Datenbanken (Einführung) • Modellierung

1. Interne Befehle von SQLite

Interne Befehle des DBMS SQLite haben nichts gemeinsames mit SQL-Anweisungen. Sie alle fangen mit einem Punkt an. Für Unterricht sind nur wenige Befehle notwendig: *.help* — Hilfe zeigen, *.ta* — Tabellen zeigen, *.da* — angeschlossene Datenbankdatei zeigen, *.read* — SQL-Skript lesen und ausführen, *.q* — DBMS beenden.

2. Projekte — Aufgabestellung

Unten sind einige Übungsaufgaben als Projekte vorgestellt. Das Projekt "Taxi" wird in dieser Anleitung und in Übungen vollständig abgearbeitet. Die anderen Projekte sind für selbständige Arbeit vorgesehen.

Projekt "Taxi".
In einer Stadt gibt es mehrere Taxi-Unternehmen, die sich von einander durch mehrere Charakteristiken unterscheiden: Firmenname (eindeutig), Anzahl der Mitarbeiter, IBAN (eindeutig). Jedes Unternehmen betreibt Autos, die folgende Merkmale haben: KFZ-Zeichen (eindeutig), Marke, Baujahr. Ziel ist es, ERM und relationales Modell zu erstellen.

Projekt "Einkaufzettel".
Sie entdecken in Ihrem Schubladen viele Einkaufszettel, auf denen ein Datum steht, wann Sie eingekauft haben, Bezeichnung und Anzahl der Produkte. Sie können davon ausgehen, dass die Preise, MwSt (7% oder 19%) sowie ein Merkmal, ob es ein Bio-Produkt war, sehr leicht aus Ihrem Geschäft zu holen sind. Einfachheit halber besuchen Sie nur ein Geschäft und nur einmal am Tag. Folgende Abfragen sind für Sie interessant: "An welchen Tagen haben Sie Milch gekauft?", "Wie viel Geld haben Sie für Bio-Produkte ausgegeben?", "Wie viel Geld haben Sie an einem bestimmten Tag ausgegeben?", "Wie viel Geld geben Sie für Bio-Produkte mit MwSt 7% an bestimmtem Tagen aus?". Erstellen Sie nach dem unten beschriebenen Schema ein ERM, ein relationales Modell und die oben erwähnten SQL-Abfragen. Verwenden Sie die Vereinfachungsregeln (Optimierung).

Projekt "Büro".
In Büroräumen (charakterisiert durch eine Zimmernummer) sitzen seit einem Zeitpunkt Mitarbeiter (Personalnummer, Name, Titel, Status) an einem bestimmten Platz. In den Zimmern sind Telefone (besitzen eine eindeutige Telefonnummer) aufgestellt, die als Hausapparat oder Amtsapparat geschaltet sind. Erstellen Sie nach dem unten beschriebenen Schema ein ERM, ein relationales Modell und denken Sie die sinnvollen SQL-Abfragen aus.

Projekt "Projektverwaltung".
In einer Abteilung (besitzt Abteilungsnummer, Name und Ort) arbeiten Mitarbeiter (charakterisiert durch Personalnummer, Name, Geburtsdatum, Privatadresse, Gehalt, Tätigkeitsbezeichnung), die an Projekten (eindeutige Projektnummer, Projektbeginn und -ende) arbeiten. Jeder Mitarbeiter arbeitet mit einer bestimmten Wochenstundenzahl an einem bestimmten Projekt und kann an mehreren Projekten gleichzeitig arbeiten. Jedes Projekt wird von einem anderen Mitarbeiter geleitet. Erstellen Sie nach dem unten beschriebenen Schema ein ERM, ein relationales Modell und denken Sie die sinnvollen SQL-Abfragen aus.

Projekt "Spielverwaltung".
In einer Fußballdatenbank sollen für alle Spieler (charakterisiert durch einen Namen) folgende Fakten gespeichert werden:

- *in welchem Spiel (Spieltag) im Einsatz*
- *wie lange (von,bis) im Einsatz*
- *auf welcher Position im Einsatz*
- *zu welcher Mannschaft gehörig*

Von der Mannschaft sind von Interesse:

- *Name der Mannschaft*
- *Name des Trainers*
- *Alter des Trainers*

3. Unvollständiges ERM

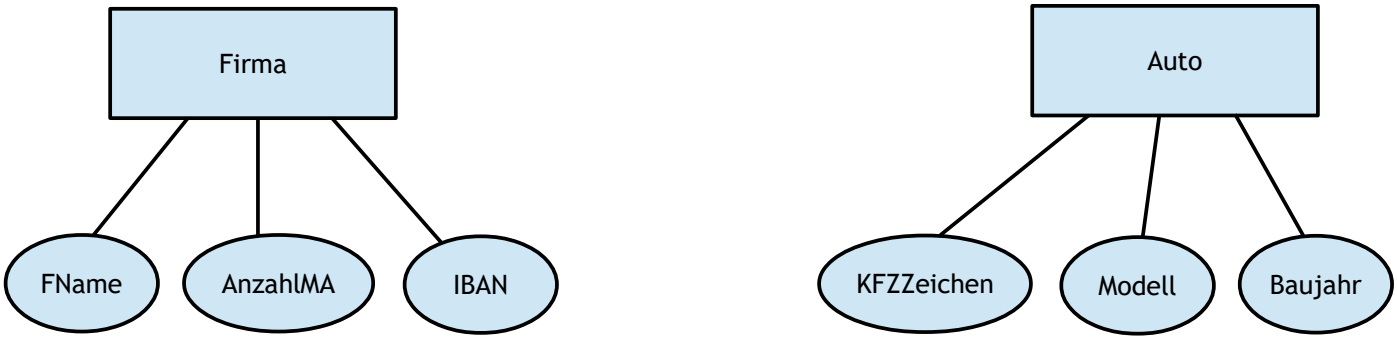
In diesem Kurs werden nur Modelle in Peter-Chen-Notation betrachtet.

Entitätstyp: Firma.

Attribute:	FName, AnzahlMA, IBAN.
Domäne für FName:	{ "Luxi-Taxi", "Maxi-Taxi", "Mini-Taxi", "Rabbit" }
Domäne für AnzahlMA:	{ 42, 53, 116, 1, 64 }
Domäne für IBAN:	{ "DE1234", "DE2345", "DE3456", "FR1234" }

Entitätstyp: Auto.

Attribute:	KFZZeichen, Modell, Baujahr.
Domäne für KFZZeichen:	{ "BAZ-1723", "BAQ-1835", "BAZ-3134", "BCD-9235", "BIM-1245", "BOM-1234" }
Domäne für Modell:	{ "VW", "Mercedes", "BMW", "Skoda", "Maserati", "Chrysler"}
Domäne für Baujahr:	{ 1988, 1986, 2000, 2010, 2015 }



ERM ist nicht vollständig — fehlt Verbindung zwischen Entitätstypen, es ist nicht klar, in welcher Beziehung stehen die zu einander. Merken Sie, wie die Entitätstypen und deren Attribute aussehen müssen. In ERM werden die Werte von Attributen nicht dargestellt. Man setzt hier keine Pfeile ein.

4. Unvollständiges relationales Modell

Es gibt mehrere Schreibweisen für ein relationales Modell. Hier werden Tabellen betrachtet.

Man kann schon jetzt diese Entitätstypen aus dem oberen ERM in die Tabellen umsetzen:

```
/*
    Taxi            taxicr.sql
*/

DROP TABLE Firma;  /* Tabellen werden gelöscht, wenn sie existieren, */
DROP TABLE Auto;   /* sonst kommt eine error-Meldung, die nicht von Bedeutung ist */

CREATE TABLE Firma
(
    FName    VARCHAR2(10), /* Eigentlich TEXT */
    AnzahlMA INTEGER,
    IBAN     TEXT  /* SQLite3 akzeptiert viele ähnliche Datentypen */
);

CREATE TABLE Auto
(
    KFZZeichen VARCHAR2(10),
    Modell     VARCHAR2(8),
    Baujahr    INTEGER
);
```

Diese Anweisungen sollen als Textdatei (z.B. mit dem Namen *taxicr.sql*) gespeichert werden, sie kann dann bei Notwendigkeit einfach und schnell aufgerufen werden:

```
SQL ==> .read taxicr.sql
```

