Datenbanksysteme Projekt

Ferdous Nasri
Yannek Nowatzky
Leli Schiestl

August 8, 2016

1 Projektiteration

1.1 ERM anhand der Projektbeschreibung und Daten erstellen

Zuerst haben wir anhand der Anforderungen ein ERM erstellt. Beim späteren Abgleichen mit der Datenbank ist uns aufgefallen, dass wir nicht alle Informationen aus der Datenbank herauslesen können, die wir laut Beschreibung benötigen.

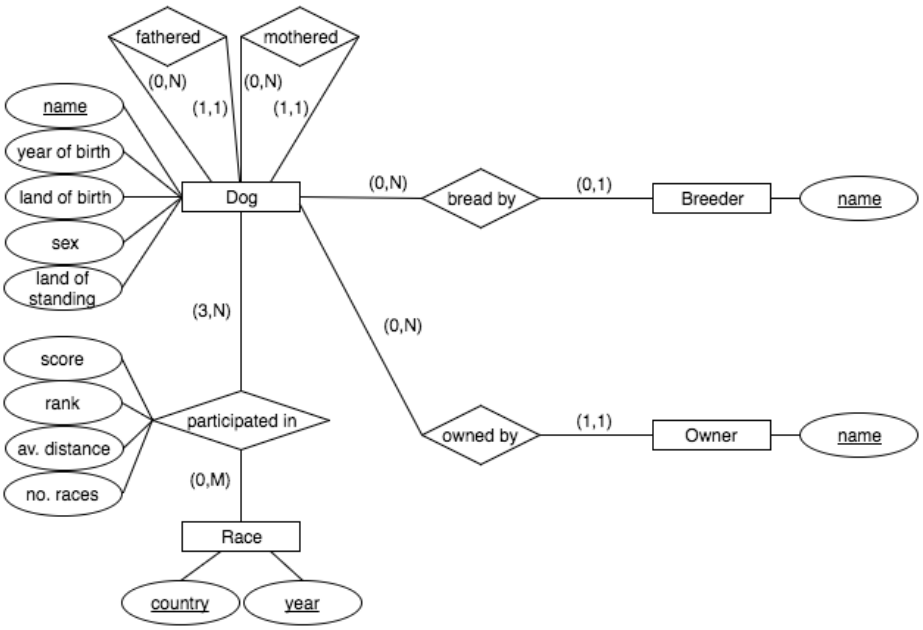

Figure 1: ERM der ersten Projektiteration der Datenbank greyhounddog

1.2 Relationales Modell ableiten

Aus dem ERM (Figure 1) leiten wir nun das Relationale Modell ab.

Dog(name, year_of_birth, land_of_birth, land_of_standing, sex, mother, father, owner, breeder)

Race(country, year)

participated_in(dog_name, race_country, race_year, score, average distance, rank, number_of_races)

Die unterstrichenen Einträge stellen die `primary keys` dar und die unterwellten Einträge sind `foreign keys`.

1.3 CREATE-Statements zur Erzeugung der Tabellen schreiben

Aus diesem rationalen Modell haben wir dann die CREATE Statements abgeleitet.

```
CREATE TABLE Dog(
   name varchar(255) NOT NULL,
   year_of_birth int,
   land_of_birth nchar(2),
   land_of_standing nchar(2),
   sex nchar(1),
   father varchar(255),
   mother varchar(255),
   owner varchar(255),
   breeder varchar(255),
   CONSTRAINT pk_dog PRIMARY KEY (name),
   CONSTRAINT fk_mother FOREIGN KEY (mother) REFERENCE Dog(name),
   CONSTRAINT fk_father FOREIGN KEY (father) REFERENCE Dog(name)
);

CREATE TABLE Race(
   country varchar(255),
   year int,
   CONSTRAINT pk_race PRIMARY KEY (country, year),
);

CREATE TABLE participated_in(
   dog_name varchar(255),
   race_country varchar(255),
   race_year int,
   score float,
   average_distance float,
   CONSTRAINT fk_dog_name FOREIGN KEY (dog_name) REFERENCES Dog(name),
   CONSTRAINT fk_race_country FOREIGN KEY (race_country, race_year)
   REFERENCE Race(country, year)
);
```

Bei der Präsentation hat unser Tutor Alexander Korzec uns in der Annahme bestätigt, dass es unmöglich ist aus der Datenbank den Besitzer zu ermitteln und nicht wirklich möglich ist den Züchter zu ermitteln. Daher haben wir für die nächsten Projektiterationen unser Modell angepasst und diese Informationen entfernt.

