Prüfungsleistung Datenbanken 2

Dokumentation Praktikum

Aufgabe 4: Vereine

Abgabedatum: 07.07.2024

Projektmitglieder:
Josephina Burger
Stephanie Wachs

Inhaltsverzeichnis

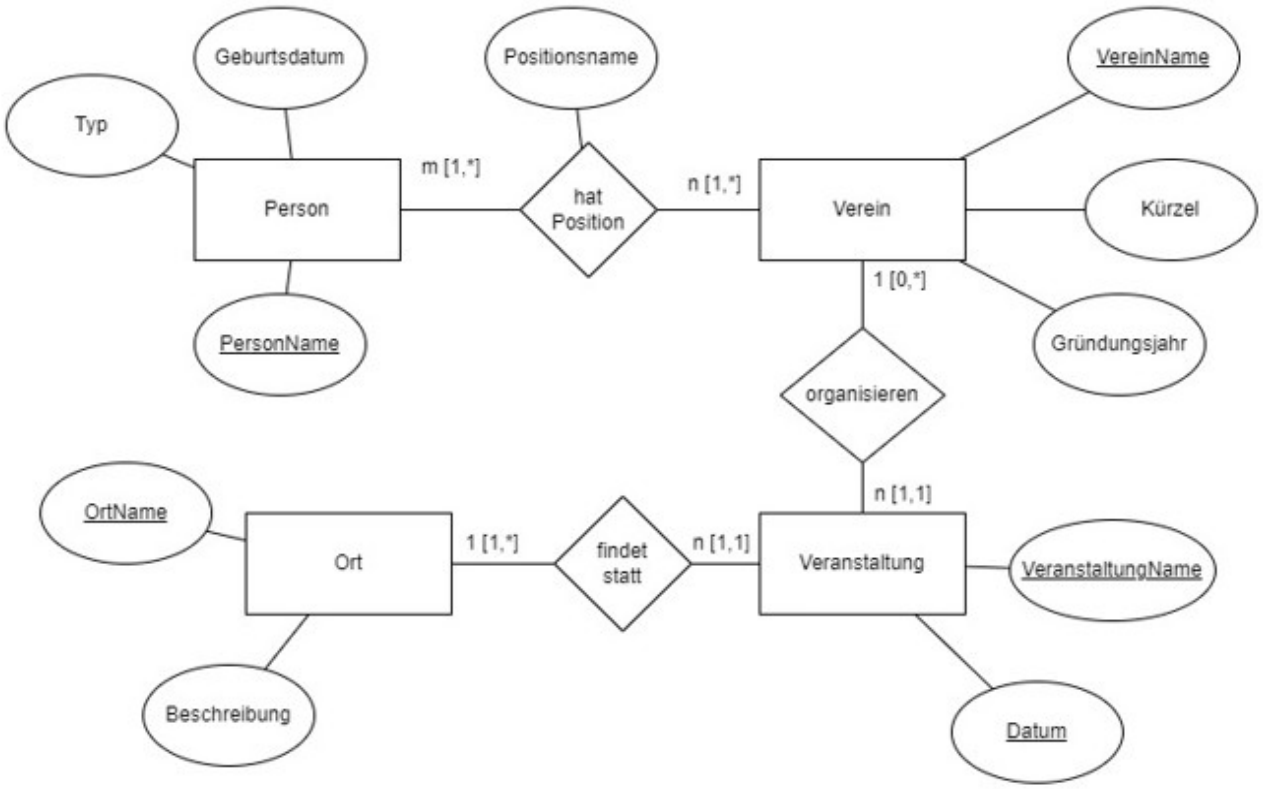
Einleitung

Im folgenden wird die Bearbeitung für die Praktikumsaufgabe Vereine in Datenbanken 2 dargestellt. Die Bearbeitung erfolgte durch Stephanie Wachs und Josephina Burger.

