Projektarbeit

Sportturniere Datenbank

Timo Johannsen, Benjamin Peiter

Abgabe 25.11.2025

Inhalt

Szenario

An vielen Hochschulen werden organisierte Liegen und Turniere für teamsportarten wie Fußball, Volleyball, Basketball angeboten.

Aktuell werden Teams, Spielpläne und Ergebnisse oft in Excel-Listen oder über mehrere Textdateien festgehalten. Solche Lösungen führen schnell zu Redundanzen, Dateninkonsistenz und hohen Pflegeaufwand führen. Besser ist es eine Datenbank zur Verwaltung zu nutzen.

Im vorliegenden Projekt wird diese Idee auf Hochschulniveau heruntergebrochen. Es wird eine Sportturnier-Datenbank entworfen, mit der das Hochschulsportzentrum

- mehrere Teamsportarten
- mit Ligen, Spieltagen, Playoffs,
- sowie Teams, Spielern, Sch iedsrichtern und Spielorten

einheitlich verwalten kann. Dabei stehen Überschaubarkeit und repräsentative Strukturen im Vordergrund, da sich das Projekt nach einem realistischen Hochschul-Liga-Szenario richtet.

Zielsetzung und Nutzen

Ziel des Projekts ist die Entwicklung eines normalisierten rationalen Datenbankschemas, das alle für die Hochschul-Teamsportliegen relevanten Informationen konsistent verwalten.

Die Datenbank soll konkret:

- Verwaltung von Teams, Spielern, Schiedsrichtern und Spielorten in einer konsistenten Struktur
- Minimierung von Redundanzen und Aktualisierungsanomalien durch Normalisierung (3NF)
- Unterstützung typischer Teamsport-Turnierlogik
    - Abbildung von Ligen und Saisons
    - Verwaltung von Spielen mit Datum, Uhrzeit, Ort, beteiligte Teams und Ergebnis
- Transparente Ergebnisverwaltung und Tabellen
    - Speicherung von Spielergebnissen
    - Berechnung von Tabellen (Spiele, Siege, Niederlagen, …)
- Auswertung für Organisation und Berichte
    - SQL-Abfragen für organisatorische Aufgaben (Hallenbelegung und Sportliche statistiken)

Abgrenzung des Projekts

Um den umfang der Projektarbeit in einem sinnvollen Rahmen zu halten, werden bewusst nur die Teamsportarten des Hochschulsports modelliert.

Betrachtet werden:

- Teamsportarten mit Teams-vs-Team-Begegnungen
- Liegen und Turniere mit
    - Saisons
    - Spieltagen
    - Ggf. Playoff-Struktur
- Verwaltung von
    - Teams
    - Spielern
    - Schiedsrichtern
    - Spiel Lokationen
    - Ergebnisse

Nicht beachtet werden:

- Finanzverwaltung
- Tickets und Zuschauerverwaltung
- Komplexe Datenanalytik
- Frontend

Anforderungsanalyse

Im Sinne der Anforderungsanalyse werden die benötigten Entitäten, ihre Eigenschaften und zusammenhänge identifiziert, die später in einer Entity-Relationship Model überführt werden.

Stammdaten

- **Sportart**
    - Attribute: Name (Fußball, Basketball), Kategorie (Hallen-/Feldsport), ggf. Standard-Teamgröße.
    - Motivation: erlaubt Erweiterungen auf weitere Teamsportarten ohne Schemaänderung.
- **Liga / Turnier**
    - Attribute: Name (z. B. „Fußball-Hochschulliga Herren“), Sportart, Saison, Modus (Liga, Liga+Playoffs).
- **Saison**

- Attribute: Bezeichnung (z. B. „SoSe 2025“), Start- und Enddatum.

- **Team**
  - Attribute: Team-Name, zugehörige Liga, Kategorie (Herren/Damen/Mixed),
- **Spieler**
  - Attribute: Name, Matrikelnummer, ggf. Studiengang, zugeordnetes Team, Rückennummer.
- **Schiedsrichter**
  - Attribute: Name
- **Spielort**
  - Attribute: Name der Halle/des Platzes, Adresse, Kapazität.

Wettbewerbsstruktur

- **Spieltag**
  - Ordnet Spiele zu einem Datum
- **Spiel**
  - Attribute: Datum, Uhrzeit, Spielort, Heim-Team, Auswärts-Team, zugehörige Liga/Saison, Runde
- **Playoff-Runde (optional)**
  - Modellierung von Viertel-, Halb- und Finalspielen mit Verknüpfung zur Liga/Saison.