2 Projektiteration

Für den zweiten Teil der Projektphase haben wir unser ERM angepasst.

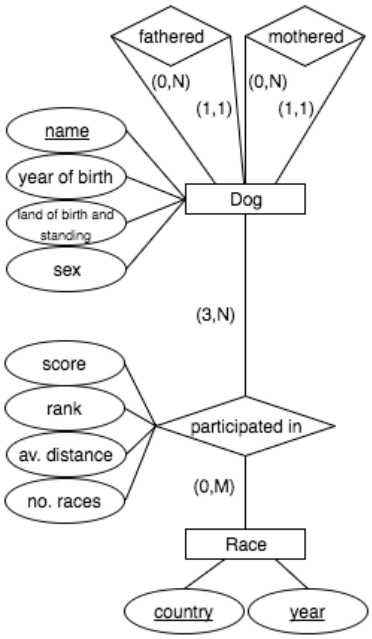

Figure 2: ERM der zweiten Projektiteration den Datenbank greyhounddog

Zum einen haben wir den Züchter und den Besitzer entfernt, da wir diese Daten nicht wirklich erfassen konnten. Zum anderen haben wir die Attribute `land_of_standing` und `land_of_birth` zu einem gemeinsamen Attribut zusammengefasst. Dies liegt daran, dass nicht alle Hunde zwei Einträge haben und wir nicht genau wissen, zu welchem der beiden Attribute die einzelnen Einträge gehören.

Dadurch hat sich natürlich auch unser relationales Modell verändert.

Dog(name, year_of_birth, land_of_birth_and_standing, sex, mother, father)

Race(country, year)

participated_in(dog_name, race_country, race_year, score, average distance, rank, number_of_races)

Beim einfügen der Daten in die Datenbank haben uns auch dagegen entschieden `mother` und `father` als `foreign keys` beizubehalten, da sonst alle Einträge für Eltern, die nicht selbst Rennhunde in der Datenbank sind `NULL` gesetzt werden würden. Dadurch gewinnt man nicht wirklich wichtige Informationen und löscht Informationen, die man hat.

2.1 Import der Daten über SQL

```
CREATE TABLE Dog(
   name varchar(255) NOT NULL,
   year_of_birth int,
   land_of_birth_and_standing nchar(5),
   sex nchar(1),
   father varchar(255),
   mother varchar(255),
   CONSTRAINT pk_dog PRIMARY KEY (name)
);

CREATE TABLE Race(
   country varchar(255),
   year int,
   CONSTRAINT pk_race PRIMARY KEY (country, year)
);

CREATE TABLE participated_in(
   dog_name VARCHAR(255),
   race_country VARCHAR(255),
   race_year INT,
   score FLOAT,
   average_distance INT,
   rank INT,
   number_of_races INT,
   CONSTRAINT fk_dog_name FOREIGN KEY (dog_name) REFERENCES Dog(name),
   CONSTRAINT fk_race FOREIGN KEY (race_country, race_year) REFERENCES
   Race(country, year)
);
```

Außerdem haben wir eine Hilfstabelle für den Import der Daten aus der Datei greyhounddata.csv erstellt.

```
CREATE TABLE import(
   i_country VARCHAR(255),
   i_year int,
   i_rang int,
   i_name VARCHAR(255),
   i_sex CHAR,
   i_vater VARCHAR(255),
   i_mutter VARCHAR(255),
   i_anzahl int,
   i_punkte DOUBLE PRECISION,
   i_durch VARCHAR(30)
);
```

