Einführung in Datenbanken

Übung 6: CREATE TABLE, Logik

Prof. Dr. Stefan Brass
PD Dr. Alexander Hinneburg

Martin-Luther-Universität Halle-Wittenberg

Wintersemester 2024/25

http://www.informatik.uni-halle.de/~brass/db24/

Inhalt

IndustrieTag InformationsTechnologie

- IT^2: Heute, 12. November, 14:15, im Raum 3.07

  Kaffeetrinken und Stände in Raum 509.

- Vorträge aus IT-Industrie und Uni
- Kaffeetrinken, „Get Together“ (Raum 5.09, ca. 16:00–17:30)
- Stände von Firmen, die neue Mitarbeiter rekrutieren wollen.
- Man muss/sollte sich vorher anmelden: [https://www.uni-halle.de/uzi/veranstaltungen/39it/]

  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).

- Besuch nötig für eine Aufgabe auf dem neuen Übungsblatt.

Logik in der Klausur (1)

- In der Klausur wird Logik (aller Voraussicht nach) nur in der SQL-Syntax drankommen.
- Es sind aber z.B. schon Aufgaben möglich, die fragen, ob zwei gegebene SQL-Anfragen äquivalent sind, also in beliebigen DB-Zuständen immer die gleiche Antwort geben.

  Dazu sollte man logische Äquivalenzen kennen. Sie können bei der Klausur zwar Anfragen ausprobieren, aber wir machen die Aufgaben natürlich so, dass ein Test im gegebenen Zustand keinen Unterschied zeigt (oder sie beziehen sich auf ein ganz anderes Schema, das im Adminer nicht angelegt ist).
- In der Klausur werden immer wieder inkonsistente Anfragen geschrieben (oder inkonsistente Teilbedingungen). Es ist nützlich, solche Konzepte zu kennen, um diesen Fehler zu vermeiden oder wenigstens das Problem zu verstehen.

Logik in der Klausur (2)

- Der komplexeste Teil von SQL-Anfragen sind die `WHERE`-Bedinungen. Das sind logische Formeln.

  Logik ist also praktisch wichtig für Datenbanken.

- Es werden (sehr wahrscheinlich) keine Definition abgefragt, sondern praktische Anwendungen verlangt.

  Sie dürfen 5 Blätter/10 Seiten mit Notizen verwenden (bei der ersten Klausur, bei der zweiten nur 3/6). Es ist also nicht direkt nötig, etwas bewusst auswendig zu lernen (zu „pauken“). Bei der Beschäftigung mit dem Stoff prägen sich natürlich gewisse Dinge ein. Die Zeit in der Klausur ist relativ knapp, Sie können daher nicht ständig in den Notizen nachschauen, und schon gar nicht länger darin suchen.

- Es gibt alte Klausuren auf der Webseite zur Vorlesung (unter „Prüfung“). Eine Probeklausur ist geplant.

Hinweis zum Adminer: Kommentare nach „;“

- Wenn Sie Ihre Anfrage mit einem Semikolon abschließen und danach noch Kommentare schreiben, führt das zumindest im Adminer zu einer Fehlermeldung.

  Nicht aber in der Kommandoschnittstelle `psql`.

- Error in query: Unknown error.

- Für den Adminer beginnt nach dem Semikolon eine neue Anfrage. Offenbar kann er mit der Antwort von PostgreSQL auf leere Anfragen (die nur aus einem Kommentar bestehen) nicht umgehen.

- SQL-Anfragen müssen nicht mit einem Semikolon abgeschlossen werden. Sie können so auch nach der eigentlichen Anfrage noch Kommentare schreiben.

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.

Präsenzaufgabe: Erste Lösung (1)

Lösung 1 (mit Fehlern):

- Rezept(Nummer, Bezeichnung, Schwierigkeitsgrad, Dauer)
- Schritt(Position, Nummer → Rezept, Anweisung)
- Zutat(Name, Nummer → Rezept)
- Zutat-pro-Schritt(Nummer → Rezept, Position → Schritt, Name → Zutat, Menge)

Frage:

- Sieht jemand Fehler/Verbesserungsmöglichkeiten?

Präsenzaufgabe: Erste Lösung (2)

Fehler I (Syntaxfehler):

- Ein Schlüssel aus zwei Attributen muss mit zwei Attributen referenziert werden.

  Anzahl (und Typen) der Fremdschlüssel-Attribute muss passen zu der Anzahl der Schlüssel-Attribute.

- `Zutat-pro-Schritt(`Nummer $\rightarrow$ `Rezept,`
  Position $\rightarrow$ `Schritt,`
  Name $\rightarrow$ `Zutat, Menge)`

- `Schritt(`Position`,` Nummer`, ...)`
  `Zutat(`Name`,` Nummer`, ...)`

Präsenzaufgabe: Erste Lösung (3)

Fehler II (Redundanz):

- Was ist der Sinn der Relation `Zutat`?
- Die Information, welche Zutat in welchem Rezept verwendet wird, ist bereits in der Relation `Zutat-pro-Schritt` enthalten.
- Vermeiden Sie die Speicherung redundanter Daten!
  - Mehr Arbeit beim Eingeben.
  - Gefahr von Inkonsistenzen.
- Sie können später aber `Zutat` als virtuelle Relation (Sicht) definieren, die aus `Zutat-pro-Schritt` berechnet wird.

Die Zusammenstellung der Zutaten pro Rezept ist ja durchaus sinnvoll.