1 Entity-Relationship-Model

Durch folgenden einleitenden Text entwarfen wir ein Entity-Relationship-Model:

Das Vereinsleben einer Hochschule zeichnet sich durch seine ausgeprägte Vielfalt aus. Vereine (mit eindeutigem Namen, Kürzel und Gründungsjahr) besitzen Mitglieder, die jeweils eine bestimmte Position im Verein innehaben. Personen (mit Name, Geburtsdatum und Typ) können dabei auch mehreren Vereinen gleichzeitig angehören, wobei jede Person mindestens in einem Verein aktiv ist. Außerdem organisieren die meisten Vereine zahlreiche Veranstaltungen (mit Name und Datum) im Jahr. Diese finden an jeweils einem festen Ort (mit Name und Beschreibung) statt. Veranstaltungen, die jedes Jahr erneut stattfinden, unterscheiden sich lediglich hinsichtlich des Veranstaltungsdatums. Veranstaltungen können auch zeitgleich in Kombination mit anderen Veranstaltungen an einem Ort stattfinden.



2 Normalisierung

2.1 Vom Entity-Relationship-Model zum relationalen Datenbankschema

Bei der Überführung des ER-Diagramms in das relationale Datenbankschema werden zunächst alle Entitäts- und Beziehungstypen in Relationen abgebildet:

Verein (VereinName, Kürzel, Gründungsjahr)
Person (PersonName, Geburtsdatum, Typ)
Veranstaltung (VeranstaltungName, Datum)

Ort (OrtName, Beschreibung)
hatPosition (VereinName → Verein (VereinName), PersonName → Person(PersonName), Positionsname)
organisieren (VeranstaltungName → Veranstaltung (VeranstaltungName), Datum → Veranstaltung (Datum), VereinName → Verein (VereinName))
findetStatt (VeranstaltungName → Veranstaltung (VeranstaltungName), Datum → Veranstaltung (Datum), OrtName → Ort (OrtName))

Anschließend wird das Relationenmodell unter der Berücksichtigung der Kardinalitäten optimiert, um Redundanzen zu vermeiden. Die Relationen **organisieren** und **findetStatt** lassen sich verschmelzen, da es sich jeweils um 1:N Beziehungen und um zwingende Beziehungen handelt ([1:1]). Dabei wird mit der N-Seite (hier der Relation **Veranstaltung**) verschmolzen. Die Relation **hatPosition** lässt sich nicht verschmelzen, da es sich um eine N:M Beziehung handelt. Es resultiert folgendes relationales Datenbankschema:

Verein (VereinName, Kürzel, Gründungsjahr)
Person (PersonName, Geburtsdatum, Typ)
Veranstaltung (VeranstaltungName, Datum, VereinName → Verein (VereinName), OrtName → Ort (OrtName))
Ort (OrtName, Beschreibung)
hatPosition (VereinName → Verein (VereinName), PersonName → Person (PersonName), Positionsname)

2.2 Erste Normalform

Das relationale Datenbankschema befindet sich bereits in der ersten Normalform, da bereits alle Werte der Attribute atomar sind. Beispielsweise besteht das Attribut *PersonName* nicht aus Vor- und Nachname, sondern nur aus dem Vornamen und ist somit atomar.

2.3 Zweite Normalform

Ein relationales Datenbankschema befindet sich in der zweiten Normalform, wenn es sich in der ersten Normalform befindet und jedes Nicht-Schlüsselattribut vom gesamten Primärschlüssel voll funktional abhängig ist.
Bei der Relation **Veranstaltung** sind die Attribute *VereinName* und *OrtName* nicht voll funktional abhängig vom gesamten Schlüsselattribut, sondern nur vom Teilschlüssel *VeranstaltungName*. Da es Veranstaltungen gibt, die wiederholt stattfinden und somit den gleichen Namen haben und am gleichen Ort stattfinden, aber nicht am selben Datum, entstehen Redundanzen, die sich vermeiden lassen. Die Relation lässt sich wie folgt aufspalten:

Veranstaltung (VeranstaltungName → Veranstaltungsinfo (VeranstaltungName), Datum)
Veranstaltungsinfo (VeranstaltungName, VereinName → Verein (VereinName), OrtName → Ort (OrtName))

