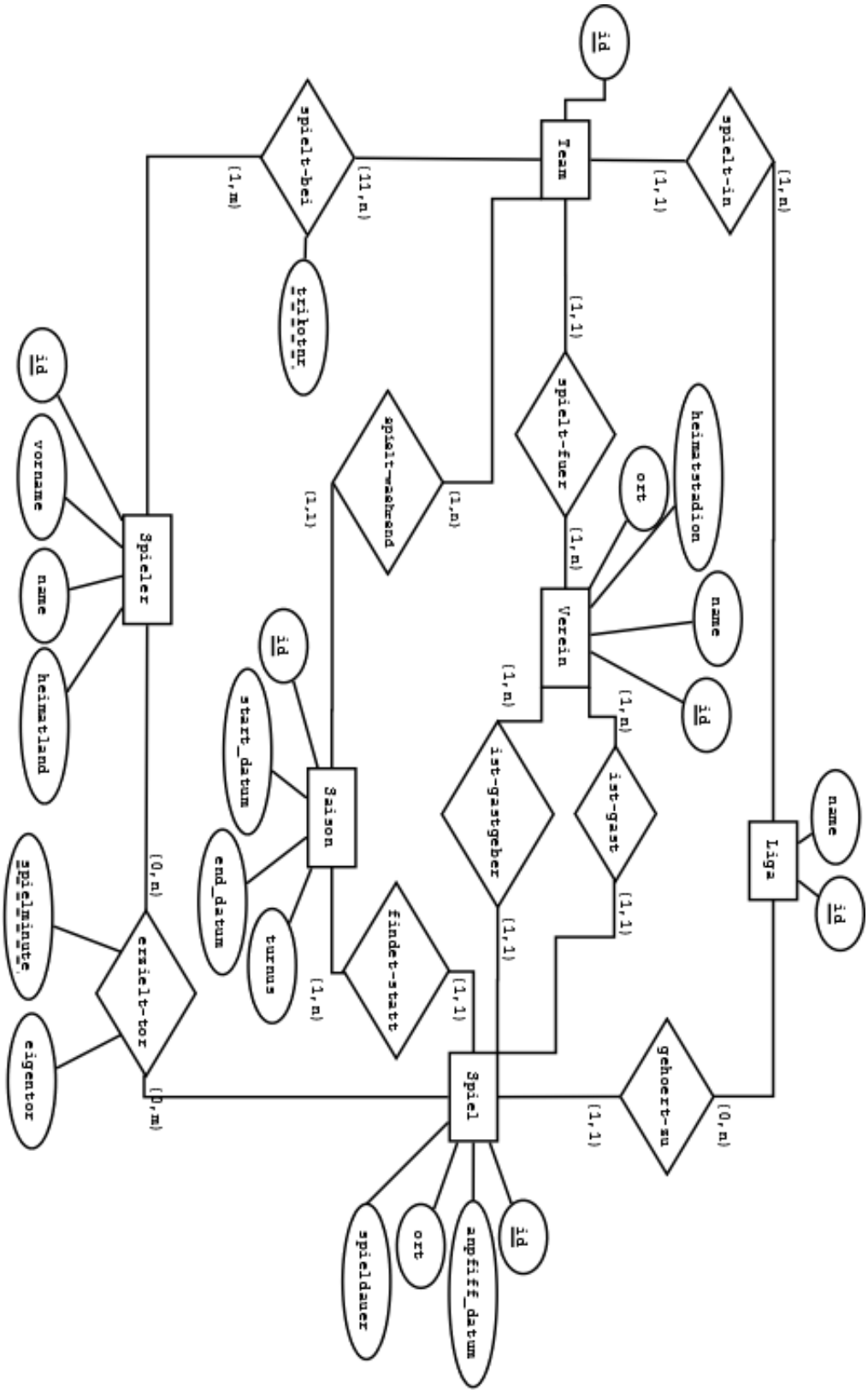
3.1 Grundkonzept



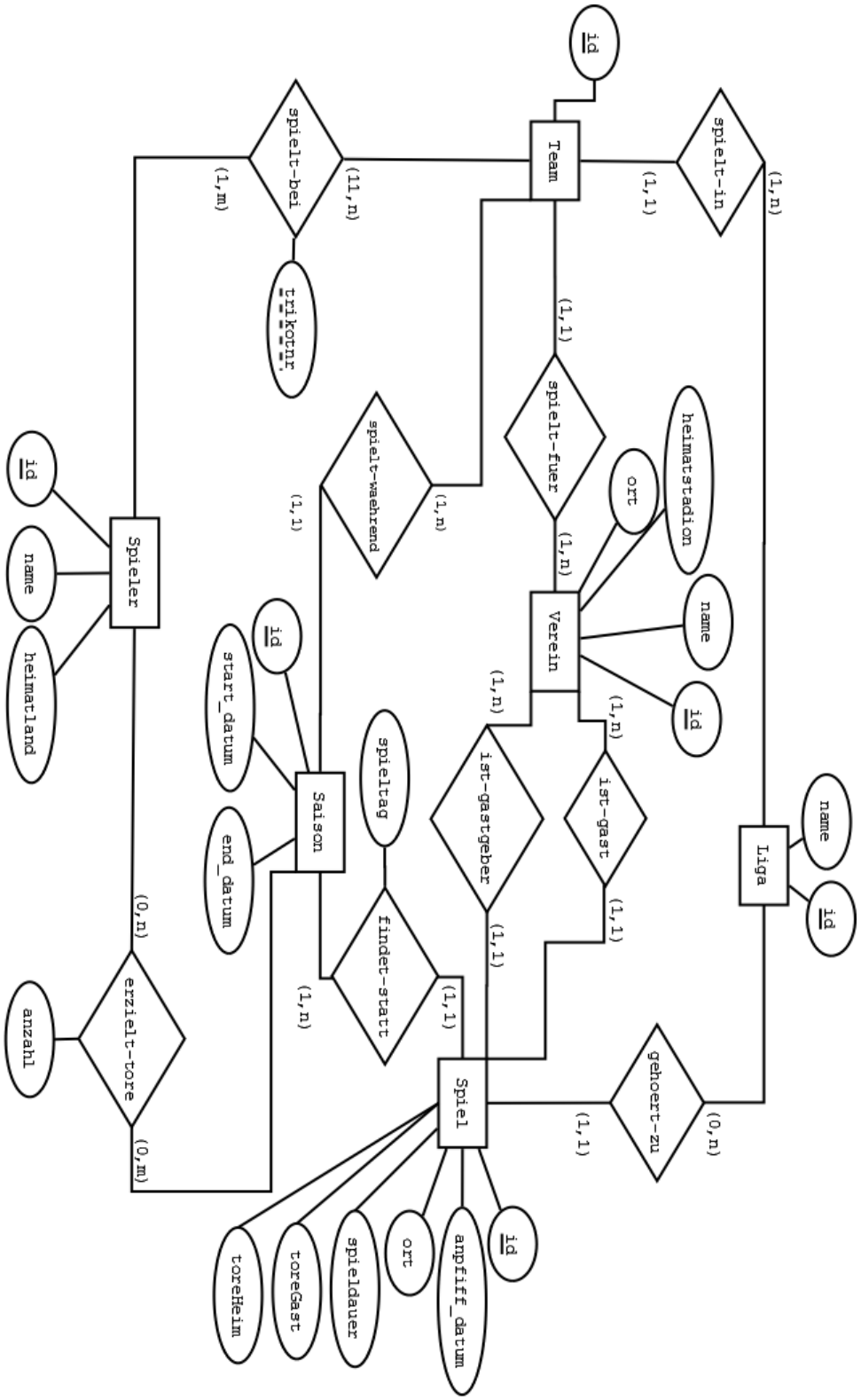
3.2 Einführung von Teams und Saison als Entität



3.3 Eigentore



3.4 Anpassung an die Bundesliga Datenbank der Universität Bayreuth



3.5 Unterstützung mehrerer Saisons