Daten müssen in die Tabellen hinzugefügt werden:

```
/*
    Taxi           taxidata.sql
*/

DELETE FROM Firma;
DELETE FROM Auto;

INSERT INTO Firma VALUES ('Maxi-Taxi', 350, 'DE1234');
INSERT INTO Firma VALUES ('Luxi-Taxi',  90, 'DE2345');
INSERT INTO Firma VALUES ('Fixi-Taxi', 120, 'FR1234');

INSERT INTO Auto VALUES ('BAZ-1718', 'VW',       2011);
INSERT INTO Auto VALUES ('BMP-1718', 'Skoda',    2015);
INSERT INTO Auto VALUES ('BKA-4253', 'Mercedes', 2000);
INSERT INTO Auto VALUES ('BAZ-9876', 'BMW',      2011);
INSERT INTO Auto VALUES ('BAZ-6789', 'VW',       2011);
```

Diese Anweisungen sollen als Textdatei (z.B. mit dem Namen *taxidata.sql*) gespeichert werden, sie kann dann bei Notwendigkeit einfach und schnell aufgerufen werden:

```
SQL ==> .read taxidata.sql
```

Man kann jetzt nur die Tabellen einzeln abfragen:

```
SELECT * FROM Firma;
SELECT * FROM Auto;
SELECT FName, IBAN FROM Firma;
SELECT Modell, KFZZeichen FROM Auto;
```

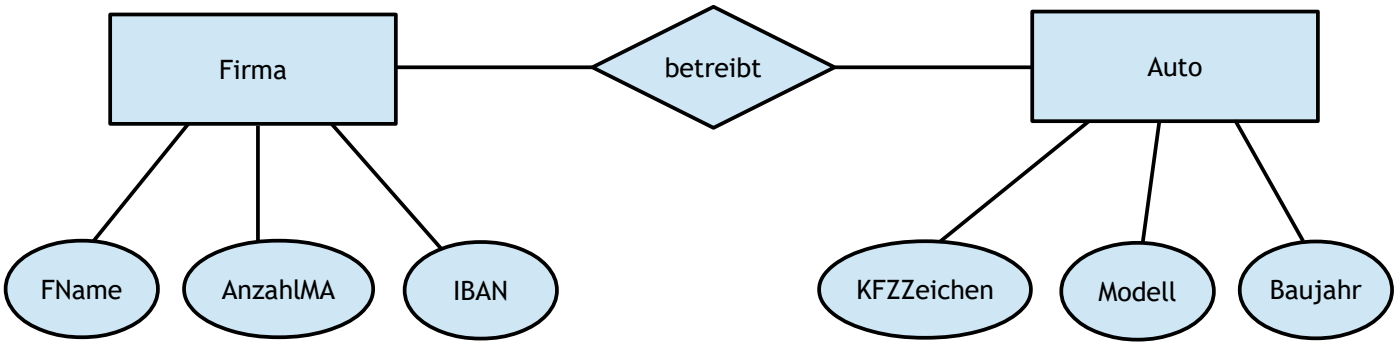
Ein Versuch, die beiden Tabellen abzufragen, bringt Unsinn:

```
SELECT FName, IBAN, Modell, KFZZeichen FROM Firma, Auto;
```

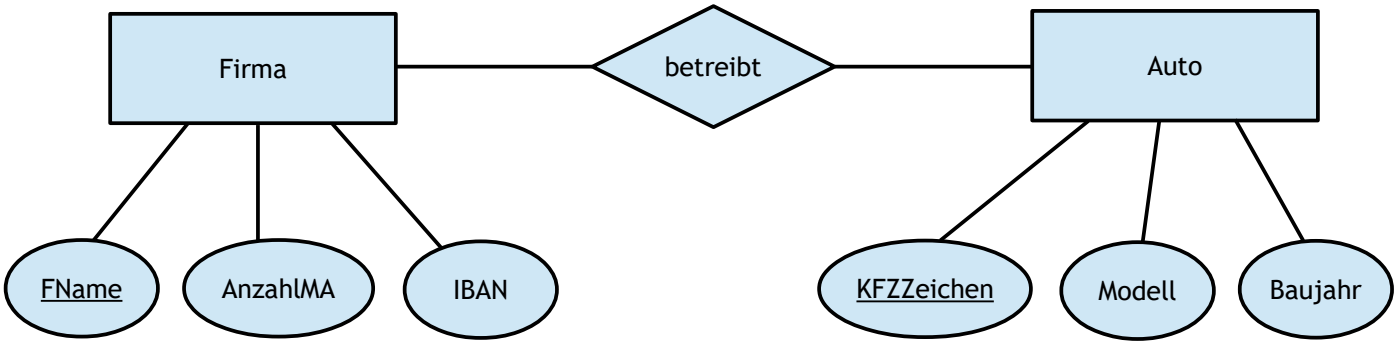
Das ist kartesisches Produkt — hier ist jede Zeile aus einer Tabelle mit jeder Zeile aus der anderen Tabelle verknüpft.

