Einführung in Datenbanken

Übung 5: Relationale DB-Schemata, CREATE TABLE

Prof. Dr. Stefan Brass
PD Dr. Alexander Hinneburg
Martin-Luther-Universität Halle-Wittenberg

Wintersemester 2023/24

http://www.informatik.uni-halle.de/~brass/db23/

Inhalt

IndustrieTag Informationstechnologie (1)

- HEUTE, 14. November, 14:00, im Raum 3.07

  Kaffeetrinken und Stände in Raum 509.

- Vorträge aus IT-Industrie und Uni
- Kaffeetrinken, „Get Together“ (Ende ca. 17:30)
- Stände von Firmen, die neue Mitarbeiter rekrutieren wollen.

  Es gibt auch ein „Job-Speeddating“.

- Man muss/sollte sich vorher anmelden:
  [https://www.uni-halle.de/uzi/veranstaltungen/37it/]

  Es ist natürlich kostenlos. Es wird ein Namensschild vorbereitet. Man wird bei Kaffee und Kuchen mitgezählt. Ohne vorherige Anmeldung muss man in einer längeren Schlange stehen (um ein „ad hoc“ Namensschild zu bekommen).

- Für Hausaufgabe 5.1 muss man einen Vortrag kennen.

Inhalt

Aufgabe 4.1: Schema in Kurznotation (1)

Aufgabe:

- Loggen Sie sich über das Adminer-Webinterface bei der PostgreSQL-Datenbank für diese Übungen ein.

  [https://dbs.informatik.uni-halle.de/edb?pgsql=db&username=student_gast&db=postgres&ns=]

- Das Passwort finden Sie in StudIP-Eintrag der Vorlesung, Reiter „Informationen“.

  Weitere Informationen zum Adminer finden Sie auf folgenden Webseiten:

  [https://www.adminer.org/de/]

  [https://en.wikipedia.org/wiki/Adminer]

  [https://linuxconfig.org/using-adminer-to-manage-your-databases]

Aufgabe 4.1: Schema in Kurznotation (2)

Aufgabe, Forts.:

- Wählen Sie in der Auswahlbox links das Schema „komponist_public“. Dies ist eine Datenbank mit Information über klassische Musikstücke und ihre Aufnahmen.

  Man kann auch im Link ns=komponist_public angeben.

- Ihre Aufgabe ist, das Schema der Datenbank herauszufinden (mit fünf Tabellen), und in der aus der Vorlesung bekannten Kurznotation aufzuschreiben, also in der Form

  R(A, B→S, C°).

- Sie können alternativ auch die ASCII-Version der Schema-Notation verwenden:

  R(#A, B->S, C?).

Aufgabe 4.1: Schema in Kurznotation (3)

Aufgabe, Forts.:

- Sie müssen auch Schlüssel und Fremdschlüssel angeben.
- Schlüssel finden Sie unter „Indizes“, was in zweierlei Hinsicht problematisch ist:
  - „Index“ ist ein Konzept des internen Schemas. Es wird u.a. benutzt, um einen Schlüssel zu implementieren. Auf Ebene des relationalen Modells interessieren wir uns nur für Schlüssel.
  - In dieser Vorlesung und vermutlich der Mehrheit der Lehrbücher wird „Indexe“ als Plural der Datenstruktur benutzt. Dagegen herrscht Einigkeit darüber, dass in $f(x_i, y_j)$ die Zahlen i und j „Indizes“ sind.

Aufgabe 4.1: Schema in Kurznotation (4)

Aufgabe, Forts.:

- Falls möglich, ordnen Sie die Tabellen so, dass Fremdschlüssel nur auf bereits vorher definierte Tabellen verweisen.
- Die alphabetische Anordnung der Tabellen (wie im Adminer) ist nicht unbedingt die beste Anordnung, um das Datenbank-Schema zu verstehen.

Hinweis:

- Bei der Klausur bekommen Sie ein neues (bisher unbekanntes) Schema in dieser Notation und einen Beispielzustand (in ILIAS und auch im Adminer).

  Sie müssen also auch lernen, sich recht zügig in eine neue Datenbank hineinzudenken.