In diese `import` importieren wir die Daten mit folgendem Befehl:

```
COPY import(i_country, i_year, i_rang, i_name, i_sex, i_vater, i_mutter,
i_anzahl, i_punkte, i_durch)
FROM '/Users/Shared/greyhounddata.csv'
WITH DELIMITER ';';
```

2.2 Bereinigung der Daten

Da die Daten nicht alle in dem von uns gewünschten Format vorlagen, haben wir eine zweite Hilfstabelle import2 angelegt.

```
CREATE TABLE import2(
   i2_country VARCHAR(255),
   i2_year INT,
   i2_rang INT,
   i2_name VARCHAR(255),
   i2_year_of_birth VARCHAR(255),
   i2_land_of_birth_and_standing VARCHAR(255),
   i2_sex CHAR,
   i2_vater VARCHAR(255),
   i2_mutter VARCHAR(255),
   i2_anzahl INT,
   i2_punkte FLOAT,
   i2_durch VARCHAR(30)
);
```

Der Lesbarkeit zu Liebe fügen wir hier nicht alle Import Befehle für die Tabelle `import2` ein und verweisen auf die Quellcode-Datein.

Die Spalten für `country`, `year`, `rang, sex, vater, mutter` und `anzahl der rennen` wurden einfach kopiert.
Da der Name in der Ursprungsdatei aus dem Namen, dem Geburtsjahr, dem Geburtsland und dem Rennland bestand, haben wir diese Information mithilfe von `substring` in die jeweiligen Spalten `i2_name`, `i2_year_of_birth`, `i2_land_of_birth_and_standing` von `import2` kopiert.

Als Beispiel führen wir hier die Gewinnung des Namens an:

```
substring(i_name for position('[' in i_name)-2)
```

Die Punkte sollten wir runden, daher haben wir beim kopieren in `import2` den Befehl `round(i_punkte)` angewandt.
Da die durchschnittliche Renndistanz mit dem Zusatz 'm' zu einem String wurde, haben wir auch hier den `substring` gebildet und alles was keine Zahl war, abgeschnitten.

Dann haben wir die Werte in die drei Tabellen `Dog`, `Race` und `participated_in` eingefügt. Dabei haben wir die beiden Strings `year_of_birth` und `average_distance` mit dem Befehl `::INTEGER` in Integer umgewandelt. Um redundante Tabelleneinträge zu vermeiden, haben wir `DISTINCT` eingefügt.

Als Beispiel führen wir hier die Tabelle `Race` an.

```
INSERT INTO Race(country, year)
  SELECT DISTINCT i2_country, i2_year FROM import2;
```

2.3 API (vorerst für Konsolenanwendungen)

API wird unter 3.2 im Rahmen der Web-GUI erläutert. In dieser Iteration wurde noch auf die GUI-Umsetzung verzichtet, die SQL-Code Eingabe erfolgt über Konsole und konnte so direkt in Java behandelt werden.

3 Projektiteration

3.1 Entwurf und Umsetzung einer Web-GUI (HTML/CSS)

Um unsere Resultaten schön darzustellen und die Query benutzerfreundlicher ausführen zu können, haben wir ein Web-GUI gebaut. Da hatten wir viel Designentscheidungen zu treffen. Wir haben uns entschieden ausschliesslich die Sprachen HTML und CSS zu benutzen (Die schon geschriebene Code auf sass, scss und ReactJS worden dann zu einfache CSS übersetzt.). Erstens haben wir unsere Dog-Query Seite erstellt und dabei gemerkt dass wir ein "Willkommen Seite" brauchen wo Users sich entschieden können, ob sie die Daten von einer spezifischer Hund sehen möchten oder ob sie die Vorhersagen von Hunden sehen möchten. Dabei haben wir die Navigationsbar auch hergestellt. Ganz zum Schluss sahen die Seiten einigermassen in Ordnung aus (nach der Behebung von viel kleine Designfehler). Anbei paar Screenschots von unsere Webseite.

