合肥大學
HEFEI UNIVERSITY

Datenbanken

41. Logisches Schema: 1. Normalform

Thomas Weise (汤卫思)
tweise@hfuu.edu.cn

Institute of Applied Optimization (IAO)
School of Artificial Intelligence and Big Data
Hefei University
Hefei, Anhui, China

应用优化研究所
人工智能与大数据学院
合肥大学
中国安徽省合肥市

Version: 2025-11-28

Databases

Dies ist ein Kurs über Datenbanken an der Universität Hefei (合肥大学).

Die Webseite mit dem Lehrmaterial dieses Kurses ist https://thomasweise.github.io/databases (siehe auch den QR-Kode unten rechts). Dort können Sie das Kursbuch (in Englisch) und diese Slides finden. Das Repository mit den Beispielen finden Sie unter https://github.com/thomasWeise/databasesCode.

Outline

Einleitung

Einleitung

- Das logische Schema einer Datenbank zu entwerfen bedeutet nicht nur, das wir ein konzeptuelles Schema nach SQL transformieren.

Einleitung

- Das logische Schema einer Datenbank zu entwerfen bedeutet nicht nur, das wir ein konzeptuelles Schema nach SQL transformieren.
- Wir können nun zwar alle Elemente eines konzeptuellen ERDs in ein logisches Modell übersetzen...

Einleitung

- Das logische Schema einer Datenbank zu entwerfen bedeutet nicht nur, das wir ein konzeptuelles Schema nach SQL transformieren.
- Wir können nun zwar alle Elemente eines konzeptuellen ERDs in ein logisches Modell übersetzen. . . . . das heißt aber noch nicht, dass dieses logische Modell dann auch *gut* ist.
- Um das zu schaffen, sollten wir mehrere Richtlinien und Best Practices befolgen.

Einleitung

- Das logische Schema einer Datenbank zu entwerfen bedeutet nicht nur, das wir ein konzeptuelles Schema nach SQL transformieren.
- Wir können nun zwar alle Elemente eines konzeptuellen ERDs in ein logisches Modell übersetzen. . . . . das heißt aber noch nicht, dass dieses logische Modell dann auch *gut* ist.
- Um das zu schaffen, sollten wir mehrere Richtlinien und Best Practices befolgen.
- Einige der wichtigsten davon betreffen *Normalisierung* (EN: *normalization*)[26,31].

Normalisierung

- Normalisierung ist ein Prozess, der darauf abziehlt, Redundanz zu verringern und Inkonsistenzen sowie Anomalitäten zu vermeiden[62,63].

Normalisierung

- Normalisierung ist ein Prozess, der darauf abziehlt, Redundanz zu verringern und Inkonsistenzen sowie Anomalitäten zu vermeiden[62,63].
- Normalisierung tauscht dafür langsame Auslesegeschwindigkeit ein.

Normalisierung

- Normalisierung ist ein Prozess, der darauf abziehlt, Redundanz zu verringern und Inkonsistenzen sowie Anomalitäten zu vermeiden[62,63].
- Normalisierung tauscht dafür langsame Auslesegeschwindigkeit ein:
- Daten, die un-normalisiert in einer einzigen Tabelle gespeichert werden könnten müssen stattdessen über `INNER JOIN`s und ähnliche Konstrukte aus mehreren Tabellen (in normalisierter Form) zusammengesetzt werden[40].

Normalisierung

- Normalisierung ist ein Prozess, der darauf abziehlt, Redundanz zu verringern und Inkonsistenzen sowie Anomalitäten zu vermeiden[62,63].
- Normalisierung tauscht dafür langsame Auslesegeschwindigkeit ein:
- Daten, die un-normalisiert in einer einzigen Tabelle gespeichert werden könnten müssen stattdessen über `INNER JOIN`s und ähnliche Konstrukte aus mehreren Tabellen (in normalisierter Form) zusammengesetzt werden[40].
- Die Frage, welche Daten normalisiert werden sollten und zu welchem Grad muss also immer unter Berücksichtigung der Performanz beantwortet werden[40].

Normalisierung

- Normalisierung ist ein Prozess, der darauf abziehlt, Redundanz zu verringern und Inkonsistenzen sowie Anomalitäten zu vermeiden[62,63].
- Normalisierung tauscht dafür langsame Auslesegeschwindigkeit ein:
- Daten, die un-normalisiert in einer einzigen Tabelle gespeichert werden könnten müssen stattdessen über `INNER JOIN`s und ähnliche Konstrukte aus mehreren Tabellen (in normalisierter Form) zusammengesetzt werden[40].
- Die Frage, welche Daten normalisiert werden sollten und zu welchem Grad muss also immer unter Berücksichtigung der Performanz beantwortet werden[40].
- Es gibt mehrere *Normalformen* (EN: *normal forms*) (NF).

Normalisierung

- Normalisierung ist ein Prozess, der darauf abziehlt, Redundanz zu verringern und Inkonsistenzen sowie Anomalitäten zu vermeiden[62,63].
- Normalisierung tauscht dafür langsame Auslesegeschwindigkeit ein:
- Daten, die un-normalisiert in einer einzigen Tabelle gespeichert werden könnten müssen stattdessen über `INNER JOIN`s und ähnliche Konstrukte aus mehreren Tabellen (in normalisierter Form) zusammengesetzt werden[40].
- Die Frage, welche Daten normalisiert werden sollten und zu welchem Grad muss also immer unter Berücksichtigung der Performanz beantwortet werden[40].
- Es gibt mehrere *Normalformen* (EN: *normal forms*) (NF).
- Wir betrachten hier die 1NF, die 2NF und die 3NF.

Normalisierung

- Normalisierung ist ein Prozess, der darauf abziehlt, Redundanz zu verringern und Inkonsistenzen sowie Anomalitäten zu vermeiden[62,63].
- Normalisierung tauscht dafür langsame Auslesegeschwindigkeit ein:
- Daten, die un-normalisiert in einer einzigen Tabelle gespeichert werden könnten müssen stattdessen über `INNER JOIN`s und ähnliche Konstrukte aus mehreren Tabellen (in normalisierter Form) zusammengesetzt werden[40].
- Die Frage, welche Daten normalisiert werden sollten und zu welchem Grad muss also immer unter Berücksichtigung der Performanz beantwortet werden[40].
- Es gibt mehrere *Normalformen* (EN: *normal forms*) (NF).
- Wir betrachten hier die 1NF, die 2NF und die 3NF.
- Höhere Normalformen sind immer mehr restriktiv als niedrige Normalformen.

Normalisierung

- Normalisierung ist ein Prozess, der darauf abziehlt, Redundanz zu verringern und Inkonsistenzen sowie Anomalitäten zu vermeiden[62,63].
- Normalisierung tauscht dafür langsame Auslesegeschwindigkeit ein:
- Daten, die un-normalisiert in einer einzigen Tabelle gespeichert werden könnten müssen stattdessen über `INNER JOIN`s und ähnliche Konstrukte aus mehreren Tabellen (in normalisierter Form) zusammengesetzt werden[40].
- Die Frage, welche Daten normalisiert werden sollten und zu welchem Grad muss also immer unter Berücksichtigung der Performanz beantwortet werden[40].
- Es gibt mehrere *Normalformen* (EN: *normal forms*) (NF).
- Wir betrachten hier die 1NF, die 2NF und die 3NF.
- Höhere Normalformen sind immer mehr restriktiv als niedrige Normalformen.
- Wenn ein Teil eines logischen Modells in einer höheren Normalform ist, dann ist es automatisch auch in allen niedrigeren Normalformen.

1. Normalform

1. Normalform

- Die 1. Normalform (1NF) geht zurück bis auf Codd's Paper [22] in dem er 1970 das relationale Datenmodell präsentierte.

1. Normalform

- Die 1. Normalform (1NF) geht zurück bis auf Codd's Paper [22] in dem er 1970 das relationale Datenmodell präsentierte.
- In Grunde verlangt sie nur, dass das Design der Tabellen dem relationalen Datenmodell ordentlich folgen soll.

1. Normalform

- Die 1. Normalform (1NF) geht zurück bis auf Codd's Paper [22] in dem er 1970 das relationale Datenmodell präsentierte.
- In Grunde verlangt sie nur, dass das Design der Tabellen dem relationalen Datenmodell ordentlich folgen soll.

Definition: 1. Normalform

Unter der 1NF müssen alle Zeilen einer Tabelle die selbe Anzahl Spalten haben und alle Spalten müssen unteilbar sein.

1. Normalform

- Die 1. Normalform (1NF) geht zurück bis auf Codd's Paper [22] in dem er 1970 das relationale Datenmodell präsentierte.
- In Grunde verlangt sie nur, dass das Design der Tabellen dem relationalen Datenmodell ordentlich folgen soll.

Definition: 1. Normalform

Unter der 1NF müssen alle Zeilen einer Tabelle die selbe Anzahl Spalten haben und alle Spalten müssen unteilbar sein.

- Das schließt also mehrwertige und zusammengesetzte Attribute aus.

1. Normalform

- Die 1. Normalform (1NF) geht zurück bis auf Codd's Paper [22] in dem er 1970 das relationale Datenmodell präsentierte.
- In Grunde verlangt sie nur, dass das Design der Tabellen dem relationalen Datenmodell ordentlich folgen soll.

Definition: 1. Normalform

Unter der 1NF müssen alle Zeilen einer Tabelle die selbe Anzahl Spalten haben und alle Spalten müssen unteilbar sein.

- Das schließt also mehrwertige und zusammengesetzte Attribute aus.
- Wie wir bereits gesagt haben, werden diese im relationalen Modell sowieso nicht unterstützt.

1. Normalform

- Die 1. Normalform (1NF) geht zurück bis auf Codd's Paper [22] in dem er 1970 das relationale Datenmodell präsentierte.
- In Grunde verlangt sie nur, dass das Design der Tabellen dem relationalen Datenmodell ordentlich folgen soll.

Definition: 1. Normalform

Unter der 1NF müssen alle Zeilen einer Tabelle die selbe Anzahl Spalten haben und alle Spalten müssen unteilbar sein.

- Das schließt also mehrwertige und zusammengesetzte Attribute aus.
- Wie wir bereits gesagt haben, werden diese im relationalen Modell sowieso nicht unterstützt.
- Warum gibt es dann diese Normalform überhaupt?

1. Normalform

- Die 1. Normalform (1NF) geht zurück bis auf Codd's Paper [22] in dem er 1970 das relationale Datenmodell präsentierte.
- In Grunde verlangt sie nur, dass das Design der Tabellen dem relationalen Datenmodell ordentlich folgen soll.

Definition: 1. Normalform

Unter der 1NF müssen alle Zeilen einer Tabelle die selbe Anzahl Spalten haben und alle Spalten müssen unteilbar sein.

- Das schließt also mehrwertige und zusammengesetzte Attribute aus.
- Wie wir bereits gesagt haben, werden diese im relationalen Modell sowieso nicht unterstützt.
- Warum gibt es dann diese Normalform überhaupt?
- Wenn sie von allen Tabellen sowieso erfüllt wird, dann brauchen wir sie auch nicht diskutieren.

1. Normalform kann verletzt werden

- Wir haben ja bereits gelernt, wie man Entitätstypen des konzeptuellen Schemas in Tabellen im logischen Schema übersetzt.

1. Normalform kann verletzt werden

- Wir haben ja bereits gelernt, wie man Entitätstypen des konzeptuellen Schemas in Tabellen im logischen Schema übersetzt.
- Wir haben gesagt, dass mehrwertge Attribute eigentständige Tabellen werden.

1. Normalform kann verletzt werden

- Wir haben ja bereits gelernt, wie man Entitätstypen des konzeptuellen Schemas in Tabellen im logischen Schema übersetzt.
- Wir haben gesagt, dass mehrwertge Attribute eigentständige Tabellen werden.
- Zusammengesetzte Attribute rekursiv in ihre unteilbaren Komponenten heruntergebrochen, die dann zu Spalten werden.

1. Normalform kann verletzt werden

- Wir haben ja bereits gelernt, wie man Entitätstypen des konzeptuellen Schemas in Tabellen im logischen Schema übersetzt.
- Wir haben gesagt, dass mehrwertge Attribute eigentständige Tabellen werden.
- Zusammengesetzte Attribute rekursiv in ihre unteilbaren Komponenten heruntergebrochen, die dann zu Spalten werden.
- Wenn wir das relationale Datenmodell verwenden, dann produzieren wir ganz natürlich logische Schemas in der 1NF.

1. Normalform kann verletzt werden

- Wir haben ja bereits gelernt, wie man Entitätstypen des konzeptuellen Schemas in Tabellen im logischen Schema übersetzt.
- Wir haben gesagt, dass mehrwertge Attribute eigentständige Tabellen werden.
- Zusammengesetzte Attribute rekursiv in ihre unteilbaren Komponenten heruntergebrochen, die dann zu Spalten werden.
- Wenn wir das relationale Datenmodell verwenden, dann produzieren wir ganz natürlich logische Schemas in der 1NF.
- Das stimmt aber nur, wenn wir mehrwertige und zusammengesetzte Attribute auch als solche *erkennen*.

1. Normalform kann verletzt werden

- Wir haben ja bereits gelernt, wie man Entitätstypen des konzeptuellen Schemas in Tabellen im logischen Schema übersetzt.
- Wir haben gesagt, dass mehrwertge Attribute eigentständige Tabellen werden.
- Zusammengesetzte Attribute rekursiv in ihre unteilbaren Komponenten heruntergebrochen, die dann zu Spalten werden.
- Wenn wir das relationale Datenmodell verwenden, dann produzieren wir ganz natürlich logische Schemas in der 1NF.
- Das stimmt aber nur, wenn wir mehrwertige und zusammengesetzte Attribute auch als solche *erkennen*.
- Wenn eine Tabelle Attribute hat, die von ihrer Bedeutung her zusammengesetzt sind, wir diese aber als einzelne Attribute implementieren, dann verletzen wir die 1NF.

1. Normalform kann verletzt werden

- Wir haben ja bereits gelernt, wie man Entitätstypen des konzeptuellen Schemas in Tabellen im logischen Schema übersetzt.
- Wir haben gesagt, dass mehrwertge Attribute eigentständige Tabellen werden.
- Zusammengesetzte Attribute rekursiv in ihre unteilbaren Komponenten heruntergebrochen, die dann zu Spalten werden.
- Wenn wir das relationale Datenmodell verwenden, dann produzieren wir ganz natürlich logische Schemas in der 1NF.
- Das stimmt aber nur, wenn wir mehrwertige und zusammengesetzte Attribute auch als solche *erkennen*.
- Wenn eine Tabelle Attribute hat, die von ihrer Bedeutung her zusammengesetzt sind, wir diese aber als einzelne Attribute implementieren, dann verletzen wir die 1NF.
- Wenn ein Attribut mehrwertig ist, wir aber versuchen, es durch mehrere Spalten zu speichern anstatt es in eine separate Tabelle zu tun, dann verletzen wir die 1NF.

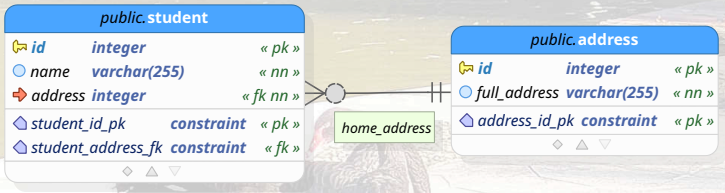
1. Normalform kann verletzt werden

- Wir haben ja bereits gelernt, wie man Entitätstypen des konzeptuellen Schemas in Tabellen im logischen Schema übersetzt.
- Wir haben gesagt, dass mehrwertge Attribute eigentständige Tabellen werden.
- Zusammengesetzte Attribute rekursiv in ihre unteilbaren Komponenten heruntergebrochen, die dann zu Spalten werden.
- Wenn wir das relationale Datenmodell verwenden, dann produzieren wir ganz natürlich logische Schemas in der 1NF.
- Das stimmt aber nur, wenn wir mehrwertige und zusammengesetzte Attribute auch als solche *erkennen*.
- Wenn eine Tabelle Attribute hat, die von ihrer Bedeutung her zusammengesetzt sind, wir diese aber als einzelne Attribute implementieren, dann verletzen wir die 1NF.
- Wenn ein Attribut mehrwertig ist, wir aber versuchen, es durch mehrere Spalten zu speichern anstatt es in eine separate Tabelle zu tun, dann verletzen wir die 1NF.
- Schauen wir uns mal an, was dann passiert.

Zusammengesetzte Attribute: Problem

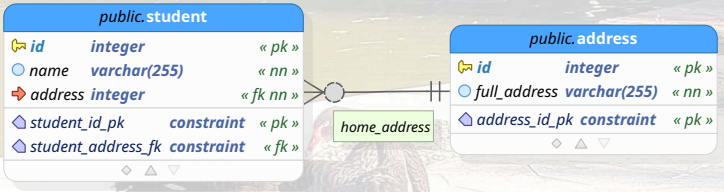
Verletzung der 1. Normalform durch Zusammengesetze Attribute

- Hier zeigen wir einen Tail eines logischen Modells, dass Studenten- und Addressdatensätze verbindet.



Verletzung der 1. Normalform durch Zusammengesetze Attribute

- Hier zeigen wir einen Tail eines logischen Modells, dass Studenten- und Addressdatensätze verbindet.
- Im Modell hat jeder Student genau eine Adresse und einen Namen.



Verletzung der 1. Normalform durch Zusammengesetze Attribute

- Hier zeigen wir einen Tail eines logischen Modells, dass Studenten- und Addressdatensätze verbindet.
- Im Modell hat jeder Student genau eine Adresse und einen Namen.
- Die Adressen sind als Text in einer eigenständigen Tabelle gespeichert.

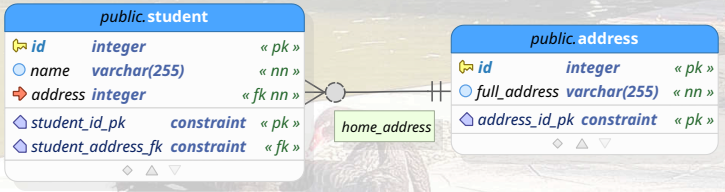


Tabelle address

- Hier sehen wir das vom PgModeler generierte Skript zum Erstellen der Tabelle `address`.

```
-- object: public.address | type: TABLE --
-- DROP TABLE IF EXISTS public.address CASCADE;
CREATE TABLE public.address (
    id integer NOT NULL GENERATED BY DEFAULT AS IDENTITY ,
    full_address varchar(255) NOT NULL,
    CONSTRAINT address_id_pk PRIMARY KEY (id)
);
-- ddl-end --
ALTER TABLE public.address OWNER TO postgres;
-- ddl-end --
```

```
$ psql "postgres://postgres:XXX@localhost/anomalies" -v ON_ERROR_STOP=1
    ↪ -ebf 03_public_address_table_5071.sql
CREATE TABLE
ALTER TABLE
# psql 16.11 succeeded with exit code 0.
```

Tabelle address

- Hier sehen wir das vom PgModeler generierte Skript zum Erstellen der Tabelle `address`.
- Die Tabelle hat zwei Spalten, nämlich den Ersatz-Primärschlüssel `id` und die Spalte `full_address`, welche variabel-langen Text von bis zu 255 Zeichen beinhaltet.

```
1  -- object: public.address | type: TABLE --
2  -- DROP TABLE IF EXISTS public.address CASCADE;
3  CREATE TABLE public.address (
4      id integer NOT NULL GENERATED BY DEFAULT AS IDENTITY ,
5      full_address varchar(255) NOT NULL,
6      CONSTRAINT address_id_pk PRIMARY KEY (id)
7  );
8  -- ddl-end --
9  ALTER TABLE public.address OWNER TO postgres;
10 -- ddl-end --
```

```
1  $ psql "postgres://postgres:XXX@localhost/anomalies" -v ON_ERROR_STOP=1
     ↪ -ebf 03_public_address_table_5071.sql
2  CREATE TABLE
3  ALTER TABLE
4  # psql 16.11 succeeded with exit code 0.
```

Tabelle address

- Hier sehen wir das vom PgModeler generierte Skript zum Erstellen der Tabelle `address`.
- Die Tabelle hat zwei Spalten, nämlich den Ersatz-Primärschlüssel `id` und die Spalte `full_address`, welche variabel-langen Text von bis zu 255 Zeichen beinhaltet.
- Wie der Name schon sagt speichert diese Tabelle die Adressen der Studenten.

```
-- object: public.address | type: TABLE --
-- DROP TABLE IF EXISTS public.address CASCADE;
CREATE TABLE public.address (
    id integer NOT NULL GENERATED BY DEFAULT AS IDENTITY ,
    full_address varchar(255) NOT NULL,
    CONSTRAINT address_id_pk PRIMARY KEY (id)
);
-- ddl-end --
ALTER TABLE public.address OWNER TO postgres;
-- ddl-end --
```

```
$ psql "postgres://postgres:XXX@localhost/anomalies" -v ON_ERROR_STOP=1
    ↪ -ebf 03_public_address_table_5071.sql
CREATE TABLE
ALTER TABLE
# psql 16.11 succeeded with exit code 0.
```

Tabelle student

- Hier sehen wir das vom PgModeler generierte Skript zum Erstellen der Tabelle `student`.

```
-- object: public.student | type: TABLE --
-- DROP TABLE IF EXISTS public.student CASCADE;
CREATE TABLE public.student (
    id integer NOT NULL GENERATED BY DEFAULT AS IDENTITY ,
    name varchar(255) NOT NULL,
    address integer NOT NULL,
    CONSTRAINT student_id_pk PRIMARY KEY (id)
);
-- ddl-end --
ALTER TABLE public.student OWNER TO postgres;
-- ddl-end --
```

```
$ psql "postgres://postgres:XXX@localhost/anomalies" -v ON_ERROR_STOP=1
    ↪ -ebf 04_public_student_table_5075.sql
CREATE TABLE
ALTER TABLE
# psql 16.11 succeeded with exit code 0.
```

Tabelle student

- Hier sehen wir das vom PgModeler generierte Skript zum Erstellen der Tabelle `student`.
- Auch sie hat einen Ersatz-Primärschlüssel `id`.

```
-- object: public.student | type: TABLE --
-- DROP TABLE IF EXISTS public.student CASCADE;
CREATE TABLE public.student (
    id integer NOT NULL GENERATED BY DEFAULT AS IDENTITY ,
    name varchar(255) NOT NULL,
    address integer NOT NULL,
    CONSTRAINT student_id_pk PRIMARY KEY (id)
);
-- ddl-end --
ALTER TABLE public.student OWNER TO postgres;
-- ddl-end --
```

```
$ psql "postgres://postgres:XXX@localhost/anomalies" -v ON_ERROR_STOP=1
    ↪ -ebf 04_public_student_table_5075.sql
CREATE TABLE
ALTER TABLE
# psql 16.11 succeeded with exit code 0.
```

Tabelle student

- Hier sehen wir das vom PgModeler generierte Skript zum Erstellen der Tabelle `student`.
- Auch sie hat einen Ersatz-Primärschlüssel `id`.
- Sie hat auch die Spalte `name` um die Namen der Studenten zu speichern.

```
-- object: public.student | type: TABLE --
-- DROP TABLE IF EXISTS public.student CASCADE;
CREATE TABLE public.student (
    id integer NOT NULL GENERATED BY DEFAULT AS IDENTITY ,
    name varchar(255) NOT NULL,
    address integer NOT NULL,
    CONSTRAINT student_id_pk PRIMARY KEY (id)
);
-- ddl-end --
ALTER TABLE public.student OWNER TO postgres;
-- ddl-end --
```

```
$ psql "postgres://postgres:XXX@localhost/anomalies" -v ON_ERROR_STOP=1
    ↪ -ebf 04_public_student_table_5075.sql
CREATE TABLE
ALTER TABLE
# psql 16.11 succeeded with exit code 0.
```

Tabelle student

- Hier sehen wir das vom PgModeler generierte Skript zum Erstellen der Tabelle `student`.
- Auch sie hat einen Ersatz-Primärschlüssel `id`.
- Sie hat auch die Spalte `name` um die Namen der Studenten zu speichern.
- Die Spalte `address` hat einen Fremdschlüssel auf die Spalte `id` der Tabelle `address`.

```
-- object: public.student | type: TABLE --
-- DROP TABLE IF EXISTS public.student CASCADE;
CREATE TABLE public.student (
    id integer NOT NULL GENERATED BY DEFAULT AS IDENTITY ,
    name varchar(255) NOT NULL,
    address integer NOT NULL,
    CONSTRAINT student_id_pk PRIMARY KEY (id)
);
-- ddl-end --
ALTER TABLE public.student OWNER TO postgres;
-- ddl-end --
```

```
$ psql "postgres://postgres:XXX@localhost/anomalies" -v ON_ERROR_STOP=1
    ↪ -ebf 04_public_student_table_5075.sql
CREATE TABLE
ALTER TABLE
# psql 16.11 succeeded with exit code 0.
```