Präsenzaufgabe: Erste Lösung (4)

Stilfrage I (Bezeichner):

- In SQL sind Bindestriche in Bezeichnern nicht zulässig: `Zutat-pro-Schritt`.
- Man kann es als „Delimited Identifier“ haben: `"Zutat-pro-Schritt"`.
- Man erleichtert sich spätere Anfragen aber, wenn man Unterstriche wählt: `Zutat_pro_Schritt`.

  In der Auflistung der Relationen erscheint es bei PostgreSQL dann allerdings ganz in Kleinbuchstaben.

- Bei der Lösung mit „Delimited Identifier“ muss man auch später in jeder Anfrage `"..."` schreiben.

Präsenzaufgabe: Erste Lösung (5)

Stilfrage II (Attribut-Reihenfolge):

- Schritt(Position, Nummer → Rezept, Anweisung)
- Man sollte die Attribute so anordnen, dass die größere Einheit (hier die Rezept-Nummer) vor der kleineren Einheit (die Position des Schrittes innerhalb des Rezepts) steht.

  Gewissermaßen nach der Priorität in der natürlichen Sortierreihenfolge.

- Man sollte gleiche Attribute möglichst auch immer in gleicher Reihenfolge anordnen:

  Zutat-pro-Schritt(Nummer → Rezept,
  Position → Schritt,
  Name → Zutat, Menge)

  Mindestens für die Fremdschlüssel in dieser Notation ist das zwingend.

  In SQL nicht. Eventuell bei `SELECT *`, `INSERT INTO` ohne Spaltenangabe.

Präsenzaufgabe: Erste Lösung (6)

Korrigierte Fassung von Lösung 1 (optimal):

- Rezept(Nummer, Bezeichnung, Schwierigkeitsgrad, Dauer)
- Schritt(Nummer → Rezept, Position, Anweisung)
- Zutat_pro_Schritt((Nummer, Position) → Schritt, Zutat_Name, Menge)

Frage:

- Braucht man in Zutat_pro_Schritt zusätzlich den Fremdschlüssel Nummer → Rezept?

Präsenzaufgabe: Zweite Lösung (1)

- ```
  CREATE TABLE REZEPTE(
        Rezeptnummer     NUMERIC(3)  NOT NULL,
        Rezeptbezeichner VARCHAR(20) NOT NULL,
        Dauer_in_min     NUMERIC(50) NOT NULL
        PRIMARY KEY (Rezeptnummer),
        UNIQUE (Rezeptbezeichner))
  ```

  Es war die Kurznotation gefordert. Schwierigkeitsgrad fehlt.

  Bezeichner nicht eindeutig (mehrere Rezepte für das gleiche Gericht).

- Sieht jemand den Syntaxfehler?
- Was bedeutet der Typ `NUMERIC(50)`?

  Viele DBMS werden das nicht unterstützen.

- `VARCHAR(20)` für den Rezeptbezeichner ist zu kurz.

  Z.B. hat „Elisenlebkuchen mit Belegkirschen“ 33 Zeichen.
  ```

Präsenzaufgabe: Zweite Lösung (2)

- ```
  CREATE TABLE ABLAUF(
          Rezeptnummer  NUMERIC(3)    NOT NULL,
          Platzierung   NUMERIC(50)   NOT NULL,
          Anweisung     VARCHAR(1000) NOT NULL,
          FOREIGN KEY (Rezeptnummer)
                        REFERENCES REZEPTE,
          FOREIGN KEY (Platzierung)
                        REFERENCES ZUTATEN)
  ```
- Anweisungen so anordnen, dass Fremdschlüssel bereits existierende Tabellen referenzieren (`ZUTATEN` vorher).

  Hier also besser zuerst `ZUTATEN`. Bei wechselseitigen Referenzen ist das nicht möglich, dann Vorlesung „Datenbank-Programmierung“ hören (`ALTER TABLE`).
- Tabelle hat keinen Schlüssel! → nächste Folie ...
  ```

Präsenzaufgabe: Zweite Lösung (3)

- Wenn man für eine Tabelle keinen Schlüssel deklariert, sind in SQL doppelte Zeilen möglich (in allen Spalten gleich).

  In der Theorie sind Relationen Mengen, in SQL aber Multimengen (von Tupeln). Mir ist bisher kein Fall begegnet, in dem eine gespeicherte Relation mit Duplikats-Tupeln wünschenswert gewesen wäre (bei Relationen als Anfrageergebnis ist das anders, da können Duplikate im Ausnahmefall wichtig sein). Es wird handhabbarer, wenn man eine Zahl als künstlichen Schlüssel hinzu fügt. Wenn man nur Duplikate ausschließen will, kann man das mit einem Schlüssel aus allen Attributen tun.

- Im konkreten Beispiel will man für eine Rezept-Nummer und eine „Platzierung“ (Schritt-Nummer) nur einen Eintrag in der Tabelle haben:

```
PRIMARY KEY(Rezeptnummer, Platzierung)
```

Präsenzaufgabe: Zweite Lösung (4)

- ```
  CREATE TABLE ZUTATEN(
          Rezeptnummer    NUMERIC(3)      NOT NULL,
          Zutat           VARCHAR(50)     NOT NULL,
          Platzierung     NUMERIC(50)     NOT NULL,
          Menge_in_g      NUMERIC(1000),
          FOREIGN KEY (Rezeptnummer)
                          REFERENCES REZEPTE)
  ```
- `NUMERIC(1000)` ist viel zu groß.
- Nullwert für Menge?
- Schlüssel fehlt.