Sodass folgendes relationales Datenbankschema entsteht:

Verein (VereinName, Kürzel, Gründungsjahr)
Person (PersonName, Geburtsdatum, Typ)
Veranstaltung (VeranstaltungName → Veranstaltungsinfo (VeranstaltungName), Datum)
Veranstaltungsinfo (VeranstaltungName, VereinName → Verein (VereinName), OrtName → Ort (OrtName))
Ort (OrtName, Beschreibung)
hatPosition (VereinName → Verein (VereinName), PersonName → Person (PersonName), Positionsname)

2.4 Dritte Normalform

Ein relationales Datenbankschema befindet sich in der dritten Normalform, wenn es sich in der zweiten Normalform befindet und kein Nicht-Schlüsselattribut von einem anderen Nicht-Schlüsselattribut abhängig ist.
Bei der Relation **Verein** ist das Attribut *Gründungsjahr* abhängig vom Primärschlüssel, könnte aber auch als abhängig vom *Kürzel* gesehen werden. In dem Fall ließe sich die Relation nochmal unterteilen:

Verein1 (VereinName, Kürzel → Verein (Kürzel))
Verein2 (Kürzel, Gründungsjahr)

Wir haben uns jedoch dazu entschieden, die Relation nicht aufzuspalten, da in der ursprünglichen Form keine redundanten Daten entstehen und somit ein Aufspalten der Relation keinen Sinn macht.
Außerdem haben wir uns dazu entschieden in der Relation **Person** eine *TypID* passend zum *Typ* hinzuzufügen. Da hier auch wieder eine Abhängigkeit zwischen *TypID* und *Typ* entsteht, spalten wir bei der Relation **Person** den *Typ* ab, sodass eine neue Relation **Typ** entsteht. Dies machen wir, um den Typ in der Relation Person als kurze ID anzugeben. Somit können hier Anomalien und Redundanzen vermieden werden.
Die Relation verändert sich damit von:
Person (PersonName, Geburtsdatum, Typ, TypID)
zu:
Person (PersonName, Geburtsdatum, TypID → Typ (TypID))
Typ (TypID, Typ)

In dritter Normalform sieht das relationale Datenbankschema wie folgt aus:
Verein (VereinName, Kürzel, Gründungsjahr)
Person (PersonName, Geburtsdatum, TypID → Typ (TypID))
Typ (TypID, Typ)
Veranstaltung (VeranstaltungName → Veranstaltungsinfo (VeranstaltungName), Datum)
Veranstaltungsinfo (VeranstaltungName, VereinName → Verein (VereinName), OrtName → Ort (OrtName))
Ort (OrtName, Beschreibung)
hatPosition (VereinName → Verein (VereinName), PersonName → Person (PersonName), Positionsname)

2.5 Weitere Änderungen am relationalen Datenbankschema

Zur eindeutigen Kennzeichnung der Tupel in den Relationen haben wir uns dazu entschieden ID's zu verwenden. So kann beim Hinzufügen neuer Datensätze ein versehentliches Überschreiben eines bereits vorhandenen Tupels vermieden werden, wie dies z.B. beim Hinzufügen eines neuen Vereinsmitgliedes mit dem gleichen *PersonNamen* der Fall wäre. Außerdem haben wir uns entschieden den *Positionsnamen* in *Position* umzubenennen.

Person (PersonID, PersonName, Geburtsdatum, TypID → Typ(TypID))
Typ (TypID, TypName)
Verein (VereinID, VereinName, Kürzel, Gründungsjahr)
Veranstaltung (VeranstaltungID, VeranstaltungsinfoID → Veranstaltungsinfo(VeranstaltungsinfoID), Datum)
Veranstaltungsinfo (VeranstaltungsinfoID, VeranstaltungName, VereinID → Verein(VereinID), OrtID → Ort(OrtID))
Ort (OrtID, OrtName, OrtBeschreibung)
hatPosition (VereinID → Verein(VereinID), PersonID → Person(PersonID), Position)

3 Importieren und Erstellen der Tabellen

3.1 Importieren der Rohdaten

Innerhalb der CSV-Datei mussten noch Kommas hinzugefügt werden, denn beim Knie Club und bei den Hochschulfriseuren hat pro Zeile ein Komma gefehlt. Diese Vereine haben bisher noch keine Veranstaltungen durchgeführt. Nur durch diese Änderung konnten die Daten in die SQL Tabelle übertragen werden.

