Einführung in Datenbanken

Übung 6: Datenbank-Entwurf, Erste Join-Anfragen

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

Klarstellung zum Vorrechnen (1)

- Sie müssen bei dieser Vorlesung nicht eine bestimmte Anzahl von Malen vorrechnen, um die Studienleistung zu bekommen.

  Die Hauptbedingung sind 50% der Punkte für die Hausaufgaben.

  Allerdings ist auch eine aktive Mitarbeit in den Übungen verlangt.

- Sie müssen aber für mindestens 20% der Aufgaben erklären, dass Sie bereit wären, vorzurechnen (und das dann auch anständig machen, wenn Sie aufgerufen werden).

- Ich wurde darauf aufmerksam gemacht, dass ich in meiner Übungsgruppe zu selten Studierende tatsächlich zum Vorrechnen aufrufe.

  In der Übungsgruppe von Herrn Hinneburg ist es vermutlich anders.

Klarstellung zum Vorrechnen (2)

- Die Sache ist noch etwas experimentell: Es sollte auch ein Gegengewicht gegen das erlaubte Kopieren von Lösungen von ChatGPT und Mitstudierenden geschaffen werden.

  Vergessen Sie dabei aber die Quellenangabe nicht!

- Die Angabe zum Vorrechnen soll natürlich auch etwas bedeuten.

- Dann muss ich aber die Aufnahme bei der Übung stoppen.

  Eine Videoaufnahme von Übungen ist ohnehin etwas problematisch.
  An Übungen sollte man life teilnehmen. Die Aufzeichnung war nur für Notfälle gedacht, geht aber leider nicht gleichzeitig mit intensiverem Vorrechnen durch die Teilnehmer/Teilnehmerinnen.

Klarstellung zum Vorrechnen (3)

- Vergessen Sie also nicht, anzugeben, ob Sie bereit wären, vorzurechnen:
  - `--Vorrechnen:5`: Sehr gerne (auch bei Fehlern möglich).
  - `--Vorrechnen:4`: Kein Problem.
  - `--Vorrechnen:3`: Bei Bedarf (nicht gerne).
  - `--Vorrechnen:2`: Ungerne, aber wenn es sein muss.

    Dies reicht zum Erreichen der 20% Quote. Aber es muss voraussichtlich mal sein.
  - `--Vorrechnen:1`: Nein.

    Das wird respektiert. Je nach Vorbereitung des Übungsleiters werden Sie möglicherweise aufgerufen, können dann aber sagen, dass Sie diese Aufgabe nicht vorrechnen wollen.
  - `--Vorrechnen:0`: Ich werde nicht da sein.

Klarstellung zum Vorrechnen (4)

- Eventuell werden Ihre Lösungen für ein Forschungsprojekt verwendet (anonymisiert).
- Dafür ist besonders wichtig, dass insbesondere Angaben zur Quelle automatisch erkannt und entfernt werden können.

  Es sollen ja keinesfalls Namen oder andere identifizierbare Information in den SQL-Anfragen zurückbleiben, die für das Forschungsprojekt verwendet werden.
- Das vorschriebene Format ist:

```
--QUELLE: Weiss, Lisa
```

- Schreiben Sie Namen, Materikelnummern, Benutzeraccounts auch nicht irgendwie anders in Ihre SQL-Anfragen!
- Ziel ist, die Lehre und damit die Klausurergebnisse zu verbessern. Vorschläge sind immer willkommen.

Inhalt

Präsenzaufgabe: Schema für Koch-/Back-Rezepte

- Entwerfen Sie ein Schema für Rezepte (zum kochen/backen) und geben Sie es in der Kurznotation ab. Anforderungen:
  - Zu jedem Rezept muss eine Nummer, eine Bezeichnung, ein Schwierigkeitsgrad, und eine Dauer (der Zubereitung) gespeichert werden. Bezeichnungen sind nicht eindeutig.
  - 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.

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(`$\underline{\texttt{Nummer}} \rightarrow$ `Rezept,`
  $\underline{\texttt{Position}} \rightarrow$ `Schritt,`
  $\underline{\texttt{Name}} \rightarrow$ `Zutat, Menge)`

- `Schritt(`$\underline{\texttt{Position}}$`,` $\underline{\texttt{Nummer}}$`, ...)`
  `Zutat(`$\underline{\texttt{Name}}$`,` $\underline{\texttt{Nummer}}$`, ...)`

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

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

Logische Analyse von Anfragen (1)

1. ```
   SELECT VORNAME, NACHNAME
   FROM   STUDENTEN
   WHERE  VORNAME = 'Tim' OR VORNAME = 'Tina'
   ```

2. ```
   SELECT VORNAME, NACHNAME
   FROM   STUDENTEN
   WHERE  NOT(VORNAME <> 'Tim' OR VORNAME <> 'Tina')
   ```