Ergebnis- und Statistikdaten

1. **Spielergebnis**
   - Attribute: erzielte Tore/Punkte je Team, ggf. Verlängerung/Overtime
   - Beziehung zu Spiel und Teams.
2. **Spielerstatistik (optional)**
   - Attribute: erzielte Punkte/Tore je Spieler, Spielzeit

Anforderungen an Datenqualität und Integrität

- **Referentielle Integrität**
    - Jedes Spiel verweist auf **genau zwei gültige Teams**, eine Liga/Saison, einen Spielort und – falls vorhanden – zugewiesene Schiedsrichter.
    - Spieler sind **immer einem Team zugeordnet**, Teamzuordnung darf nicht auf nicht existierende Teams verweisen.
- **Normalisierung**
    - Das Schema wird mindestens bis zur **Dritten Normalform (3NF)** entworfen

Leitfragen für SQL-Abfragen

1. **Ligatabelle und Qualifikation für Playoffs**
    - Wie sieht die Abschlusstabelle einer ausgewählten Liga aus?
2. **Top-Scorer er jeweiligen Saison und Sportart?**
    - Welche Spieler sind in einer gewählten Saison und Sportart die Top-Scorer.
3. **Wie oft gewinnt die Heim-Mannschaft (in Prozent)?**

Entity-Relationship Modell Diagramm

Siehe ERM.pdf

Beziehungen und Kardinalitäten

SPORT (1) → COMPETITION (0,N)
Ein Sport kann mehrere Wettbewerbe haben; ein Wettbewerb gehört genau zu einer Sportart.

SEASON (1) → COMPETITION (0,N)
Eine Saison umfasst beliebig viele Wettbewerbe; jeder Wettbewerb gehört zu einer Saison.

COMPETITION (1) → TEAM (0,N)
Wettbewerb enthält mehrere Teams

TEAM (1) → COMPETITION (0,N)
Ein Team kann an mehreren Wettbewerben teilnehmen; pro Teilnahme gibt es Attribute wie *registration_date* und *group_name*.
→ klassische M:N-Beziehung mit Zwischentabelle.

COMPETITION (1) → GAME (0,N)
Ein Wettbewerb besteht aus mehreren Spielen; ein Spiel gehört genau zu einem Wettbewerb.

VENUE (1) → GAME (0,N)
Ein Austragungsort ist für viele Spiele nutzbar; jedes Spiel findet an genau einem Venue statt.

TEAM (1) → GAME (0,N)
Jedes Spiel hat exakt einen Heim- und einen Auswärtsverein.

TEAM (1) → PLAYER (0,N)
Spieler können mehreren Teams über die Zeit angehören (durch *from_date*, *to_date*).
→ quasi historisierte 1:N-Beziehung (Zeitintervall).

GAME (1) → GAME_EVENT (0,N)
Ein Spiel hat beliebig viele Events (Tor, Foul, Karte, etc.). Ein Event gehört genau zu einem Spiel.

TEAM (1) → GAME_EVENT (0,N)
Ein Event kann einem Team zugeordnet sein, z. B. Tor für Team A.

PLAYER (1) → GAME_EVENT (0,N)
Ein Event hat einen ausführenden Spieler (Schütze, Foulender etc.).

Normalisierung

Alle Tabellen erfüllen die Anforderungen der 3. Normalform (3NF), also enthalten die Tabelle keine Transitiven Abhängigkeiten. Dadurch stellen wir sicher, dass jedes nichtschlüsselabhängiges Attribut nur vom Primärschlüssel abhängt, somit wird Inkonsistenz verhindert. Damit haben wir ein verständliches Modell ohne weitere Komplexität, der höheren Normalformen.

Begründung Datenbankentwurfs

1. Separate Tabelle für Sport, Saison
    - Gute Skalierbarkeit für zukünftige Wettbewerbe in anderen Sportarten
2. M:N Beziehung TEAM – COMPETITION
    - Teams können in mehreren Wettbewerben teilnehmen
    - Ermöglicht Zusatzattribute (Registrierungsdatum, Gruppeneinteilung -> Praktisch für Turnierorganisation)
3. M:N Beziehung TEAM - PLAYER mit Attribute from_date, to_date
    - Historische Daten von Spieler in Teams werden festgehalten
    - Zeitintervalle vermeiden Redundanzen und ermöglischen rückwirkende Auswertung
4. Beziehungen zwischen Klassen
    - Verhindern Redundanzen, da keine Attribute mehrfach gepeichert sind
    - Beispiel: Team-Name steht nur in TEAM, nicht im GAME -> da steht nur die ID vom Team