Tabelle student

- Hier sehen wir das vom PgModeler generierte Skript zum Erstellen der Tabelle `student`.
- Auch sie hat einen Ersatz-Primärschlüssel `id`.
- Sie hat auch die Spalte `name` um die Namen der Studenten zu speichern.
- Die Spalte `address` hat einen Fremdschlüssel auf die Spalte `id` der Tabelle `address`.
- Dieser wird über eine `REFERENCES`-Einschränkung gesichert.

```
1  -- object: public.student | type: TABLE --
2  -- DROP TABLE IF EXISTS public.student CASCADE;
3  CREATE TABLE public.student (
4      id integer NOT NULL GENERATED BY DEFAULT AS IDENTITY ,
5      name varchar(255) NOT NULL,
6      address integer NOT NULL,
7      CONSTRAINT student_id_pk PRIMARY KEY (id)
8  );
9  -- ddl-end --
10 ALTER TABLE public.student OWNER TO postgres;
11 -- ddl-end --
```

```
1  $ psql "postgres://postgres:XXX@localhost/anomalies" -v ON_ERROR_STOP=1
       ↪ -ebf 04_public_student_table_5075.sql
2  CREATE TABLE
3  ALTER TABLE
4  # psql 16.11 succeeded with exit code 0.
```

Tabelle student

- Hier sehen wir das vom PgModeler generierte Skript zum Erstellen der Tabelle `student`.
- Auch sie hat einen Ersatz-Primärschlüssel `id`.
- Sie hat auch die Spalte `name` um die Namen der Studenten zu speichern.
- Die Spalte `address` hat einen Fremdschlüssel auf die Spalte `id` der Tabelle `address`.
- Dieser wird über eine `REFERENCES`-Einschränkung gesichert.
- Die Skripte dafür und zum Erstellen bzw. später zum Löschen der Datenbank lassen wir hier aber weg.

```
-- object: public.student | type: TABLE --
-- DROP TABLE IF EXISTS public.student CASCADE;
CREATE TABLE public.student (
    id integer NOT NULL GENERATED BY DEFAULT AS IDENTITY ,
    name varchar(255) NOT NULL,
    address integer NOT NULL,
    CONSTRAINT student_id_pk PRIMARY KEY (id)
);
-- ddl-end --
ALTER TABLE public.student OWNER TO postgres;
-- ddl-end --
```

```
$ psql "postgres://postgres:XXX@localhost/anomalies" -v ON_ERROR_STOP=1
    ↪ -ebf 04_public_student_table_5075.sql
CREATE TABLE
ALTER TABLE
# psql 16.11 succeeded with exit code 0.
```

Daten Einfügen

- Jetzt fügen wir Daten in die Datenbank ein.

```
/** Insert data into the database. */

-- Insert several address records.
INSERT INTO address (full_address) VALUES
    ('Hefei University, Hefei 230601 Jinkaiqu, Hefei, Anhui, China'),
    ('Am Rathaus 1, 09111 Zentrum, Chemnitz, Sachsen, Deutschland'),
    ('Canal Street 4, Chinatown, New York, NY, USA'),
    ('West Street, Licheng District, Quanzhou 362002, Fujian, PRC');

-- Create the student records.
INSERT INTO student (name, address) VALUES
    ('Bibbo', 1), ('Bebbo', 2), ('Bibbi', 3), ('Babbo', 4);
```

```
$ psql "postgres://postgres:XXX@localhost/anomalies" -v ON_ERROR_STOP=1
  ↪ -ebf insert.sql
INSERT 0 4
INSERT 0 4
# psql 16.11 succeeded with exit code 0.
```

Daten Einfügen

- Jetzt fügen wir Daten in die Datenbank ein.
- Erstmal erstellen wir vier Addresse.

```
/** Insert data into the database. */

-- Insert several address records.
INSERT INTO address (full_address) VALUES
    ('Hefei University, Hefei 230601 Jinkaiqu, Hefei, Anhui, China'),
    ('Am Rathaus 1, 09111 Zentrum, Chemnitz, Sachsen, Deutschland'),
    ('Canal Street 4, Chinatown, New York, NY, USA'),
    ('West Street, Licheng District, Quanzhou 362002, Fujian, PRC');

-- Create the student records.
INSERT INTO student (name, address) VALUES
    ('Bibbo', 1), ('Bebbo', 2), ('Bibbi', 3), ('Babbo', 4);
```

```
$ psql "postgres://postgres:XXX@localhost/anomalies" -v ON_ERROR_STOP=1
  ↪ -ebf insert.sql
INSERT 0 4
INSERT 0 4
# psql 16.11 succeeded with exit code 0.
```

Daten Einfügen

- Jetzt fügen wir Daten in die Datenbank ein.
- Erstmal erstellen wir vier Addresse:
  1. eine in unserer Universität Hefei (合肥大学) in der schönen Stadt Hefei (合肥市) in China.

```sql
/** Insert data into the database. */

-- Insert several address records.
INSERT INTO address (full_address) VALUES
    ('Hefei University, Hefei 230601 Jinkaiqu, Hefei, Anhui, China'),
    ('Am Rathaus 1, 09111 Zentrum, Chemnitz, Sachsen, Deutschland'),
    ('Canal Street 4, Chinatown, New York, NY, USA'),
    ('West Street, Licheng District, Quanzhou 362002, Fujian, PRC');

-- Create the student records.
INSERT INTO student (name, address) VALUES
    ('Bibbo', 1), ('Bebbo', 2), ('Bibbi', 3), ('Babbo', 4);
```

```
$ psql "postgres://postgres:XXX@localhost/anomalies" -v ON_ERROR_STOP=1
  ↪ -ebf insert.sql
INSERT 0 4
INSERT 0 4
# psql 16.11 succeeded with exit code 0.
```

Daten Einfügen

- Jetzt fügen wir Daten in die Datenbank ein.
- Erstmal erstellen wir vier Addresse:
  1. eine in unserer Universität Hefei (合肥大学) in der schönen Stadt Hefei (合肥市) in China,
  2. eine in meiner Heimatstadt Chemnitz in Deutschland.

```sql
/** Insert data into the database. */

-- Insert several address records.
INSERT INTO address (full_address) VALUES
    ('Hefei University, Hefei 230601 Jinkaiqu, Hefei, Anhui, China'),
    ('Am Rathaus 1, 09111 Zentrum, Chemnitz, Sachsen, Deutschland'),
    ('Canal Street 4, Chinatown, New York, NY, USA'),
    ('West Street, Licheng District, Quanzhou 362002, Fujian, PRC');

-- Create the student records.
INSERT INTO student (name, address) VALUES
    ('Bibbo', 1), ('Bebbo', 2), ('Bibbi', 3), ('Babbo', 4);
```

```
$ psql "postgres://postgres:XXX@localhost/anomalies" -v ON_ERROR_STOP=1
  ↪ -ebf insert.sql
INSERT 0 4
INSERT 0 4
# psql 16.11 succeeded with exit code 0.
```

Daten Einfügen

- Jetzt fügen wir Daten in die Datenbank ein.
- Erstmal erstellen wir vier Addresse:
  1. eine in unserer Universität Hefei (合肥大学) in der schönen Stadt Hefei (合肥市) in China,
  2. eine in meiner Heimatstadt Chemnitz in Deutschland,
  3. eine in der Chinatown von New York, USA.

```
/** Insert data into the database. */

-- Insert several address records.
INSERT INTO address (full_address) VALUES
    ('Hefei University, Hefei 230601 Jinkaiqu, Hefei, Anhui, China'),
    ('Am Rathaus 1, 09111 Zentrum, Chemnitz, Sachsen, Deutschland'),
    ('Canal Street 4, Chinatown, New York, NY, USA'),
    ('West Street, Licheng District, Quanzhou 362002, Fujian, PRC');

-- Create the student records.
INSERT INTO student (name, address) VALUES
    ('Bibbo', 1), ('Bebbo', 2), ('Bibbi', 3), ('Babbo', 4);
```

```
$ psql "postgres://postgres:XXX@localhost/anomalies" -v ON_ERROR_STOP=1
  ↪ -ebf insert.sql
INSERT 0 4
INSERT 0 4
# psql 16.11 succeeded with exit code 0.
```

Daten Einfügen

- Jetzt fügen wir Daten in die Datenbank ein.
- Erstmal erstellen wir vier Addresse:
  1. eine in unserer Universität Hefei (合肥大学) in der schönen Stadt Hefei (合肥市) in China,
  2. eine in meiner Heimatstadt Chemnitz in Deutschland,
  3. eine in der Chinatown von New York, USA und
  4. eine in der Stadt Quanzhou (泉州市), China.

```
/** Insert data into the database. */

-- Insert several address records.
INSERT INTO address (full_address) VALUES
    ('Hefei University, Hefei 230601 Jinkaiqu, Hefei, Anhui, China'),
    ('Am Rathaus 1, 09111 Zentrum, Chemnitz, Sachsen, Deutschland'),
    ('Canal Street 4, Chinatown, New York, NY, USA'),
    ('West Street, Licheng District, Quanzhou 362002, Fujian, PRC');

-- Create the student records.
INSERT INTO student (name, address) VALUES
    ('Bibbo', 1), ('Bebbo', 2), ('Bibbi', 3), ('Babbo', 4);
```

```
$ psql "postgres://postgres:XXX@localhost/anomalies" -v ON_ERROR_STOP=1
  ↪ -ebf insert.sql
INSERT 0 4
INSERT 0 4
# psql 16.11 succeeded with exit code 0.
```

Daten Einfügen

- Jetzt fügen wir Daten in die Datenbank ein.
- Erstmal erstellen wir vier Addresse.
- Dann erstellen wir vier `student`-Datensätze, für Herrn Bibbo, Herrn Bebbo, Frau Bibbi und Herrn Bebbo.

```
/** Insert data into the database. */

-- Insert several address records.
INSERT INTO address (full_address) VALUES
    ('Hefei University, Hefei 230601 Jinkaiqu, Hefei, Anhui, China'),
    ('Am Rathaus 1, 09111 Zentrum, Chemnitz, Sachsen, Deutschland'),
    ('Canal Street 4, Chinatown, New York, NY, USA'),
    ('West Street, Licheng District, Quanzhou 362002, Fujian, PRC');

-- Create the student records.
INSERT INTO student (name, address) VALUES
    ('Bibbo', 1), ('Bebbo', 2), ('Bibbi', 3), ('Babbo', 4);
```

```
$ psql "postgres://postgres:XXX@localhost/anomalies" -v ON_ERROR_STOP=1
  ↪ -ebf insert.sql
INSERT 0 4
INSERT 0 4
# psql 16.11 succeeded with exit code 0.
```

Daten Einfügen

- Jetzt fügen wir Daten in die Datenbank ein.
- Erstmal erstellen wir vier Addresse.
- Dann erstellen wir vier `student`-Datensätze, für Herrn Bibbo, Herrn Bebbo, Frau Bibbi und Herrn Bebbo.
- Diese sind über ihre Fremdschlüssel mit genau den Adressen oben verbunden.

```sql
/** Insert data into the database. */

-- Insert several address records.
INSERT INTO address (full_address) VALUES
    ('Hefei University, Hefei 230601 Jinkaiqu, Hefei, Anhui, China'),
    ('Am Rathaus 1, 09111 Zentrum, Chemnitz, Sachsen, Deutschland'),
    ('Canal Street 4, Chinatown, New York, NY, USA'),
    ('West Street, Licheng District, Quanzhou 362002, Fujian, PRC');

-- Create the student records.
INSERT INTO student (name, address) VALUES
    ('Bibbo', 1), ('Bebbo', 2), ('Bibbi', 3), ('Babbo', 4);
```

```
$ psql "postgres://postgres:XXX@localhost/anomalies" -v ON_ERROR_STOP=1
  ↪ -ebf insert.sql
INSERT 0 4
INSERT 0 4
# psql 16.11 succeeded with exit code 0.
```

Daten Einfügen

- Jetzt fügen wir Daten in die Datenbank ein.
- Erstmal erstellen wir vier Addresse.
- Dann erstellen wir vier `student`-Datensätze, für Herrn Bibbo, Herrn Bebbo, Frau Bibbi und Herrn Bebbo.
- Diese sind über ihre Fremdschlüssel mit genau den Adressen oben verbunden.
- Das sieht erstmal gut aus.

```sql
/** Insert data into the database. */

-- Insert several address records.
INSERT INTO address (full_address) VALUES
    ('Hefei University, Hefei 230601 Jinkaiqu, Hefei, Anhui, China'),
    ('Am Rathaus 1, 09111 Zentrum, Chemnitz, Sachsen, Deutschland'),
    ('Canal Street 4, Chinatown, New York, NY, USA'),
    ('West Street, Licheng District, Quanzhou 362002, Fujian, PRC');

-- Create the student records.
INSERT INTO student (name, address) VALUES
    ('Bibbo', 1), ('Bebbo', 2), ('Bibbi', 3), ('Babbo', 4);
```

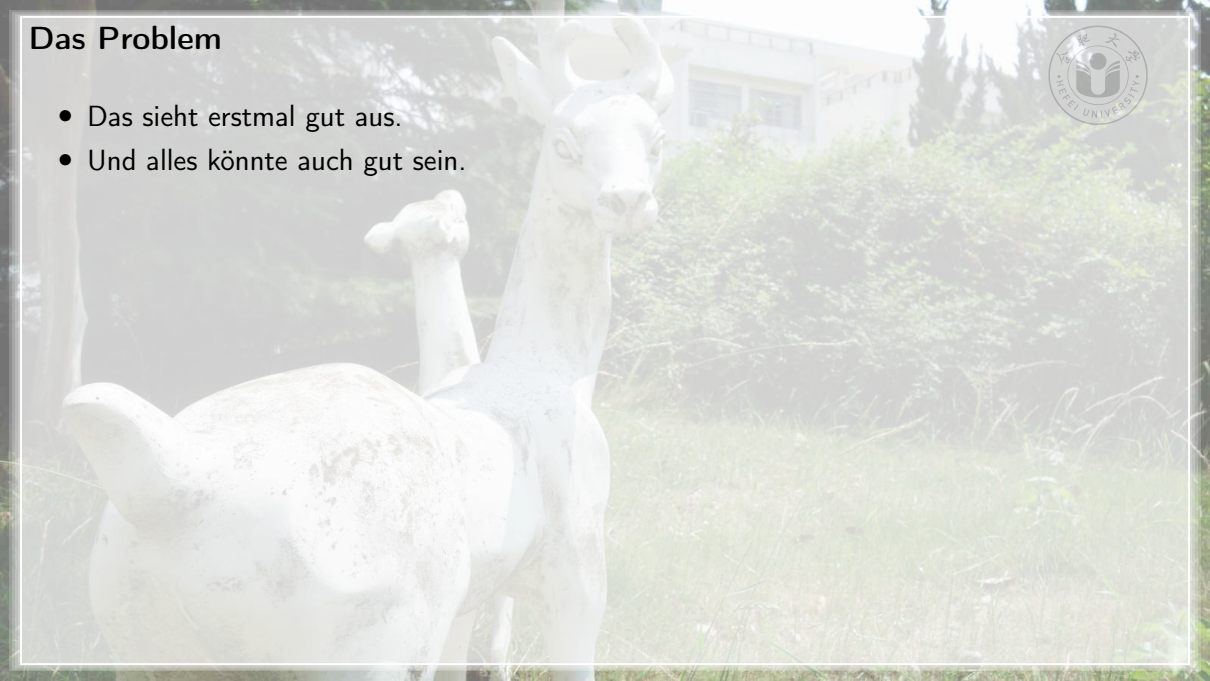
```
$ psql "postgres://postgres:XXX@localhost/anomalies" -v ON_ERROR_STOP=1
  ↪ -ebf insert.sql
INSERT 0 4
INSERT 0 4
# psql 16.11 succeeded with exit code 0.
```

Das Problem

- Das sieht erstmal gut aus.

Das Problem

- Das sieht erstmal gut aus.
- Und alles könnte auch gut sein.



Das Problem

- Das sieht erstmal gut aus.
- Und alles könnte auch gut sein.
- Wenn wir die Adressen immer als einzelne, unstrukturierte Zeichenketten verwenden würden.

Das Problem

- Das sieht erstmal gut aus.
- Und alles könnte auch gut sein.
- Wenn wir die Adressen immer als einzelne, unstrukturierte Zeichenketten verwenden würden.
- Aber das ist nicht unbedingt so, besonders in unserem Lehre-Management-System.

Das Problem

- Das sieht erstmal gut aus.
- Und alles könnte auch gut sein.
- Wenn wir die Adressen immer als einzelne, unstrukturierte Zeichenketten verwenden würden.
- Aber das ist nicht unbedingt so, besonders in unserem Lehre-Management-System.
- In unserem Beispiel wohnt Herr Bibbo in unserer Uni und Herr Babbo kommt aus Quanzhou.

Das Problem

- Das sieht erstmal gut aus.
- Und alles könnte auch gut sein.
- Wenn wir die Adressen immer als einzelne, unstrukturierte Zeichenketten verwenden würden.
- Aber das ist nicht unbedingt so, besonders in unserem Lehre-Management-System.
- In unserem Beispiel wohnt Herr Bibbo in unserer Uni und Herr Babbo kommt aus Quanzhou.
- Herr Bebbo und Frau Bibbi sind aber ausländische Austauschstudenten (留学生) aus Deutschland bzw. den USA.

Das Problem

- Das sieht erstmal gut aus.
- Und alles könnte auch gut sein.
- Wenn wir die Adressen immer als einzelne, unstrukturierte Zeichenketten verwenden würden.
- Aber das ist nicht unbedingt so, besonders in unserem Lehre-Management-System.
- In unserem Beispiel wohnt Herr Bibbo in unserer Uni und Herr Babbo kommt aus Quanzhou.
- Herr Bebbo und Frau Bibbi sind aber ausländische Austauschstudenten (留学生) aus Deutschland bzw. den USA.
- Stellen Sie sich vor, die Tabellen wären viel größer?

Das Problem

- Das sieht erstmal gut aus.
- Und alles könnte auch gut sein.
- Wenn wir die Adressen immer als einzelne, unstrukturierte Zeichenketten verwenden würden.
- Aber das ist nicht unbedingt so, besonders in unserem Lehre-Management-System.
- In unserem Beispiel wohnt Herr Bibbo in unserer Uni und Herr Babbo kommt aus Quanzhou.
- Herr Bebbo und Frau Bibbi sind aber ausländische Austauschstudenten (留学生) aus Deutschland bzw. den USA.
- Stellen Sie sich vor, die Tabellen wären viel größer?
- Was wäre, wenn wir fragen würden, welche unserer Studenten denn eine chinesische Adresse haben?

Das Problem

- Das sieht erstmal gut aus.
- Und alles könnte auch gut sein.
- Wenn wir die Adressen immer als einzelne, unstrukturierte Zeichenketten verwenden würden.
- Aber das ist nicht unbedingt so, besonders in unserem Lehre-Management-System.
- In unserem Beispiel wohnt Herr Bibbo in unserer Uni und Herr Babbo kommt aus Quanzhou.
- Herr Bebbo und Frau Bibbi sind aber ausländische Austauschstudenten (留学生) aus Deutschland bzw. den USA.
- Stellen Sie sich vor, die Tabellen wären viel größer?
- Was wäre, wenn wir fragen würden, welche unserer Studenten denn eine chinesische Adresse haben?
- Wir würden wir das tun?

Daten auslesen (1)

- Nun, wir haben genau sowas schonmal im Kontext unseres Fabrikbeispiels gemacht.

```
/** Get a list of students from China. */

-- Fails to get addresses from the PRC and includes addresses from
-- Chinatown, New York.
SELECT name, full_address as address_attempt_1 FROM student
    INNER JOIN address ON student.address = address.id
    WHERE full_address ILIKE '%china%';
```

```
$ psql "postgres://postgres:XXX@localhost/anomalies" -v ON_ERROR_STOP=1
    ↪ -ebf select_1.sql
 name  |                    address_attempt_1
-------+-----------------------------------------------------------
 Bibbo | Hefei University, Hefei 230601 Jinkaiqu, Hefei, Anhui, China
 Bibbi | Canal Street 4, Chinatown, New York, NY, USA
(2 rows)

# psql 16.11 succeeded with exit code 0.
```

Daten auslesen (1)

- Nun, wir haben genau sowas schonmal im Kontext unseres Fabrikbeispiels gemacht.
- Damals haben wir einen `ILIKE`-Ausdruck[50] geschrieben und das machen wir jetzt auch.

```
/** Get a list of students from China. */

-- Fails to get addresses from the PRC and includes addresses from
-- Chinatown, New York.
SELECT name, full_address as address_attempt_1 FROM student
    INNER JOIN address ON student.address = address.id
    WHERE full_address ILIKE '%china%';
```

```
$ psql "postgres://postgres:XXX@localhost/anomalies" -v ON_ERROR_STOP=1
    ↪ -ebf select_1.sql
 name  |                    address_attempt_1
-------+-----------------------------------------------------------
 Bibbo | Hefei University, Hefei 230601 Jinkaiqu, Hefei, Anhui, China
 Bibbi | Canal Street 4, Chinatown, New York, NY, USA
(2 rows)

# psql 16.11 succeeded with exit code 0.
```

Daten auslesen (1)

- Nun, wir haben genau sowas schonmal im Kontext unseres Fabrikbeispiels gemacht.
- Damals haben wir einen `ILIKE`-Ausdruck[50] geschrieben und das machen wir jetzt auch.
- Wir kombinieren die Tabellen `student` und `address` mit einem `INNER JOIN`.

```
/** Get a list of students from China. */

-- Fails to get addresses from the PRC and includes addresses from
-- Chinatown, New York.
SELECT name, full_address as address_attempt_1 FROM student
    INNER JOIN address ON student.address = address.id
    WHERE full_address ILIKE '%china%';
```

```
$ psql "postgres://postgres:XXX@localhost/anomalies" -v ON_ERROR_STOP=1
    ↪ -ebf select_1.sql
 name  |                    address_attempt_1
-------+-----------------------------------------------------------
 Bibbo | Hefei University, Hefei 230601 Jinkaiqu, Hefei, Anhui, China
 Bibbi | Canal Street 4, Chinatown, New York, NY, USA
(2 rows)

# psql 16.11 succeeded with exit code 0.
```

Daten auslesen (1)

- Nun, wir haben genau sowas schonmal im Kontext unseres Fabrikbeispiels gemacht.
- Damals haben wir einen `ILIKE`-Ausdruck[50] geschrieben und das machen wir jetzt auch.
- Wir kombinieren die Tabellen `student` und `address` mit einem `INNER JOIN`.
- Dann behalten wir nur die Zeilen mit `full_address ILIKE '%china%'`.

```
/** Get a list of students from China. */

-- Fails to get addresses from the PRC and includes addresses from
-- Chinatown, New York.
SELECT name, full_address as address_attempt_1 FROM student
    INNER JOIN address ON student.address = address.id
    WHERE full_address ILIKE '%china%';
```

```
$ psql "postgres://postgres:XXX@localhost/anomalies" -v ON_ERROR_STOP=1
    ↪ -ebf select_1.sql
 name  |                    address_attempt_1
-------+-----------------------------------------------------------
 Bibbo | Hefei University, Hefei 230601 Jinkaiqu, Hefei, Anhui, China
 Bibbi | Canal Street 4, Chinatown, New York, NY, USA
(2 rows)

# psql 16.11 succeeded with exit code 0.
```

Daten auslesen (1)

- Nun, wir haben genau sowas schonmal im Kontext unseres Fabrikbeispiels gemacht.
- Damals haben wir einen `ILIKE`-Ausdruck[50] geschrieben und das machen wir jetzt auch.
- Wir kombinieren die Tabellen `student` und `address` mit einem `INNER JOIN`.
- Dann behalten wir nur die Zeilen mit `full_address ILIKE '%china%'`.
- In anderen Worten, wir behalten die Zeilen, wo das Wort „china“ irgendwo in der Spalte `full_address`, gleichgültig ob es groß- oder klein geschrieben ist.

```sql
/** Get a list of students from China. */

-- Fails to get addresses from the PRC and includes addresses from
-- Chinatown, New York.
SELECT name, full_address as address_attempt_1 FROM student
    INNER JOIN address ON student.address = address.id
    WHERE full_address ILIKE '%china%';
```

```
$ psql "postgres://postgres:XXX@localhost/anomalies" -v ON_ERROR_STOP=1
    ↪ -ebf select_1.sql
 name  |                    address_attempt_1
-------+-----------------------------------------------------------
 Bibbo | Hefei University, Hefei 230601 Jinkaiqu, Hefei, Anhui, China
 Bibbi | Canal Street 4, Chinatown, New York, NY, USA
(2 rows)

# psql 16.11 succeeded with exit code 0.
```

Daten auslesen (1)