Nachfolgend unser Befehl zum erstellen der Tabelle, in die die CSV-Daten importiert werden sollen:

```
CREATE TABLE importtabelle (
    Vereinsname VARCHAR(50),
```

```
    Vereinskürzel VARCHAR(10),
    Vereinsgründung INT,
    Name VARCHAR(20),
    Geburtsdatum DATE,
    Typ VARCHAR(20),
    Position VARCHAR(20),
    Veranstaltungsname VARCHAR(250) DEFAULT NULL,
    Veranstaltungsdatum DATE DEFAULT NULL,
    Veranstaltungsort VARCHAR(250) DEFAULT NULL,
    Ortsbeschreibung VARCHAR(1000) DEFAULT NULL
);
```

Anschließend haben wir die Daten von der CSV-Datei in die eben erstellte SQL Tabelle importiert:

```
COPY importtabelle (
    Vereinsname,
    Vereinskürzel,
    Vereinsgründung,
    Name,
    Geburtsdatum,
    Typ,
    Position,
    Veranstaltungsname,
    Veranstaltungsdatum,
    Veranstaltungsort,
    Ortsbeschreibung
)
FROM 'C:\\Program Files\\PostgreSQL\\vereineKorrigiert.csv'
DELIMITER ','
CSV HEADER
NULL '';
```

Zur Überprüfung konnte dann folgender Befehl ausgeführt werden:

```
SELECT * FROM importtabelle;
```

3.2 Erstellen der neuen Tabellen

Mit Hilfe des CREATE TABLE Befehls haben wir die neuen Tabellen in SQL erstellt. Wichtig ist auch die Reihenfolge, in der die Tabellen erstellt werden, da sonst die Schlüsselverknüpfungen (Primär- und Fremdschlüssel) nicht richtig zugeordnet werden können.

```
CREATE TABLE Typ (
    TypID SERIAL,
    TypName VARCHAR(20) NOT NULL,
    PRIMARY KEY (TypID)
);

CREATE TABLE Person (
    PersonID SERIAL,
    PersonName VARCHAR(20) NOT NULL,
    Geburtsdatum DATE NOT NULL,
    TypID INT NOT NULL,
    PRIMARY KEY (PersonID),
    FOREIGN KEY (TypID) REFERENCES Typ(TypID)
);
```

```
CREATE TABLE Verein (
	VereinID SERIAL,
	VereinName VARCHAR(50) NOT NULL,
	Kürzel VARCHAR(10) NOT NULL,
	Gründungsjahr INT NOT NULL,
	PRIMARY KEY (VereinID)
);

CREATE TABLE Ort (
	OrtID SERIAL,
	OrtName VARCHAR(20) NOT NULL,
	OrtBeschreibung VARCHAR(1000) NOT NULL,
	PRIMARY KEY (OrtID)
);

CREATE TABLE Veranstaltungsinfo (
	VeranstaltungsinfoID SERIAL,
	VeranstaltungName VARCHAR(250) NOT NULL,
	VereinID INT NOT NULL,
	OrtID INT NOT NULL,
	PRIMARY KEY (VeranstaltungsinfoID),
	FOREIGN KEY (OrtID) REFERENCES Ort(OrtID),
	FOREIGN KEY (VereinID) REFERENCES Verein(VereinID)
);

CREATE TABLE Veranstaltung (
	VeranstaltungID SERIAL,
	VeranstaltungsinfoID INT NOT NULL,
	Datum DATE NOT NULL DEFAULT current_Date,
	PRIMARY KEY (VeranstaltungID),
	FOREIGN KEY (VeranstaltungsinfoID) REFERENCES Veranstaltungsinfo(VeranstaltungsinfoID)
);

CREATE TABLE hatPosition (
	VereinID INT,
	PersonID INT,
	Position VARCHAR(20) NOT NULL,
	PRIMARY KEY (VereinID, PersonID),
	FOREIGN KEY (VereinID) REFERENCES Verein(VereinID),
	FOREIGN KEY (PersonID) REFERENCES Person(PersonID)
);
```