Aufgabe 4.1: Schema in Kurznotation (5)

Schema der CD-Datenbank (klassische Musik):

- KOMPONIST(KNR, NAME, VORNAME°, GEBOREN°, GESTORBEN°)
- STUECK(SNR, KNR°→KOMPONIST, TITEL, TONART°, OPUS°)
- CD(CDNR, NAME, HERSTELLER°, ANZ_CDS°, GESAMTSPIELZEIT°)
- AUFNAHME(CDNR→CD, SNR→STUECK, ORCHESTER°, LEITUNG°)
- SOLIST((CDNR, SNR)→AUFNAHME, NAME, INSTRUMENT°)

Anmerkung:

- Die Groß-/Kleinschreibung ist beliebig.

  PostgreSQL wandelt alle Bezeichner ohne "..." in Kleinbuchstaben um, Oracle alle in Großbuchstaben. Die Original-Schreibweise aus dem `CREATE TABLE` kann man mindestens bei diesen beiden Systemen nicht mehr herausfinden.

Aufgabe 4.1: Schema in Kurznotation (6)

- Während bei der Spalte `GESTORBEN` (Todesjahr) klar ist, dass ein Nullwert nötig ist (für noch lebende Komponisten), ist das bei `GEBOREN` weniger klar.
- Aber man muss sich an das gegebene Schema halten.

  Es wäre ja möglich, dass von irgendeinem sehr frühen Komponisten die genauen Lebensdaten unbekannt sind.
- Im aktuellen Datenbank-Zustand kommt es nicht vor.

  `SELECT * FROM KOMPONIST WHERE GEBOREN IS NULL` liefert die leere Menge.
- Das Schema erlaubt auch unbekannte Vornamen, aber auch das kommt in den Daten nicht vor.

  Anfragen, z.B. mit `NACHNAME || ', ' || VORNAME`, werden dadurch komplizierter: Man muss den möglichen Nullwert speziell behandeln.
- Insgesamt erlauben wohl etwas viele Spalten Nullwerte.

Aufgabe 4.1: Schema in Kurznotation (7)

- Die Auflistung der Tabellen im Adminer ist alphabetisch.
  - Der Adminer holt die Daten aus Tabellen des Systemkatalogs (Data Dictionary).
  - Im relationalen Modell bedeutet die Reihenfolge der Tabellenzeilen nichts.
  - Deswegen haben die Programmierer des Adminers sich für die alphabetische Reihenfolge entschieden.
- Wenn man ein DB-Schema verständlich darstellen will, ist die alphabetische Reihenfolge normalerweise nicht hilfreich.
  - Z.B. sollten Fremdschlüssel möglichst auf vorher definierte Tabellen verweisen.

    Der `CREATE TABLE` Befehl würde auch einen Fehler geben, wenn man auf eine noch nicht definierte Tabelle verweist (zykl. Ref. $\rightarrow$ DBP).

Aufgabe 4.1: Schema in Kurznotation (8)

- Im Beispiel könnte CD weiter nach vorne (insbesondere ganz an den Anfang, vor KOMPONIST), ansonsten liegt die Reihenfolge durch die Fremdschlüssel fest.

  CD könnte auch zwischen KOMPONIST und STUECK, aber da STUECK einen Fremdschlüssel auf KOMPONIST enthält, und mit CD nichts zu tun hat, wäre das auch eine unglückliche Reihenfolge.

- STUECK(SNR, KNR°→KOMPONIST, TITEL, TONART°, OPUS°)
  Der Fremdschlüssel von STUECK nach KOMPONIST zeigt, dass
  - jedes Stück nur von einem Komponisten geschrieben sein kann (es gibt nur einen KNR-Wert bei jedem Stück),
  - eventuell auch gar keinem (das Fremdschlüssel-Attribut KNR in STUECK erlaubt Nullwerte).
  - Umgekehrt kann ein Komponist dagegen mehrere Stücke geschrieben haben („Eins-zu-viele Beziehung“).

Aufgabe 4.1: Schema in Kurznotation (9)

- AUFNAHME(CDNR→CD, SNR→STUECK, ORCHESTER°, LEITUNG°)
  Hier ist der Schlüssel aus zwei Fremdschlüsseln zusammengesetzt (wie bei BEWERTUNG in der Vorlesung):
  - Es kann also mehrere Aufnahmen des gleichen Stücks auf verschiedenen CDs geben.
  - Selbstverständlich kann die gleiche CD Aufnahmen mehrerer Stücke enthalten.
  - Eine CD kann aber nicht mehrere Aufnahmen des gleichen Stücks enthalten.