- Damals haben wir einen `ILIKE`-Ausdruck[50] geschrieben und das machen wir jetzt auch.
- Wir kombinieren die Tabellen `student` und `address` mit einem `INNER JOIN`.
- Dann behalten wir nur die Zeilen mit `full_address ILIKE '%china%'`.
- In anderen Worten, wir behalten die Zeilen, wo das Wort „china“ irgendwo in der Spalte `full_address`, gleichgültig ob es groß- oder klein geschrieben ist.
- Wir bekommen zwei Studenten heraus, Herrn Bibbo und Frau Bibbi.

```
/** Get a list of students from China. */

-- Fails to get addresses from the PRC and includes addresses from
-- Chinatown, New York.
SELECT name, full_address as address_attempt_1 FROM student
    INNER JOIN address ON student.address = address.id
    WHERE full_address ILIKE '%china%';
```

```
$ psql "postgres://postgres:XXX@localhost/anomalies" -v ON_ERROR_STOP=1
    ↪ -ebf select_1.sql
 name  |                    address_attempt_1
-------+-----------------------------------------------------------
 Bibbo | Hefei University, Hefei 230601 Jinkaiqu, Hefei, Anhui, China
 Bibbi | Canal Street 4, Chinatown, New York, NY, USA
(2 rows)

# psql 16.11 succeeded with exit code 0.
```

Daten auslesen (1)

- Wir kombinieren die Tabellen `student` und `address` mit einem `INNER JOIN`.
- Dann behalten wir nur die Zeilen mit `full_address ILIKE '%china%'`.
- In anderen Worten, wir behalten die Zeilen, wo das Wort „china“ irgendwo in der Spalte `full_address`, gleichgültig ob es groß- oder klein geschrieben ist.
- Wir bekommen zwei Studenten heraus, Herrn Bibbo und Frau Bibbi.
- Frau Bibbi ist aber eine ausländische Studentin.

```sql
/** Get a list of students from China. */

-- Fails to get addresses from the PRC and includes addresses from
-- Chinatown, New York.
SELECT name, full_address as address_attempt_1 FROM student
    INNER JOIN address ON student.address = address.id
    WHERE full_address ILIKE '%china%';
```

```
$ psql "postgres://postgres:XXX@localhost/anomalies" -v ON_ERROR_STOP=1
    ↪ -ebf select_1.sql
 name  |                    address_attempt_1
-------+-----------------------------------------------------------
 Bibbo | Hefei University, Hefei 230601 Jinkaiqu, Hefei, Anhui, China
 Bibbi | Canal Street 4, Chinatown, New York, NY, USA
(2 rows)

# psql 16.11 succeeded with exit code 0.
```

Daten auslesen (1)

- Dann behalten wir nur die Zeilen mit `full_address ILIKE '%china%'`.
- In anderen Worten, wir behalten die Zeilen, wo das Wort „china“ irgendwo in der Spalte `full_address`, gleichgültig ob es groß- oder klein geschrieben ist.
- Wir bekommen zwei Studenten heraus, Herrn Bibbo und Frau Bibbi.
- Frau Bibbi ist aber eine ausländische Studentin.
- Sie wohnt in *China*town, New York.

```
1  /** Get a list of students from China. */
2
3  -- Fails to get addresses from the PRC and includes addresses from
4  -- Chinatown, New York.
5  SELECT name, full_address as address_attempt_1 FROM student
6      INNER JOIN address ON student.address = address.id
7      WHERE full_address ILIKE '%china%';
```

```
1  $ psql "postgres://postgres:XXX@localhost/anomalies" -v ON_ERROR_STOP=1
       ↪ -ebf select_1.sql
2   name  |                    address_attempt_1
3  -------+-----------------------------------------------------------
4   Bibbo | Hefei University, Hefei 230601 Jinkaiqu, Hefei, Anhui, China
5   Bibbi | Canal Street 4, Chinatown, New York, NY, USA
6  (2 rows)
7
8  # psql 16.11 succeeded with exit code 0.
```

Daten auslesen (1)

- In anderen Worten, wir behalten die Zeilen, wo das Wort „china“ irgendwo in der Spalte `full_address`, gleichgültig ob es groß- oder klein geschrieben ist.
- Wir bekommen zwei Studenten heraus, Herrn Bibbo und Frau Bibbi.
- Frau Bibbi ist aber eine ausländische Studentin.
- Sie wohnt in *China*town, New York.
- Herr Babbo wurde gar nicht gelistet, denn er hat in seiner Adresse „PRC“ als Land angegeben, also die Volksrepublik China (中华人民共和国).

```
/** Get a list of students from China. */

-- Fails to get addresses from the PRC and includes addresses from
-- Chinatown, New York.
SELECT name, full_address as address_attempt_1 FROM student
    INNER JOIN address ON student.address = address.id
    WHERE full_address ILIKE '%china%';
```

```
$ psql "postgres://postgres:XXX@localhost/anomalies" -v ON_ERROR_STOP=1
    ↪ -ebf select_1.sql
 name  |                    address_attempt_1
-------+-----------------------------------------------------------
 Bibbo | Hefei University, Hefei 230601 Jinkaiqu, Hefei, Anhui, China
 Bibbi | Canal Street 4, Chinatown, New York, NY, USA
(2 rows)

# psql 16.11 succeeded with exit code 0.
```

Die Probleme

- Wir haben zwei Probleme.

Die Probleme

- Wir haben zwei Probleme:
  1. Wir haben keine Möglichkeit, zu entscheiden welcher Teil von `full_address` das Land ist und welcher nicht.

Die Probleme

- Wir haben zwei Probleme:
  1. Wir haben keine Möglichkeit, zu entscheiden welcher Teil von `full_address` das Land ist und welcher nicht.
  2. Es gibt viele verschiedene Möglichkeiten, den Name eines Landes wie China anzugeben.

Die Probleme

- Wir haben zwei Probleme:
  1. Wir haben keine Möglichkeit, zu entscheiden welcher Teil von `full_address` das Land ist und welcher nicht.
  2. Es gibt viele verschiedene Möglichkeiten, den Name eines Landes wie China anzugeben.
- Das erste Problem resultiert direkt aus der Verletzung der 1NF.

Die Probleme

- Wir haben zwei Probleme:
  1. Wir haben keine Möglichkeit, zu entscheiden welcher Teil von `full_address` das Land ist und welcher nicht.
  2. Es gibt viele verschiedene Möglichkeiten, den Name eines Landes wie China anzugeben.
- Das erste Problem resultiert direkt aus der Verletzung der 1NF.
- Wir haben das Adress-Attribut *nicht* als zusammengesetztes Attribut modelliert.

Die Probleme

- Wir haben zwei Probleme:
  1. Wir haben keine Möglichkeit, zu entscheiden welcher Teil von `full_address` das Land ist und welcher nicht.
  2. Es gibt viele verschiedene Möglichkeiten, den Name eines Landes wie China anzugeben.
- Das erste Problem resultiert direkt aus der Verletzung der 1NF.
- Wir haben das Adress-Attribut *nicht* als zusammengesetztes Attribut modelliert.
- Wir haben es als unteilbares Attribut modelliert, was falsch war, weil wir eben doch auf seine Komponenten zugreifen wollten.

Die Probleme

- Wir haben zwei Probleme:
  1. Wir haben keine Möglichkeit, zu entscheiden welcher Teil von `full_address` das Land ist und welcher nicht.
  2. Es gibt viele verschiedene Möglichkeiten, den Name eines Landes wie China anzugeben.
- Das erste Problem resultiert direkt aus der Verletzung der 1NF.
- Wir haben das Adress-Attribut *nicht* als zusammengesetztes Attribut modelliert.
- Wir haben es als unteilbares Attribut modelliert, was falsch war, weil wir eben doch auf seine Komponenten zugreifen wollten.
- Ein unteilbares Attribut hat aber keine Komponenten.

Die Probleme

- Wir haben zwei Probleme:
  1. Wir haben keine Möglichkeit, zu entscheiden welcher Teil von `full_address` das Land ist und welcher nicht.
  2. Es gibt viele verschiedene Möglichkeiten, den Name eines Landes wie China anzugeben.
- Das erste Problem resultiert direkt aus der Verletzung der 1NF.
- Wir haben das Adress-Attribut *nicht* als zusammengesetztes Attribut modelliert.
- Wir haben es als unteilbares Attribut modelliert, was falsch war, weil wir eben doch auf seine Komponenten zugreifen wollten.
- Ein unteilbares Attribut hat aber keine Komponenten.
- Der Fehler lag also wahrscheinlich schon im konzeptuellen Modell (was wir hier nicht diskutiert haben).

Daten auslesen (2)

- Unsere Datenbank kann trotzdem funktionieren.

```
/** Get a list of students from China. */

-- Gets everything right, but is still error prone.
-- For examples, what if China was written in Chinese?
-- What with other words or names that include China?
SELECT name, full_address as address_attempt_2 FROM student
    INNER JOIN address ON student.address = address.id
    WHERE ((full_address ILIKE '%china%') AND NOT
           (full_address ILIKE '%chinatown%'))
          OR (full_address ILIKE '%PRC%')
          OR (full_address ILIKE '%P.R.C.%');
```

```
$ psql "postgres://postgres:XXX@localhost/anomalies" -v ON_ERROR_STOP=1
    ↪ -ebf select_2.sql
 name  |                    address_attempt_2
-------+-----------------------------------------------------------
 Bibbo | Hefei University, Hefei 230601 Jinkaiqu, Hefei, Anhui, China
 Babbo | West Street, Licheng District, Quanzhou 362002, Fujian, PRC
(2 rows)

# psql 16.11 succeeded with exit code 0.
```

Daten auslesen (2)

- Unsere Datenbank kann trotzdem funktionieren.
- Wir können die Anfrage nach dem Land umbauen um mit den Spezialfällen von oben umzugehen.

```
/** Get a list of students from China. */

-- Gets everything right, but is still error prone.
-- For examples, what if China was written in Chinese?
-- What with other words or names that include China?
SELECT name, full_address as address_attempt_2 FROM student
    INNER JOIN address ON student.address = address.id
    WHERE ((full_address ILIKE '%china%') AND NOT
           (full_address ILIKE '%chinatown%'))
          OR (full_address ILIKE '%PRC%')
          OR (full_address ILIKE '%P.R.C.%');
```

```
$ psql "postgres://postgres:XXX@localhost/anomalies" -v ON_ERROR_STOP=1
  ↪ -ebf select_2.sql
 name  |                      address_attempt_2
-------+-----------------------------------------------------------
 Bibbo | Hefei University, Hefei 230601 Jinkaiqu, Hefei, Anhui, China
 Babbo | West Street, Licheng District, Quanzhou 362002, Fujian, PRC
(2 rows)

# psql 16.11 succeeded with exit code 0.
```

Daten auslesen (2)

- Unsere Datenbank kann trotzdem funktionieren.
- Wir können die Anfrage nach dem Land umbauen um mit den Spezialfällen von oben umzugehen.
- Wir können Adressen aussortieren, die sowohl China als auch Chinatown beinhalten.

```sql
/** Get a list of students from China. */

-- Gets everything right, but is still error prone.
-- For examples, what if China was written in Chinese?
-- What with other words or names that include China?
SELECT name, full_address as address_attempt_2 FROM student
    INNER JOIN address ON student.address = address.id
    WHERE ((full_address ILIKE '%china%') AND NOT
           (full_address ILIKE '%chinatown%'))
          OR (full_address ILIKE '%PRC%')
          OR (full_address ILIKE '%P.R.C.%');
```

```
$ psql "postgres://postgres:XXX@localhost/anomalies" -v ON_ERROR_STOP=1
    ↪ -ebf select_2.sql
 name  |                      address_attempt_2
-------+-----------------------------------------------------------
 Bibbo | Hefei University, Hefei 230601 Jinkaiqu, Hefei, Anhui, China
 Babbo | West Street, Licheng District, Quanzhou 362002, Fujian, PRC
(2 rows)

# psql 16.11 succeeded with exit code 0.
```

Daten auslesen (2)

- Unsere Datenbank kann trotzdem funktionieren.
- Wir können die Anfrage nach dem Land umbauen um mit den Spezialfällen von oben umzugehen.
- Wir können Adressen aussortieren, die sowohl China als auch Chinatown beinhalten.
- Und wir können Adressen dazunehmen, die PRC oder P.R.C. beinhalten.

```sql
/** Get a list of students from China. */

-- Gets everything right, but is still error prone.
-- For examples, what if China was written in Chinese?
-- What with other words or names that include China?
SELECT name, full_address as address_attempt_2 FROM student
    INNER JOIN address ON student.address = address.id
    WHERE ((full_address ILIKE '%china%') AND NOT
           (full_address ILIKE '%chinatown%'))
          OR (full_address ILIKE '%PRC%')
          OR (full_address ILIKE '%P.R.C.%');
```

```
$ psql "postgres://postgres:XXX@localhost/anomalies" -v ON_ERROR_STOP=1
  ↪ -ebf select_2.sql
 name  |                    address_attempt_2
-------+-----------------------------------------------------------
 Bibbo | Hefei University, Hefei 230601 Jinkaiqu, Hefei, Anhui, China
 Babbo | West Street, Licheng District, Quanzhou 362002, Fujian, PRC
(2 rows)

# psql 16.11 succeeded with exit code 0.
```

Daten auslesen (2)

- Unsere Datenbank kann trotzdem funktionieren.
- Wir können die Anfrage nach dem Land umbauen um mit den Spezialfällen von oben umzugehen.
- Wir können Adressen aussortieren, die sowohl China als auch Chinatown beinhalten.
- Und wir können Adressen dazunehmen, die PRC oder P.R.C. beinhalten.
- Das sind aber nur Krücken und keine wirklichen Lösngen.

```sql
/** Get a list of students from China. */

-- Gets everything right, but is still error prone.
-- For examples, what if China was written in Chinese?
-- What with other words or names that include China?
SELECT name, full_address as address_attempt_2 FROM student
    INNER JOIN address ON student.address = address.id
    WHERE ((full_address ILIKE '%china%') AND NOT
           (full_address ILIKE '%chinatown%'))
          OR (full_address ILIKE '%PRC%')
          OR (full_address ILIKE '%P.R.C.%');
```

```
$ psql "postgres://postgres:XXX@localhost/anomalies" -v ON_ERROR_STOP=1
  ↪ -ebf select_2.sql
 name  |                    address_attempt_2
-------+-----------------------------------------------------------
 Bibbo | Hefei University, Hefei 230601 Jinkaiqu, Hefei, Anhui, China
 Babbo | West Street, Licheng District, Quanzhou 362002, Fujian, PRC
(2 rows)

# psql 16.11 succeeded with exit code 0.
```

Daten auslesen (2)

- Wir können die Anfrage nach dem Land umbauen um mit den Spezialfällen von oben umzugehen.
- Wir können Adressen aussortieren, die sowohl China als auch Chinatown beinhalten.
- Und wir können Adressen dazunehmen, die PRC oder P.R.C. beinhalten.
- Das sind aber nur Krücken und keine wirklichen Lösngen.
- Wir können immer noch leicht Adressen finden, die falsch klassifiziert werden würden.

```
/** Get a list of students from China. */

-- Gets everything right, but is still error prone.
-- For examples, what if China was written in Chinese?
-- What with other words or names that include China?
SELECT name, full_address as address_attempt_2 FROM student
    INNER JOIN address ON student.address = address.id
    WHERE ((full_address ILIKE '%china%') AND NOT
           (full_address ILIKE '%chinatown%'))
          OR (full_address ILIKE '%PRC%')
          OR (full_address ILIKE '%P.R.C.%');
```

```
$ psql "postgres://postgres:XXX@localhost/anomalies" -v ON_ERROR_STOP=1
    ↪ -ebf select_2.sql
 name  |                    address_attempt_2
-------+-----------------------------------------------------------
 Bibbo | Hefei University, Hefei 230601 Jinkaiqu, Hefei, Anhui, China
 Babbo | West Street, Licheng District, Quanzhou 362002, Fujian, PRC
(2 rows)

# psql 16.11 succeeded with exit code 0.
```

Daten auslesen (2)

- Wir können Adressen aussortieren, die sowohl China als auch Chinatown beinhalten.
- Und wir können Adressen dazunehmen, die PRC oder P.R.C. beinhalten.
- Das sind aber nur Krücken und keine wirklichen Lösngen.
- Wir können immer noch leicht Adressen finden, die falsch klassifiziert werden würden.
- Z. B. was machen wir mit der „Embassy of China in Berlin, Germany“?

```sql
/** Get a list of students from China. */

-- Gets everything right, but is still error prone.
-- For examples, what if China was written in Chinese?
-- What with other words or names that include China?
SELECT name, full_address as address_attempt_2 FROM student
    INNER JOIN address ON student.address = address.id
    WHERE ((full_address ILIKE '%china%') AND NOT
           (full_address ILIKE '%chinatown%'))
          OR (full_address ILIKE '%PRC%')
          OR (full_address ILIKE '%P.R.C.%');
```

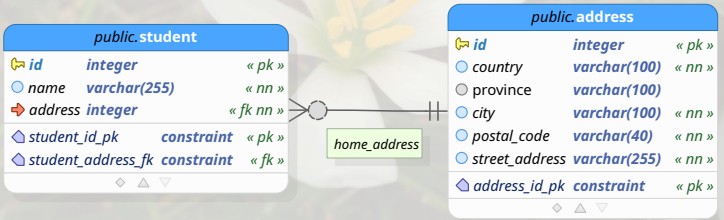
```
$ psql "postgres://postgres:XXX@localhost/anomalies" -v ON_ERROR_STOP=1
  ↪ -ebf select_2.sql
 name  |                      address_attempt_2
-------+-----------------------------------------------------------
 Bibbo | Hefei University, Hefei 230601 Jinkaiqu, Hefei, Anhui, China
 Babbo | West Street, Licheng District, Quanzhou 362002, Fujian, PRC
(2 rows)

# psql 16.11 succeeded with exit code 0.
```

Zusammengesetzte Attribute: Lösung

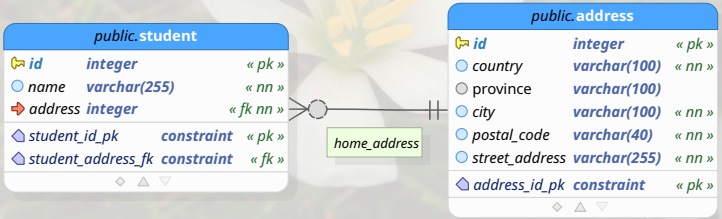
Zusammengesetzte Attribute: Lösung

- Lösen wir das Problem.



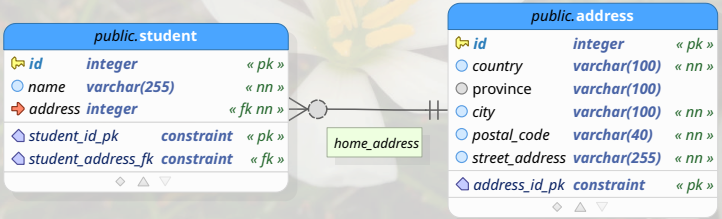
Zusammengesetzte Attribute: Lösung

- Lösen wir das Problem.
- Eine vernünftige Lösung muss sein, die Adresse als zusammengesetztes Attribut zu modellieren.



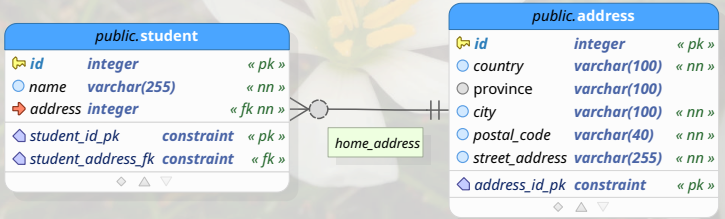
Zusammengesetzte Attribute: Lösung

- Lösen wir das Problem.
- Eine vernünftige Lösung muss sein, die Adresse als zusammengesetztes Attribut zu modellieren.
- Wenigstens das Land muss abgeteilt werden.



Zusammengesetzte Attribute: Lösung

- Lösen wir das Problem.
- Eine vernünftige Lösung muss sein, die Adresse als zusammengesetztes Attribut zu modellieren.
- Wenigstens das Land muss abgeteilt werden.
- Vielleicht könnte man auch noch die Provinz als Komponente modellieren.



Zusammengesetzte Attribute: Lösung

- Lösen wir das Problem.
- Eine vernünftige Lösung muss sein, die Adresse als zusammengesetztes Attribut zu modellieren.
- Wenigstens das Land muss abgeteilt werden.
- Vielleicht könnte man auch noch die Provinz als Komponente modellieren.
- Und weil wir schon dabei sind, teilen wir auch noch Stadt und Postleitzahl ab.

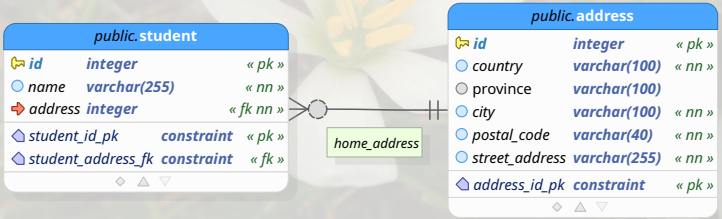


Tabelle address

- Hier sehen wir das vom PgModeler generierte Skript zum Erstellen der Tabelle `address`.

```
-- object: public.address | type: TABLE --
-- DROP TABLE IF EXISTS public.address CASCADE;
CREATE TABLE public.address (
    id integer NOT NULL GENERATED BY DEFAULT AS IDENTITY ,
    country varchar(100) NOT NULL,
    province varchar(100),
    city varchar(100) NOT NULL,
    postal_code varchar(40) NOT NULL,
    street_address varchar(255) NOT NULL,
    CONSTRAINT address_id_pk PRIMARY KEY (id)
);
-- ddl-end --
ALTER TABLE public.address OWNER TO postgres;
-- ddl-end --
```

```
$ psql "postgres://postgres:XXX@localhost/fixed" -v ON_ERROR_STOP=1 -ebf
    ↪ 03_public_address_table_5071.sql
CREATE TABLE
ALTER TABLE
# psql 16.11 succeeded with exit code 0.
```

Tabelle address

- Hier sehen wir das vom PgModeler generierte Skript zum Erstellen der Tabelle `address`.
- Jetzt haben wir die Spalten `country`, `province`, `city`, `postal_code` und `street_address`.

```
-- object: public.address | type: TABLE --
-- DROP TABLE IF EXISTS public.address CASCADE;
CREATE TABLE public.address (
    id integer NOT NULL GENERATED BY DEFAULT AS IDENTITY ,
    country varchar(100) NOT NULL,
    province varchar(100),
    city varchar(100) NOT NULL,
    postal_code varchar(40) NOT NULL,
    street_address varchar(255) NOT NULL,
    CONSTRAINT address_id_pk PRIMARY KEY (id)
);
-- ddl-end --
ALTER TABLE public.address OWNER TO postgres;
-- ddl-end --
```

```
$ psql "postgres://postgres:XXX@localhost/fixed" -v ON_ERROR_STOP=1 -ebf
    ↪ 03_public_address_table_5071.sql
CREATE TABLE
ALTER TABLE
# psql 16.11 succeeded with exit code 0.
```

Tabelle address

- Hier sehen wir das vom PgModeler generierte Skript zum Erstellen der Tabelle `address`.
- Jetzt haben wir die Spalten `country`, `province`, `city`, `postal_code` und `street_address`.
- Alle von ihnen sind `VARCHAR`s passender Länge.

```
-- object: public.address | type: TABLE --
-- DROP TABLE IF EXISTS public.address CASCADE;
CREATE TABLE public.address (
    id integer NOT NULL GENERATED BY DEFAULT AS IDENTITY ,
    country varchar(100) NOT NULL,
    province varchar(100),
    city varchar(100) NOT NULL,
    postal_code varchar(40) NOT NULL,
    street_address varchar(255) NOT NULL,
    CONSTRAINT address_id_pk PRIMARY KEY (id)
);
-- ddl-end --
ALTER TABLE public.address OWNER TO postgres;
-- ddl-end --
```

```
$ psql "postgres://postgres:XXX@localhost/fixed" -v ON_ERROR_STOP=1 -ebf
    ↪ 03_public_address_table_5071.sql
CREATE TABLE
ALTER TABLE
# psql 16.11 succeeded with exit code 0.
```