3.3 Einfügen der Daten in die neuen Tabellen

Nachdem die neuen Tabellen erstellt wurden, müssen nun die importierten Daten aus der CSV Datei in die neuen Tabellen übertragen werden. Auch hier ist es wichtig, dass die Daten in der richtigen Reihenfolge importiert werden. Dies funktioniert mit folgenden Befehlen:

```
INSERT INTO Typ (TypName)
SELECT DISTINCT Typ FROM importtabelle;

INSERT INTO Person (PersonName, Geburtsdatum, TypID)
SELECT DISTINCT Name, Geburtsdatum, TypID FROM importtabelle AS i
```

```
JOIN Typ ON Typ.TypName = i.Typ;

INSERT INTO Verein (VereinName, Kürzel, Gründungsjahr)
SELECT DISTINCT vereinsname, vereinskürzel, vereinsgründung FROM importtabelle;

INSERT INTO Ort (OrtName, OrtBeschreibung)
SELECT DISTINCT veranstaltungsort, ortsbeschreibung FROM importtabelle
WHERE veranstaltungsort IS NOT NULL;

INSERT INTO Veranstaltungsinfo (VeranstaltungName, VereinID, OrtID)
SELECT DISTINCT veranstaltungsname, VereinID, OrtID FROM importtabelle AS i
JOIN Verein AS v ON v.VereinName = i.Vereinsname
JOIN Ort AS o ON o.OrtName = i.veranstaltungsort;

INSERT INTO Veranstaltung (VeranstaltungsinfoID, Datum)
SELECT DISTINCT VeranstaltungsinfoID, veranstaltungsdatum FROM importtabelle AS i
JOIN Veranstaltungsinfo AS v ON v.VeranstaltungName = i.veranstaltungsname;

INSERT INTO hatPosition (VereinID, PersonID, Position)
SELECT DISTINCT VereinID, PersonID, position FROM importtabelle AS i
JOIN Verein AS v ON v.VereinName = i.vereinsname
JOIN Person AS p ON p.PersonName = i.name;
```

Zum Test, ob alle Importdaten in den richtigen Zieltabellen gelandet sind, haben wir dann noch folgenden Befehl eingegeben:

```
SELECT v.VereinName, v.Kürzel, v.Gründungsjahr, p.PersonName, p.Geburtsdatum, t.TypName, h.Position,
    vei.VeranstaltungName, ve.Datum, o.OrtName, o.OrtBeschreibung
FROM Verein AS v
    JOIN Veranstaltungsinfo AS vei ON vei.VereinID = v.VereinID
    JOIN Veranstaltung AS ve ON ve.VeranstaltungsinfoID = vei.VeranstaltungsinfoID
    JOIN Ort AS o ON o.OrtID = vei.OrtID
    JOIN hatPosition AS h ON h.VereinID = v.VereinID
    JOIN Person AS p ON p.PersonID = h.PersonID
    JOIN Typ AS t ON t.TypID = p.TypID
EXCEPT
SELECT * FROM importtabelle;
```