    Wenn das eine Anforderung war, ist der Datenbank-Entwurf falsch. Letztendlich ist das Maß für die Korrektheit eines Datenbank-Schemas aber die reale Welt. Vielleicht wären Tracknummern besser gewesen.

- „Viele-zu-viele Beziehung“ zwischen Stücken und CDs.

  Ein Stück hat Aufnahmen auf vielen CDs, eine CD enthält viele Stücke.

Aufgabe 4.1: Schema in Kurznotation (10)

- SOLIST((CDNR, SNR)→AUFNAHME, NAME, INSTRUMENT)
  - Für eine Aufnahme kann man mehrere Solisten speichern.

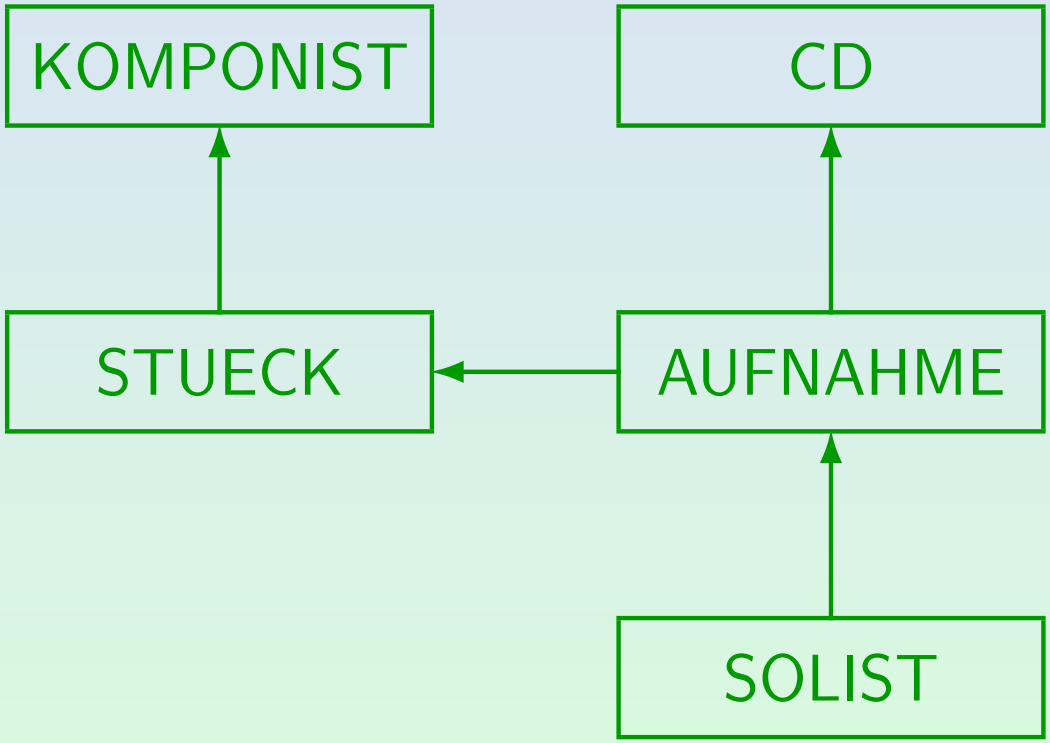
    Es ist auch eine Art „Eins-zu-viele Beziehung“: Jede Solisten-Information bezieht sich auf genau ein Stück. Ggf. Rest von viele-zu-viele mit Musiker.
  - Innerhalb einer Aufnahme sind die Solisten über den Namen identifiziert.

    Das ist ein Unterschied zur „Eins-zu-viele Beziehung“ zwischen `KOMPONIST` und `STUECK`: Dort wäre es möglich, das ein Komponist zwei Stücke mit gleichem Titel hat (das ist vielleicht ein Fehler — wird aber durch Stücke mit unbekanntem Komponisten schwierig).
  - Es kann nicht gespeichert werden, dass ein Solist in der gleichen Aufnahme verschiedene Instrumente spielt.