Tabelle address

- Hier sehen wir das vom PgModeler generierte Skript zum Erstellen der Tabelle `address`.
- Jetzt haben wir die Spalten `country`, `province`, `city`, `postal_code` und `street_address`.
- Alle von ihnen sind `VARCHAR`s passender Länge.
- Wir erlauben, dass `province` `NULL` sein darf, weil manche Länder keine Provinzen haben.

```
-- object: public.address | type: TABLE --
-- DROP TABLE IF EXISTS public.address CASCADE;
CREATE TABLE public.address (
    id integer NOT NULL GENERATED BY DEFAULT AS IDENTITY ,
    country varchar(100) NOT NULL,
    province varchar(100),
    city varchar(100) NOT NULL,
    postal_code varchar(40) NOT NULL,
    street_address varchar(255) NOT NULL,
    CONSTRAINT address_id_pk PRIMARY KEY (id)
);
-- ddl-end --
ALTER TABLE public.address OWNER TO postgres;
-- ddl-end --
```

```
$ psql "postgres://postgres:XXX@localhost/fixed" -v ON_ERROR_STOP=1 -ebf
    ↪ 03_public_address_table_5071.sql
CREATE TABLE
ALTER TABLE
# psql 16.11 succeeded with exit code 0.
```

Tabelle address

- Hier sehen wir das vom PgModeler generierte Skript zum Erstellen der Tabelle `address`.
- Jetzt haben wir die Spalten `country`, `province`, `city`, `postal_code` und `street_address`.
- Alle von ihnen sind `VARCHAR`s passender Länge.
- Wir erlauben, dass `province` `NULL` sein darf, weil manche Länder keine Provinzen haben.
- Alle anderen Felder müssen `NOT NULL` sein.

```
-- object: public.address | type: TABLE --
-- DROP TABLE IF EXISTS public.address CASCADE;
CREATE TABLE public.address (
    id integer NOT NULL GENERATED BY DEFAULT AS IDENTITY ,
    country varchar(100) NOT NULL,
    province varchar(100),
    city varchar(100) NOT NULL,
    postal_code varchar(40) NOT NULL,
    street_address varchar(255) NOT NULL,
    CONSTRAINT address_id_pk PRIMARY KEY (id)
);
-- ddl-end --
ALTER TABLE public.address OWNER TO postgres;
-- ddl-end --
```

```
$ psql "postgres://postgres:XXX@localhost/fixed" -v ON_ERROR_STOP=1 -ebf
    ↪ 03_public_address_table_5071.sql
CREATE TABLE
ALTER TABLE
# psql 16.11 succeeded with exit code 0.
```

Tabelle address

- Jetzt haben wir die Spalten `country`, `province`, `city`, `postal_code` und `street_address`.
- Alle von ihnen sind `VARCHAR`s passender Länge.
- Wir erlauben, dass `province` `NULL` sein darf, weil manche Länder keine Provinzen haben.
- Alle anderen Felder müssen `NOT NULL` sein.
- An der Tabelle `student` ändert sich nichts, also schauen wir sie uns nicht nochmal an.

```
-- object: public.address | type: TABLE --
-- DROP TABLE IF EXISTS public.address CASCADE;
CREATE TABLE public.address (
    id integer NOT NULL GENERATED BY DEFAULT AS IDENTITY ,
    country varchar(100) NOT NULL,
    province varchar(100),
    city varchar(100) NOT NULL,
    postal_code varchar(40) NOT NULL,
    street_address varchar(255) NOT NULL,
    CONSTRAINT address_id_pk PRIMARY KEY (id)
);
-- ddl-end --
ALTER TABLE public.address OWNER TO postgres;
-- ddl-end --
```

```
$ psql "postgres://postgres:XXX@localhost/fixed" -v ON_ERROR_STOP=1 -ebf
    ↪ 03_public_address_table_5071.sql
CREATE TABLE
ALTER TABLE
# psql 16.11 succeeded with exit code 0.
```

Daten Einfügen

- Jetzt fügen wir Daten in die Datenbank ein.

```
/** Insert data into the database. */

-- Insert several address records.
INSERT INTO address (
        country, province, city, postal_code, street_address) VALUES
    ('China', 'Anhui', 'Hefei', '230601', 'Jinkaiqu, Hefei University'),
    ('Deutschland', 'Sachsen', 'Chemnitz', '09111', 'Am Rathaus 1'),
    ('USA', 'NY', 'New York', '10013', 'Canal Street 4, Chinatown'),
    ('PRC', 'Fujian', 'Quanzhou', '362002', 'West Street'),
    ('P.R.C.', NULL, 'Beijing', '100084', 'Tsinghua University');

-- Create the student records.
INSERT INTO student (name, address) VALUES
    ('Bibbo', 1), ('Bebbo', 2), ('Bibbi', 3), ('Babbo', 4), ('Bebbe', 5)
      ↪ ;
```

```
$ psql "postgres://postgres:XXX@localhost/fixed" -v ON_ERROR_STOP=1 -ebf
   ↪ insert.sql
INSERT 0 5
INSERT 0 5
# psql 16.11 succeeded with exit code 0.
```

Daten Einfügen

- Jetzt fügen wir Daten in die Datenbank ein.
- Wir haben wieder Datensätze Herrn Bibbo, Herrn Bebbo, Frau Bibbi und Herrn Bebbo.

```
/** Insert data into the database. */

-- Insert several address records.
INSERT INTO address (
        country, province, city, postal_code, street_address) VALUES
    ('China', 'Anhui', 'Hefei', '230601', 'Jinkaiqu, Hefei University'),
    ('Deutschland', 'Sachsen', 'Chemnitz', '09111', 'Am Rathaus 1'),
    ('USA', 'NY', 'New York', '10013', 'Canal Street 4, Chinatown'),
    ('PRC', 'Fujian', 'Quanzhou', '362002', 'West Street'),
    ('P.R.C.', NULL, 'Beijing', '100084', 'Tsinghua University');

-- Create the student records.
INSERT INTO student (name, address) VALUES
    ('Bibbo', 1), ('Bebbo', 2), ('Bibbi', 3), ('Babbo', 4), ('Bebbe', 5)
      ↪ ;
```

```
$ psql "postgres://postgres:XXX@localhost/fixed" -v ON_ERROR_STOP=1 -ebf
   ↪ insert.sql
INSERT 0 5
INSERT 0 5
# psql 16.11 succeeded with exit code 0.
```

Daten Einfügen

- Jetzt fügen wir Daten in die Datenbank ein.
- Wir haben wieder Datensätze Herrn Bibbo, Herrn Bebbo, Frau Bibbi und Herrn Bebbo.
- Neu dazu kommt Frau Bebbe.

```
/** Insert data into the database. */

-- Insert several address records.
INSERT INTO address (
        country, province, city, postal_code, street_address) VALUES
    ('China', 'Anhui', 'Hefei', '230601', 'Jinkaiqu, Hefei University'),
    ('Deutschland', 'Sachsen', 'Chemnitz', '09111', 'Am Rathaus 1'),
    ('USA', 'NY', 'New York', '10013', 'Canal Street 4, Chinatown'),
    ('PRC', 'Fujian', 'Quanzhou', '362002', 'West Street'),
    ('P.R.C.', NULL, 'Beijing', '100084', 'Tsinghua University');

-- Create the student records.
INSERT INTO student (name, address) VALUES
    ('Bibbo', 1), ('Bebbo', 2), ('Bibbi', 3), ('Babbo', 4), ('Bebbe', 5)
      ↪ ;
```

```
$ psql "postgres://postgres:XXX@localhost/fixed" -v ON_ERROR_STOP=1 -ebf
   ↪ insert.sql
INSERT 0 5
INSERT 0 5
# psql 16.11 succeeded with exit code 0.
```

Daten Einfügen

- Jetzt fügen wir Daten in die Datenbank ein.
- Wir haben wieder Datensätze Herrn Bibbo, Herrn Bebbo, Frau Bibbi und Herrn Bebbo.
- Neu dazu kommt Frau Bebbe.
- Die Adressen von den ersten vier Studenten bleiben gleich, müssen jedoch in ihre Komponenten zerlegt werden.

```sql
/** Insert data into the database. */

-- Insert several address records.
INSERT INTO address (
        country, province, city, postal_code, street_address) VALUES
    ('China', 'Anhui', 'Hefei', '230601', 'Jinkaiqu, Hefei University'),
    ('Deutschland', 'Sachsen', 'Chemnitz', '09111', 'Am Rathaus 1'),
    ('USA', 'NY', 'New York', '10013', 'Canal Street 4, Chinatown'),
    ('PRC', 'Fujian', 'Quanzhou', '362002', 'West Street'),
    ('P.R.C.', NULL, 'Beijing', '100084', 'Tsinghua University');

-- Create the student records.
INSERT INTO student (name, address) VALUES
    ('Bibbo', 1), ('Bebbo', 2), ('Bibbi', 3), ('Babbo', 4), ('Bebbe', 5)
       ↪ ;
```

```
$ psql "postgres://postgres:XXX@localhost/fixed" -v ON_ERROR_STOP=1 -ebf
   ↪ insert.sql
INSERT 0 5
INSERT 0 5
# psql 16.11 succeeded with exit code 0.
```

Daten Einfügen

- Jetzt fügen wir Daten in die Datenbank ein.
- Wir haben wieder Datensätze Herrn Bibbo, Herrn Bebbo, Frau Bibbi und Herrn Bebbo.
- Neu dazu kommt Frau Bebbe.
- Die Adressen von den ersten vier Studenten bleiben gleich, müssen jedoch in ihre Komponenten zerlegt werden.
- Frau Bebbe wohnt in Beijing (北京).

```
/** Insert data into the database. */

-- Insert several address records.
INSERT INTO address (
        country, province, city, postal_code, street_address) VALUES
    ('China', 'Anhui', 'Hefei', '230601', 'Jinkaiqu, Hefei University'),
    ('Deutschland', 'Sachsen', 'Chemnitz', '09111', 'Am Rathaus 1'),
    ('USA', 'NY', 'New York', '10013', 'Canal Street 4, Chinatown'),
    ('PRC', 'Fujian', 'Quanzhou', '362002', 'West Street'),
    ('P.R.C.', NULL, 'Beijing', '100084', 'Tsinghua University');

-- Create the student records.
INSERT INTO student (name, address) VALUES
    ('Bibbo', 1), ('Bebbo', 2), ('Bibbi', 3), ('Babbo', 4), ('Bebbe', 5)
      ↪ ;
```

```
$ psql "postgres://postgres:XXX@localhost/fixed" -v ON_ERROR_STOP=1 -ebf
   ↪ insert.sql
INSERT 0 5
INSERT 0 5
# psql 16.11 succeeded with exit code 0.
```

Nachteil der 1. Normalform

- Auf der vorigen Slide haben wir auch gleich den (kleinen) Nachteil der Normalformen kennengelernt.

Nachteil der 1. Normalform

- Auf der vorigen Slide haben wir auch gleich den (kleinen) Nachteil der Normalformen kennengelernt.
- Sie brechen zusammenhängende Daten auf ihre unabhängigen Einzelteile herunter.

Nachteil der 1. Normalform

- Auf der vorigen Slide haben wir auch gleich den (kleinen) Nachteil der Normalformen kennengelernt.
- Sie brechen zusammenhängende Daten auf ihre unabhängigen Einzelteile herunter.
- Wenn wir später wieder die zusammenhängend Daten „am Stück“ brauchen, dann müssen wir sie wieder zusammensetzen.

Nachteil der 1. Normalform

- Auf der vorigen Slide haben wir auch gleich den (kleinen) Nachteil der Normalformen kennengelernt.
- Sie brechen zusammenhängende Daten auf ihre unabhängigen Einzelteile herunter.
- Wenn wir später wieder die zusammenhängend Daten „am Stück“ brauchen, dann müssen wir sie wieder zusammensetzen.
- Wenn wir also den gesamten Adress-String brauchen, dann müssen wir ihn wieder zusammenbauen, wahrscheinlich mit Hilfe des String-Konkatenations-Operators `||`[69].

Daten auslesen

- Wie Sie sehen können, können wir nun eine Liste von allen Studenten mit einer Adresse in China viel einfacher bekommen.

```
/** Get a list of students from China. */

-- Much easier due to 1NF, but still a bit problematic due to multiple
-- names for same country.
SELECT name, country || ', ' || COALESCE(province || ', ', '')
                     || postal_code || ' ' || city || ', '
                     || street_address AS address FROM student
    INNER JOIN address ON student.address = address.id
    WHERE country ILIKE ANY(ARRAY['%china%', '%PRC%', '%P.R.C.%']);
```

```
$ psql "postgres://postgres:XXX@localhost/fixed" -v ON_ERROR_STOP=1 -ebf
    ↪ select.sql
 name  |                          address
-------+-----------------------------------------------------------
 Bibbo | China, Anhui, 230601 Hefei, Jinkaiqu, Hefei University
 Babbo | PRC, Fujian, 362002 Quanzhou, West Street
 Bebbe | P.R.C., 100084 Beijing, Tsinghua University
(3 rows)

# psql 16.11 succeeded with exit code 0.
```

Daten auslesen

- Wie Sie sehen können, können wir nun eine Liste von allen Studenten mit einer Adresse in China viel einfacher bekommen.
- Wir müssen natürlich noch mit dem Problem umgehen, dass Leute den Ländernamen verschieden schreiben können.

```sql
/** Get a list of students from China. */

-- Much easier due to 1NF, but still a bit problematic due to multiple
-- names for same country.
SELECT name, country || ', ' || COALESCE(province || ', ', '')
                     || postal_code || ' ' || city || ', '
                     || street_address AS address FROM student
    INNER JOIN address ON student.address = address.id
    WHERE country ILIKE ANY(ARRAY['%china%', '%PRC%', '%P.R.C.%']);
```

```
$ psql "postgres://postgres:XXX@localhost/fixed" -v ON_ERROR_STOP=1 -ebf
    ↪ select.sql
 name  |                          address
-------+-----------------------------------------------------------
 Bibbo | China, Anhui, 230601 Hefei, Jinkaiqu, Hefei University
 Babbo | PRC, Fujian, 362002 Quanzhou, West Street
 Bebbe | P.R.C., 100084 Beijing, Tsinghua University
(3 rows)

# psql 16.11 succeeded with exit code 0.
```

Daten auslesen

- Wie Sie sehen können, können wir nun eine Liste von allen Studenten mit einer Adresse in China viel einfacher bekommen.
- Wir müssen natürlich noch mit dem Problem umgehen, dass Leute den Ländernamen verschieden schreiben können.
- Aber wir können nicht aus Versehen jemanden aus Chinatown in San Francisco als jemanden der in China wohnt misklassifizieren.

```sql
/** Get a list of students from China. */

-- Much easier due to 1NF, but still a bit problematic due to multiple
-- names for same country.
SELECT name, country || ', ' || COALESCE(province || ', ', '')
                     || postal_code || ' ' || city || ', '
                     || street_address AS address FROM student
    INNER JOIN address ON student.address = address.id
    WHERE country ILIKE ANY(ARRAY['%china%', '%PRC%', '%P.R.C.%']);
```

```
$ psql "postgres://postgres:XXX@localhost/fixed" -v ON_ERROR_STOP=1 -ebf
    ↪ select.sql
 name  |                          address
-------+-----------------------------------------------------------
 Bibbo | China, Anhui, 230601 Hefei, Jinkaiqu, Hefei University
 Babbo | PRC, Fujian, 362002 Quanzhou, West Street
 Bebbe | P.R.C., 100084 Beijing, Tsinghua University
(3 rows)

# psql 16.11 succeeded with exit code 0.
```

Daten auslesen

- Wie Sie sehen können, können wir nun eine Liste von allen Studenten mit einer Adresse in China viel einfacher bekommen.
- Wir müssen natürlich noch mit dem Problem umgehen, dass Leute den Ländernamen verschieden schreiben können.
- Aber wir können nicht aus Versehen jemanden aus Chinatown in San Francisco als jemanden der in China wohnt misklassifizieren.
- Und wenn wir sowieso dabei sind, lernen wir noch etwas mehr SQL – was aber mit der 1NF nichts zu tun hat.

```sql
/** Get a list of students from China. */

-- Much easier due to 1NF, but still a bit problematic due to multiple
-- names for same country.
SELECT name, country || ', ' || COALESCE(province || ', ', '')
                     || postal_code || ' ' || city || ', '
                     || street_address AS address FROM student
    INNER JOIN address ON student.address = address.id
    WHERE country ILIKE ANY(ARRAY['%china%', '%PRC%', '%P.R.C.%']);
```

```
$ psql "postgres://postgres:XXX@localhost/fixed" -v ON_ERROR_STOP=1 -ebf
    ↪ select.sql
 name  |                          address
-------+-----------------------------------------------------------
 Bibbo | China, Anhui, 230601 Hefei, Jinkaiqu, Hefei University
 Babbo | PRC, Fujian, 362002 Quanzhou, West Street
 Bebbe | P.R.C., 100084 Beijing, Tsinghua University
(3 rows)

# psql 16.11 succeeded with exit code 0.
```

Daten auslesen

- Wir müssen natürlich noch mit dem Problem umgehen, dass Leute den Ländernamen verschieden schreiben können.
- Aber wir können nicht aus Versehen jemanden aus Chinatown in San Francisco als jemanden der in China wohnt misklassifizieren.
- Und wenn wir sowieso dabei sind, lernen wir noch etwas mehr SQL – was aber mit der 1NF nichts zu tun hat.
- Wenn Sie die Anfrage lesen, dann sehen Sie, dass das Zusammenbauen der vollen Adresse doch schwieriger war, als einfach `||` zu verwenden.

```sql
/** Get a list of students from China. */

-- Much easier due to 1NF, but still a bit problematic due to multiple
-- names for same country.
SELECT name, country || ', ' || COALESCE(province || ', ', '')
                     || postal_code || ' ' || city || ', '
                     || street_address AS address FROM student
    INNER JOIN address ON student.address = address.id
    WHERE country ILIKE ANY(ARRAY['%china%', '%PRC%', '%P.R.C.%']);
```

```
$ psql "postgres://postgres:XXX@localhost/fixed" -v ON_ERROR_STOP=1 -ebf
    ↪ select.sql
 name  |                          address
-------+-----------------------------------------------------------
 Bibbo | China, Anhui, 230601 Hefei, Jinkaiqu, Hefei University
 Babbo | PRC, Fujian, 362002 Quanzhou, West Street
 Bebbe | P.R.C., 100084 Beijing, Tsinghua University
(3 rows)

# psql 16.11 succeeded with exit code 0.
```

Daten auslesen

- Aber wir können nicht aus Versehen jemanden aus Chinatown in San Francisco als jemanden der in China wohnt misklassifizieren.
- Und wenn wir sowieso dabei sind, lernen wir noch etwas mehr SQL – was aber mit der 1NF nichts zu tun hat.
- Wenn Sie die Anfrage lesen, dann sehen Sie, dass das Zusammenbauen der vollen Adresse doch schwieriger war, als einfach `||` zu verwenden.
- Das ist weil die Spalte `province` `NULL` sein kann.

```sql
/** Get a list of students from China. */

-- Much easier due to 1NF, but still a bit problematic due to multiple
-- names for same country.
SELECT name, country || ', ' || COALESCE(province || ', ', '')
                     || postal_code || ' ' || city || ', '
                     || street_address AS address FROM student
    INNER JOIN address ON student.address = address.id
    WHERE country ILIKE ANY(ARRAY['%china%', '%PRC%', '%P.R.C.%']);
```

```
$ psql "postgres://postgres:XXX@localhost/fixed" -v ON_ERROR_STOP=1 -ebf
    ↪ select.sql
 name  |                          address
-------+-----------------------------------------------------------
 Bibbo | China, Anhui, 230601 Hefei, Jinkaiqu, Hefei University
 Babbo | PRC, Fujian, 362002 Quanzhou, West Street
 Bebbe | P.R.C., 100084 Beijing, Tsinghua University
(3 rows)

# psql 16.11 succeeded with exit code 0.
```

Daten auslesen

- Und wenn wir sowieso dabei sind, lernen wir noch etwas mehr SQL – was aber mit der 1NF nichts zu tun hat.
- Wenn Sie die Anfrage lesen, dann sehen Sie, dass das Zusammenbauen der vollen Adresse doch schwieriger war, als einfach `||` zu verwenden.
- Das ist weil die Spalte `province` `NULL` sein kann.
- Wir haben sogar so einen Fall in unserer Tabelle.

```
/** Get a list of students from China. */

-- Much easier due to 1NF, but still a bit problematic due to multiple
-- names for same country.
SELECT name, country || ', ' || COALESCE(province || ', ', '')
                     || postal_code || ' ' || city || ', '
                     || street_address AS address FROM student
    INNER JOIN address ON student.address = address.id
    WHERE country ILIKE ANY(ARRAY['%china%', '%PRC%', '%P.R.C.%']);
```

```
$ psql "postgres://postgres:XXX@localhost/fixed" -v ON_ERROR_STOP=1 -ebf
    ↪ select.sql
 name  |                          address
-------+-----------------------------------------------------------
 Bibbo | China, Anhui, 230601 Hefei, Jinkaiqu, Hefei University
 Babbo | PRC, Fujian, 362002 Quanzhou, West Street
 Bebbe | P.R.C., 100084 Beijing, Tsinghua University
(3 rows)

# psql 16.11 succeeded with exit code 0.
```

Daten auslesen

- Und wenn wir sowieso dabei sind, lernen wir noch etwas mehr SQL – was aber mit der 1NF nichts zu tun hat.
- Wenn Sie die Anfrage lesen, dann sehen Sie, dass das Zusammenbauen der vollen Adresse doch schwieriger war, als einfach `||` zu verwenden.
- Das ist weil die Spalte `province` `NULL` sein kann.
- Wir haben sogar so einen Fall in unserer Tabelle.
- Frau Bebbe ist aus Beijing (北京市).

```
/** Get a list of students from China. */

-- Much easier due to 1NF, but still a bit problematic due to multiple
-- names for same country.
SELECT name, country || ', ' || COALESCE(province || ', ', '')
                     || postal_code || ' ' || city || ', '
                     || street_address AS address FROM student
    INNER JOIN address ON student.address = address.id
    WHERE country ILIKE ANY(ARRAY['%china%', '%PRC%', '%P.R.C.%']);
```

```
$ psql "postgres://postgres:XXX@localhost/fixed" -v ON_ERROR_STOP=1 -ebf
    ↪ select.sql
 name  |                          address
-------+-----------------------------------------------------------
 Bibbo | China, Anhui, 230601 Hefei, Jinkaiqu, Hefei University
 Babbo | PRC, Fujian, 362002 Quanzhou, West Street
 Bebbe | P.R.C., 100084 Beijing, Tsinghua University
(3 rows)

# psql 16.11 succeeded with exit code 0.
```

Daten auslesen

- Wenn Sie die Anfrage lesen, dann sehen Sie, dass das Zusammenbauen der vollen Adresse doch schwieriger war, als einfach `||` zu verwenden.
- Das ist weil die Spalte `province` `NULL` sein kann.
- Wir haben sogar so einen Fall in unserer Tabelle.
- Frau Bebbe ist aus Beijing (北京市).
- Beijing gehört zu keiner Provinz, sondern ist eine Stadt unter der direkten Kontrolle der Zentralregierung von China.

```sql
/** Get a list of students from China. */

-- Much easier due to 1NF, but still a bit problematic due to multiple
-- names for same country.
SELECT name, country || ', ' || COALESCE(province || ', ', '')
                     || postal_code || ' ' || city || ', '
                     || street_address AS address FROM student
    INNER JOIN address ON student.address = address.id
    WHERE country ILIKE ANY(ARRAY['%china%', '%PRC%', '%P.R.C.%']);
```

```
$ psql "postgres://postgres:XXX@localhost/fixed" -v ON_ERROR_STOP=1 -ebf
    ↪ select.sql
 name  |                          address
-------+-----------------------------------------------------------
 Bibbo | China, Anhui, 230601 Hefei, Jinkaiqu, Hefei University
 Babbo | PRC, Fujian, 362002 Quanzhou, West Street
 Bebbe | P.R.C., 100084 Beijing, Tsinghua University
(3 rows)

# psql 16.11 succeeded with exit code 0.
```

Daten auslesen

- Das ist weil die Spalte `province` `NULL` sein kann.
- Wir haben sogar so einen Fall in unserer Tabelle.
- Frau Bebbe ist aus Beijing (北京市).
- Beijing gehört zu keiner Provinz, sondern ist eine Stadt unter der direkten Kontrolle der Zentralregierung von China.
- Deshalb hatten wir ihre `province`-Spalte auf `NULL` gelassen.

```sql
/** Get a list of students from China. */

-- Much easier due to 1NF, but still a bit problematic due to multiple
-- names for same country.
SELECT name, country || ', ' || COALESCE(province || ', ', '')
                     || postal_code || ' ' || city || ', '
                     || street_address AS address FROM student
    INNER JOIN address ON student.address = address.id
    WHERE country ILIKE ANY(ARRAY['%china%', '%PRC%', '%P.R.C.%']);
```

```
$ psql "postgres://postgres:XXX@localhost/fixed" -v ON_ERROR_STOP=1 -ebf
    ↪ select.sql
 name  |                          address
-------+-----------------------------------------------------------
 Bibbo | China, Anhui, 230601 Hefei, Jinkaiqu, Hefei University
 Babbo | PRC, Fujian, 362002 Quanzhou, West Street
 Bebbe | P.R.C., 100084 Beijing, Tsinghua University
(3 rows)

# psql 16.11 succeeded with exit code 0.
```