  Der in `ABLAUF` deklarierte `FOREIGN KEY(Platzierung) REFERENCES ZUTATEN` ist nur möglich, wenn der Schlüssel hier ausschließlich aus `Platzierung` besteht. Dann aber nicht in einem Schritt mehrere Zutaten. Der Fremdschlüssel müsste eigentlich in der anderen Richtung verweisen, von `ZUTAT` zu `ABLAUF`.
  ```

Präsenzaufgabe: Zweite Lösung (5)

Korrigiert:

- REZEPTE(Rezeptnummer, Rezeptbezeichner, Schwierigkeitsgrad, Dauer)
- ABLAUF(Rezeptnummer → REZEPTE, Platzierung, Anweisung)
- ZUTATEN((Rezeptnummer, Platzierung) → ABLAUF, Zutat, Menge_in_g)
- Datentypen (Vorschläge, es gibt Bereich sinnvoller Längen):

Rezeptnummer	NUMERIC(5)
Platzierung	NUMERIC(2)
Schwierigkeitsgrad	NUMERIC(1)
Menge_in_g	NUMERIC(4)
Rezeptbezeichner	VARCHAR(50)

Präsenzaufgabe: Zweite Lösung (6)

- ```
  CREATE TABLE REZEPTE(
        Rezeptnummer        NUMERIC(5)   NOT NULL,
        Rezeptbezeichner    VARCHAR(50)  NOT NULL,
        Schwierigkeitsgrad  NUMERIC(1)   NOT NULL,
        Dauer_in_min        NUMERIC(3)   NOT NULL,
        PRIMARY KEY (Rezeptnummer))
  ```

- ```
  CREATE TABLE ABLAUF(
        Rezeptnummer  NUMERIC(5)     NOT NULL,
        Platzierung   VARCHAR(2)     NOT NULL,
        Anweisung     VARCHAR(1000)  NOT NULL,
        PRIMARY KEY (Rezeptnummer, Platzierung),
        FOREIGN KEY (Rezeptnummer)
                    REFERENCES REZEPTE)
  ```

Präsenzaufgabe: Zweite Lösung (7)

- ```
  CREATE TABLE ZUTATEN(
          Rezeptnummer   NUMERIC(5)  NOT NULL,
          Platzierung    VARCHAR(2)  NOT NULL,
          Zutat          VARCHAR(30) NOT NULL,
          Menge_in_g     NUMERIC(4)  NOT NULL,
          PRIMARY KEY (Rezeptnummer, Platzierung,
                       Zutat),
          FOREIGN KEY (Rezeptnummer, Platzierung)
                       REFERENCES ABLAUF)
  ```

- Eigentlich gibt es keinen Grund, warum die Tabellennamen ganz groß geschrieben sind, aber Spaltennamen groß/klein.

  PostgreSQL wandelt sowieso alles in Kleinbuchstaben um, solange man nicht `"..."` verwendet. Aber wenigstens in der Dokumentation will man es natürlich schön (und einheitlich) schreiben. Etwas Geschmackssache.
  ```

Präsenzaufgabe: Dritte Lösung

- Rezept(RID, Bezeichnung, Schwierigkeit, Dauer)
- Zutat(ZID, Name)
- Zubereitung((RID → Rezept, ZID → Zutat, Schritt, Menge, Text)
- Man hat so pro Zutat in einem Schritt einen Text, nicht pro Schritt (des Rezepts).
  Wie will man die Texte bei mehreren Zutaten ordnen?

  Z.B., um das Rezept zu drucken.
- Text zu Schritten ohne Zutat gar nicht speicherbar!
- Extra Zutaten-Tabelle: Tippfehler/Schreibvarianten würden vermutlich besser bemerkt, als wenn Zutaten-Namen direkt im Rezept. Eventuell Mengen-Einheit hier (g/Stück)?

  Ist aber auch eine Frage der Benutzerschnittstelle. Größerer Aufwand.

Präsenzaufgabe: Weitere Fragen (1)

- Kann in ABLAUF auch nur die Schrittnummer Schlüssel sein?

  ABLAUF(Schrittnummer, Anweisung, Rezeptnummer → REZEPTE)

  - Im Prinzip ja, auch so haben die Schritte eines Rezepts eine definierte Reihenfolge.
  - Aber der erste Schritt hat dann nicht die Schrittnummer 1 (höchstens für ein Rezept), sondern z.B. 1234.
  - Die Schrittnummer muss jetzt ja global eindeutig sein (in der ganzen Tabelle), nicht nur innerhalb eines Rezepts.
- Woher Notation A^* für Schlüssel?

Präsenzaufgabe: Weitere Fragen (2)

- Kann man nicht auch einfach einen künstlichen Schlüssel (fortlaufende Nummer) für jede Relation nehmen?
  - REZEPTE(RID, Bezeichnung, Schwierigkeitsgrad, Dauer)
  - SCHRITTE(SID, Reihenfolge, Anweisung, RID → REZEPTE)
  - ZUTATEN(ZID, Zutat, Menge, SID → SCHRITTE)
- Im Prinzip ja, aber dann werden zusätzliche UNIQUE-Schlüssel besonders wichtig: So kann z.B. die gleiche Zutat im gleichen Schritt mehrfach angegeben werden.
- Braucht man in SCHRITTE eine extra Reihenfolge, wenn man schon die SID hat? Redundant?