5. Vervollständigung des ERM

Man muss das obere ERM auf jeden Fall vervollständigen, nämlich eine Beziehung hinzufügen. Achten Sie, wie ein Beziehungstyp aussieht. In einem relationalen Modell muss man einen Beziehungstyp ebenfalls in eine Tabelle umsetzen. Aber zuvor muss man die primären Schlüssel jeweils in jedem Entitätstyp definieren.



Die primären Schlüssel müssen unterstrichen werden:



6. Vervollständigung des relationalen Modells

Die Tabelle für eine Beziehung bildet man aus primären Schlüsseln der beiden Entitätstypen:

```
/*
    Taxi           taxicr.sql
*/

DROP TABLE betreibt;

CREATE TABLE betreibt
(
    FName      VARCHAR2(10), /* Spalten können dieselben Namen haben */
    KFZZeichen VARCHAR2(10)  /* wie in ursprünglichen Tabellen       */
);
```

Die Daten muss man hier hinzufügen, aber dabei berücksichtigen, dass *FName* und *KFZZeichen* übereinstimmen:

```
/*
    Taxi           taxidata.sql
*/

DELETE FROM betreibt;

INSERT INTO betreibt VALUES ('Maxi-Taxi', 'BAZ-1718');
INSERT INTO betreibt VALUES ('Maxi-Taxi', 'BMP-1718');
INSERT INTO betreibt VALUES ('Luxi-Taxi', 'BKA-4253');
INSERT INTO betreibt VALUES ('Luxi-Taxi', 'BAZ-9876');
INSERT INTO betreibt VALUES ('Fixi-Taxi', 'BAZ-6789');
```

Die beiden letzten Codes sollen in die oben erstellten Skripte *taxicr.sql* und *taxidata.sql* hinzugefügt werden.

Jetzt kann man die beiden Tabellen mit Entitäten abfragen, aber nicht vergessen, auch die Beziehungstabelle auf jeden Fall zu verwenden, weil nur sie die Entitätstypen verbindet:

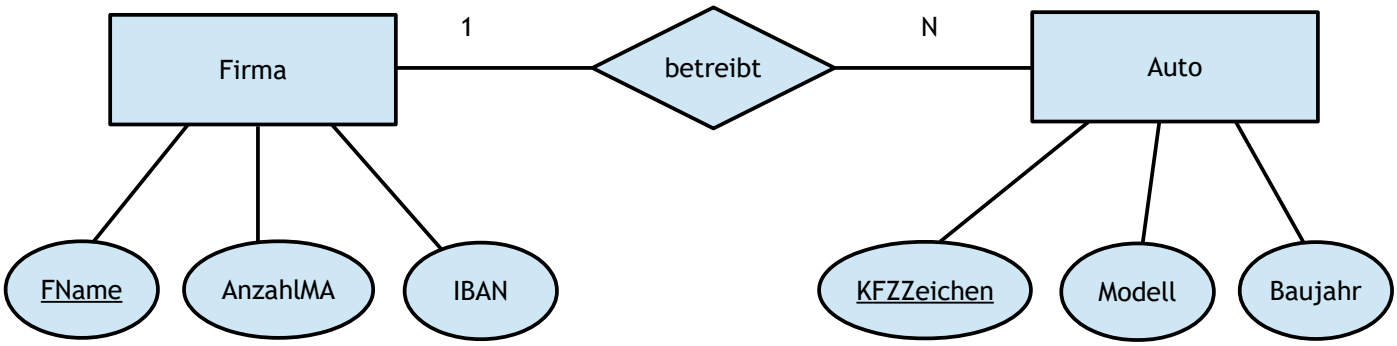
```
SELECT f.FName, f.IBAN, a.Modell, a.KFZZeichen       /* Aliase sind obligatorisch */
FROM Firma f, Auto a, betreibt b             /* Reihenfolge der Tabellen ist egal */
WHERE f.FName=b.FName AND b.KFZZeichen=a.KFZZeichen; /* Aliase sind obligatorisch */
   /*  access-Prädikat          access-Prädikat  */
```