Daten auslesen

- Wir haben sogar so einen Fall in unserer Tabelle.
- Frau Bebbe ist aus Beijing (北京市).
- Beijing gehört zu keiner Provinz, sondern ist eine Stadt unter der direkten Kontrolle der Zentralregierung von China.
- Deshalb hatten wir ihre `province`-Spalte auf `NULL` gelassen.
- Wenn Sie in PostgreSQL strings mit `NULL` zusammenfügen, dann bekommen Sie `NULL` als Ergebnis.

```sql
/** Get a list of students from China. */

-- Much easier due to 1NF, but still a bit problematic due to multiple
-- names for same country.
SELECT name, country || ', ' || COALESCE(province || ', ', '')
                     || postal_code || ' ' || city || ', '
                     || street_address AS address FROM student
    INNER JOIN address ON student.address = address.id
    WHERE country ILIKE ANY(ARRAY['%china%', '%PRC%', '%P.R.C.%']);
```

```
$ psql "postgres://postgres:XXX@localhost/fixed" -v ON_ERROR_STOP=1 -ebf
    ↪ select.sql
 name  |                          address
-------+-----------------------------------------------------------
 Bibbo | China, Anhui, 230601 Hefei, Jinkaiqu, Hefei University
 Babbo | PRC, Fujian, 362002 Quanzhou, West Street
 Bebbe | P.R.C., 100084 Beijing, Tsinghua University
(3 rows)

# psql 16.11 succeeded with exit code 0.
```

Daten auslesen

- Frau Bebbe ist aus Beijing (北京市).
- Beijing gehört zu keiner Provinz, sondern ist eine Stadt unter der direkten Kontrolle der Zentralregierung von China.
- Deshalb hatten wir ihre `province`-Spalte auf `NULL` gelassen.
- Wenn Sie in PostgreSQL strings mit `NULL` zusammenfügen, dann bekommen Sie `NULL` als Ergebnis.
- Wenn wir also einfach alle Adressfelder mit `||` zusammenfügen, dann bekommen wir `NULL` als die Adresse von Frau Bebbe.

```sql
/** Get a list of students from China. */

-- Much easier due to 1NF, but still a bit problematic due to multiple
-- names for same country.
SELECT name, country || ', ' || COALESCE(province || ', ', '')
                     || postal_code || ' ' || city || ', '
                     || street_address AS address FROM student
    INNER JOIN address ON student.address = address.id
    WHERE country ILIKE ANY(ARRAY['%china%', '%PRC%', '%P.R.C.%']);
```

```
$ psql "postgres://postgres:XXX@localhost/fixed" -v ON_ERROR_STOP=1 -ebf
    ↪ select.sql
 name  |                          address
-------+-----------------------------------------------------------
 Bibbo | China, Anhui, 230601 Hefei, Jinkaiqu, Hefei University
 Babbo | PRC, Fujian, 362002 Quanzhou, West Street
 Bebbe | P.R.C., 100084 Beijing, Tsinghua University
(3 rows)

# psql 16.11 succeeded with exit code 0.
```

Daten auslesen

- Beijing gehört zu keiner Provinz, sondern ist eine Stadt unter der direkten Kontrolle der Zentralregierung von China.
- Deshalb hatten wir ihre `province`-Spalte auf `NULL` gelassen.
- Wenn Sie in PostgreSQL strings mit `NULL` zusammenfügen, dann bekommen Sie `NULL` als Ergebnis.
- Wenn wir also einfach alle Adressfelder mit `||` zusammenfügen, dann bekommen wir `NULL` als die Adresse von Frau Bebbe.
- Um mit dem `NULL` in der `province` umzugehen, benutzen wir die Funktion `COALESCE`[24].

```
/** Get a list of students from China. */

-- Much easier due to 1NF, but still a bit problematic due to multiple
-- names for same country.
SELECT name, country || ', ' || COALESCE(province || ', ', '')
                     || postal_code || ' ' || city || ', '
                     || street_address AS address FROM student
    INNER JOIN address ON student.address = address.id
    WHERE country ILIKE ANY(ARRAY['%china%', '%PRC%', '%P.R.C.%']);
```

```
$ psql "postgres://postgres:XXX@localhost/fixed" -v ON_ERROR_STOP=1 -ebf
    ↪ select.sql
 name  |                          address
-------+-----------------------------------------------------------
 Bibbo | China, Anhui, 230601 Hefei, Jinkaiqu, Hefei University
 Babbo | PRC, Fujian, 362002 Quanzhou, West Street
 Bebbe | P.R.C., 100084 Beijing, Tsinghua University
(3 rows)

# psql 16.11 succeeded with exit code 0.
```

Daten auslesen

- Deshalb hatten wir ihre `province`-Spalte auf `NULL` gelassen.
- Wenn Sie in PostgreSQL strings mit `NULL` zusammenfügen, dann bekommen Sie `NULL` als Ergebnis.
- Wenn wir also einfach alle Adressfelder mit `||` zusammenfügen, dann bekommen wir `NULL` als die Adresse von Frau Bebbe.
- Um mit dem `NULL` in der `province` umzugehen, benutzen wir die Funktion `COALESCE`[24].
- Diese akzeptiert beliebig veiele Argumente und liefert das erste Argument zurück, das nicht `NULL` ist (oder `NULL` wenn `NULL` sind).

```sql
/** Get a list of students from China. */

-- Much easier due to 1NF, but still a bit problematic due to multiple
-- names for same country.
SELECT name, country || ', ' || COALESCE(province || ', ', '')
                     || postal_code || ' ' || city || ', '
                     || street_address AS address FROM student
    INNER JOIN address ON student.address = address.id
    WHERE country ILIKE ANY(ARRAY['%china%', '%PRC%', '%P.R.C.%']);
```

```
$ psql "postgres://postgres:XXX@localhost/fixed" -v ON_ERROR_STOP=1 -ebf
    ↪ select.sql
 name  |                          address
-------+-----------------------------------------------------------
 Bibbo | China, Anhui, 230601 Hefei, Jinkaiqu, Hefei University
 Babbo | PRC, Fujian, 362002 Quanzhou, West Street
 Bebbe | P.R.C., 100084 Beijing, Tsinghua University
(3 rows)

# psql 16.11 succeeded with exit code 0.
```

Daten auslesen

- Um mit dem `NULL` in der `province` umzugehen, benutzen wir die Funktion `COALESCE`[24].
- Diese akzeptiert beliebig veiele Argumente und liefert das erste Argument zurück, das nicht `NULL` ist (oder `NULL` wenn `NULL` sind).
- Wenn Sie die Anfrage lesen, dann finden Sie noch eine weitere Veränderung.
- Wir könnten `OR` verwenden, um die drei Bedingungen `country ILIKE '%china%'`, `country ILIKE '%PRC%'` und `country ILIKE '%P.R.C.%'` zu verbinden.

```sql
/** Get a list of students from China. */

-- Much easier due to 1NF, but still a bit problematic due to multiple
-- names for same country.
SELECT name, country || ', ' || COALESCE(province || ', ', '')
                     || postal_code || ' ' || city || ', '
                     || street_address AS address FROM student
    INNER JOIN address ON student.address = address.id
    WHERE country ILIKE ANY(ARRAY['%china%', '%PRC%', '%P.R.C.%']);
```

```
$ psql "postgres://postgres:XXX@localhost/fixed" -v ON_ERROR_STOP=1 -ebf
    ↪ select.sql
 name  |                          address
-------+-----------------------------------------------------------
 Bibbo | China, Anhui, 230601 Hefei, Jinkaiqu, Hefei University
 Babbo | PRC, Fujian, 362002 Quanzhou, West Street
 Bebbe | P.R.C., 100084 Beijing, Tsinghua University
(3 rows)

# psql 16.11 succeeded with exit code 0.
```

Daten auslesen

- Wenn Sie die Anfrage lesen, dann finden Sie noch eine weitere Veränderung.
- Wir könnten `OR` verwenden, um die drei Bedingungen `country ILIKE '%china%'`, `country ILIKE '%PRC%'` und `country ILIKE '%P.R.C.%'` zu verbinden.
- Stattdessen habe ich geschrieben `country ILIKE ANY(ARRAY['%china%', '%PRC%', '%P.R.C.%'])`, was das gleiche bedeutet[2,57]:

```sql
/** Get a list of students from China. */

-- Much easier due to 1NF, but still a bit problematic due to multiple
-- names for same country.
SELECT name, country || ', ' || COALESCE(province || ', ', '')
                     || postal_code || ' ' || city || ', '
                     || street_address AS address FROM student
    INNER JOIN address ON student.address = address.id
    WHERE country ILIKE ANY(ARRAY['%china%', '%PRC%', '%P.R.C.%']);
```

```
$ psql "postgres://postgres:XXX@localhost/fixed" -v ON_ERROR_STOP=1 -ebf
    ↪ select.sql
 name  |                          address
-------+-----------------------------------------------------------
 Bibbo | China, Anhui, 230601 Hefei, Jinkaiqu, Hefei University
 Babbo | PRC, Fujian, 362002 Quanzhou, West Street
 Bebbe | P.R.C., 100084 Beijing, Tsinghua University
(3 rows)

# psql 16.11 succeeded with exit code 0.
```

Daten auslesen

- Wenn Sie die Anfrage lesen, dann finden Sie noch eine weitere Veränderung.
- Wir könnten `OR` verwenden, um die drei Bedingungen `country ILIKE '%china%'`, `country ILIKE '%PRC%'` und `country ILIKE '%P.R.C.%'` zu verbinden.
- Stattdessen habe ich geschrieben `country ILIKE ANY(ARRAY['%china%', '%PRC%', '%P.R.C.%'])`, was das gleiche bedeutet[2,57]:
- Wir können einen Array der Werte `a`, `b`, `c` und `d` über `ARRAY[a, b, c, d]` deklarieren.

```sql
/** Get a list of students from China. */

-- Much easier due to 1NF, but still a bit problematic due to multiple
-- names for same country.
SELECT name, country || ', ' || COALESCE(province || ', ', '')
                     || postal_code || ' ' || city || ', '
                     || street_address AS address FROM student
    INNER JOIN address ON student.address = address.id
    WHERE country ILIKE ANY(ARRAY['%china%', '%PRC%', '%P.R.C.%']);
```

```
$ psql "postgres://postgres:XXX@localhost/fixed" -v ON_ERROR_STOP=1 -ebf
    ↪ select.sql
 name  |                          address
-------+-----------------------------------------------------------
 Bibbo | China, Anhui, 230601 Hefei, Jinkaiqu, Hefei University
 Babbo | PRC, Fujian, 362002 Quanzhou, West Street
 Bebbe | P.R.C., 100084 Beijing, Tsinghua University
(3 rows)

# psql 16.11 succeeded with exit code 0.
```

Daten auslesen

- Stattdessen habe ich geschrieben `country ILIKE ANY(ARRAY['%china%', '%PRC%', '%P.R.C.%'])`, was das gleiche bedeutet[2,57]:
- Wir können einen Array der Werte `a`, `b`, `c` und `d` über `ARRAY[a, b, c, d]` deklarieren.
- Der Ausdruck `XXX operator ANY(ARRAY[...])` wird dann `TRUE`, wenn `XXX operator YYY` auch `TRUE` für irgendein – also mindestens ein – `YYY` im Array gilt[57].

```sql
/** Get a list of students from China. */

-- Much easier due to 1NF, but still a bit problematic due to multiple
-- names for same country.
SELECT name, country || ', ' || COALESCE(province || ', ', '')
                     || postal_code || ' ' || city || ', '
                     || street_address AS address FROM student
    INNER JOIN address ON student.address = address.id
    WHERE country ILIKE ANY(ARRAY['%china%', '%PRC%', '%P.R.C.%']);
```

```
$ psql "postgres://postgres:XXX@localhost/fixed" -v ON_ERROR_STOP=1 -ebf
    ↪ select.sql
 name  |                          address
-------+-----------------------------------------------------------
 Bibbo | China, Anhui, 230601 Hefei, Jinkaiqu, Hefei University
 Babbo | PRC, Fujian, 362002 Quanzhou, West Street
 Bebbe | P.R.C., 100084 Beijing, Tsinghua University
(3 rows)

# psql 16.11 succeeded with exit code 0.
```

Daten auslesen

- Stattdessen habe ich geschrieben `country ILIKE ANY(ARRAY['%china%', '%PRC%', '%P.R.C.%'])`, was das gleiche bedeutet[2,57]:
- Wir können einen Array der Werte `a`, `b`, `c` und `d` über `ARRAY[a, b, c, d]` deklarieren.
- Der Ausdruck `XXX operator ANY(ARRAY[...])` wird dann `TRUE`, wenn `XXX operator YYY` auch `TRUE` für irgendein – also mindestens ein – `YYY` im Array gilt[57].
- In unserem Fall ist `XXX` is `country` und `operator` ist `ILIKE`.

```sql
/** Get a list of students from China. */

-- Much easier due to 1NF, but still a bit problematic due to multiple
-- names for same country.
SELECT name, country || ', ' || COALESCE(province || ', ', '')
                     || postal_code || ' ' || city || ', '
                     || street_address AS address FROM student
    INNER JOIN address ON student.address = address.id
    WHERE country ILIKE ANY(ARRAY['%china%', '%PRC%', '%P.R.C.%']);
```

```
$ psql "postgres://postgres:XXX@localhost/fixed" -v ON_ERROR_STOP=1 -ebf
    ↪ select.sql
 name  |                          address
-------+-----------------------------------------------------------
 Bibbo | China, Anhui, 230601 Hefei, Jinkaiqu, Hefei University
 Babbo | PRC, Fujian, 362002 Quanzhou, West Street
 Bebbe | P.R.C., 100084 Beijing, Tsinghua University
(3 rows)

# psql 16.11 succeeded with exit code 0.
```

Daten auslesen

- Wir können einen Array der Werte `a`, `b`, `c` und `d` über `ARRAY[a, b, c, d]` deklarieren.
- Der Ausdruck `XXX operator ANY(ARRAY[...])` wird dann `TRUE`, wenn `XXX operator YYY` auch `TRUE` für irgendein – also mindestens ein – `YYY` im Array gilt[57].
- In unserem Fall ist `XXX` is `country` und `operator` ist `ILIKE`.
- Ähnliche Ausdrücke können mit `ALL` statt `ANY` gebaut werden, wobei der Operator dann für *alle* elemente des Arrays `TRUE` liefern müss[57].

```
/** Get a list of students from China. */

-- Much easier due to 1NF, but still a bit problematic due to multiple
-- names for same country.
SELECT name, country || ', ' || COALESCE(province || ', ', '')
                     || postal_code || ' ' || city || ', '
                     || street_address AS address FROM student
    INNER JOIN address ON student.address = address.id
    WHERE country ILIKE ANY(ARRAY['%china%', '%PRC%', '%P.R.C.%']);
```

```
$ psql "postgres://postgres:XXX@localhost/fixed" -v ON_ERROR_STOP=1 -ebf
    ↪ select.sql
 name  |                          address
-------+-----------------------------------------------------------
 Bibbo | China, Anhui, 230601 Hefei, Jinkaiqu, Hefei University
 Babbo | PRC, Fujian, 362002 Quanzhou, West Street
 Bebbe | P.R.C., 100084 Beijing, Tsinghua University
(3 rows)

# psql 16.11 succeeded with exit code 0.
```

Daten auslesen

- Der Ausdruck `XXX operator ANY(ARRAY[...])` wird dann `TRUE`, wenn `XXX operator YYY` auch `TRUE` für irgendein – also mindestens ein – `YYY` im Array gilt[57].
- In unserem Fall ist `XXX` is `country` und `operator` ist `ILIKE`.
- Ähnliche Ausdrücke können mit `ALL` statt `ANY` gebaut werden, wobei der Operator dann für *alle* elemente des Arrays `TRUE` liefern müss[57].
- Mit diesem Konstrukt können wir also etwas Platz sparen.

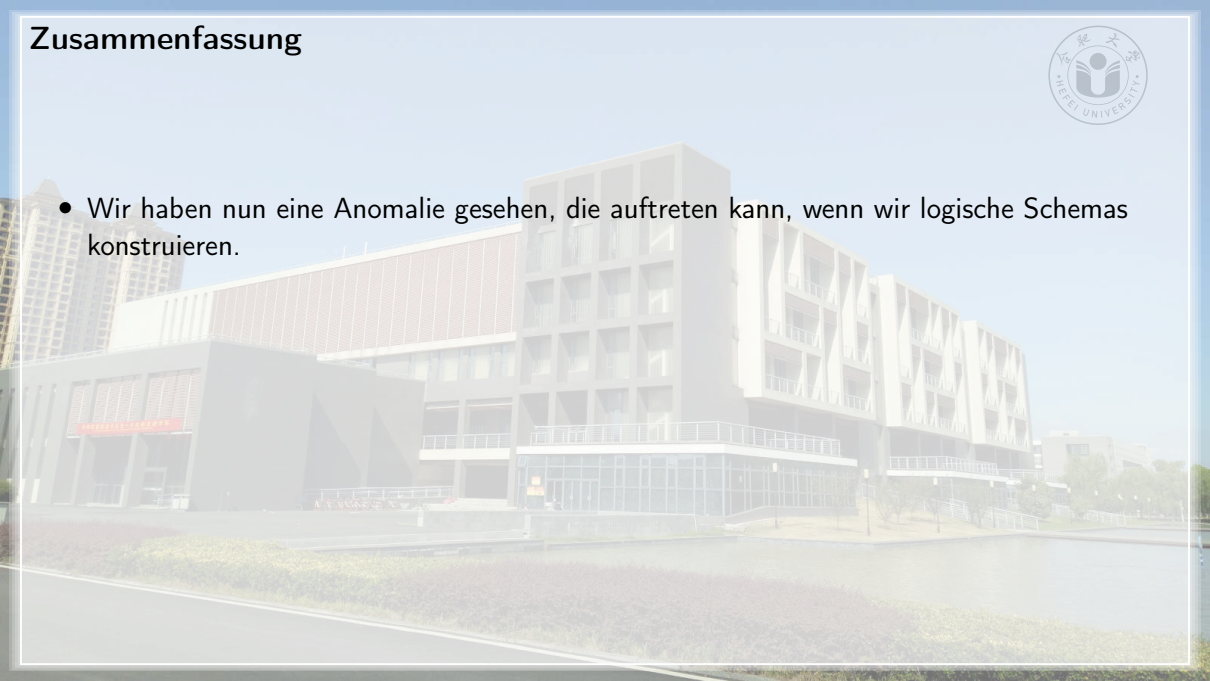
```sql
/** Get a list of students from China. */

-- Much easier due to 1NF, but still a bit problematic due to multiple
-- names for same country.
SELECT name, country || ', ' || COALESCE(province || ', ', '')
                     || postal_code || ' ' || city || ', '
                     || street_address AS address FROM student
    INNER JOIN address ON student.address = address.id
    WHERE country ILIKE ANY(ARRAY['%china%', '%PRC%', '%P.R.C.%']);
```

```
$ psql "postgres://postgres:XXX@localhost/fixed" -v ON_ERROR_STOP=1 -ebf
    ↪ select.sql
 name  |                          address
-------+-----------------------------------------------------------
 Bibbo | China, Anhui, 230601 Hefei, Jinkaiqu, Hefei University
 Babbo | PRC, Fujian, 362002 Quanzhou, West Street
 Bebbe | P.R.C., 100084 Beijing, Tsinghua University
(3 rows)

# psql 16.11 succeeded with exit code 0.
```

Zusammenfassung



- Wir haben nun eine Anomalie gesehen, die auftreten kann, wenn wir logische Schemas konstruieren.

Zusammenfassung

- Wir haben nun eine Anomalie gesehen, die auftreten kann, wenn wir logische Schemas konstruieren.
- Wenn wir zusammengesetzte Attribute fälschlicherweise als unteilbare Attribute modellieren, dann können wir schnelle in einer Hölle aus Spezielfällen und Krücken landen.

Zusammenfassung

- Wir haben nun eine Anomalie gesehen, die auftreten kann, wenn wir logische Schemas konstruieren.
- Wenn wir zusammengesetzte Attribute fälschlicherweise als unteilbare Attribute modellieren, dann können wir schnelle in einer Hölle aus Spezielfällen und Krücken landen.
- Die Natur von Attributen richtig zu erkennen und der 1NF string zu folgen kann also die Menge möglicher Fehler stark verringern.

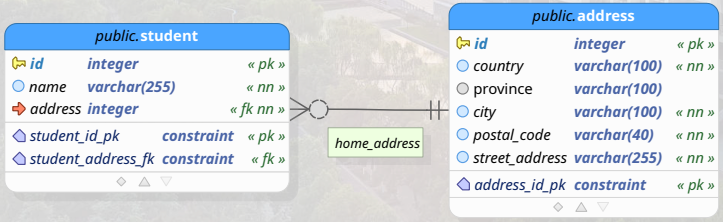
Zusammenfassung

- Wir haben nun eine Anomalie gesehen, die auftreten kann, wenn wir logische Schemas konstruieren.
- Wenn wir zusammengesetzte Attribute fälschlicherweise als unteilbare Attribute modellieren, dann können wir schnelle in einer Hölle aus Spezielfällen und Krücken landen.
- Die Natur von Attributen richtig zu erkennen und der 1NF string zu folgen kann also die Menge möglicher Fehler stark verringern.
- Wir bezahlen dafür mit etwas komplizierteren Anfragen, die die Daten dann wieder zusammensetzen müssen.

Mehrwertige Attribute: Problem

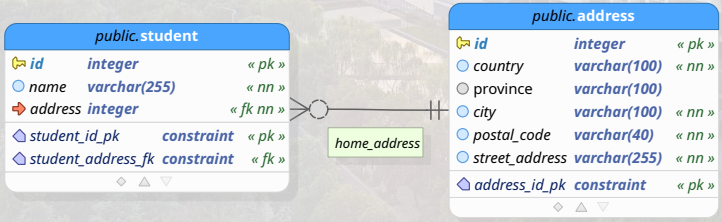
Verletzung der 1. Normalform durch Mehrwertige Attribute

- Setzen wir unser Beispiel von vorhin fort.



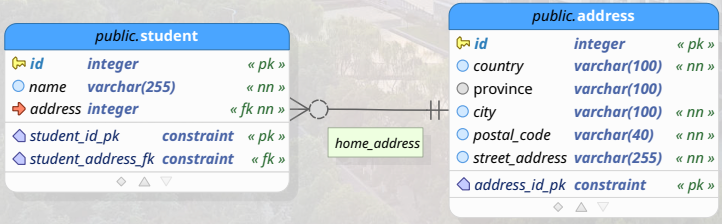
Verletzung der 1. Normalform durch Mehrwertige Attribute

- Setzen wir unser Beispiel von vorhin fort.
- Wir hatten eine Tabellenstruktur zum Speichern von Adressen von Studenten entwickelt.



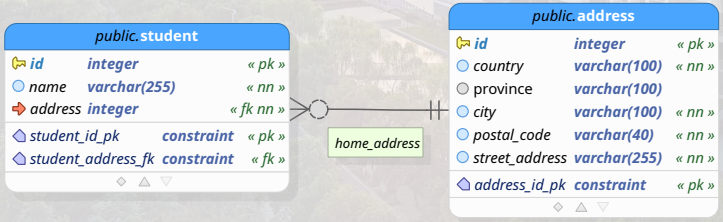
Verletzung der 1. Normalform durch Mehrwertige Attribute

- Setzen wir unser Beispiel von vorhin fort.
- Wir hatten eine Tabellenstruktur zum Speichern von Adressen von Studenten entwickelt.
- Jeder Student hatte genau eine Adresse.



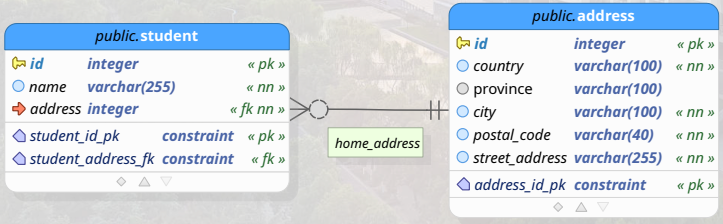
Verletzung der 1. Normalform durch Mehrwertige Attribute

- Setzen wir unser Beispiel von vorhin fort.
- Wir hatten eine Tabellenstruktur zum Speichern von Adressen von Studenten entwickelt.
- Jeder Student hatte genau eine Adresse.
- In der realen Welt ist das aber nicht so . . . dort können Studenten mehrere Adressen haben.