3.2 Java API – Schnittstelle zwischen Datenbankzugriff (SQL) und GUI (JSP/HTML)

Um dem Benutzer graphisch anschaulich eine Verbindung zur Datenbank anzubieten, standen wir vor der herausfordernden Aufgabe HTML User-Input in eine SQL-Query zu übersetzen und ihm in den Output der Datenbank wieder in HTML-Format auszugeben. Dazu verwendeten wir JSP, welche ein Schnittstelle zu Java anbot. Mit der useBean-Funktion ließen sich so (Bean) Klassen in Java ansteuern und Variable via Set-Funktion zu übergeben. Wir haben in den Klassen QManager.java und Datamining.java den JSP-Input aus den Textfeldern in Conditions übersetzt. Daraus konnten wir dann eine SQL Anfrage in String-Form basteln. Über die API.java haben wir dann einen Call auf die Datenbank ausgeführt und ein Statement mit SQL-String executed und ein Resultset (enthält Inhalt des SQL-Tables) erhalten. Dieses Resultset haben wir dann in eine Matrix (in Java mit ArrayList¡ ArrayList ¡String¿¿ realisiert). Über die useBean:get-Funktion ließen sich nun Einträge der Matrix anfragen und in HTML zu printen. Mit eine JSP-for-Schleife haben wir ebendiese in einen Table (¡table¿) grafisch ansprechend eingefügt. Neben kleineren Problembewältigugen (falscher User-Input wie zB. String bei Year-Eingabe) und Realisierungen ist dies Kern der User-Datenbank-Interaktion.

3.3 Data-Mining-Aufgabe

Wir sollten eine Anwendung zur Platzierungsprognose erstellen. Hierzu sollten wir zunächst die Leistungskurve eines durchschnittlichen Rüden, bzw. einer durchschnittlichen Hündin eines Zwingers berechnen.
Nachdem wir aber wie schon oben beschrieben nicht wirklich Informationen zu den Zwingern haben und auch die Trennung nach Geschlecht bei der Datenlage (gerade mal 3 Hunden, die mindestens 4 Rennen gelaufen sind, 106 Hunden, die mindestens 3 Rennen gelaufen sind und nur 1006 Hunde der insgesamt 5285 Hunde sind mindestens zwei Rennen gelaufen) in unseren Augen keinen Sinn macht, haben wir über alle Hunde gemittelt eine Leistungskurve erstellt .

Wir haben überlegt welche der Daten sich für diese Prognose am besten eignen

könnte. Punkte pro Lauf in einem Rennen haben wir nicht gewählt, da es einige Hunde gab, die nur einen Lauf in einem Rennen hatten, bei diesem aber 15 Punkte gemacht haben. (Hier haben wir auch ein Problem mit Widersprüchen in der Aufgabenstellung. An einem Lauf sollen maximal 6 Hunde teilnehmen können und der Gewinner bekommt die Anzahl der Teilnehmer + einen Punkt für den Sieg, was maximal 7 Punkte ergeben könnte...).
Auch die Platzierung eignet sich unserer Meinung nach nicht besonders gut für die Vorhersage, da die Platzierung ja nicht nur von der eigenen Leistung sondern auch von der Konkurrenz abhängt. Das ist zwar auch bei den kumulierten Punkten der Fall, sollte sich aber besser verteilen und zwischen Hunden aus verschiedenen Rennländern besser vergleichen lassen.

Zunächst haben wir eine Tabelle `Leistungskurve_alter` erstellt in wir für jeden Hund die Jahreszahlen der Jahre in denen er laufen durfte eingetragen haben (auch hier gibt es ein paar Ungenauigkeiten, da die Aussagen sich auf Monate beziehen und wir nur das Geburtsjahr zur Verfügung haben, aber ein Hund darf laut Aufgabenstellung zwischen 15 und 96 Monaten an Rennen teilnehmen). Gerundet haben wir also Hunde ein Jahr nach ihrem Geburtstag für rennfähig erklärt und am 8. Geburtstag ausscheiden lassen.

Anschließend haben wir die Tabelle `Leistungskurve_Punkte` erstellt in die wir für jeden Hund die Punkte, die er in dem Jahr laut `Leistungskurve_alter` erreicht hat eingetragen und haben diese dann für die Tabelle `Leistungskurve_verlauf` alle gemittelt. Das heißt wir haben alle Punkte die Hunde in ihrem ersten Jahr nach dem Geburtsjahr erreicht haben addiert und durch die Anzahl der Hunde die in ihrem ersten Jahr gelaufen sind dividiert. Diesen Wert haben wir als 100% definiert.