Präsenzaufgabe: Weitere Fragen (3)

- Geht auch dies?

  Zutaten(Zutat°, Menge,
  (SchrittNr,Nummer) → Schritt)

  - Nein, ein Primärschlüssel-Attribut, das auch Nullwerte erlaubt, ist automatisch ein Syntaxfehler.
  - Wenn ein Schritt keine Zutaten hat, braucht er auch keinen Eintrag in der Zutaten-Tabelle.

Inhalt

5. Übungsblatt: Aufgabe 1 (1)

- Wählen Sie einen Fachvortrag vom Industrietag Informationstechnologie IT^2, der am 12.11.2024 von 14 bis 18 Uhr stattfindet (Vorträge 14–16 im Raum 3.07).
- Modellieren Sie eine im gewählten Vortrag beschriebene Anwendung, einen relevanten Teil, oder eine Beispiel-Anwendung aus dem Vortrag als relationales Datenbankschema mit etwa 2 bis 3 Tabellen.
  - Geben Sie die Tabellen mit Primär- und Fremdschlüsseln in Kurznotation wie auf Folie 6-37 an.
  - Geben Sie weiterhin für alle Tabellen Tupel an, die verdeutlichen, was der Inhalten der einzelnen Spalten ist.

    Der gesamte Datenbankzustand soll etwa 8–12 Tupel umfassen.

5. Übungsblatt: Aufgabe 1 (2)

- Anforderungen, Forts.:
  - Beschreiben Sie knapp und verständlich, was die Attribute der Tabellen bedeuten sollen.

    Soweit es aus den Spaltennamen und den Beispieldaten nicht schon völlig klar ist.
  - Beschreiben Sie weiterhin mindestens einen Anwendungsprozess, der Daten in die Tabellen einfügt, liest oder aktualisiert.
- Sie können die Lösung als `.txt` oder als PDF-Datei abgeben.
- **Hinweis:** Für den Besuch des IT-Tags müssen (sollten) Sie sich frühzeitig über die Web-Seite anmelden.

  [https://www.uni-halle.de/uzi/veranstaltungen/39it/]

5. Übungsblatt, Aufgabe 2: Logik-Rätsel (1)

- Ein König lässt eine junge Dame einsperren, weil sie für mehr Umweltschutz demonstriert hat. Ihr Verlobter möchte sie gern befreien. Der König ist ein Fan von Logik-Rätseln, und stellt den Verlobten vor die Wahl zwischen drei Türen:
  - In einem der drei Räume ist seine Herzens-Dame. Wenn er diese Tür öffnet, darf er mit ihr gehen.
  - In einem anderen Raum ist ein hungriger Tiger. Wenn er diese Tür öffnet, wird er höchstwahrscheinlich aufgefressen.
  - Der dritte Raum ist leer. Er überlebt dann zwar, muss aber ohne seine Liebste gehen.

Dieses Logikrätsel stammt aus dem Buch „Dame oder Tiger“ von Raymond Smullyan. Sie können Auszüge aus dem titel-gebenden Abschnitt des Buches auf dieser Webseite finden: [https://emath.de/Referate/Smullyan.pdf]
Ich habe die Geschichte und das Rätsel leicht verändert.

5. Übungsblatt, Aufgabe 2: Logik-Rätsel (2)

- An den drei Räumen gibt es Schilder mit folgenden Inschriften:
  - Raum 1: „Dieser Raum ist leer.“
  - Raum 2: „Der Tiger ist in Raum 1.“
  - Raum 3: „Die Dame ist in Raum 1.“
- Um die Sache etwas komplizierter zu machen, gelten folgende Regeln:
  - Wenn die Dame in dem Raum ist, ist die Inschrift des Schildes wahr.
  - Die Inschrift des Schildes an dem Raum, in dem der Tiger ist, ist falsch.
  - Beim leeren Raum kann die Inschrift des Schildes wahr oder falsch sein.

5. Übungsblatt, Aufgabe 2: Logik-Rätsel (3)

- Wir codieren die Möglichkeiten als Zeilen einer Tabelle:

MOEGLICH		
R1	R2	R3
D	T	
D		T
T	D	
	D	T
T		D
	T	D

5. Übungsblatt, Aufgabe 2: Logik-Rätsel (4)

- Aus den Inschriften und den Zusatzbedingungen ergeben sich folgende logischen Formeln (jeweils zwei Wenn-Dann-Bedingungen pro Schild):
  - `R1 = 'D'` $\rightarrow$ `R1 = ' '`
  - `R1 = 'T'` $\rightarrow \neg$ `R1 = ' '`
  - `R2 = 'D'` $\rightarrow$ `R1 = 'T'`
  - `R2 = 'T'` $\rightarrow \neg$ `R1 = 'T'`
  - `R3 = 'D'` $\rightarrow$ `R1 = 'D'`
  - `R3 = 'T'` $\rightarrow \neg$ `R1 = 'D'`

In SQL gibt es leider kein „$\rightarrow$“ (wenn-dann), aber `A` $\rightarrow$ `B` ist bekanntlich äquivalent zu $\neg$ `A` $\vee$ `B`, und kann damit in SQL mit `NOT` und `OR` ausgedrückt werden. Wenn ein Tiger im jeweiligen Raum ist, ist die Inschrift des Schildes falsch und wird deswegen mit $\neg$ (`NOT` in SQL) negiert.