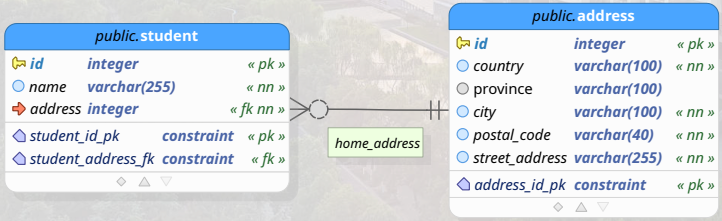
Verletzung der 1. Normalform durch Mehrwertige Attribute

- Setzen wir unser Beispiel von vorhin fort.
- Wir hatten eine Tabellenstruktur zum Speichern von Adressen von Studenten entwickelt.
- Jeder Student hatte genau eine Adresse.
- In der realen Welt ist das aber nicht so . . . dort können Studenten mehrere Adressen haben.
- Vielleicht haben sie ein Zimmer im Wohnheim der Uni und als zweite Adresse die Wohnung ihrer Eltern.



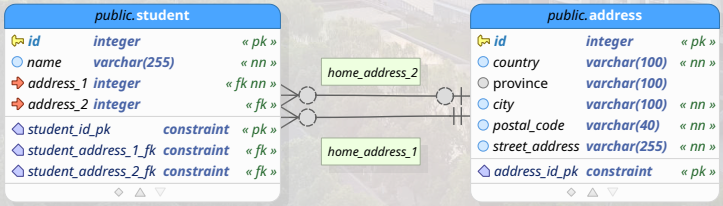
Verletzung der 1. Normalform durch Mehrwertige Attribute

- Wir hatten eine Tabellenstruktur zum Speichern von Adressen von Studenten entwickelt.
- Jeder Student hatte genau eine Adresse.
- In der realen Welt ist das aber nicht so . . . dort können Studenten mehrere Adressen haben.
- Vielleicht haben sie ein Zimmer im Wohnheim der Uni und als zweite Adresse die Wohnung ihrer Eltern.
- Das kann mit unserem Modell nicht implementiert werden.



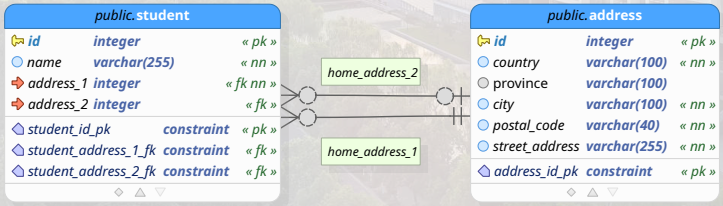
Verletzung der 1. Normalform durch Mehrwertige Attribute

- Jeder Student hatte genau eine Adresse.
- In der realen Welt ist das aber nicht so ... dort können Studenten mehrere Adressen haben.
- Vielleicht haben sie ein Zimmer im Wohnheim der Uni und als zweite Adresse die Wohnung ihrer Eltern.
- Das kann mit unserem Modell nicht implementiert werden.
- Jemand hatte deshalb die Idee, einfach zwei Spalten für Adressen zu machen.



Verletzung der 1. Normalform durch Mehrwertige Attribute

- In der realen Welt ist das aber nicht so ... dort können Studenten mehrere Adressen haben.
- Vielleicht haben sie ein Zimmer im Wohnheim der Uni und als zweite Adresse die Wohnung ihrer Eltern.
- Das kann mit unserem Modell nicht implementiert werden.
- Jemand hatte deshalb die Idee, einfach zwei Spalten für Adressen zu machen.
- Die Tabelle `student` hat nun die Spalten `address_1` and `address_2`.



Verletzung der 1. Normalform durch Mehrwertige Attribute

- Vielleicht haben sie ein Zimmer im Wohnheim der Uni und als zweite Adresse die Wohnung ihrer Eltern.
- Das kann mit unserem Modell nicht implementiert werden.
- Jemand hatte deshalb die Idee, einfach zwei Spalten für Adressen zu machen.
- Die Tabelle `student` hat nun die Spalten `address_1` and `address_2`.
- Und sie verletzte die 1NF.

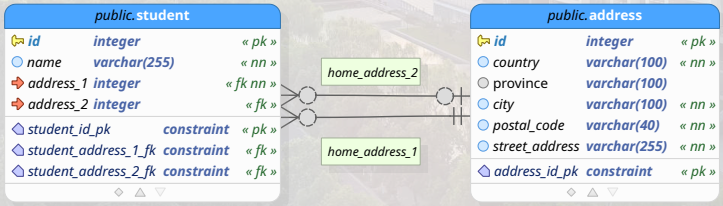


Tabelle student

- Hier sehen wir das vom PgModeler generierte Skript zum Erstellen der neuen Variante der Tabelle `student`.

```
-- object: public.student | type: TABLE --
-- DROP TABLE IF EXISTS public.student CASCADE;
CREATE TABLE public.student (
    id integer NOT NULL GENERATED BY DEFAULT AS IDENTITY ,
    name varchar(255) NOT NULL,
    address integer NOT NULL,
    CONSTRAINT student_id_pk PRIMARY KEY (id)
);
-- ddl-end --
ALTER TABLE public.student OWNER TO postgres;
-- ddl-end --
```

```
$ psql "postgres://postgres:XXX@localhost/anomalies" -v ON_ERROR_STOP=1
    ↪ -ebf 04_public_student_table_5075.sql
CREATE TABLE
ALTER TABLE
# psql 16.11 succeeded with exit code 0.
```

Tabelle student

- Hier sehen wir das vom PgModeler generierte Skript zum Erstellen der neuen Variante der Tabelle `student`.
- Sie hat nochimer Ersatz-Primärschlüssel `id`.

```
-- object: public.student | type: TABLE --
-- DROP TABLE IF EXISTS public.student CASCADE;
CREATE TABLE public.student (
    id integer NOT NULL GENERATED BY DEFAULT AS IDENTITY ,
    name varchar(255) NOT NULL,
    address integer NOT NULL,
    CONSTRAINT student_id_pk PRIMARY KEY (id)
);
-- ddl-end --
ALTER TABLE public.student OWNER TO postgres;
-- ddl-end --
```

```
$ psql "postgres://postgres:XXX@localhost/anomalies" -v ON_ERROR_STOP=1
    ↪ -ebf 04_public_student_table_5075.sql
CREATE TABLE
ALTER TABLE
# psql 16.11 succeeded with exit code 0.
```

Tabelle student

- Hier sehen wir das vom PgModeler generierte Skript zum Erstellen der neuen Variante der Tabelle `student`.
- Sie hat nochimer Ersatz-Primärschlüssel `id`.
- Sie hat nach wie vor auch die Spalte `name` um die Namen der Studenten zu speichern.

```
-- object: public.student | type: TABLE --
-- DROP TABLE IF EXISTS public.student CASCADE;
CREATE TABLE public.student (
    id integer NOT NULL GENERATED BY DEFAULT AS IDENTITY ,
    name varchar(255) NOT NULL,
    address integer NOT NULL,
    CONSTRAINT student_id_pk PRIMARY KEY (id)
);
-- ddl-end --
ALTER TABLE public.student OWNER TO postgres;
-- ddl-end --
```

```
$ psql "postgres://postgres:XXX@localhost/anomalies" -v ON_ERROR_STOP=1
    ↪ -ebf 04_public_student_table_5075.sql
CREATE TABLE
ALTER TABLE
# psql 16.11 succeeded with exit code 0.
```

Tabelle student

- Hier sehen wir das vom PgModeler generierte Skript zum Erstellen der neuen Variante der Tabelle `student`.
- Sie hat nochimer Ersatz-Primärschlüssel `id`.
- Sie hat nach wie vor auch die Spalte `name` um die Namen der Studenten zu speichern.
- Wir haben jetzt zwei Fremdschlüssel, die auf die Tabelle `address` zeigen.

```
1  -- object: public.student | type: TABLE --
2  -- DROP TABLE IF EXISTS public.student CASCADE;
3  CREATE TABLE public.student (
4      id integer NOT NULL GENERATED BY DEFAULT AS IDENTITY ,
5      name varchar(255) NOT NULL,
6      address integer NOT NULL,
7      CONSTRAINT student_id_pk PRIMARY KEY (id)
8  );
9  -- ddl-end --
10 ALTER TABLE public.student OWNER TO postgres;
11 -- ddl-end --
```

```
1  $ psql "postgres://postgres:XXX@localhost/anomalies" -v ON_ERROR_STOP=1
       ↪ -ebf 04_public_student_table_5075.sql
2  CREATE TABLE
3  ALTER TABLE
4  # psql 16.11 succeeded with exit code 0.
```

Tabelle student

- Hier sehen wir das vom PgModeler generierte Skript zum Erstellen der neuen Variante der Tabelle `student`.
- Sie hat nochimer Ersatz-Primärschlüssel `id`.
- Sie hat nach wie vor auch die Spalte `name` um die Namen der Studenten zu speichern.
- Wir haben jetzt zwei Fremdschlüssel, die auf die Tabelle `address` zeigen.
- Diese werden über zwei `REFERENCES`-Einschränkungen gesichert.

```
1  -- object: public.student | type: TABLE --
2  -- DROP TABLE IF EXISTS public.student CASCADE;
3  CREATE TABLE public.student (
4      id integer NOT NULL GENERATED BY DEFAULT AS IDENTITY ,
5      name varchar(255) NOT NULL,
6      address integer NOT NULL,
7      CONSTRAINT student_id_pk PRIMARY KEY (id)
8  );
9  -- ddl-end --
10 ALTER TABLE public.student OWNER TO postgres;
11 -- ddl-end --
```

```
1  $ psql "postgres://postgres:XXX@localhost/anomalies" -v ON_ERROR_STOP=1
       ↪ -ebf 04_public_student_table_5075.sql
2  CREATE TABLE
3  ALTER TABLE
4  # psql 16.11 succeeded with exit code 0.
```

Tabelle student

- Sie hat nochimer Ersatz-Primärschlüssel `id`.
- Sie hat nach wie vor auch die Spalte `name` um die Namen der Studenten zu speichern.
- Wir haben jetzt zwei Fremdschlüssel, die auf die Tabelle `address` zeigen.
- Diese werden über zwei `REFERENCES`-Einschränkungen gesichert.
- Die Skripte dafür und zum Erstellen bzw. später zum Löschen der Datenbank lassen wir hier aber weg.

```
-- object: public.student | type: TABLE --
-- DROP TABLE IF EXISTS public.student CASCADE;
CREATE TABLE public.student (
    id integer NOT NULL GENERATED BY DEFAULT AS IDENTITY ,
    name varchar(255) NOT NULL,
    address integer NOT NULL,
    CONSTRAINT student_id_pk PRIMARY KEY (id)
);
-- ddl-end --
ALTER TABLE public.student OWNER TO postgres;
-- ddl-end --
```

```
$ psql "postgres://postgres:XXX@localhost/anomalies" -v ON_ERROR_STOP=1
    ↪ -ebf 04_public_student_table_5075.sql
CREATE TABLE
ALTER TABLE
# psql 16.11 succeeded with exit code 0.
```

Tabelle student

- Sie hat nach wie vor auch die Spalte `name` um die Namen der Studenten zu speichern.
- Wir haben jetzt zwei Fremdschlüssel, die auf die Tabelle `address` zeigen.
- Diese werden über zwei `REFERENCES`-Einschränkungen gesichert.
- Die Skripte dafür und zum Erstellen bzw. später zum Löschen der Datenbank lassen wir hier aber weg.
- Die Tabelle `address` lassen wir unverändert, weshalb wir auch das Skript dafür nicht nochmal angucken.

```
-- object: public.student | type: TABLE --
-- DROP TABLE IF EXISTS public.student CASCADE;
CREATE TABLE public.student (
    id integer NOT NULL GENERATED BY DEFAULT AS IDENTITY ,
    name varchar(255) NOT NULL,
    address integer NOT NULL,
    CONSTRAINT student_id_pk PRIMARY KEY (id)
);
-- ddl-end --
ALTER TABLE public.student OWNER TO postgres;
-- ddl-end --
```

```
$ psql "postgres://postgres:XXX@localhost/anomalies" -v ON_ERROR_STOP=1
    ↪ -ebf 04_public_student_table_5075.sql
CREATE TABLE
ALTER TABLE
# psql 16.11 succeeded with exit code 0.
```

Probleme (1)

- Diese Verletzung der 1NF kann verschiedene Probleme auslösen.

Probleme (1)

- Diese Verletzung der 1NF kann verschiedene Probleme auslösen.
- Zuest gibt es da Probleme auf der Ebene des Entwurfs.

Probleme (1)

- Diese Verletzung der 1NF kann verschiedene Probleme auslösen.
- Zuest gibt es da Probleme auf der Ebene des Entwurfs.
- Was machen wir z. B., wenn ein Student eine dritte Adresse braucht?

Probleme (1)

- Diese Verletzung der 1NF kann verschiedene Probleme auslösen.
- Zuest gibt es da Probleme auf der Ebene des Entwurfs.
- Was machen wir z. B., wenn ein Student eine dritte Adresse braucht?
- Vielleicht hat ein Student ja getrennt lebende Eltern und eine eigene Adresse, also insgesamt drei Adressen.

Probleme (1)

- Diese Verletzung der 1NF kann verschiedene Probleme auslösen.
- Zuest gibt es da Probleme auf der Ebene des Entwurfs.
- Was machen wir z. B., wenn ein Student eine dritte Adresse braucht?
- Vielleicht hat ein Student ja getrennt lebende Eltern und eine eigene Adresse, also insgesamt drei Adressen.
- Wenn wir die „wiederholende Gruppe“ (EN: *repeating group*)-Methode weitermachen, würden wir einfach eine weitere Spalte `address_3` hinzufügen.

Probleme (1)

- Diese Verletzung der 1NF kann verschiedene Probleme auslösen.
- Zuest gibt es da Probleme auf der Ebene des Entwurfs.
- Was machen wir z. B., wenn ein Student eine dritte Adresse braucht?
- Vielleicht hat ein Student ja getrennt lebende Eltern und eine eigene Adresse, also insgesamt drei Adressen.
- Wenn wir die „wiederholende Gruppe“ (EN: *repeating group*)-Methode weitermachen, würden wir einfach eine weitere Spalte `address_3` hinzufügen.
- Was passiert, wenn eine Person vier Adressen hat?

Probleme (1)

- Diese Verletzung der 1NF kann verschiedene Probleme auslösen.
- Zuest gibt es da Probleme auf der Ebene des Entwurfs.
- Was machen wir z. B., wenn ein Student eine dritte Adresse braucht?
- Vielleicht hat ein Student ja getrennt lebende Eltern und eine eigene Adresse, also insgesamt drei Adressen.
- Wenn wir die „wiederholende Gruppe“ (EN: *repeating group*)-Methode weitermachen, würden wir einfach eine weitere Spalte `address_3` hinzufügen.
- Was passiert, wenn eine Person vier Adressen hat?
- Werden wir immer mehr Spalten hinzufügen, wenn weitere Spezialfälle auftreten?

Probleme (1)

- Diese Verletzung der 1NF kann verschiedene Probleme auslösen.
- Zuest gibt es da Probleme auf der Ebene des Entwurfs.
- Was machen wir z. B., wenn ein Student eine dritte Adresse braucht?
- Vielleicht hat ein Student ja getrennt lebende Eltern und eine eigene Adresse, also insgesamt drei Adressen.
- Wenn wir die „wiederholende Gruppe“ (EN: *repeating group*)-Methode weitermachen, würden wir einfach eine weitere Spalte `address_3` hinzufügen.
- Was passiert, wenn eine Person vier Adressen hat?
- Werden wir immer mehr Spalten hinzufügen, wenn weitere Spezialfälle auftreten?
- Solche Veränderungen sind ja nicht nur einfach einzelne, isolierte Änderungen.

Probleme (1)

- Diese Verletzung der 1NF kann verschiedene Probleme auslösen.
- Zuest gibt es da Probleme auf der Ebene des Entwurfs.
- Was machen wir z. B., wenn ein Student eine dritte Adresse braucht?
- Vielleicht hat ein Student ja getrennt lebende Eltern und eine eigene Adresse, also insgesamt drei Adressen.
- Wenn wir die „wiederholende Gruppe“ (EN: *repeating group*)-Methode weitermachen, würden wir einfach eine weitere Spalte `address_3` hinzufügen.
- Was passiert, wenn eine Person vier Adressen hat?
- Werden wir immer mehr Spalten hinzufügen, wenn weitere Spezialfälle auftreten?
- Solche Veränderungen sind ja nicht nur einfach einzelne, isolierte Änderungen.
- Wir können ja viele Anfragen und Anwendungen haben, die die Adressen verwenden.

Probleme (1)

- Diese Verletzung der 1NF kann verschiedene Probleme auslösen.
- Zuest gibt es da Probleme auf der Ebene des Entwurfs.
- Was machen wir z. B., wenn ein Student eine dritte Adresse braucht?
- Vielleicht hat ein Student ja getrennt lebende Eltern und eine eigene Adresse, also insgesamt drei Adressen.
- Wenn wir die „wiederholende Gruppe“ (EN: *repeating group*)-Methode weitermachen, würden wir einfach eine weitere Spalte `address_3` hinzufügen.
- Was passiert, wenn eine Person vier Adressen hat?
- Werden wir immer mehr Spalten hinzufügen, wenn weitere Spezialfälle auftreten?
- Solche Veränderungen sind ja nicht nur einfach einzelne, isolierte Änderungen.
- Wir können ja viele Anfragen und Anwendungen haben, die die Adressen verwenden.
- Wir würden also jede von ihnen updaten müssen.

Probleme (2)

- Ein weiteres Problem ist, wie wir dann wissen können, wie viele Adressen ein Student hat?

Probleme (2)

- Ein weiteres Problem ist, wie wir dann wissen können, wie viele Adressen ein Student hat?
- In unserem logischen Modell haben wir gesagt, dass `address_1` `NOT NULL` sein muss.

Probleme (2)

- Ein weiteres Problem ist, wie wir dann wissen können, wie viele Adressen ein Student hat?
- In unserem logischen Modell haben wir gesagt, dass `address_1` `NOT NULL` sein muss.
- Die Spalte `address_2` muss natürlich `NULL` sein dürfen.

Probleme (2)

- Ein weiteres Problem ist, wie wir dann wissen können, wie viele Adressen ein Student hat?
- In unserem logischen Modell haben wir gesagt, dass `address_1` `NOT NULL` sein muss.
- Die Spalte `address_2` muss natürlich `NULL` sein dürfen.
- Ein Student hat also eine oder zwei Adressen.

Probleme (2)

- Ein weiteres Problem ist, wie wir dann wissen können, wie viele Adressen ein Student hat?
- In unserem logischen Modell haben wir gesagt, dass `address_1` `NOT NULL` sein muss.
- Die Spalte `address_2` muss natürlich `NULL` sein dürfen.
- Ein Student hat also eine oder zwei Adressen.
- Sagen wir, dass wir jetzt eine dritte Adressspalte hinzufügen.

Probleme (2)

- Ein weiteres Problem ist, wie wir dann wissen können, wie viele Adressen ein Student hat?
- In unserem logischen Modell haben wir gesagt, dass `address_1` `NOT NULL` sein muss.
- Die Spalte `address_2` muss natürlich `NULL` sein dürfen.
- Ein Student hat also eine oder zwei Adressen.
- Sagen wir, dass wir jetzt eine dritte Adressspalte hinzufügen.
- Dann könnten wir eine Zeile bekommen, wo die zweite Addressspalte `NULL` ist aber die dritte nicht, oder andersherum.

Probleme (2)

- Ein weiteres Problem ist, wie wir dann wissen können, wie viele Adressen ein Student hat?
- In unserem logischen Modell haben wir gesagt, dass `address_1` `NOT NULL` sein muss.
- Die Spalte `address_2` muss natürlich `NULL` sein dürfen.
- Ein Student hat also eine oder zwei Adressen.
- Sagen wir, dass wir jetzt eine dritte Adressspalte hinzufügen.
- Dann könnten wir eine Zeile bekommen, wo die zweite Addressspalte `NULL` ist aber die dritte nicht, oder andersherum.
- Wir müssten also die Anfragen komplizierter aufbauen.

Probleme (2)

- Ein weiteres Problem ist, wie wir dann wissen können, wie viele Adressen ein Student hat?
- In unserem logischen Modell haben wir gesagt, dass `address_1` `NOT NULL` sein muss.
- Die Spalte `address_2` muss natürlich `NULL` sein dürfen.
- Ein Student hat also eine oder zwei Adressen.
- Sagen wir, dass wir jetzt eine dritte Adressspalte hinzufügen.
- Dann könnten wir eine Zeile bekommen, wo die zweite Addressspalte `NULL` ist aber die dritte nicht, oder andersherum.
- Wir müssten also die Anfragen komplizierter aufbauen.
- Weil wir mehrere Adressspalten haben werden sie ja sowieso komplizierter.

Probleme (2)

- Ein weiteres Problem ist, wie wir dann wissen können, wie viele Adressen ein Student hat?
- In unserem logischen Modell haben wir gesagt, dass `address_1` `NOT NULL` sein muss.
- Die Spalte `address_2` muss natürlich `NULL` sein dürfen.
- Ein Student hat also eine oder zwei Adressen.
- Sagen wir, dass wir jetzt eine dritte Adressspalte hinzufügen.
- Dann könnten wir eine Zeile bekommen, wo die zweite Addressspalte `NULL` ist aber die dritte nicht, oder andersherum.
- Wir müssten also die Anfragen komplizierter aufbauen.
- Weil wir mehrere Adressspalten haben werden sie ja sowieso komplizierter.
- Was machen wir, wenn wir eine Liste von Studenten mit mindestens einer Adresse in China brauchen?

Probleme (2)

- Ein weiteres Problem ist, wie wir dann wissen können, wie viele Adressen ein Student hat?
- In unserem logischen Modell haben wir gesagt, dass `address_1` `NOT NULL` sein muss.
- Die Spalte `address_2` muss natürlich `NULL` sein dürfen.
- Ein Student hat also eine oder zwei Adressen.
- Sagen wir, dass wir jetzt eine dritte Adressspalte hinzufügen.
- Dann könnten wir eine Zeile bekommen, wo die zweite Addressspalte `NULL` ist aber die dritte nicht, oder andersherum.
- Wir müssten also die Anfragen komplizierter aufbauen.
- Weil wir mehrere Adressspalten haben werden sie ja sowieso komplizierter.
- Was machen wir, wenn wir eine Liste von Studenten mit mindestens einer Adresse in China brauchen?
- Ugh...