Annahmen

1. In der Datenbank werden nur Team-Sportarten und Turniere gespeichert
2. Scorewerte werden in der GAME Tabelle redundant gespeichert, um Datenbank übersichtlich zu halten -> bewusste Denormalisierung
3. Es gibt genau zwei Teams pro Spiel

Data Dictionary

Tabelle „game“

Attributsname	**Daten-typ**	**Länge**	**Format**	**Default-wert**	**NULL**	**Schlüssel**	**Beschreibung**
game_id	Int	11		-	N	Primär	Identifikationsnummer fürs Spiel
competition_id	Int	11		-	N	Fremd	Identifikationsnummer des Tuniers, um das Spiel einem Turnier zuzuordnen
venue_id	Int	11		-	N	Fremd	Ort an dem das Spiel stattfindet
gameday_number	Int	11		NULL	Y	Nein	Spieltag nummer des Tuniers
stage	Varchar	15	Wort	NULL	Y	Nein	Phase von Turnier
scheduled_datetime	Date time	19	YYYY-MM-DD HH:MM:SS	-	N	Nein	Geplanntes Stattfinden des Spiels (Datum + Uhrzeit)
away_team_id	Int	11		-	N	Fremd	Identifikationsnummer vom Auswärtsteam
home_team_id	Int	11		-	N	Fremd	Identifikationsnummer vom Heimteam
home_score	Int	11		NULL	Y	Nein	Erzielte Tore/Punkte des Heimteams
away_score	Int	11		NULL	Y	Nein	Erzielte Tore/Punkte des Auswärtsteams
status	Varchar	15	Wort	“Planned”	N	Nein	Status des Spiels (planned, canceled, ongoing, finished)

Rollen und Berechtigungen

Rolle	Berechtigung	Tabellen	Begründung
Vereinsmanager	Insert, update, select	Player, Player_Team	Muss Spieler hinzufügen können, wenn ein neuer Spieler ins Team kommt
	Select	Game, Competition	Einsehen von kommenden Spielen und Turnieren
Turnierveranstalter	Insert, update, select	Competition, Game, Comp_Team	Muss Turniere und Spiele einrichten können und beteiligte Teams hinzufügen
	Select	Team, Venue	Teams und Venue einsehen zum planen
Spielberichtserstatter	Insert, Update, Select	Game_Event	Muss Spielereignisse hinzufügenkönnen
	Update	Game(home_score, away_score)	Updatet Spielstand
	Select	Player, Game	Einsehen von Spieler und Game, zum überprüfen der Daten
Viewer	Select	All	Einsehen der Datenbank
Datenpfleger	Select, Update, Insert, Delete	Venue, Team	Vollständige Pflege von Stammdaten
	Select, Update, Insert	Season, Sport	
	Select, CREATE VIEW, SHOW VIEW	*	Erstellung von Views
Admin	All	All	

SQL-Abfragen

Aufgrund des Verwendungszwecks der SQL-Abfragen werden die meisten in Form einer View im Projekt festgehalten.

Abschlusstabelle Fußball:

Erstellungs-Datei: „view_AbschlusstabelleFussball.sql"

View: „vw_hochschulcup_2025_abschluss_Fussball"

Diese View sammelt alle Spiele des Wettbewerbs und berechnet für jedes Team die Anzahl der Siege, die erzielten Tore und die kassierten Tore. Anschließend wird daraus die Platzierung ermittelt (Anzahlt Tore > erzielte Tore > kassierte Tore).

Funktionsweise:

1. Zuerst läd ein CTE (comp) die passende competition_id über den Namen des Turniers.
2. Ein zweiter CTE (game_results) erzeugt pro Spiel zwei Zeilen:
   - Eine für das Heimteam
   - Eine für das Auswärtsteam
3. Dann werden im CTE (team_stats) pro Mannschaft alle Spiele summiert mit:
   - Spiele
   - Siege
   - Tore und Gegentore
4. Über eine ROW_NUMBER()-Sortierung entsteht die Platzierung
5. Tabelle wird ausgegeben

Abschlusstabelle Basketball:

Erstellungs-Datei: „view_AbschlusstabelleBasketball.sql"

View: „vw_hochschulcup_2025_abschluss_Basketball"

Diese View sammelt die aktuelle Abschlusstabelle für den Hochschulcup Basketball 2025. Es werde alle bereits beendeten Spiele gewertet und gibt die momentanen Platzierungen aus. Im Unterschied zur Fußball-View werden die Punktestande nicht aus der game-Table sondern aus den game_events berechnet.