5. Übungsblatt, Aufgabe 2: Logik-Rätsel (5)

- Ihre Aufgabe ist nun, die obigen Bedingungen in eine `WHERE`-Bedingung in SQL zu überführen (alle sechs Bedingungen müssen gelten, also mit `AND` verknüpft werden).
- Sie dürfen, wenn Sie möchten, `NOT` entfernen, indem Sie den jeweiligen Vergleich invertieren.
- Ansonsten bleiben Sie bitte möglichst nahe an den gegebenen Bedingungen (in der Geschichte wäre es ja fatal, wenn irgendein Fehler geschehen würde).

5. Übungsblatt, Aufgabe 2: Logik-Rätsel (6)

- Leider gibt es die Tabelle MOEGLICH nicht in unserer Adminer-Installation, aber man kann auf folgende Weise eine lokale Tabelle nur für die eine Anfrage definieren:

```
WITH MOEGLICH(R1, R2, R3) AS
    (VALUES
        ('D', 'T', ' '),
        ('D', ' ', 'T'),
        ('T', 'D', ' '),
        (' ', 'D', 'T'),
        ('T', ' ', 'D'),
        (' ', 'T', 'D'))
SELECT *
FROM MOEGLICH
```

[https://users.informatik.uni-halle.de/~brass/db24/homework/h5_dt.sql]

5. Übungsblatt, Aufgabe 2: Logik-Rätsel (7)

- Für diese Aufgabe ist wesentlich, dass man die folgende Äquivalenz kennt:
  - $A \to B$ ist äquivalent zu $\neg A \vee B$, bzw. in SQL: `NOT A OR B`.
- Eine „Wenn-Dann-Bedingung“ ist erfüllt gdw.
  - Die Voraussetzung falsch ist, oder

    Dann ist egal, ob die Folgerung wahr oder falsch ist. Es ist ja keine „Genau-dann-wenn-Bedingung“.
  - die Folgerung wahr ist.

    Dann ist egal, ob die Voraussetzung wahr oder falsch ist.
- Dies ist eine nützliche Äquivalenz, die man sich merken sollte.

  Man braucht diese Äquivalenz z.B. für `CHECK`-Constraints, weil es kein $\to$ in SQL gibt, aber natürlich schon `NOT` und `OR`.

5. Übungsblatt, Aufgabe 3: Erster Join (1)

- Diese Aufgabe bezieht sich auf die schon bekannte Datenbank mit Informationen über Musik-CDs (klassische Musik). Sie finden Sie im Adminer, Schema „komponist_public“. Die Webadresse ist:

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

- Sie benötigen die Tabellen
  - KOMPONIST(KNR, NAME, VORNAME°, GEBOREN°, GESTORBEN°)
  - STUECK(SNR, KNR°→KOMPONIST, TITEL, TONART°, OPUS°)

5. Übungsblatt, Aufgabe 3: Erster Join (2)

- Sie wollen wissen, welche Stücke in der Tonart „F-dur“ von Johann Sebastian Bach und Georg Friedrich Händel in der Datenbank sind.
- Geben Sie Name und Vorname des Komponisten sowie den Titel des Stückes aus.
- Nennen Sie die Spalten der Ausgabe bitte „`Nachname`“, „`Vorname`“ und „`Titel`“ (auch in genau dieser Groß-/Kleinschreibung).
- Tipp: Sie brauchen über beiden Tabellen jeweils eine Tupelvariable.
- Das erwartete Ergebnis steht auf der nächsten Folie.

5. Übungsblatt, Aufgabe 3: Erster Join (3)

Nachname	Vorname	Titel
Händel	Georg Friedrich	Wassermusik, Suite in F-dur
Händel	Georg Friedrich	Concerto grosso op.3 Nr.4
Händel	Georg Friedrich	Concerto grosso op.6 Nr.2
Händel	Georg Friedrich	Concerto grosso op.6 Nr.9
Bach	Johann Sebastian	Brandenburg... Konzert Nr.1
Bach	Johann Sebastian	Brandenburg... Konzert Nr.2

5. Übungsblatt, Aufgabe 3: Erster Join (4)

Hinweise:

- Sie dürfen nur die Informationen im Aufgabentext verwenden (und ein wenig Allgemeinbildung).
  - Sie dürfen z.B. nicht die Komponistennummern der beiden Komponisten erst in der Beispieldatenbank nachschlagen, und dann in die Anfrage einsetzen.
  - Ihre Anfrage muss auch mit anderen Testzuständen funktionieren, nicht nur mit dem gegebenen Beispiel-Zustand.
- Bitte verwenden Sie möglichst nur die in der Vorlesung bereits vorgestellten Konstrukte von SQL.

Inhalt

4. Übungsblatt, Aufgabe 1: Schema-Notation (1)

- 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=komponist_public]

  Diese Aufgabe bezieht sich auf das Schema „`komponist_public`“.

  Wir haben auf dem zweiten Übungsblatt schon die Tabelle `KOMPONIST` verwendet.

- Das Schema der Tabelle „`KOMPONIST`“ in der Kurznotation der Vorlesung ist:

  `KOMPONIST(KNr, Name, Vorname°, geboren°, gestorben°)`

- In der Variante nur mit ASCII-Zeichen ist es:

  ```
  KOMPONIST(#KNr, Name, Vorname?, geboren?,
                                  gestorben?)
  ```

4. Übungsblatt, Aufgabe 1: Schema-Notation (2)

- Man kann darüber streiten, ob es angemessen ist, Nullwerte in der Spalte „`Vorname`“ zu erlauben.