Daten Einfügen

- Herr Bibbo hat nun zwei Adressen in Hefei, eine an unserer Hefei University (合肥大学) und eine bei der University of Science and Technology of China (中国科学技术大学, USTC).

```
/** Insert data into the database. */

-- Insert several address records.
INSERT INTO address (
        country, province, city, postal_code, street_address) VALUES
    ('China', 'Anhui', 'Hefei', '230601', 'Jinkaiqu, Hefei University'),
    ('China', 'Anhui', 'Hefei', '230026', 'USTC'),
    ('Deutschland', 'Sachsen', 'Chemnitz', '09111', 'Am Rathaus 1'),
    ('USA', 'NY', 'New York', '10013', 'Canal Street 4, Chinatown'),
    ('Deutschland', 'Sachsen', 'Chemnitz', '09111', 'TU Chemnitz'),
    ('PRC', 'Fujian', 'Quanzhou', '362002', 'West Street'),
    ('P.R.C.', NULL, 'Beijing', '100084', 'Tsinghua University'),
    ('Spain', 'Andalusia', 'Granada', '18009', 'Alhambra de Granada');


-- Create the student records.
INSERT INTO student (name, address_1, address_2) VALUES
    ('Bibbo', 1, 2), ('Bebbo', 3, NULL), ('Bibbi', 4, NULL),
    ('Babbo', 5, 6), ('Bebbe', 7, 8);
```

```
$ psql "postgres://postgres:XXX@localhost/anomalies" -v ON_ERROR_STOP=1
    ↪ -ebf insert.sql
INSERT 0 8
INSERT 0 5
# psql 16.11 succeeded with exit code 0.
```

Daten Einfügen

- Herr Bibbo hat nun zwei Adressen in Hefei, eine an unserer Hefei University (合肥大学) und eine bei der University of Science and Technology of China (中国科学技术大学, USTC).
- Herr Bebbo hat immer noch nur eine Adresse, in Chemnitz, Deutschland.

```
/** Insert data into the database. */

-- Insert several address records.
INSERT INTO address (
        country, province, city, postal_code, street_address) VALUES
    ('China', 'Anhui', 'Hefei', '230601', 'Jinkaiqu, Hefei University'),
    ('China', 'Anhui', 'Hefei', '230026', 'USTC'),
    ('Deutschland', 'Sachsen', 'Chemnitz', '09111', 'Am Rathaus 1'),
    ('USA', 'NY', 'New York', '10013', 'Canal Street 4, Chinatown'),
    ('Deutschland', 'Sachsen', 'Chemnitz', '09111', 'TU Chemnitz'),
    ('PRC', 'Fujian', 'Quanzhou', '362002', 'West Street'),
    ('P.R.C.', NULL, 'Beijing', '100084', 'Tsinghua University'),
    ('Spain', 'Andalusia', 'Granada', '18009', 'Alhambra de Granada');


-- Create the student records.
INSERT INTO student (name, address_1, address_2) VALUES
    ('Bibbo', 1, 2), ('Bebbo', 3, NULL), ('Bibbi', 4, NULL),
    ('Babbo', 5, 6), ('Bebbe', 7, 8);
```

```
$ psql "postgres://postgres:XXX@localhost/anomalies" -v ON_ERROR_STOP=1
    ↪ -ebf insert.sql
INSERT 0 8
INSERT 0 5
# psql 16.11 succeeded with exit code 0.
```

Daten Einfügen

- Herr Bibbo hat nun zwei Adressen in Hefei, eine an unserer Hefei University (合肥大学) und eine bei der University of Science and Technology of China (中国科学技术大学, USTC).
- Herr Bebbo hat immer noch nur eine Adresse, in Chemnitz, Deutschland.
- Frau Bibbi lebt nur in Chinatown, New York, USA.

```sql
/** Insert data into the database. */

-- Insert several address records.
INSERT INTO address (
        country, province, city, postal_code, street_address) VALUES
    ('China', 'Anhui', 'Hefei', '230601', 'Jinkaiqu, Hefei University'),
    ('China', 'Anhui', 'Hefei', '230026', 'USTC'),
    ('Deutschland', 'Sachsen', 'Chemnitz', '09111', 'Am Rathaus 1'),
    ('USA', 'NY', 'New York', '10013', 'Canal Street 4, Chinatown'),
    ('Deutschland', 'Sachsen', 'Chemnitz', '09111', 'TU Chemnitz'),
    ('PRC', 'Fujian', 'Quanzhou', '362002', 'West Street'),
    ('P.R.C.', NULL, 'Beijing', '100084', 'Tsinghua University'),
    ('Spain', 'Andalusia', 'Granada', '18009', 'Alhambra de Granada');


-- Create the student records.
INSERT INTO student (name, address_1, address_2) VALUES
    ('Bibbo', 1, 2), ('Bebbo', 3, NULL), ('Bibbi', 4, NULL),
    ('Babbo', 5, 6), ('Bebbe', 7, 8);
```

```
$ psql "postgres://postgres:XXX@localhost/anomalies" -v ON_ERROR_STOP=1
    ↪ -ebf insert.sql
INSERT 0 8
INSERT 0 5
# psql 16.11 succeeded with exit code 0.
```

Daten Einfügen

- Herr Bibbo hat nun zwei Adressen in Hefei, eine an unserer Hefei University (合肥大学) und eine bei der University of Science and Technology of China (中国科学技术大学, USTC).
- Herr Bebbo hat immer noch nur eine Adresse, in Chemnitz, Deutschland.
- Frau Bibbi lebt nur in Chinatown, New York, USA.
- Herr Babbo hat auch eine Adresse in Chemnitz, Deutschland, aber eben auch eine in Quanzhou (福建省泉州市), China.

```sql
/** Insert data into the database. */

-- Insert several address records.
INSERT INTO address (
        country, province, city, postal_code, street_address) VALUES
    ('China', 'Anhui', 'Hefei', '230601', 'Jinkaiqu, Hefei University'),
    ('China', 'Anhui', 'Hefei', '230026', 'USTC'),
    ('Deutschland', 'Sachsen', 'Chemnitz', '09111', 'Am Rathaus 1'),
    ('USA', 'NY', 'New York', '10013', 'Canal Street 4, Chinatown'),
    ('Deutschland', 'Sachsen', 'Chemnitz', '09111', 'TU Chemnitz'),
    ('PRC', 'Fujian', 'Quanzhou', '362002', 'West Street'),
    ('P.R.C.', NULL, 'Beijing', '100084', 'Tsinghua University'),
    ('Spain', 'Andalusia', 'Granada', '18009', 'Alhambra de Granada');


-- Create the student records.
INSERT INTO student (name, address_1, address_2) VALUES
    ('Bibbo', 1, 2), ('Bebbo', 3, NULL), ('Bibbi', 4, NULL),
    ('Babbo', 5, 6), ('Bebbe', 7, 8);
```

```
$ psql "postgres://postgres:XXX@localhost/anomalies" -v ON_ERROR_STOP=1
    ↪ -ebf insert.sql
INSERT 0 8
INSERT 0 5
# psql 16.11 succeeded with exit code 0.
```

Daten Einfügen

- Herr Bibbo hat nun zwei Adressen in Hefei, eine an unserer Hefei University (合肥大学) und eine bei der University of Science and Technology of China (中国科学技术大学, USTC).
- Herr Bebbo hat immer noch nur eine Adresse, in Chemnitz, Deutschland.
- Frau Bibbi lebt nur in Chinatown, New York, USA.
- Herr Babbo hat auch eine Adresse in Chemnitz, Deutschland, aber eben auch eine in Quanzhou (福建省泉州市), China.
- Die erste Adresse von Frau Bebbe ist in Beijing (北京), aber sie hat noch eine Zweitadresse in Spanien.

```sql
/** Insert data into the database. */

-- Insert several address records.
INSERT INTO address (
        country, province, city, postal_code, street_address) VALUES
    ('China', 'Anhui', 'Hefei', '230601', 'Jinkaiqu, Hefei University'),
    ('China', 'Anhui', 'Hefei', '230026', 'USTC'),
    ('Deutschland', 'Sachsen', 'Chemnitz', '09111', 'Am Rathaus 1'),
    ('USA', 'NY', 'New York', '10013', 'Canal Street 4, Chinatown'),
    ('Deutschland', 'Sachsen', 'Chemnitz', '09111', 'TU Chemnitz'),
    ('PRC', 'Fujian', 'Quanzhou', '362002', 'West Street'),
    ('P.R.C.', NULL, 'Beijing', '100084', 'Tsinghua University'),
    ('Spain', 'Andalusia', 'Granada', '18009', 'Alhambra de Granada');


-- Create the student records.
INSERT INTO student (name, address_1, address_2) VALUES
    ('Bibbo', 1, 2), ('Bebbo', 3, NULL), ('Bibbi', 4, NULL),
    ('Babbo', 5, 6), ('Bebbe', 7, 8);
```

```
$ psql "postgres://postgres:XXX@localhost/anomalies" -v ON_ERROR_STOP=1
    ↪ -ebf insert.sql
INSERT 0 8
INSERT 0 5
# psql 16.11 succeeded with exit code 0.
```

Daten Einfügen

- Herr Bebbo hat immer noch nur eine Adresse, in Chemnitz, Deutschland.
- Frau Bibbi lebt nur in Chinatown, New York, USA.
- Herr Babbo hat auch eine Adresse in Chemnitz, Deutschland, aber eben auch eine in Quanzhou (福建省泉州市), China.
- Die erste Adresse von Frau Bebbe ist in Beijing (北京), aber sie hat noch eine Zweitadresse in Spanien.
- Das Beispiel deckt also alle möglichen Kombinationen ab.

```
/** Insert data into the database. */

-- Insert several address records.
INSERT INTO address (
        country, province, city, postal_code, street_address) VALUES
    ('China', 'Anhui', 'Hefei', '230601', 'Jinkaiqu, Hefei University'),
    ('China', 'Anhui', 'Hefei', '230026', 'USTC'),
    ('Deutschland', 'Sachsen', 'Chemnitz', '09111', 'Am Rathaus 1'),
    ('USA', 'NY', 'New York', '10013', 'Canal Street 4, Chinatown'),
    ('Deutschland', 'Sachsen', 'Chemnitz', '09111', 'TU Chemnitz'),
    ('PRC', 'Fujian', 'Quanzhou', '362002', 'West Street'),
    ('P.R.C.', NULL, 'Beijing', '100084', 'Tsinghua University'),
    ('Spain', 'Andalusia', 'Granada', '18009', 'Alhambra de Granada');


-- Create the student records.
INSERT INTO student (name, address_1, address_2) VALUES
    ('Bibbo', 1, 2), ('Bebbo', 3, NULL), ('Bibbi', 4, NULL),
    ('Babbo', 5, 6), ('Bebbe', 7, 8);
```

```
$ psql "postgres://postgres:XXX@localhost/anomalies" -v ON_ERROR_STOP=1
    ↪ -ebf insert.sql
INSERT 0 8
INSERT 0 5
# psql 16.11 succeeded with exit code 0.
```

Daten Einfügen

- Frau Bibbi lebt nur in Chinatown, New York, USA.
- Herr Babbo hat auch eine Adresse in Chemnitz, Deutschland, aber eben auch eine in Quanzhou (福建省泉州市), China.
- Die erste Adresse von Frau Bebbe ist in Beijing (北京), aber sie hat noch eine Zweitadresse in Spanien.
- Das Beispiel deckt also alle möglichen Kombinationen ab.
- Manche Leute haben keine Adresse in China, bei anderen ist nur die erste oder zweite Adresse in China, und manche haben zwei chinesische Adressen.

```
/** Insert data into the database. */

-- Insert several address records.
INSERT INTO address (
        country, province, city, postal_code, street_address) VALUES
    ('China', 'Anhui', 'Hefei', '230601', 'Jinkaiqu, Hefei University'),
    ('China', 'Anhui', 'Hefei', '230026', 'USTC'),
    ('Deutschland', 'Sachsen', 'Chemnitz', '09111', 'Am Rathaus 1'),
    ('USA', 'NY', 'New York', '10013', 'Canal Street 4, Chinatown'),
    ('Deutschland', 'Sachsen', 'Chemnitz', '09111', 'TU Chemnitz'),
    ('PRC', 'Fujian', 'Quanzhou', '362002', 'West Street'),
    ('P.R.C.', NULL, 'Beijing', '100084', 'Tsinghua University'),
    ('Spain', 'Andalusia', 'Granada', '18009', 'Alhambra de Granada');


-- Create the student records.
INSERT INTO student (name, address_1, address_2) VALUES
    ('Bibbo', 1, 2), ('Bebbo', 3, NULL), ('Bibbi', 4, NULL),
    ('Babbo', 5, 6), ('Bebbe', 7, 8);
```

```
$ psql "postgres://postgres:XXX@localhost/anomalies" -v ON_ERROR_STOP=1
    ↪ -ebf insert.sql
INSERT 0 8
INSERT 0 5
# psql 16.11 succeeded with exit code 0.
```

Daten Einfügen

- Herr Babbo hat auch eine Adresse in Chemnitz, Deutschland, aber eben auch eine in Quanzhou (福建省泉州市), China.
- Die erste Adresse von Frau Bebbe ist in Beijing (北京), aber sie hat noch eine Zweitadresse in Spanien.
- Das Beispiel deckt also alle möglichen Kombinationen ab.
- Manche Leute haben keine Adresse in China, bei anderen ist nur die erste oder zweite Adresse in China, und manche haben zwei chinesische Adressen.
- Von den fünf Studenten haben die Herren bibbo und Babbo sowie Frau Bebbe eine Adresse in China.

```sql
/** Insert data into the database. */

-- Insert several address records.
INSERT INTO address (
        country, province, city, postal_code, street_address) VALUES
    ('China', 'Anhui', 'Hefei', '230601', 'Jinkaiqu, Hefei University'),
    ('China', 'Anhui', 'Hefei', '230026', 'USTC'),
    ('Deutschland', 'Sachsen', 'Chemnitz', '09111', 'Am Rathaus 1'),
    ('USA', 'NY', 'New York', '10013', 'Canal Street 4, Chinatown'),
    ('Deutschland', 'Sachsen', 'Chemnitz', '09111', 'TU Chemnitz'),
    ('PRC', 'Fujian', 'Quanzhou', '362002', 'West Street'),
    ('P.R.C.', NULL, 'Beijing', '100084', 'Tsinghua University'),
    ('Spain', 'Andalusia', 'Granada', '18009', 'Alhambra de Granada');


-- Create the student records.
INSERT INTO student (name, address_1, address_2) VALUES
    ('Bibbo', 1, 2), ('Bebbo', 3, NULL), ('Bibbi', 4, NULL),
    ('Babbo', 5, 6), ('Bebbe', 7, 8);
```

```
$ psql "postgres://postgres:XXX@localhost/anomalies" -v ON_ERROR_STOP=1
    ↪ -ebf insert.sql
INSERT 0 8
INSERT 0 5
# psql 16.11 succeeded with exit code 0.
```

Daten auslesen

- Wie bekommen wir nun eine Liste mit diesen drei Studenten?

```sql
/** Get a list of students with at least one address in China. */

-- Select the student id, student name, and address as three columns.
     SELECT name, address_1 AS adr FROM student
UNION SELECT name, address_2 AS adr FROM student
    WHERE address_2 IS NOT NULL;
```

```
$ psql "postgres://postgres:XXX@localhost/anomalies" -v ON_ERROR_STOP=1
    ↪ -ebf select_1.sql
 name  | adr
-------+-----
 Bebbe |   7
 Babbo |   5
 Bibbo |   1
 Bibbo |   2
 Bebbo |   3
 Bebbe |   8
 Bibbi |   4
 Babbo |   6
(8 rows)

# psql 16.11 succeeded with exit code 0.
```

Daten auslesen

- Wie bekommen wir nun eine Liste mit diesen drei Studenten?
- Egal wie wir das Problem angehen, wir müssen irgendwie den selben Ausdruck auf beide Adressspalten anwenden.

```sql
/** Get a list of students with at least one address in China. */

-- Select the student id, student name, and address as three columns.
     SELECT name, address_1 AS adr FROM student
UNION SELECT name, address_2 AS adr FROM student
    WHERE address_2 IS NOT NULL;
```

```
$ psql "postgres://postgres:XXX@localhost/anomalies" -v ON_ERROR_STOP=1
    ↪ -ebf select_1.sql
 name  | adr
-------+-----
 Bebbe |   7
 Babbo |   5
 Bibbo |   1
 Bibbo |   2
 Bebbo |   3
 Bebbe |   8
 Bibbi |   4
 Babbo |   6
(8 rows)

# psql 16.11 succeeded with exit code 0.
```

Daten auslesen

- Wie bekommen wir nun eine Liste mit diesen drei Studenten?
- Egal wie wir das Problem angehen, wir müssen irgendwie den selben Ausdruck auf beide Adressspalten anwenden.
- Zuerst wollen wir daher versuchen, die Daten irgendwie in eine Form zu bringen dass wir pro Zeile den Namen eines Studenten und einen zugehörigen Fremdschlüsselwert auf die Adresstabelle haben.

```
/** Get a list of students with at least one address in China. */

-- Select the student id, student name, and address as three columns.
      SELECT name, address_1 AS adr FROM student
UNION SELECT name, address_2 AS adr FROM student
    WHERE address_2 IS NOT NULL;
```

```
$ psql "postgres://postgres:XXX@localhost/anomalies" -v ON_ERROR_STOP=1
  ↪ -ebf select_1.sql
 name  | adr
-------+-----
 Bebbe |   7
 Babbo |   5
 Bibbo |   1
 Bibbo |   2
 Bebbo |   3
 Bebbe |   8
 Bibbi |   4
 Babbo |   6
(8 rows)

# psql 16.11 succeeded with exit code 0.
```

Daten auslesen

- Wie bekommen wir nun eine Liste mit diesen drei Studenten?
- Egal wie wir das Problem angehen, wir müssen irgendwie den selben Ausdruck auf beide Adressspalten anwenden.
- Zuerst wollen wir daher versuchen, die Daten irgendwie in eine Form zu bringen dass wir pro Zeile den Namen eines Studenten und einen zugehörigen Fremdschlüsselwert auf die Adresstabelle haben.
- Das können wir mit einer `UNION`-Anfrage machen.

```sql
/** Get a list of students with at least one address in China. */

-- Select the student id, student name, and address as three columns.
     SELECT name, address_1 AS adr FROM student
UNION SELECT name, address_2 AS adr FROM student
    WHERE address_2 IS NOT NULL;
```

```
$ psql "postgres://postgres:XXX@localhost/anomalies" -v ON_ERROR_STOP=1
  ↪ -ebf select_1.sql
 name  | adr
-------+-----
 Bebbe |   7
 Babbo |   5
 Bibbo |   1
 Bibbo |   2
 Bebbo |   3
 Bebbe |   8
 Bibbi |   4
 Babbo |   6
(8 rows)

# psql 16.11 succeeded with exit code 0.
```

Daten auslesen

- Egal wie wir das Problem angehen, wir müssen irgendwie den selben Ausdruck auf beide Adressspalten anwenden.
- Zuerst wollen wir daher versuchen, die Daten irgendwie in eine Form zu bringen dass wir pro Zeile den Namen eines Studenten und einen zugehörigen Fremdschlüsselwert auf die Adresstabelle haben.
- Das können wir mit einer `UNION`-Anfrage machen.
- Im ersten Schritt machen wir eine Anfrage, die den Studenname `name` und die erste Addressspalte `address_1` (die wir mit `AS` in `adr` umbennen) auswählt.

```sql
/** Get a list of students with at least one address in China. */

-- Select the student id, student name, and address as three columns.
     SELECT name, address_1 AS adr FROM student
UNION SELECT name, address_2 AS adr FROM student
    WHERE address_2 IS NOT NULL;
```

```
$ psql "postgres://postgres:XXX@localhost/anomalies" -v ON_ERROR_STOP=1
    ↪ -ebf select_1.sql
 name  | adr
-------+-----
 Bebbe |   7
 Babbo |   5
 Bibbo |   1
 Bibbo |   2
 Bebbo |   3
 Bebbe |   8
 Bibbi |   4
 Babbo |   6
(8 rows)

# psql 16.11 succeeded with exit code 0.
```

Daten auslesen

- Das können wir mit einer `UNION`-Anfrage machen.
- Im ersten Schritt machen wir eine Anfrage, die den Studenname `name` und die erste Addressspalte `address_1` (die wir mit `AS` in `adr` umbennen) auswählt.
- Dann hängen wir das Ergebnis einer zweiten Anfrage an, die fast das selbe macht aber `address_2` als `adr` auswählt.

```
/** Get a list of students with at least one address in China. */

-- Select the student id, student name, and address as three columns.
    SELECT name, address_1 AS adr FROM student
UNION SELECT name, address_2 AS adr FROM student
    WHERE address_2 IS NOT NULL;
```

```
$ psql "postgres://postgres:XXX@localhost/anomalies" -v ON_ERROR_STOP=1
    ↪ -ebf select_1.sql
 name  | adr
-------+-----
 Bebbe |   7
 Babbo |   5
 Bibbo |   1
 Bibbo |   2
 Bebbo |   3
 Bebbe |   8
 Bibbi |   4
 Babbo |   6
(8 rows)

# psql 16.11 succeeded with exit code 0.
```

Daten auslesen

- Das können wir mit einer `UNION`-Anfrage machen.
- Im ersten Schritt machen wir eine Anfrage, die den Studenname `name` und die erste Addressspalte `address_1` (die wir mit `AS` in `adr` umbennen) auswählt.
- Dann hängen wir das Ergebnis einer zweiten Anfrage an, die fast das selbe macht aber `address_2` als `adr` auswählt.
- Die beiden Anfragen werden über `UNION` kombiniert.

```
1  /** Get a list of students with at least one address in China. */
2
3  -- Select the student id, student name, and address as three columns.
4      SELECT name, address_1 AS adr FROM student
5  UNION SELECT name, address_2 AS adr FROM student
6      WHERE address_2 IS NOT NULL;
```

```
1  $ psql "postgres://postgres:XXX@localhost/anomalies" -v ON_ERROR_STOP=1
     ↪ -ebf select_1.sql
2   name  | adr
3  -------+-----
4   Bebbe |   7
5   Babbo |   5
6   Bibbo |   1
7   Bibbo |   2
8   Bebbo |   3
9   Bebbe |   8
10  Bibbi |   4
11  Babbo |   6
12 (8 rows)
13
14 # psql 16.11 succeeded with exit code 0.
```

Daten auslesen

- Das können wir mit einer `UNION`-Anfrage machen.
- Im ersten Schritt machen wir eine Anfrage, die den Studenname `name` und die erste Addressspalte `address_1` (die wir mit `AS` in `adr` umbennen) auswählt.
- Dann hängen wir das Ergebnis einer zweiten Anfrage an, die fast das selbe macht aber `address_2` als `adr` auswählt.
- Die beiden Anfragen werden über `UNION` kombiniert.
- Wir bekommen acht Zeilen ... und diese Relation ist in der 1NF.

```
/** Get a list of students with at least one address in China. */

-- Select the student id, student name, and address as three columns.
    SELECT name, address_1 AS adr FROM student
UNION SELECT name, address_2 AS adr FROM student
    WHERE address_2 IS NOT NULL;
```

```
$ psql "postgres://postgres:XXX@localhost/anomalies" -v ON_ERROR_STOP=1
    ↪ -ebf select_1.sql
 name  | adr
-------+-----
 Bebbe |   7
 Babbo |   5
 Bibbo |   1
 Bibbo |   2
 Bebbo |   3
 Bebbe |   8
 Bibbi |   4
 Babbo |   6
(8 rows)

# psql 16.11 succeeded with exit code 0.
```

Daten auslesen

- Im ersten Schritt machen wir eine Anfrage, die den Studenname `name` und die erste Addressspalte `address_1` (die wir mit `AS` in `adr` umbennen) auswählt.
- Dann hängen wir das Ergebnis einer zweiten Anfrage an, die fast das selbe macht aber `address_2` als `adr` auswählt.
- Die beiden Anfragen werden über `UNION` kombiniert.
- Wir bekommen acht Zeilen ... und diese Relation ist in der 1NF.
- Mit der wiederhergestellten 1NF können wir nun die Leute mit einer Adresse in China herausfiltern.

```
/** Get a list of students with at least one address in China. */

-- Select the student id, student name, and address as three columns.
     SELECT name, address_1 AS adr FROM student
UNION SELECT name, address_2 AS adr FROM student
    WHERE address_2 IS NOT NULL;
```

```
$ psql "postgres://postgres:XXX@localhost/anomalies" -v ON_ERROR_STOP=1
  ↪ -ebf select_1.sql
 name  | adr
-------+-----
 Bebbe |   7
 Babbo |   5
 Bibbo |   1
 Bibbo |   2
 Bebbo |   3
 Bebbe |   8
 Bibbi |   4
 Babbo |   6
(8 rows)

# psql 16.11 succeeded with exit code 0.
```

Daten auslesen

- Dann hängen wir das Ergebnis einer zweiten Anfrage an, die fast das selbe macht aber `address_2` als `adr` auswählt.
- Die beiden Anfragen werden über `UNION` kombiniert.
- Wir bekommen acht Zeilen ... und diese Relation ist in der 1NF.
- Mit der wiederhergestellten 1NF können wir nun die Leute mit einer Adresse in China herausfiltern.
- Wir können das fast genauso machen, wie vorhin, nämlich mit einem `INNER JOIN` und dem Array-basierten `ILIKE`.

```
/** Get a list of students with at least one address in China. */

-- Use the previous query as *subquery* to construct the overall result.
SELECT name, city || ', ' || street_address AS address FROM (
              SELECT name, address_1 AS adr FROM student
        UNION SELECT name, address_2 AS adr FROM student)
    INNER JOIN address ON adr = address.id
    WHERE country ILIKE ANY(ARRAY['%china%', '%PRC%', '%P.R.C.%']);
```

```
$ psql "postgres://postgres:XXX@localhost/anomalies" -v ON_ERROR_STOP=1
    ↪ -ebf select_2.sql
 name  |              address
-------+-----------------------------------
 Bebbe | Beijing, Tsinghua University
 Bibbo | Hefei, Jinkaiqu, Hefei University
 Bibbo | Hefei, USTC
 Babbo | Quanzhou, West Street
(4 rows)

# psql 16.11 succeeded with exit code 0.
```

Daten auslesen

- Die beiden Anfragen werden über `UNION` kombiniert.
- Wir bekommen acht Zeilen ... und diese Relation ist in der 1NF.
- Mit der wiederhergestellten 1NF können wir nun die Leute mit einer Adresse in China herausfiltern.
- Wir können das fast genauso machen, wie vorhin, nämlich mit einem `INNER JOIN` und dem Array-basierten `ILIKE`.
- Wir können nur nicht die Tabelle `student` als Datenquelle nehmen.

```sql
/** Get a list of students with at least one address in China. */

-- Use the previous query as *subquery* to construct the overall result.
SELECT name, city || ', ' || street_address AS address FROM (
              SELECT name, address_1 AS adr FROM student
        UNION SELECT name, address_2 AS adr FROM student)
    INNER JOIN address ON adr = address.id
    WHERE country ILIKE ANY(ARRAY['%china%', '%PRC%', '%P.R.C.%']);
```

```
$ psql "postgres://postgres:XXX@localhost/anomalies" -v ON_ERROR_STOP=1
    ↪ -ebf select_2.sql
 name  |              address
-------+-----------------------------------
 Bebbe | Beijing, Tsinghua University
 Bibbo | Hefei, Jinkaiqu, Hefei University
 Bibbo | Hefei, USTC
 Babbo | Quanzhou, West Street
(4 rows)

# psql 16.11 succeeded with exit code 0.
```