Kreuzen Sie die erste zutreffende Antwort an:

A. Die Anfragen liefern immer die gleiche Antwort (äquivalent).

B. Bis auf Duplikate die gleiche Antwort.

C. Anfrage 1 liefert immer $\emptyset$ (inkonsistent).

D. Anfrage 2 liefert immer $\emptyset$ (inkonsistent).

E. Keine der Aussagen trifft zu.

Logische Analyse von Anfragen (2)

1.
```
SELECT S.SID -- Spaltenüberschrift ist SID
FROM   STUDENTEN S, BEWERTUNGEN B
WHERE  S.SID=B.SID AND B.ATYP='H' AND B.ANR=1
```

2.
```
SELECT SID
FROM   BEWERTUNGEN
WHERE  ANR = 1 AND ATYP = 'H'
```

Kreuzen Sie die erste zutreffende Antwort an:

A. Die Anfragen liefern immer die gleiche Antwort (äquivalent).

B. Bis auf Duplikate die gleiche Antwort.

C. Anfrage 1 liefert immer $\emptyset$ (inkonsistent).

D. Anfrage 2 liefert immer $\emptyset$ (inkonsistent).

E. Keine der Aussagen trifft zu.

Logische Analyse von Anfragen (3)

1.
```
SELECT B.SID
FROM   BEWERTUNGEN B, STUDENTEN S
WHERE  B.ATYP = 'H' AND B.ANR = 1
```

2.
```
SELECT SID
FROM   BEWERTUNGEN
WHERE  ANR = 1 AND ATYP = 'H'
```

Kreuzen Sie die erste zutreffende Antwort an:

A. Die Anfragen liefern immer die gleiche Antwort (äquivalent).

B. Bis auf Duplikate die gleiche Antwort.

C. Anfrage 1 liefert immer $\emptyset$ (inkonsistent).

D. Anfrage 2 liefert immer $\emptyset$ (inkonsistent).

E. Keine der Aussagen trifft zu.

Logische Analyse von Anfragen (4)

1. ```
   SELECT S.NACHNAME
   FROM   STUDENTEN S, BEWERTUNGEN B
   WHERE  S.VORNAME = 'Lisa'
   ```

2. ```
   SELECT NACHNAME
   FROM   STUDENTEN
   WHERE  VORNAME = 'Lisa'
   ```

Kreuzen Sie die erste zutreffende Antwort an:

A. Die Anfragen liefern immer die gleiche Antwort (äquivalent).

B. Bis auf Duplikate die gleiche Antwort.

C. Anfrage 1 liefert immer $\emptyset$ (inkonsistent).

D. Anfrage 2 liefert immer $\emptyset$ (inkonsistent).

E. Keine der Aussagen trifft zu.

Logische Analyse von Anfragen (5)

1.
```
SELECT B.SID, A.ANR AS NR
FROM   BEWERTUNGEN B, AUFGABEN A
WHERE  B.ATYP = A.ATYP AND A.ATYP = 'H'
AND    A.ANR = B.ANR AND B.PUNKTE = A.MAXPT
```

2.
```
SELECT Y.SID, X.ANR AS NR
FROM   AUFGABEN X, BEWERTUNGEN Y
WHERE  X.MAXPT = Y.PUNKTE
AND    X.ATYP='H' AND Y.ATYP='H' AND X.ANR=Y.ANR
```

Kreuzen Sie die erste zutreffende Antwort an:

A. Die Anfragen liefern immer die gleiche Antwort (äquivalent).
B. Bis auf Duplikate die gleiche Antwort.
C. Anfrage 1 liefert immer $\emptyset$ (inkonsistent).
D. Anfrage 2 liefert immer $\emptyset$ (inkonsistent).
E. Keine der Aussagen trifft zu.

Logische Analyse von Anfragen (6)

1.
```
SELECT A.ANR, A.THEMA
FROM   AUFGABEN A
WHERE  A.ATYP = 'H'
```

2.
```
SELECT DISTINCT A.ANR, A.THEMA
FROM   AUFGABEN A, BEWERTUNGEN B
WHERE  A.ATYP = B.ATYP AND B.ATYP = 'H'
```

Kreuzen Sie die erste zutreffende Antwort an:

A. Die Anfragen liefern immer die gleiche Antwort (äquivalent).

B. Bis auf Duplikate die gleiche Antwort.

C. Anfrage 1 liefert immer eine Teilmenge von Anfrage 2 ($\subseteq$).

D. Anfrage 1 liefert immer eine Obermenge von Anfrage 2 ($\supseteq$).

E. Keine der Aussagen trifft zu.

Logische Analyse von Anfragen (7)