  In den Daten kommt es nicht vor. Es gibt aber nicht in allen Kulturen Vornamen nach deutschem Muster. Die Datenbank enthält teils auch relativ alte Musikstücke von kaum bekannten Komponisten — wahrscheinlich war befürchtet worden, dass die Trennung von Namen in Vor- und Nachname nicht immer möglich ist.

- Bei der Spalte „`gestorben`“ braucht man Nullwerte sicher für noch lebende Komponisten.

- Es ist außerdem auch plausibel, dass man eventuell nicht von jedem Komponisten die Lebensdaten ermitteln kann.

4. Übungsblatt, Aufgabe 1: Schema-Notation (3)

- Ihre Aufgabe ist, das Schema der übrigen vier Tabellen dieser Datenbank herauszufinden (`Stueck`, `CD`, `Aufnahme`, `Solist`), und in dieser Kurznotation aufzuschreiben (entsprechend Kap. 6 der Vorlesung).
- Wählen Sie dafür am besten die ASCII-Variante der Notation und geben Sie das Schema als `.txt`-Datei ab.
- Wenn Sie im Adminer auf den Tabellennamen links klicken, wird das Schema der jeweiligen Tabelle angezeigt.
- Die Datentypen der Spalten sind für diese Aufgabe nicht relevant.
- Schlüssel und Fremdschlüssel müssen Sie aber angeben.

4. Übungsblatt, Aufgabe 1: Schema-Notation (4)

- 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.
- Die Groß-/Kleinschreibung von Tabellen- und Spaltennamen ist egal.

  Der Adminer zeigt die Namen aus dem Systemkatalog von PostgreSQL an, und PostgreSQL konvertiert alle Namen in Kleinbuchstaben (außer bei `"..."`).

4. Übungsblatt, Aufgabe 1: Schema-Notation (5)

Lösung (Variante mit Unterstreichen):

- 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.

4. Übungsblatt, Aufgabe 1: Schema-Notation (6)

Lösung (ASCII-Variante):

- 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?)

4. Übungsblatt, Aufgabe 1: Schema-Notation (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).

4. Übungsblatt, Aufgabe 1: Schema-Notation (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“).

4. Übungsblatt, Aufgabe 1: Schema-Notation (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.

4. Übungsblatt, Aufgabe 1: Schema-Notation (10)

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

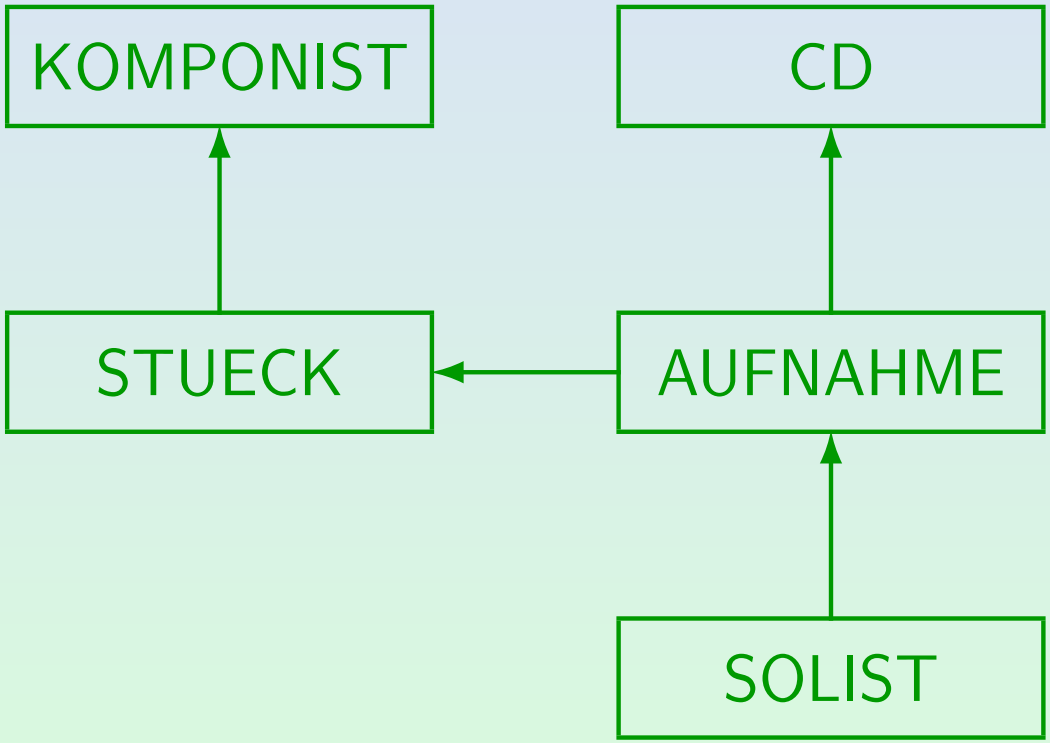
    Es ist auch eine Art „Eins-zu-viele Beziehung“: Jede Solisten-Information bezieht sich auf genau ein Stück. Ggf. Rest von viele-zu-viele mit Musiker.
  - Innerhalb einer Aufnahme sind die Solisten über den Namen identifiziert.