4 SQL Anfragen

In den nachfolgenden Abschnitten werden die SQL Anfragen formuliert, die in der Aufgabenstellung gefragt sind:

a) In welchem Jahr wurde der Verein ”Kulinarische Koordinierung” gegründet?

```
SELECT Gründungsjahr
FROM Verein
WHERE VereinName = 'Kulinarische Koordinierung';
```

b) Welcher Verein (Vereinsname, Anzahl) hat die meisten Mitglieder?

```
SELECT VereinName, COUNT(p.PersonID) AS Anzahl
FROM Verein AS v
JOIN hatPosition AS h ON h.VereinID = v.VereinID
```

```
JOIN Person AS p ON p.PersonID = h.PersonID
GROUP BY VereinName
ORDER BY Anzahl DESC
LIMIT 1;
```

c) Welche Person (Name, Anzahl) ist in den meisten Vereinen gleichzeitig aktiv?

```
SELECT PersonName, COUNT(v.VereinID) AS Anzahl
FROM Person AS p
JOIN hatPosition AS h ON h.PersonID = p.PersonID
JOIN Verein AS v ON v.VereinID = h.VereinID
GROUP BY PersonName
ORDER BY Anzahl DESC
LIMIT 1;
```

d) Erstellen Sie eine Statistik bestehend aus Typ und Anzahl, wie viele Studenten, Mitarbeiter und Externe in Vereinen aktiv sind.

```
SELECT VereinName, TypName, COUNT(p.PersonID) AS Anzahl
FROM Typ AS t
JOIN Person AS p ON p.TypID = t.TypID
JOIN hatPosition AS h ON h.PersonID = p.PersonID
JOIN Verein AS v ON v.VereinID = h.VereinID
GROUP BY VereinName, TypName
ORDER BY VereinName, TypName;
```

e) Welcher Verein (Name, Anzahl) organisiert die meisten Veranstaltungen?

```
SELECT VereinName, COUNT(ve.VeranstaltungID) AS Anzahl
FROM Verein AS v
JOIN Veranstaltungsinfo AS vei ON vei.VereinID = v.VereinID
JOIN Veranstaltung AS ve ON ve.VeranstaltungsinfoID = vei.VeranstaltungsinfoID
GROUP BY VereinName
ORDER BY Anzahl DESC
LIMIT 1;
```

f) An welchem Ort (Name, Anzahl) finden die meisten verschiedenen Veranstaltungen statt?

```
SELECT OrtName, COUNT(VeranstaltungsinfoID) AS Anzahl
FROM Ort AS o
JOIN Veranstaltungsinfo AS v ON v.OrtID = o.OrtID
GROUP BY OrtName
ORDER BY Anzahl DESC
LIMIT 1;
```

g) Welcher Verein (Name, Anzahl) hat die meisten studentischen Mitglieder?

```
SELECT VereinName, COUNT(p.PersonID) AS Anzahl
FROM Verein AS v
JOIN hatPosition AS h ON h.VereinID = v.VereinID
JOIN Person AS p ON p.PersonID = h.PersonID
```

```
JOIN Typ AS t ON t.TypID = p.TypID
WHERE t.TypName = 'Student'
GROUP BY VereinName
ORDER BY Anzahl DESC
LIMIT 1;
```

h) Ermitteln Sie den Verein (Name, Verhältnis), der im Verhätnis zu der Anzahl seiner Mitglieder die meisten Veranstaltungen organisiert.

```
WITH Veranstaltungsanzahl AS (
    SELECT VereinName, COUNT (va.VeranstaltungID) AS AnzahlVeranstaltungen
    FROM Verein AS v
    JOIN Veranstaltungsinfo AS vai ON v.VereinID = vai.VereinID
    JOIN Veranstaltung AS va ON va.VeranstaltungsinfoID = vai.VeranstaltungsinfoID
    GROUP BY VereinName
),
Mitgliederanzahl AS (
    SELECT VereinName, COUNT (p.PersonID) AS AnzahlMitglieder
    FROM hatPosition AS h
    JOIN Person AS p ON h.PersonID = p.PersonID
    JOIN Verein AS v ON h.VereinID = v.VereinID
    GROUP BY VereinName
)

SELECT a.VereinName,
    CAST(a.AnzahlVeranstaltungen as decimal)/CAST(m.AnzahlMitglieder as decimal) AS Verhältnis
FROM Veranstaltungsanzahl AS a, Mitgliederanzahl AS m
WHERE a.VereinName = m.VereinName
ORDER BY Verhältnis DESC
LIMIT 1;
```