Mit diesen prozentualen Veränderungen haben wir mehrere Zukunftstabellen erstellt, abhängig davon, in welchem Jahr der Hund, den wir betrachten gelaufen ist. Wir haben für jedes Rennergebnis eine Prognose für alle folgenden Jahre erstellt, in denen der Hund noch aktiv ist und diese anschließend in der Tabelle `zukunft_sauber` gemittelt. Auf diese Tabelle greifen wir mit unserer Webanwendung zu, um Vorhersagen über die Rennen unserer zufällig gewählten Hunde zu treffen.

In postgres würde eine Anfrage für ein zufälliges Rennen im Jahr 2016 folgendermaßen aussehen:

```
SELECT zs_name, zs_score, zs_year
  FROM zukunft_sauber
  WHERE zs_year = 2016 AND zs_name IN (
   SELECT name
    FROM dog
    WHERE year_of_birth > 2008
    ORDER BY random() LIMIT 6)
  ORDER BY zs_score DESC;
```

Die Einschränkung, dass die Hunde nach 2008 geboren sein müssen, sorgt dafür, dass wir nur zufällige Hunde wählen, die bei dem ersten imaginären Rennen noch

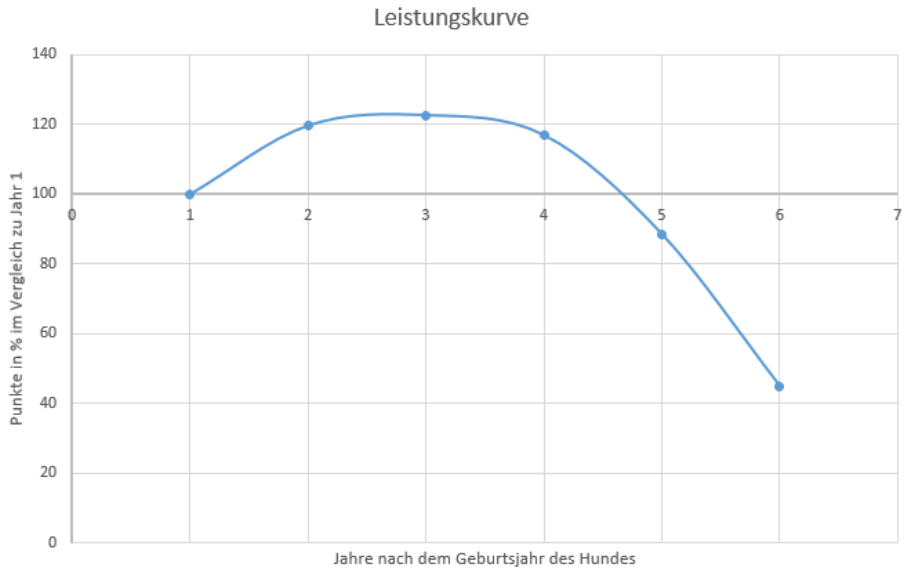

Figure 3: Leistungskurve über die durschnittliche Entwicklung der Hunde

mitlaufen können und nicht vor Beginn der Prognose schon ausgeschieden sind.

Da es keine Hunde hab, die in ihrem 7. Lebensjahr noch gelaufen wären, haben wir für diesen Zeitraum keine Vorhersagen und bekommen die Ausgabe `NULL` obwohl die Hunde noch renntauglich wären.

3.4 Erweiterung der Web-GUI um Data-Mining-Aufgabe

Einmal die Web-GUI für generische SQL-Queries geschrieben, haben wir einfach eine neue JSP Datei erstellt, die speziell auf die Data Mining anfrage gemünzt wurd. Hier nun Eingabefelder für die 4 Hunde und Auswahlmöglichkeit des prognostizierten Jahres. Der Zugriff auf die Scoring Table für jeden Hund erfogte dann auf gleiche Art und Weise, wie die normalen Queries.