    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.

4. Übungsblatt, Aufgabe 1: Schema-Notation (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.

4. Übungsblatt, Aufgabe 1: Schema-Notation (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.

4. Übungsblatt, Aufgabe 2: CREATE TABLE (1)

- Es soll eine Datenbank mit allen Modulen eines Studiengangs erstellt werden, die auch die Voraussetzungen enthält.
- Ihre Aufgabe ist es, `CREATE TABLE`-Anweisungen für die beiden Tabellen dieser Datenbank zu erstellen.

4. Übungsblatt, Aufgabe 2: CREATE TABLE (2)

- Die Tabelle „Module“ sieht so aus:

Module						
ID	Titel	LP	Art	Fach	Sem	D.
INF.00677.09	OOP	5	P		1	1
MAT.02372.02	Mathematik B	15	P		1	2
INF.06483.05	Einf. Datenbanken	5	P		3	1
INF.06484.03	DB-Programmierung	5	W		4	1
WIW.00388.05	Grundlagen BWL	5	A	BWL	3	1

  - Die Werte in der Spalte ID sind Zeichenketten, die immer genau 12 Zeichen lang sind. Diese Spalte ist Primärschlüssel der Tabelle.
  - Die Werte in der Spalte Titel sind Zeichenketten bis zur Länge 80. Die Werte in dieser Spalte sind ebenfalls eindeutig.

4. Übungsblatt, Aufgabe 2: CREATE TABLE (3)

- Spalten der Tabelle `Module`, Forts.:
  - In die Spalte `LP` werden nicht-negative ganze Zahlen eingetragen.

    Stellen Sie bitte mit einem `CHECK`-Constraint sicher, dass die Einfügung negativer Zahlen zu einem Fehler führt. Der Wert `0` soll möglich sein.
  - In der Spalte „`Art`“ steht einer der folgenden Werte (jeweils ein Zeichen):
    - „`P`“ für ein Pflichtmodul,
    - „`W`“ für ein Wahlpflichtmodul der Informatik.

      Auch Bio- und Wirtschaftsinformatik.
    - „`A`“ für ein Modul eines Anwendungsfaches (das in der Spalte „`Fach`“ steht).

    Schreiben Sie bitte einen `CHECK`-Constraint, der sicherstellt, dass man keine anderen Werte in die Spalte einfügen kann. Sie können dafür auch die logische Verknüpfung `OR` („oder“) verwenden.

4. Übungsblatt, Aufgabe 2: CREATE TABLE (4)

- Spalten der Tabelle `Module`, Forts.:
  - Die Spalte „`Fach`“ soll Zeichenketten bis zur Länge 60 erlauben, und kann aber auch einen Nullwert enthalten.
  - Sie bekommen einen Bonuspunkt, wenn Sie mit einem `CHECK`-Constraint sicherstellen, dass die Spalte genau dann einen Nullwert enthält, wenn in der Spalte „`Art`“ nicht der Wert „`A`“ steht.

    D.h. die Spalte soll nur für Anwendungsfach-Module verwendet werden. Sie können mit den Bedingungen „`Fach IS NULL`“ und „`Fach IS NOT NULL`“ die Spalte auf den Nullwert testen.

4. Übungsblatt, Aufgabe 2: CREATE TABLE (5)

- Spalten der Tabelle `Module`, Forts.:
  - In der Spalte „`Sem`“ steht das empfohlene Semester, eine positive ganze Zahl.

    Der DB-Entwurf ist etwas vereinfacht, da es auch mehrere Semester zur Auswahl geben kann — das ist in dieser Tabellenstruktur nicht möglich. Die Spalte kann auch einen Nullwert enthalten, wenn das Modul z.B. nur ganz unregelmäßig angeboten wird.
  - Die Spalte „`Dauer`“ gibt die Dauer des Moduls in Semestern an. Erlaubte Werte sind nur `1` und `2`.

    Schreiben Sie wieder einen entsprechenden `CEHCK`-Constraint.

4. Übungsblatt, Aufgabe 2: CREATE TABLE (6)

Lösung:

- Erste Hälfte mit Spaltendeklarationen:

```
CREATE TABLE Module(
        ID     CHAR(12)     NOT NULL,
        Titel  VARCHAR(80)  NOT NULL,
        LP     INTEGER      NOT NULL,
        Art    CHAR(1)      NOT NULL,
        Fach   VARCHAR(60)  NULL,
        Sem    INTEGER      NULL,
        Dauer  INTEGER      NOT NULL,
```

4. Übungsblatt, Aufgabe 2: CREATE TABLE (7)

Lösung, Forts.:

- Zweite Hälfte mit Integritätsbedingungen:

```
        PRIMARY KEY (ID),
        UNIQUE (TITEL),
        CHECK(LP >= 0),
        CHECK(Art = 'P' OR Art = 'W' OR
              Art = 'A'),
        CHECK(Sem > 0),
        CHECK(Dauer = 1 OR Dauer = 2),
        CHECK(Art = 'A' AND Fach IS NOT NULL OR
                Art <> 'A' AND Fach IS NULL)
);
```

4. Übungsblatt, Aufgabe 2: CREATE TABLE (8)

Lösung, Anmerkungen:

- Man kann Spaltendeklarationen und Integritätsbedingungen beliebig mischen, also z.B. den `CHECK`-Constraint für eine Spalte direkt nach der Spaltendeklaration schreiben.