Diese Abfrage hat schon Sinn — sie zeigt alle Unternehmen und deren Autos. Man kann solche Abfragen auch weiter filtern, z.B. nur ausgewählte Unternehmen, oder Unternehmen mit ausgewählten IBAN, oder nur bestimmte Modelle zeigen. Die Bedingung in *WHERE* ist aber wichtig und darf nicht geändert werden, weil sie zwei Tabellen verknüpft. Dazu können aber weitere Bedingungen kommen, die Abfrage spezifizieren. Im Allgemeinen gilt folgende empirische Regel: Stehen in der Klausel *FROM* mehrere (N) Tabellen, dann müssen N-1 Vergleichsoperatoren = vorhanden sein. Gibt es in den Tabellen primäre Schlüssel, die aus mehreren Feldern bestehen, dann erhöht sich entsprechend die Anzahl der Operatoren.

```
SELECT f.FName, f.IBAN, a.Modell, a.KFZZeichen
FROM Firma f, Auto a, betreibt b
WHERE f.FName=b.FName AND b.KFZZeichen=a.KFZZeichen AND /* Dieselben Infos wie oben */
      f.FName='Luxi-Taxi';                        /* aber nur über Firma Luxi-Taxi */
   /*  filter-Prädikat  */
```

7. Vereinfachung des relationalen Modells

Wie es aus Theorie bekannt ist, kann man auf die Beziehungstabellen verzichten, wenn eine 1:N-Funktionalität vorliegt. Genau das gilt in diesem Fall: man nehme eine Firma — sie betreibt mehrere Autos, man nehme ein Auto — es gehört exakt zu einer Firma.



Auf die Beziehungstabelle kann man verzichten, aber der primäre Schlüssel der Entitätstabelle an der 1-Seite muss in die Entitätstabelle an der N-Seite als Fremdschlüssel hinzugefügt werden.

Dann müssen die SQL-Dateien *taxicr.sql* und *taxidata.sql* geändert werden:

```
/*
    Taxi            taxicr.sql
*/

DROP TABLE Firma;  /* Tabellen werden gelöscht, wenn sie existieren, */
DROP TABLE Auto;   /* sonst kommt eine error-Meldung, die nicht von Bedeutung ist */

CREATE TABLE Firma
(
    FName    VARCHAR2(10), /* Primärschlüssel */
    AnzahlMA INTEGER,
    IBAN     TEXT
);

CREATE TABLE Auto
(
    KFZZeichen VARCHAR2(10), /* Primärschlüssel */
    FName      VARCHAR2(10), /* Fremdschlüssel = Primärschlüssel aus Firma */
    Modell     VARCHAR2(8),
    Baujahr    INTEGER
);
```

```
/*
    Taxi            taxidata.sql
*/

DELETE FROM Firma;
DELETE FROM Auto;

INSERT INTO Firma VALUES ('Maxi-Taxi', 350, 'DE1234');
INSERT INTO Firma VALUES ('Luxi-Taxi',  90, 'DE2345');
INSERT INTO Firma VALUES ('Fixi-Taxi', 120, 'FR1234');

INSERT INTO Auto VALUES ('BAZ-1718', 'Maxi-Taxi', 'VW',       2011);
INSERT INTO Auto VALUES ('BMP-1718', 'Maxi-Taxi', 'Skoda',    2015);
INSERT INTO Auto VALUES ('BKA-4253', 'Luxi-Taxi', 'Mercedes', 2000);
INSERT INTO Auto VALUES ('BAZ-9876', 'Luxi-Taxi', 'BMW',      2011);
INSERT INTO Auto VALUES ('BAZ-6789', 'Fixi-Taxi', 'VW',       2011);
```

Jetzt sieht man deutlich die Vorteile der SQL-Skripte: man startet nur

```
SQL ==> .read taxicr.sql
SQL ==> .read taxidata.sql
```

und man ändert sofort die Struktur der Datenbank inkl. Daten. Die Abfragen sehen jetzt auch einfacher aus.

```
SELECT f.FName, f.IBAN, a.Modell, a.KFZZeichen
FROM Firma f, Auto a
WHERE f.FName=a.FName AND  /*  access-Prädikat (Bedingung) verknüpft zwei Tabellen */
      f.FName='Luxi-Taxi';     /*  filter-Prädikat holt Infos über Firma Luxi-Taxi */
```

Fazit über vereinfachtes Modell: weniger Daten werden in Tabellen gespeichert und die Abfragen werden einfacher.