    Da die Liste der Instrumente nicht vorgegeben ist, könnte man allerdings dort z.B. „`Violine/Viola`“ eingeben.

Aufgabe 4.1: Schema in Kurznotation (11)

- Es kann nützlich sein, die Tabellen und Beziehungen über Fremdschlüssel graphisch zu visualisieren:



Das war natürlich nicht Teil der Hausaufgabe, aber es geht hier auch darum, wie man sich zügig in neue DB-Schemata einarbeiten kann.

Aufgabe 4.1: Schema in Kurznotation (12)

- Der Adminer zeigt keine `CHECK`-Constraints an.
- Sie werden ja ebenso wie Datentypen in der Kurznotation nicht aufgelistet, daher ist es für diese Aufgabe kein Problem.
- Im Schema wurde aber natürlich sichergestellt, dass z.B. Komponisten-Nummern nicht negativ sind.

  Bei PostgreSQL sind alle Constraints in `pg_catalog.pg_constraint` gelistet, die Bedingung allerdings in einer internen Darstellung.
  Bis PostgreSQL 11 gab es die Spalte `consrc`. Jetzt muss man die `OID` des Constraints als Eingabe der Funktion `pg_get_constraintdef(oid)` verwenden. [Adminer]
- Tatsächlich gibt es die `CREATE TABLE`- und `INSERT`-Anweisungen für diese Datenbank in einem SQL-Skript unten auf der Übungs-Webseite.

Aufgabe 4.1: Schema in Kurznotation (13)

- Das Datenbank-Schema in der ASCII-Variante der Kurznotation:
  - `KOMPONIST(#KNR, NAME, VORNAME?, GEBOREN?, GESTORBEN?)`
  - `STUECK(#SNR, KNR? -> KOMPONIST, TITEL, TONART?, OPUS?)`
  - `CD(#CDNR, NAME, HERSTELLER?, ANZ_CDS?, GESAMTSPIELZEIT?)`
  - `AUFNAHME(#CDNR -> CD, #SNR -> STUECK, ORCHESTER?, LEITUNG?)`
  - `SOLIST((#CDNR, #SNR) -> AUFNAHME, #NAME, INSTRUMENT?)`
- Man sollte das zumindest lesen können.

Aufgabe 4.1: Schema in Kurznotation (14)

- Mindestens ein Student hat bei Fremdschlüsseln, die auch Teil des Schlüssels sind, das #-Zeichen weggelassen.
  - Das ist falsch: Fremdschlüssel bedeutet nicht automatisch auch Primärschlüssel-Attribut.
- Oft wurden die Markierungen für den möglichen Nullwert vergessen (° oder ?).
- Manchmal wurden bei Fremdschlüsseln nicht nur die referenzierte Tabelle, sondern auch die Spalte mit angegeben.

  Es soll eine Kurz-Notation sein. Dabei werden immer die Primärschlüssel referenziert. Ich würde das nur im Notfall angeben (wenn andere Reihenfolge der Attribute oder wenn ein Sekundärschlüssel referenziert wird).
- Die Datentypen sollten nicht mit angegeben werden.

  Man gibt sie in dieser Notation nur bei Bedarf an (übersichtlicher).

Inhalt

Aufgabe 4.2a: CREATE TABLE (1)

- Aufgabe: Schreiben Sie eine `CREATE TABLE` Anweisung für eine Tabelle, die das Angebot eines Feuerwerkers an Silvester-Artikeln aufnehmen kann:

SILVESTER_ANGEBOT				
FIRMA	ART_NR	NAME	PREIS	NEM°
Zink	102-2	Schüttraketen ...	35.00	200
Zink	786-1	... Super-Vulkan 2	10.00	250
Zink	789-1	... Super-Vulkan 5	10.00	250
Jorge	JW4059	Malachit ...	30.00	450
Blackboxx	20045	Eissphinx ...	10.00	132
Blackboxx	20146	Blender ...	45.00	490
Blackboxx	22030	Erzengel Diavolo	150.00	1514
JGW Berckholtz	04272	Feuertopf Pulsar	6.00	

Aufgabe 4.2a: CREATE TABLE (2)

Anforderungen:

- Es soll möglich sein, die im Beispiel gezeigten Daten in der Tabelle so abzuspeichern.
- Sie sollen den Entwurf nicht verbessern.
- Die `NEM` (Netto-Effektiv-Masse, Menge an chemischer Füllung in Gramm) ist manchmal unbekannt.
  Alle anderen Spalten enthalten keine Nullwerte.
- Die Länge der Zeichenketten können Sie sinnvoll wählen.
  Die obigen Daten ergeben eine Mindestlänge.
  - `FIRMA`: 14 Zeichen
  - `ART_NR`: 6 Zeichen
  - `NAME`: 30 Zeichen.