  Es wäre wahrscheinlich gut, sich in Integritätsbedingungen nur auf zuvor deklarierte Spalten zu beziehen, aber in PostgreSQL ist das tatsächlich nicht nötig. In SQLite schon.

- Der komplexe `CHECK`-Constraint, dass Fach einen Wert enthält gdw. `Art = 'A'`, kann auch so aufgeschrieben werden:

```
CHECK((Art <> 'A' OR Fach IS NOT NULL) AND
      (Art =  'A' OR Fach IS NULL))
```

4. Übungsblatt, Aufgabe 2: CREATE TABLE (9)

- Die Tabelle „Voraussetzungen“ sieht so aus:

Voraussetzungen		
zuerst	danach	Studienleistung
INF.00677.09	INF.06483.05	X
INF.00677.09	INF.06484.03	
INF.06483.05	INF.06484.03	

- Diese Beispiel-Zeilen bedeuten:
  - Die Studienleistung von „Objektorientierte Programmierung“ ist Voraussetzung für „Einführung in Datenbanken“.
  - Das Modul „Objektorientierte Programmierung“ (also mit Prüfung) ist Vorausetzung für „Datenbank-Programmierung“.
  - Das Modul „Einführung in Datenbanken“ (wieder mit Prüfung) ist Voraussetzung für „Datenbank-Programmierung“.

4. Übungsblatt, Aufgabe 2: CREATE TABLE (10)

- Schreiben Sie eine `CREATE TABLE`-Anweisung für diese Tabelle:
  - Die Spalten „`zuerst`“ und „`danach`“ sind jeweils ein Fremdschlüssel, der die Tabelle „`Module`“ referenziert.
  - Die Kombination aus „`zuerst`“ und „`danach`“ ist Primärschlüssel dieser Tabelle.
  - In der Spalte „`Studienleistung`“ sind nur die folgenden beiden Werte erlaubt:
    - „`X`“, wenn nur die Studienleistung vorausgesetzt wird, und
    - „ “ (ein Leerzeichen) für den Normalfall, dass das ganze Modul abgeschlossen sein muss.
- Alle drei Spalten erlauben keine Nullwerte.

4. Übungsblatt, Aufgabe 2: CREATE TABLE (11)

Lösung:

- Tabellendeklaration in SQL:

```
CREATE TABLE Voraussetzungen(
        zuerst CHAR(12) NOT NULL,
        danach CHAR(12) NOT NULL,
        Studienleistung CHAR(1) NOT NULL,
        PRIMARY KEY (zuerst, danach),
        FOREIGN KEY (zuerst)
                REFERENCES Module(ID),
        FOREIGN KEY (danach)
                REFERENCES Module(ID),
        CHECK(Studienleistung = 'X' OR
              Studienleistung = ' ')
);
```

4. Übungsblatt, Aufgabe 2: CREATE TABLE (12)

- Geben Sie eine Textdatei ab mit Endung `.sql`, die die beiden `CREATE TABLE`-Anweisungen enthält.
- 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. Beim Adminer bekommen Sie einen Link „`Create Schema`“ angezeigt, wenn links kein Schema ausgewählt ist. Sie können aber auch den SQL-Befehl „`CREATE SCHEMA xyz`“ eingeben. Wählen Sie dann dieses Schema in der Auswahlbox links.

4. Übungsblatt, Aufgabe 2: CREATE TABLE (13)

- Anschließend können Sie Ihre `CREATE TABLE`-Statements testen.
- Wenn Sie wollen, können Sie Ihre Tabellen anschließend mit den Beispieldaten füllen, indem Sie die folgende Datei mit „Import“ im Adminer ausführen (oder den Inhalt mit Copy&Paste ins SQL-Fenster einfügen):

  [https://users.informatik.uni-halle.de/~brass/db24/homework/h4_insert.sql]

Inhalt

Präsenzaufgabe: Variablenbelegungen

- Wählen Sie das Schema „studentenaufgaben_public“ im Adminer:
  [https://dbs.informatik.uni-halle.de/edb?pgsql=db&username=student_gast&db=postgres&ns=]
- Aufgabe: Suchen Sie eine FROM-Klausel aus (gern auch mit mehreren Tupelvariablen), und eine WHERE-Bedingung, so dass

```
SELECT COUNT(*)
FROM   ____________________________
WHERE  ____________________________
```

genau den Wert 40 liefert.

Es wird so die Anzahl der Variablenbelegungen gezählt, die die WHERE-Bedingung erfüllen. Die Tabellen-Größen sind: AUFGABEN: 3, STUDENTEN: 4, BEWERTUNGEN: 8. Es gibt 5 Bewertungen mit ATYP = 'H' und 2 Aufgaben mit ATYP = 'H'. Mehrere Tupelvariablen über einer Tabelle sind möglich.

Beispiel-Datenbank

STUDENTEN

SID	VORNAME	NACHNAME	EMAIL
101	Lisa	Weiss	...
102	Michael	Grau	NULL
103	Daniel	Sommer	...
104	Iris	Winter	...

BEWERTUNGEN

SID	ATYP	ANR	PUNKTE
101	H	1	10
101	H	2	8
101	Z	1	12
102	H	1	9
102	H	2	9
102	Z	1	10
103	H	1	5
103	Z	1	7

AUFGABEN

ATYP	ANR	THEMA	MAXPT
H	1	ER	10
H	2	SQL	10
Z	1	SQL	14