1. ```
   SELECT VORNAME, NACHNAME
   FROM   STUDENTEN
   WHERE  VORNAME = 'Lisa'
   ```

2. ```
   SELECT 'Lisa' AS VORNAME, NACHNAME
   FROM   STUDENTEN
   WHERE  NOT 1=0 AND 'Lisa' = VORNAME
   ```

Kreuzen Sie die erste zutreffende Antwort an:

A. Die Anfragen liefern immer die gleiche Antwort (äquivalent).

B. Bis auf Duplikate die gleiche Antwort.

C. Anfrage 1 liefert immer eine Teilmenge von Anfrage 2 ($\subseteq$).

D. Anfrage 1 liefert immer eine Obermenge von Anfrage 2 ($\supseteq$).

E. Keine der Aussagen trifft zu.

Tupel-Variablen

- Welche Tupelvariablen werden hier deklariert?

```
SELECT ...
FROM   STUDENTEN S, BEWERTUNGEN
WHERE  ...
```

A. Nur die Tupelvariable `S`.

B. Die Tupelvariable `S` und eine anonyme Tupelvariable.

C. Die Tupelvariablen `S` und `BEWERTUNGEN`.

D. Die Tupelvariablen `STUDENTEN`, `S` und `BEWERTUNGEN`.

- Ist diese Anfrage syntaktisch korrekt (ja/nein)?

```
SELECT STUDENTEN.NACHNAME
FROM   STUDENTEN S
```

Inhalt

Aufgabe 5.1: DB-Entwurf (1)

Aufgabe:

- Wählen Sie einen Fachvortrag vom Informatik-Industrietag, der am 14.11.2023 von 14 bis 18 Uhr in der Informatik stattfindet.

  Die Vorträge sind in der Zeit der EDB-Übung von 14 bis 16 Uhr.

- Modellieren Sie eine im gewählten Vortrag beschriebene Anwendung, einen relevanten Teil, oder eine Beispiel-Anwendung aus dem Vortrag als relationales Datenbankschema mit etwa 4 Tabellen.

Aufgabe 5.1: DB-Entwurf (2)

Aufgabe, Forts.:

- 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 12–15 Tupel umfassen.
- Beschreiben Sie zum Schluss knapp und verständlich, was die Attribute der Tabellen bedeuten sollen.
- Beschreiben Sie weiterhin die Anwendungsprozesse, die die Daten in die Tabellen einfügen, lesen und ggf. aktualisieren.

Aufgabe 5.1: DB-Entwurf (3)

Vorträge:

- Thomas Schmid (Uni Halle):
  Deep Learning zur Anomalie-Erkennung in Getreide-Saatgut.
- Christoph Hillmann (PROLOGA GmbH):
  Automatische Tourenplanung versus menschliche Arbeitsweise — Die Kunst, Algorithmen und menschliche Arbeitsweise in Einklang zu bringen.
- Michael König (ipoque GmbH, a Rohde&Schwarz Company):
  Cyber Security and Network Analytics by ipoque.
- Maik Boltze und Alexander Kampf (Relaxdays GmbH):
  CI/CD Pipelines — Deployments alle 2 Minuten.

Inhalt

EMP-DEPT-Datenbank (1)

- Klassische Beispiel-Datenbank von Oracle.
- Schema „empdept_public“ im Adminer:
  - dept(deptno, dname, loc)
  - emp(empno, ename, job, mgr°→emp, hiredate, sal, comm°, deptno°→dept)
- dept: „Department“ (Abteilungen einer Firma)
- emp: „Employee“ (Angestellte der Firma)
- Der Link zum Adminer ist:

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

EMP-DEPT-Datenbank (2)

- Die Tabelle dept hat die Spalten
  - deptno: Abteilungsnummer (Department Number),
  - dname: Name der Abteilung (Department Name),
  - loc: Ort/Sitz der Abteilung (Location):

DEPT		
DEPTNO	DNAME	LOC
10	ACCOUNTING	NEW YORK
20	RESEARCH	DALLAS
30	SALES	CHICAGO
40	OPERATIONS	BOSTON

EMP-DEPT-Datenbank (3)

- Die Tabelle `emp` hat folgende Spalten:
  - `empno`: Angestellten-Nummer (identifiziert den Angestellten)
  - `ename`: Name des Angestellten.
  - `job`: Berufsbezeichnung des Angestellten.
  - `mgr`: Angestellten-Nummer des direkten Vorgesetzten.

    `mgr` von „manager“.
  - `hiredate`: Datum der Einstellung.
  - `sal`: Gehalt des Angestellten („salary“).
  - `comm`: Provision (nur für Verkäufer) („commission“).
  - `deptno`: Abteilung des Angestellten.