Aufgabe 4.2a: CREATE TABLE (3)

Anforderungen, Forts.:

- Stellen Sie über einen Schlüssel sicher, dass es für die gleiche `FIRMA` und die gleiche `ART_NR` nur einen Eintrag gibt.
- Obwohl es in den Daten nicht vorkommt, muss man davon ausgehen, dass verschiedene Firmen die gleiche Artikel-Nummer verwenden können.

  Es ist die Artikelnummer des Lieferanten, nicht die des Feuerwerkers.
- Außerdem sind die Namen der Feuerwerksartikel eindeutig (es ist ja nur das Angebot eines Feuerwerkers, nicht alle auf dem Markt befindlichen Feuerwerksartikel).

Aufgabe 4.2a: CREATE TABLE (4)

Anforderungen, Forts.:

- Stellen Sie außerdem durch `CHECK`-Constraints sicher, dass
  - der Preis positiv (> 0) ist und
  - die NEM nicht-negativ (≥ 0).
- Der mögliche Nullwert braucht bei `CHECK`-Constraints nicht berücksichtigt zu werden.
  - Wie später in der Vorlesung erläutert wird, gibt der Vergleich mit einem Nullwert den dritten Wahrheitswert „unbekannt“.
  - `CHECK`-Constraints lösen nur dann eine Fehlermeldung aus, wenn der Wahrheitswert „falsch“ ist.
  - Das ist sinnvoll, weil `NOT NULL` unabhängig spezifiziert werden kann.

Aufgabe 4.2a: CREATE TABLE (5)

- Die Tabelle soll natürlich auch zukünftige Werte aufnehmen, man muss also herausfinden, was denn sinnvolle Maximallängen wären.

  Diese müssen natürlich $\geq$ den im Beispiel vorkommenden Längen sein.

```
CREATE TABLE SILVESTER_ANGEBOT(
       FIRMA  VARCHAR(14)  NOT NULL,
       ART_NR VARCHAR(6)   NOT NULL,
       NAME   VARCHAR(50)  NOT NULL,
       PREIS  NUMERIC(5,2) NOT NULL,
       NEM    NUMERIC(4),
       PRIMARY KEY(FIRMA, ART_NR),
       UNIQUE(NAME),
       CHECK(PREIS > 0),
       CHECK(NEM >= 0))
```

Aufgabe 4.2a: CREATE TABLE (6)

- Ich habe danach in meiner Artikel-Datenbank gesucht.
  - Dort kommt als Firma „`Pyrocentury/AP-Feuerwerke`“ vor mit 25 Zeichen.
  - Die längste `ART_NR` hat 23 Zeichen.

    „`ZX8121 (FB-01-01M-0500)`“ für „Feuerball blau“ von Pyrogenie.
  - Der längste `NAME` sind 226 Zeichen, enthält allerdings auch eine Beschreibung (durch eine sinnvolle Längen-Beschränkung hätte man das verhindert).

    ```
    Das Ding (25x30mm): Bukets aus roten Dahliensternen und
    blauen Sternen mit rot blinkendem Kometenaufstieg
    abwechselnd mit Weidenbuketts aus leicht blinkenden
    Goldstaubwedeln mit Cracklingspitzen und
    Silberglitter-Kometenaufstieg
    ```

Aufgabe 4.2a: CREATE TABLE (7)

- Im `CREATE TABLE` schreibt man kein „°“ zur Markierung von Spalten, die einen Nullwert erlauben.

  Wenn man `NOT NULL` nicht hinschreibt, und die Spalte auch nicht Teil eines Primärschlüssels ist, erlaubt sie Nullwerte. In der Tabellenüberschrift war die Spalte `NEM°` geschrieben, aber der ° ist hier nicht Teil des Spaltennamens.