Funktionsweise:

1. Zuerst läd ein CTE (comp) die passende competition_id über den Namen des Turniers.

2. Ein zweiter CTE (game_points) zählt die Punkte pro Team und Spiel aus den Events zusammen
3. Im CTE game_results wird jedes Spiel in zwei Zeilen dargestellt
    - Aus Sicht Heimteam
    - Aus Sicht Auswärtsteam
    - Für jede Seite wird berechnet:
        - Punkte für
        - Punkte gegen
        - Sieg (Punke > gegnerische Punkte)
4. CTE (team_stats) fasst alles pro Mannschaft zusammen
5. Ranking durch ROW_NUMBER() nach Siegen > erzielte Punkte > kassierte Punkte
6. Ausgabe als Tabelle

Top Scorer Fußball:

Erstellungs-Datei: „view_TopScorer_Fussball.sql"

View: „vw_top_scorer_2025_fussball"

Diese View erstellt eine Top-Scorer-Liste für den Hochschulcup Fußball 2025. Sie zeigt alle Spieler die mindestens ein Tor erzielt haben und sortiert nach der Anzahl der Treffer.

Funktionsweise:

1. Eine CTE (comp) wählt die passende competition_id über den Namen.
2. Zählt die Toren in diesem Turnier (goal_events). Die game_event-Einträge werden mit den passenden Spielen verknüpft, nach Competition gefiltert und pro Spieler gruppiert.
3. CTE (tournament_players) ermittelt, welche Spieler am Turnier teilnehmen
4. Zusammenführen der Turnierspieler und Torstatistikek
    - Jedem Turnierspieler wird die Zahl seiner Tore zugeordnet
    - Spieler ohne Tor Wert = 0
5. Sortieren der Ausgabe

Top Scorer Basketball:

Erstellungs-Datei: „view_TopScorer_Basketball.sql"

View: „ vw_top_scorer_2025_basketball"

Diese View erstellt eine Liste der Top-Scorer des Hochschulcup Basketball 2025. Sie berechnet für jeden Spieler, wie viele Punkte er im Turnier erzielt hat (nach korrekter Wurfart).

Funktionsweise:

1. Eine CTE (comp) wählt die passende competition_id über den Namen.
2. Punkte Pro Spieler berechnen (points_per_player). Hier werden die Tabellen game_event, game, competition, player und team verknüpft um daraus die Punkte- und Wurfzählung zu generieren.
3. CTE (ranked) sortiert diese nach:
    - total_points > three_pt_made > two_pt_made > free_throws_made > alphabetisch nach Namen
4. ROW_NUMBERS() sorgt für richtige Platzierung

Heim-Siegesrate Fussball:

Erstellungs-Datei: „view_home_win_rateFussball.sql"

View: „ vw_hochschulcup_2025_home_win_rateFussball"

Diese View untersucht, wie erfolgreich die Teams bei ihren Heimspielen im Hochschulcup Fussball 2025 sind.

Funktionsweise:

1. Eine CTE (comp) wählt die passende competition_id über den Namen.
2. Heimspiele pro Team zusammenzählen (home_games). Hier werden alle Spiele einer Competition betrachtet, bei denen Team Heimteam ist.
3. Die Ergebnisse werden mit der team-Tabelle verknüpft und Siegquote berechnet.
4. Tabelle wird sortiert nach home_win_percentage

Heim-Siegesrate Basketball:

Erstellungs-Datei: „view_home_win_rateBasketball.sql"

View: „vw_hochschulcup_basketball_2025_home_win_rateBaskteball"

Diese View untersucht, wie erfolgreich die Teams bei ihren Heimspielen im Hochschulcup Basketball 2025 sind.

Funktionsweise:

1. Eine CTE (comp) wählt die passende competition_id über den Namen.
2. Heimspiele pro Team zusammenzählen (home_games). Hier werden alle Spiele einer Competition betrachtet, bei denen Team Heimteam ist.
3. Die Ergebnisse werden mit der team-Tabelle verknüpft und Siegquote berechnet.
4. Tabelle wird sortiert nach home_win_percentage

Alle Spieler eines Teams anzeigen:

Mit folgender SQL-Anfrage kann man alle Spieler eines gewählten Teams mit Vor- und Nachname anzeigen lassen.

```
select
    player.first_name,
    player.last_name
from
    team_player
        join team on team.team_id = team_player.team_id
        join player on player.player_id = team_player.player_id
where team.name like "Informatik Kickers%"; -- Hier Team wählen
```