EMP-DEPT-Datenbank (4)

EMP					
EMPNO	ENAME	JOB	MGR	SAL	DEPTNO
7369	SMITH	CLERK	7902	800	NULL
7499	ALLEN	SALESMAN	7698	1600	30
7521	WARD	SALESMAN	7698	1250	30
7566	JONES	MANAGER	7839	2975	20
7654	MARTIN	SALESMAN	7698	1250	30
7698	BLAKE	MANAGER	7839	2850	30
7782	CLARK	MANAGER	7839	2450	10
7788	SCOTT	ANALYST	7566	3000	20
7839	KING	PRESIDENT		5000	10
7844	TURNER	SALESMAN	7698	1500	30
7876	ADAMS	CLERK	7788	1100	20
7900	JAMES	CLERK	7698	950	30
7902	FORD	ANALYST	7566	3000	20
7934	MILLER	CLERK	7782	1300	10

Aufgabe 5.2: Angestellte in Dallas (1)

Aufgabe:

- Geben Sie den Namen und den Job der Angestellten aus,
  - die mehr als 1400 Dollar verdienen und
  - in einer Abteilung in DALLAS arbeiten.

Schema:

- DEPT(DEPTNO, DNAME, LOC)
- EMP(EMPNO, ENAME, JOB, MGR°→EMP, HIREDATE, SAL, COMM°, DEPTNO°→DEPT)

Aufgabe 5.2: Angestellte in Dallas (2)

- Mögliche Lösung:

```
SELECT E.ENAME, E.JOB
FROM   EMP E, DEPT D
WHERE  E.DEPTNO = D.DEPTNO
AND    E.SAL > 1400
AND    D.LOC = 'DALLAS'
```

- Ergebnis:

ename	job
JONES	MANAGER
SCOTT	ANALYST
FORD	ANALYST

Aufgabe 5.3: Disjunktionen (1)

Aufgabe:

- Geben Sie den Namen und das Einstellungsdatum der Angestellten aus,
  - die Analysten oder Manager sind und
  - deren Name mit J oder F beginnt und
  - in einer RESEARCH-Abteilung beschäftigt sind.

Schema:

- DEPT(DEPTNO, DNAME, LOC)
- EMP(EMPNO, ENAME, JOB, MGR°→EMP, HIREDATE, SAL, COMM°, DEPTNO°→DEPT)

Aufgabe 5.3: Disjunktionen (2)

- Mögliche Lösung:

```
SELECT E.ENAME, E.HIREDATE
FROM   EMP E, DEPT D
WHERE  E.DEPTNO = D.DEPTNO
AND    (E.JOB = 'ANALYST' OR E.JOB = 'MANAGER')
AND    (E.ENAME LIKE 'J%' OR E.ENAME LIKE 'F%')
AND    D.DNAME = 'RESEARCH'
```

- Ergebnis:

ename	hiredate
JONES	1981-04-02
FORD	1981-12-03

Aufgabe 5.4: Selbstverbund (1)

Aufgabe:

- Welche Jobs haben die Angestellten,
  - deren Manager in einer SALES-Abteilung arbeitet?

Schema:

- DEPT(DEPTNO, DNAME, LOC)
- EMP(EMPNO, ENAME, JOB, MGR$^\circ \rightarrow$EMP, HIREDATE, SAL, COMM$^\circ$, DEPTNO$^\circ \rightarrow$DEPT)

Aufgabe 5.4: Selbstverbund (2)

- Mögliche Lösung:

```
SELECT DISTINCT E.JOB
FROM   EMP E, EMP M, DEPT D
WHERE  E.MGR = M.EMPNO
AND    M.DEPTNO = D.DEPTNO
AND    D.DNAME = 'SALES'
```

- Ergebnis:

job
CLERK
SALESMAN

Aufgabe 5.5: Einfacher Verbund (1)

Aufgabe:

- Welche Angestellten arbeiten in Abteilungen für ACCOUNTING?
- Geben Sie die Personalnummer, den Namen und den Arbeitsort aus.

Schema:

- DEPT(DEPTNO, DNAME, LOC)
- EMP(EMPNO, ENAME, JOB, MGR°→EMP, HIREDATE, SAL, COMM°, DEPTNO°→DEPT)

Aufgabe 5.5: Einfacher Verbund (2)

- Mögliche Lösung:

```
SELECT E.EMPNO, E.ENAME, D.LOC
FROM   EMP E, DEPT D
WHERE  E.DEPTNO = D.DEPTNO
AND    D.DNAME = 'ACCOUNTING'
```

- Ergebnis:

empno	ename	loc
7782	CLARK	NEW YORK
7839	KING	NEW YORK
7934	MILLER	NEW YORK

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 150 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 Bewertungen mit SID = 103.

Tipp: Mehrere Tupelvariablen über einer Tabelle sind möglich.