- Die Spezialbehandlung des Nullwertes ist überflüssig:

  ```
  CHECK(NEM >= 0 OR NEM IS NULL)
  ```

  Vergleiche mit einem Nullwert liefern den dritten Wahrheitswert „unbekannt“. Die Bedingung gilt nur dann als verletzt, wenn sie den Wahrheitswert „falsch“ liefert. Das ist meist, was man will. (Details später in der Vorlesung.)

- Beide Bedingungenzusammen sind möglich, aber die Fehlermeldungen werden unspezifischer:

  ```
  CHECK(PREIS > 0 AND NEM >= 0)
  ```

  Dagegen ist `CHECK(PREIS > 0, NEM >= 0)` ein Syntaxfehler.

Aufgabe 4.2a: CREATE TABLE (8)

- Der Typ `INTEGER(6)` ist nicht portabel:
  Bei PostgreSQL gibt es einen Syntaxfehler.

  In MySQL ist der Parameter die maximale Ausgabebreite.

- Der Student, der die Anweisung offenbar mit MySQL ausprobiert hat, hat offenbar angenommen, dass `NAME` dort ein reserviertes Wort ist (und es deshalb `"name"` geschrieben).

  Nach [https://dev.mysql.com/doc/refman/8.0/en/keywords.html] ist es nur ein Keyword, aber nicht reserved. Bei MariaDB 5.5.68 funktionierte es auch. (Ergänzung: Der Student hat gesagt, sein Editor stellte das Wort in der Farbe von Schlüsselworten dar, und er wollte Probleme vermeiden.)

Aufgabe 4.2a: CREATE TABLE (9)

- Der Datentyp `NUMERIC(4)` für den Preis ist falsch:
  Es werden keine Nachkommastellen gespeichert. Das gilt auch für den fast äquivalenten Typ `DECIMAL(4)`.
- Der Typ `CHARACTER(30)` oder kurz `CHAR(30)` für den Namen ist schlecht: Es werden immer 30 Zeichen gespeichert, auch bei wesentlich kürzeren Zeichenketten.
- Der Typ `NUMERIC` ohne Parameter hat nach dem Standard keine Nachkommastellen (und Oracle macht das auch so).

  PostgreSQL macht das anders (beliebige Anzahl Nachkommastellen innerhalb der Implementierungs-Grenzen). Aus dem Handbuch: „If you're concerned about portability, always specify the precision and scale explicitly.“
- Oracle hat keinen Datentyp `MONEY` und keinen Datentyp `TEXT`. SQL-2016 auch nicht.

Aufgabe 4.2a: CREATE TABLE (10)

- Man sollte die Groß-/Kleinschreibung konsistent handhaben, selbst wenn sie eigentlich egal ist (`FIRMA` in Spaltendeklaration, `Firma` in Schlüssel).
- Die ganze Anweisung in eine Zeile ist nicht freundlich dem Leser gegenüber.

  Viele Tabulator-Zeichen am Anfang jeder Zeile, die offensichtlich von der Tabulator-Breite 4 ausgehen, auch nicht. Meine Tabulator-Breite ist 8.

- `PRIMARY KEY(FIRMA), PRIMARY KEY(ART_NR)` ist ein Syntaxfehler: Man kann nur einen Primärschlüssel pro Tabelle deklarieren.
- `PRIMARY KEY(FIRMA,ART_NR)` und `UNIQUE(FIRMA,ART_NR)` ist eine unnötige Doppelung.

Aufgabe 4.2b: CREATE TABLE mit FS (1)

- Der Feuerwerker möchte nun auch Preise einiger Webshops abspeichern, um zu prüfen, dass er eher günstige Preise hat (oder zumindest nicht auffällig teuer ist).
- Entwickeln Sie dafür eine Tabelle `PREIS_INFO`, die folgende Anforderungen erfüllt:
  - Natürlich interessiert sich der Feuerwerker nur für Artikel, die er auch selbst im Angebot hat. Jede Zeile der Tabelle `PREIS_INFO` soll sich also auf eine Zeile in `SILVESTER_ANGEBOT` beziehen.
  - In jeder Zeile steht der Name des Webshops (z.B. „`Röder`“, „`Pyroland`“), der jeweilige Preis und die Information, ob der Artikel lieferbar ist (Preise für nicht lieferbare Artikel können veraltet sein und sind keine echte Konkurrenz).

Aufgabe 4.2b: CREATE TABLE mit FS (2)

Anforderungen, Forts.:

- Die Lieferbarkeits-Information speichern Sie bitte als ein Zeichen, das nur die Werte `J` und `N` annehmen kann.

  Sie können in einem `CHECK`-Constraint zwei Bedingungen mit `OR` verknüpfen (d.h. mindestens eine der Bedingungen muss gelten).

- Natürlich kann es zu einem Artikel (d.h. einer Zeile in `SILVESTER_ANGEBOT`) mehrere Angebote verschiedener Webshops geben.

- Es soll aber nicht möglich sein, für den gleichen Artikel und den gleichen Webshop mehrere Preise abzuspeichern.

- Stellen Sie wieder sicher, dass auch in dieser Tabelle alle Preise positiv sind.

Aufgabe 4.2b: CREATE TABLE mit FS (3)

- Beispiel-Daten:
  - Z.B. kostet der „Schweizer Supervulkan No. 2“ bei „Röder“ 11.95 €,
  - und die Batterie „Malachit“ 29.95 € (ist aber derzeit nicht verfügbar).
  - Bei Pyroland kostet der „Schweizer Supervulkan No. 2“ 14.03 €.
- Es war nicht verlangt, `INSERT`-Anweisungen zu schreiben.
- Es könnte aber zum Test nützlich sein.

Aufgabe 4.2b: CREATE TABLE mit FS (4)

- Mögliche Lösung:

```
CREATE TABLE PREIS_INFO(
        SHOP       VARCHAR(20)  NOT NULL,
        NAME       VARCHAR(50)  NOT NULL,
        PREIS      NUMERIC(5,2) NOT NULL,
        LIEFERBAR  CHAR(1)      NOT NULL,
        PRIMARY KEY(SHOP, NAME),
        FOREIGN KEY (NAME)
            REFERENCES SILVESTER_ANGEBOT(NAME),
        CHECK(PREIS > 0),
        CHECK(LIEFERBAR = 'J' OR LIEFERBAR = 'N')
);
```

Aufgabe 4.2b: CREATE TABLE mit FS (5)

- Die verlangte Beziehung zum eigenen Angebot kann alternativ auch über `FIRMA` und `ART_NR` hergestellt werden.
- Mir schien die Preis-Information so lesbarer.

  Sonst müsste man öfters einen Join/Verbund mit `SILVESTER_ANGEBOT` machen, um die Namen der Feuerwerksartikel anzuzeigen.
- Es ist aber ungewöhnlich, mit einem Fremdschlüssel auf einen Sekundär-Schlüssel (`UNIQUE`) zu verweisen.

  Es ist technisch möglich und nicht verboten. Bei der algorithmischen Übersetzung aus dem ER-Modell würden solche Fremdschlüssel aber nie auftreten.

Aufgabe 4.2b: CREATE TABLE mit FS (6)

- Es könnte konsequent sein, dann auch `NAME` zum Primärschlüssel von `SILVESTER_ANGEBOT` zu machen und die Kombination aus `FIRMA` und `ART_NR` zum Alternativschlüssel.
  - Da ein Primärschlüssel typischerweise ganz vorne steht, würde man dann vielleicht auch die Spalten vertauschen.
  - Auch dies ist aber nur eine Stilfrage, keine Bedingung.
  - In der Aufgabe war die Reihenfolge der Spalten vorgegeben.
- Bei der Lösung mit `FIRMA` und `ART_NR` als Fremdschlüssel könnte man eine Sicht (virtuelle Tabelle) definieren, die den Join/Verbund macht (so dass man ihn nicht immer wieder aufschreiben muss).

  Die Sicht wäre aber nicht updatebar. Beim `INSERT` muss man `FIRMA` und `ART_NR` angeben.

Aufgabe 4.2b: CREATE TABLE mit FS (7)

- SQL hat erst seit SQL-99 einen Datentyp `BOOLEAN`.
- Dieser wird in PostgreSQL auch unterstützt, aber z.B. nicht in Oracle.

  Für moderne Programmierer sieht es zunächst komisch aus, dass der Typ `boolean` irgendwie in Frage stehen könnte. In der Prädikatenlogik hat man aber eine strikte Trennung zwischen logischen Formeln (die einen Wahrheitswert liefern) und Termen (die einen Datenwert liefern). Durch Einführung dieses Datentyps wird die Trennung aufgehoben: Auch komplexe `EXISTS`-Unteranfragen können jetzt Eingabe von Funktionen mit Argument vom Typ `boolean` sein. `MAX`- und `MIN`-Aggregation über Werten vom Typ `boolean` lassen plötzlich Disjunktionen (oder-Verknüpfungen) und Konjunktionen (und-Verknüpfungen) von beliebig vielen Operanden zu.
- Man könnte `BOOLEAN` für die Spalte `LIEFERBAR` verwenden (wenn es in der Aufgabe nicht anders stünde). Man muss den Nutzen aber gegen die geringere Portabilität abwägen.

Eigene Datenbank

- Sie haben per EMail Benutzernamen und Passwort für eine Datenbank im Adminer bekommen, in der Sie auch eigene Tabellen anlegen können.
- Der Benutzername ist gleichzeitig auch Name der Datenbank (Sie müssen ihn also in die Eingabefelder „`Username`“ und „`Database`“ beim Adminer eingeben).
- Sie müssen zunächst ein eigenes Schema anlegen, da Sie für das Standard-Schema `public` nicht das Recht haben, darin Tabellen anzulegen.
- Hat das geklappt?

Inhalt

Fehlersuche im CREATE TABLE (1)

```
(1)   CREATE TABLE R(
(2)       A NUMERIC(1) NOT NULL,
(3)       B VARCHAR(5) NOT NULL,
(4)       C NUMERIC(3),
(5)       D NUMERIC(1),
(6)       D CHAR(10),
(7)       PRIMARY KEY(A,B)
(8)   );
(9)
(10)  CREATE TABLE S(
(11)      E VARCHAR(5) NOT NULL,
(12)      F NUMERIC(1) NOT NULL,
(13)      G NUMERIC(3) NOT NULL,
(14)      C NUMERIC(3) NOT NULL,
(15)      PRIMARY KEY(G),
(16)      FOREIGN KEY(C) REFERENCES R(C),
(17)      FOREIGN KEY(E,F) REFERENCES R
(18)  );
```

Fehlersuche im CREATE TABLE (1)

- In der Tabelle `R` gibt es zwei Spalten mit Namen `D`.
  Das geht nicht:
  - Innerhalb einer Tabelle müssen die Spaltennamen eindeutig sein.
- Der erste Fremdschlüssel in `S` referenziert `R(C)`.
  Das geht nicht:
  - Man kann nur Schlüssel referenzieren (`PRIMARY KEY` oder `UNIQUE`).
- Beim zweiten Fremdschlüssel in `S` passen die Datentypen nicht: Z.B. ist `E` vom Typ `VARCHAR(5)`, aber die zugehörige Schlüssel-Spalte `A` ist vom Typ `NUMERIC(1)`.
  - Datentypen vom Fremdschlüssel und zugehöriger Schlüssel-Spalte müssen „kompatibel“ sein.

Inhalt

Präsenzaufgabe: Schema für Koch-/Back-Rezepte

- Entwerfen Sie ein Schema für Rezepte (zum Kochen/Backen):
  - Zu jedem Rezept muss eine Nummer, eine Bezeichnung, ein Schwierigkeitsgrad, und eine Dauer (der Zubereitung) gespeichert werden. Bezeichnungen sind nicht eindeutig.

    Die Nummer soll das Rezept eindeutig identifizieren.
  - Ein Rezept besteht aus mehreren Schritten. Es ist die Reihenfolge der Schritte festzuhalten, außerdem ein Text (Anweisung für den Schritt).
  - Für jeden Schritt ist zu speichern, welche Zutaten in welcher Menge benötigt werden. In einen Schritt können auch mehrere Zutaten benötigt werden (oder keine).

    Die gleiche Zutat in verschiedenen Schritten eines Rezepts ist möglich.
- Geben Sie das Schema in Kurznotation als `.txt`-Datei ab.