Daten auslesen

- Wir bekommen acht Zeilen ... und diese Relation ist in der 1NF.
- Mit der wiederhergestellten 1NF können wir nun die Leute mit einer Adresse in China herausfiltern.
- Wir können das fast genauso machen, wie vorhin, nämlich mit einem `INNER JOIN` und dem Array-basierten `ILIKE`.
- Wir können nur nicht die Tabelle `student` als Datenquelle nehmen.
- Stattdessen verwenden wir die `UNION`-Anfrage als *Unteranfrage*.

```sql
/** Get a list of students with at least one address in China. */

-- Use the previous query as *subquery* to construct the overall result.
SELECT name, city || ', ' || street_address AS address FROM (
             SELECT name, address_1 AS adr FROM student
        UNION SELECT name, address_2 AS adr FROM student)
    INNER JOIN address ON adr = address.id
    WHERE country ILIKE ANY(ARRAY['%china%', '%PRC%', '%P.R.C.%']);
```

```
$ psql "postgres://postgres:XXX@localhost/anomalies" -v ON_ERROR_STOP=1
    ↪ -ebf select_2.sql
 name  |              address
-------+-----------------------------------
 Bebbe | Beijing, Tsinghua University
 Bibbo | Hefei, Jinkaiqu, Hefei University
 Bibbo | Hefei, USTC
 Babbo | Quanzhou, West Street
(4 rows)

# psql 16.11 succeeded with exit code 0.
```

Daten auslesen

- Mit der wiederhergestellten 1NF können wir nun die Leute mit einer Adresse in China herausfiltern.
- Wir können das fast genauso machen, wie vorhin, nämlich mit einem `INNER JOIN` und dem Array-basierten `ILIKE`.
- Wir können nur nicht die Tabelle `student` als Datenquelle nehmen.
- Stattdessen verwenden wir die `UNION`-Anfrage als *Unteranfrage*.
- In SQL kann man auch die Ergebnisse eines `SELECT...FROM` als Quelle für eine weitere Anfrage nehmen.

```sql
/** Get a list of students with at least one address in China. */

-- Use the previous query as *subquery* to construct the overall result.
SELECT name, city || ', ' || street_address AS address FROM (
              SELECT name, address_1 AS adr FROM student
        UNION SELECT name, address_2 AS adr FROM student)
    INNER JOIN address ON adr = address.id
    WHERE country ILIKE ANY(ARRAY['%china%', '%PRC%', '%P.R.C.%']);
```

```
$ psql "postgres://postgres:XXX@localhost/anomalies" -v ON_ERROR_STOP=1
    ↪ -ebf select_2.sql
 name  |              address
-------+-----------------------------------
 Bebbe | Beijing, Tsinghua University
 Bibbo | Hefei, Jinkaiqu, Hefei University
 Bibbo | Hefei, USTC
 Babbo | Quanzhou, West Street
(4 rows)

# psql 16.11 succeeded with exit code 0.
```

Daten auslesen

- Wir können das fast genauso machen, wie vorhin, nämlich mit einem `INNER JOIN` und dem Array-basierten `ILIKE`.
- Wir können nur nicht die Tabelle `student` als Datenquelle nehmen.
- Stattdessen verwenden wir die `UNION`-Anfrage als *Unteranfrage*.
- In SQL kann man auch die Ergebnisse eines `SELECT...FROM` als Quelle für eine weitere Anfrage nehmen.
- Alles, was dafür notwendig ist, ist die Unteranfrage in Klammern `(...)` zu schreiben!

```sql
/** Get a list of students with at least one address in China. */

-- Use the previous query as *subquery* to construct the overall result.
SELECT name, city || ', ' || street_address AS address FROM (
              SELECT name, address_1 AS adr FROM student
        UNION SELECT name, address_2 AS adr FROM student)
    INNER JOIN address ON adr = address.id
    WHERE country ILIKE ANY(ARRAY['%china%', '%PRC%', '%P.R.C.%']);
```

```
$ psql "postgres://postgres:XXX@localhost/anomalies" -v ON_ERROR_STOP=1
    ↪ -ebf select_2.sql
 name  |              address
-------+-----------------------------------
 Bebbe | Beijing, Tsinghua University
 Bibbo | Hefei, Jinkaiqu, Hefei University
 Bibbo | Hefei, USTC
 Babbo | Quanzhou, West Street
(4 rows)

# psql 16.11 succeeded with exit code 0.
```

Daten auslesen

- Wir können nur nicht die Tabelle `student` als Datenquelle nehmen.
- Stattdessen verwenden wir die `UNION`-Anfrage als *Unteranfrage*.
- In SQL kann man auch die Ergebnisse eines `SELECT...FROM` als Quelle für eine weitere Anfrage nehmen.
- Alles, was dafür notwendig ist, ist die Unteranfrage in Klammern `(...)` zu schreiben!
- Die zweite Anfrage sieht also unserer alten Anfrage von der Bedingung her recht ähnlich.

```sql
/** Get a list of students with at least one address in China. */

-- Use the previous query as *subquery* to construct the overall result.
SELECT name, city || ', ' || street_address AS address FROM (
             SELECT name, address_1 AS adr FROM student
       UNION SELECT name, address_2 AS adr FROM student)
    INNER JOIN address ON adr = address.id
    WHERE country ILIKE ANY(ARRAY['%china%', '%PRC%', '%P.R.C.%']);
```

```
$ psql "postgres://postgres:XXX@localhost/anomalies" -v ON_ERROR_STOP=1
    ↪ -ebf select_2.sql
 name  |              address
-------+-----------------------------------
 Bebbe | Beijing, Tsinghua University
 Bibbo | Hefei, Jinkaiqu, Hefei University
 Bibbo | Hefei, USTC
 Babbo | Quanzhou, West Street
(4 rows)

# psql 16.11 succeeded with exit code 0.
```

Daten auslesen

- Stattdessen verwenden wir die `UNION`-Anfrage als *Unteranfrage*.
- In SQL kann man auch die Ergebnisse eines `SELECT...FROM` als Quelle für eine weitere Anfrage nehmen.
- Alles, was dafür notwendig ist, ist die Unteranfrage in Klammern `(...)` zu schreiben!
- Die zweite Anfrage sieht also unserer alten Anfrage von der Bedingung her recht ähnlich.
- Wir haben nur die Datenquelle (damals die Tabelle `student`) mit unserer `UNION`-Anfrage in Klammern ersetzt (und ein paar Spaltennamen geändert).

```sql
/** Get a list of students with at least one address in China. */

-- Use the previous query as *subquery* to construct the overall result.
SELECT name, city || ', ' || street_address AS address FROM (
             SELECT name, address_1 AS adr FROM student
       UNION SELECT name, address_2 AS adr FROM student)
    INNER JOIN address ON adr = address.id
    WHERE country ILIKE ANY(ARRAY['%china%', '%PRC%', '%P.R.C.%']);
```

```
$ psql "postgres://postgres:XXX@localhost/anomalies" -v ON_ERROR_STOP=1
    ↪ -ebf select_2.sql
 name  |              address
-------+-----------------------------------
 Bebbe | Beijing, Tsinghua University
 Bibbo | Hefei, Jinkaiqu, Hefei University
 Bibbo | Hefei, USTC
 Babbo | Quanzhou, West Street
(4 rows)

# psql 16.11 succeeded with exit code 0.
```

Daten auslesen

- In SQL kann man auch die Ergebnisse eines `SELECT...FROM` als Quelle für eine weitere Anfrage nehmen.
- Alles, was dafür notwendig ist, ist die Unteranfrage in Klammern `(...)` zu schreiben!
- Die zweite Anfrage sieht also unserer alten Anfrage von der Bedingung her recht ähnlich.
- Wir haben nur die Datenquelle (damals die Tabelle `student`) mit unserer `UNION`-Anfrage in Klammern ersetzt (und ein paar Spaltennamen geändert).
- Das funktioniert gut, mit einer Ausnahme.

```
1  /** Get a list of students with at least one address in China. */
2
3  -- Use the previous query as *subquery* to construct the overall result.
4  SELECT name, city || ', ' || street_address AS address FROM (
5                SELECT name, address_1 AS adr FROM student
6          UNION SELECT name, address_2 AS adr FROM student)
7      INNER JOIN address ON adr = address.id
8      WHERE country ILIKE ANY(ARRAY['%china%', '%PRC%', '%P.R.C.%']);
```

```
1  $ psql "postgres://postgres:XXX@localhost/anomalies" -v ON_ERROR_STOP=1
     ↪ -ebf select_2.sql
2   name  |              address
3  -------+-----------------------------------
4   Bebbe | Beijing, Tsinghua University
5   Bibbo | Hefei, Jinkaiqu, Hefei University
6   Bibbo | Hefei, USTC
7   Babbo | Quanzhou, West Street
8  (4 rows)
9
10 # psql 16.11 succeeded with exit code 0.
```

Daten auslesen

- Alles, was dafür notwendig ist, ist die Unteranfrage in Klammern `(...)` zu schreiben!
- Die zweite Anfrage sieht also unserer alten Anfrage von der Bedingung her recht ähnlich.
- Wir haben nur die Datenquelle (damals die Tabelle `student`) mit unserer `UNION`-Anfrage in Klammern ersetzt (und ein paar Spaltennamen geändert).
- Das funktioniert gut, mit einer Ausnahme.
- Herr Bibbo taucht zweimal auf, weil er zwei Adressen in China hat.

```sql
/** Get a list of students with at least one address in China. */

-- Use the previous query as *subquery* to construct the overall result.
SELECT name, city || ', ' || street_address AS address FROM (
              SELECT name, address_1 AS adr FROM student
        UNION SELECT name, address_2 AS adr FROM student)
    INNER JOIN address ON adr = address.id
    WHERE country ILIKE ANY(ARRAY['%china%', '%PRC%', '%P.R.C.%']);
```

```
$ psql "postgres://postgres:XXX@localhost/anomalies" -v ON_ERROR_STOP=1
    ↪ -ebf select_2.sql
 name  |              address
-------+-----------------------------------
 Bebbe | Beijing, Tsinghua University
 Bibbo | Hefei, Jinkaiqu, Hefei University
 Bibbo | Hefei, USTC
 Babbo | Quanzhou, West Street
(4 rows)

# psql 16.11 succeeded with exit code 0.
```

Daten auslesen

- Die zweite Anfrage sieht also unserer alten Anfrage von der Bedingung her recht ähnlich.
- Wir haben nur die Datenquelle (damals die Tabelle `student`) mit unserer `UNION`-Anfrage in Klammern ersetzt (und ein paar Spaltennamen geändert).
- Das funktioniert gut, mit einer Ausnahme.
- Herr Bibbo taucht zweimal auf, weil er zwei Adressen in China hat.
- Zum Glück gibt es das Schlüsselwort `DISTINCT`[59].

```
/** Get a list of students with at least one address in China. */

-- Remove double student entries.
SELECT DISTINCT ON (sid)
    name, city || ', ' || street_address AS address FROM (
              SELECT id AS sid, name, address_1 AS adr FROM student
        UNION SELECT id AS sid, name, address_2 AS adr FROM student)
    INNER JOIN address ON adr = address.id
    WHERE country ILIKE ANY(ARRAY['%china%', '%PRC%', '%P.R.C.%']);
```

```
$ psql "postgres://postgres:XXX@localhost/anomalies" -v ON_ERROR_STOP=1
    ↪ -ebf select_3.sql
 name  |            address
-------+-------------------------------
 Bibbo | Hefei, USTC
 Babbo | Quanzhou, West Street
 Bebbe | Beijing, Tsinghua University
(3 rows)

# psql 16.11 succeeded with exit code 0.
```

Daten auslesen

- Wir haben nur die Datenquelle (damals die Tabelle `student`) mit unserer `UNION`-Anfrage in Klammern ersetzt (und ein paar Spaltennamen geändert).
- Das funktioniert gut, mit einer Ausnahme.
- Herr Bibbo taucht zweimal auf, weil er zwei Adressen in China hat.
- Zum Glück gibt es das Schlüsselwort `DISTINCT`[59].
- `SELECT DISTINCT` löscht alle doppelten Zeilen aus der Anfrage.

```
/** Get a list of students with at least one address in China. */

-- Remove double student entries.
SELECT DISTINCT ON (sid)
    name, city || ', ' || street_address AS address FROM (
              SELECT id AS sid, name, address_1 AS adr FROM student
        UNION SELECT id AS sid, name, address_2 AS adr FROM student)
    INNER JOIN address ON adr = address.id
    WHERE country ILIKE ANY(ARRAY['%china%', '%PRC%', '%P.R.C.%']);
```

```
$ psql "postgres://postgres:XXX@localhost/anomalies" -v ON_ERROR_STOP=1
    ↪ -ebf select_3.sql
 name  |            address
-------+-------------------------------
 Bibbo | Hefei, USTC
 Babbo | Quanzhou, West Street
 Bebbe | Beijing, Tsinghua University
(3 rows)

# psql 16.11 succeeded with exit code 0.
```

Daten auslesen

- Das funktioniert gut, mit einer Ausnahme.
- Herr Bibbo taucht zweimal auf, weil er zwei Adressen in China hat.
- Zum Glück gibt es das Schlüsselwort `DISTINCT`[59].
- `SELECT DISTINCT` löscht alle doppelten Zeilen aus der Anfrage.
- Mit anderen Worten, wenn zwei Ergebniszeilen einer Anfrage komplett gleiche Werte haben, dann wird nur eine davon zurückgegeben und die andere weggeworfen.

```
/** Get a list of students with at least one address in China. */

-- Remove double student entries.
SELECT DISTINCT ON (sid)
    name, city || ', ' || street_address AS address FROM (
            SELECT id AS sid, name, address_1 AS adr FROM student
        UNION SELECT id AS sid, name, address_2 AS adr FROM student)
    INNER JOIN address ON adr = address.id
    WHERE country ILIKE ANY(ARRAY['%china%', '%PRC%', '%P.R.C.%']);
```

```
$ psql "postgres://postgres:XXX@localhost/anomalies" -v ON_ERROR_STOP=1
    ↪ -ebf select_3.sql
 name  |            address
-------+-------------------------------
 Bibbo | Hefei, USTC
 Babbo | Quanzhou, West Street
 Bebbe | Beijing, Tsinghua University
(3 rows)

# psql 16.11 succeeded with exit code 0.
```

Daten auslesen

- Zum Glück gibt es das Schlüsselwort `DISTINCT`[59].
- `SELECT DISTINCT` löscht alle doppelten Zeilen aus der Anfrage.
- Mit anderen Worten, wenn zwei Ergebniszeilen einer Anfrage komplett gleiche Werte haben, dann wird nur eine davon zurückgegeben und die andere weggeworfen.
- `SELECT DISTINCT ON (col1, col2, ...)` benutzt nur eine Untermenge der Spalten (hier `col1`, `col2`, ...) um zu entscheiden, welche Zeile ein Duplikat ist und welche nicht.

```
/** Get a list of students with at least one address in China. */

-- Remove double student entries.
SELECT DISTINCT ON (sid)
    name, city || ', ' || street_address AS address FROM (
              SELECT id AS sid, name, address_1 AS adr FROM student
        UNION SELECT id AS sid, name, address_2 AS adr FROM student)
    INNER JOIN address ON adr = address.id
    WHERE country ILIKE ANY(ARRAY['%china%', '%PRC%', '%P.R.C.%']);
```

```
$ psql "postgres://postgres:XXX@localhost/anomalies" -v ON_ERROR_STOP=1
    ↪ -ebf select_3.sql
 name  |            address
-------+-------------------------------
 Bibbo | Hefei, USTC
 Babbo | Quanzhou, West Street
 Bebbe | Beijing, Tsinghua University
(3 rows)

# psql 16.11 succeeded with exit code 0.
```

Daten auslesen

- Zum Glück gibt es das Schlüsselwort `DISTINCT`[59].
- `SELECT DISTINCT` löscht alle doppelten Zeilen aus der Anfrage.
- Mit anderen Worten, wenn zwei Ergebniszeilen einer Anfrage komplett gleiche Werte haben, dann wird nur eine davon zurückgegeben und die andere weggeworfen.
- `SELECT DISTINCT ON (col1, col2, ...)` benutzt nur eine Untermenge der Spalten (hier `col1`, `col2`, ...) um zu entscheiden, welche Zeile ein Duplikat ist und welche nicht.
- Nun, Namen könnten mehrdeutig sein.

```
1  /** Get a list of students with at least one address in China. */
2
3  -- Remove double student entries.
4  SELECT DISTINCT ON (sid)
5      name, city || ', ' || street_address AS address FROM (
6              SELECT id AS sid, name, address_1 AS adr FROM student
7          UNION SELECT id AS sid, name, address_2 AS adr FROM student)
8      INNER JOIN address ON adr = address.id
9      WHERE country ILIKE ANY(ARRAY['%china%', '%PRC%', '%P.R.C.%']);
```

```
1  $ psql "postgres://postgres:XXX@localhost/anomalies" -v ON_ERROR_STOP=1
       ↪ -ebf select_3.sql
2   name  |            address
3  -------+-------------------------------
4   Bibbo | Hefei, USTC
5   Babbo | Quanzhou, West Street
6   Bebbe | Beijing, Tsinghua University
7  (3 rows)
8
9  # psql 16.11 succeeded with exit code 0.
```

Daten auslesen

- Mit anderen Worten, wenn zwei Ergebniszeilen einer Anfrage komplett gleiche Werte haben, dann wird nur eine davon zurückgegeben und die andere weggeworfen.
- `SELECT DISTINCT ON (col1, col2, ...)` benutzt nur eine Untermenge der Spalten (hier `col1`, `col2`, ...) um zu entscheiden, welche Zeile ein Duplikat ist und welche nicht.
- Nun, Namen könnten mehrdeutig sein.
- Wir verändern also unsere Unteranfragen so, dass sie auch die Studenten-IDs (umbenannt zu `sid`) mit zurückliefern.

```
/** Get a list of students with at least one address in China. */

-- Remove double student entries.
SELECT DISTINCT ON (sid)
    name, city || ', ' || street_address AS address FROM (
            SELECT id AS sid, name, address_1 AS adr FROM student
        UNION SELECT id AS sid, name, address_2 AS adr FROM student)
    INNER JOIN address ON adr = address.id
    WHERE country ILIKE ANY(ARRAY['%china%', '%PRC%', '%P.R.C.%']);
```

```
$ psql "postgres://postgres:XXX@localhost/anomalies" -v ON_ERROR_STOP=1
    ↪ -ebf select_3.sql
 name  |            address
-------+-------------------------------
 Bibbo | Hefei, USTC
 Babbo | Quanzhou, West Street
 Bebbe | Beijing, Tsinghua University
(3 rows)

# psql 16.11 succeeded with exit code 0.
```

Daten auslesen

- `SELECT DISTINCT ON (col1, col2, ...)` benutzt nur eine Untermenge der Spalten (hier `col1`, `col2`, ...) um zu entscheiden, welche Zeile ein Duplikat ist und welche nicht.
- Nun, Namen könnten mehrdeutig sein.
- Wir verändern also unsere Unteranfragen so, dass sie auch die Studenten-IDs (umbenannt zu `sid`) mit zurückliefern.
- Mit dem `DISTINCT ON (sid)` am Anfang unserer Anfrage können wir sicherstellen, dass jeder Student nur einmal auftaucht.

```
/** Get a list of students with at least one address in China. */

-- Remove double student entries.
SELECT DISTINCT ON (sid)
    name, city || ', ' || street_address AS address FROM (
              SELECT id AS sid, name, address_1 AS adr FROM student
        UNION SELECT id AS sid, name, address_2 AS adr FROM student)
    INNER JOIN address ON adr = address.id
    WHERE country ILIKE ANY(ARRAY['%china%', '%PRC%', '%P.R.C.%']);
```

```
$ psql "postgres://postgres:XXX@localhost/anomalies" -v ON_ERROR_STOP=1
    ↪ -ebf select_3.sql
 name  |            address
-------+-------------------------------
 Bibbo | Hefei, USTC
 Babbo | Quanzhou, West Street
 Bebbe | Beijing, Tsinghua University
(3 rows)

# psql 16.11 succeeded with exit code 0.
```

Daten auslesen

- `SELECT DISTINCT ON (col1, col2, ...)` benutzt nur eine Untermenge der Spalten (hier `col1`, `col2`, ...) um zu entscheiden, welche Zeile ein Duplikat ist und welche nicht.
- Nun, Namen könnten mehrdeutig sein.
- Wir verändern also unsere Unteranfragen so, dass sie auch die Studenten-IDs (umbenannt zu `sid`) mit zurückliefern.
- Mit dem `DISTINCT ON (sid)` am Anfang unserer Anfrage können wir sicherstellen, dass jeder Student nur einmal auftaucht.
- Das Ergebnis stimmt nun.

```
/** Get a list of students with at least one address in China. */

-- Remove double student entries.
SELECT DISTINCT ON (sid)
    name, city || ', ' || street_address AS address FROM (
              SELECT id AS sid, name, address_1 AS adr FROM student
        UNION SELECT id AS sid, name, address_2 AS adr FROM student)
    INNER JOIN address ON adr = address.id
    WHERE country ILIKE ANY(ARRAY['%china%', '%PRC%', '%P.R.C.%']);
```

```
$ psql "postgres://postgres:XXX@localhost/anomalies" -v ON_ERROR_STOP=1
    ↪ -ebf select_3.sql
 name  |            address
-------+-------------------------------
 Bibbo | Hefei, USTC
 Babbo | Quanzhou, West Street
 Bebbe | Beijing, Tsinghua University
(3 rows)

# psql 16.11 succeeded with exit code 0.
```

Daten auslesen

- Wir verändern also unsere Unteranfragen so, dass sie auch die Studenten-IDs (umbenannt zu `sid`) mit zurückliefern.
- Mit dem `DISTINCT ON (sid)` am Anfang unserer Anfrage können wir sicherstellen, dass jeder Student nur einmal auftaucht.
- Das Ergebnis stimmt nun.

```sql
/** Get a list of students with at least one address in China. */

-- Select the student id, student name, and address as three columns.
      SELECT name, address_1 AS adr FROM student
UNION SELECT name, address_2 AS adr FROM student
    WHERE address_2 IS NOT NULL;

-- Use the above as *subquery* to construct the overall result.
SELECT name, city || ', ' || street_address AS address FROM (
            SELECT name, address_1 AS adr FROM student
       UNION SELECT name, address_2 AS adr FROM student)
    INNER JOIN address ON adr = address.id
    WHERE country ILIKE ANY(ARRAY['%china%', '%PRC%', '%P.R.C.%']);

-- Remove double student entries.
SELECT DISTINCT ON (sid)
    name, city || ', ' || street_address AS address FROM (
              SELECT id AS sid, name, address_1 AS adr FROM student
       UNION SELECT id AS sid, name, address_2 AS adr FROM student)
    INNER JOIN address ON adr = address.id
    WHERE country ILIKE ANY(ARRAY['%china%', '%PRC%', '%P.R.C.%']);
```

```
$ psql "postgres://postgres:XXX@localhost/anomalies" -v ON_ERROR_STOP=1
    ↪ -ebf select.sql
 name  | adr
-------+-----
 Bebbe |   7
 Babbo |   5
 Bibbo |   1
 Bibbo |   2
 Bebbo |   3
 Bebbe |   8
 Bibbi |   4
 Babbo |   6
(8 rows)

 name  |               address
-------+--------------------------------
 Bebbe | Beijing, Tsinghua University
 Bibbo | Hefei, Jinkaiqu, Hefei University
 Bibbo | Hefei, USTC
 Babbo | Quanzhou, West Street
(4 rows)

 name  |            address
-------+-------------------------------
 Bibbo | Hefei, USTC
 Babbo | Quanzhou, West Street
 Bebbe | Beijing, Tsinghua University
(3 rows)

# psql 16.11 succeeded with exit code 0.
```

Lehre

- Wir haben gelernt: Wenn man irgendetwas Sinnvolles mit mehrwertigen Attributen, die als „wiederholte Gruppe“ dargestellt sind, machen will . . . dann muss man sie in die 1NF bringen.

Lehre

- Wir haben gelernt: Wenn man irgendetwas Sinnvolles mit mehrwertigen Attributen, die als „wiederholte Gruppe“ dargestellt sind, machen will . . . dann muss man sie in die 1NF bringen.
- Das bedeutet, dass wir sie als separate Relation darstellen müssen.

Lehre

- Wir haben gelernt: Wenn man irgendetwas Sinnvolles mit mehrwertigen Attributen, die als „wiederholte Gruppe“ dargestellt sind, machen will . . . dann muss man sie in die 1NF bringen.
- Das bedeutet, dass wir sie als separate Relation darstellen müssen.
- Also genau in der Form, die wir beim Übersetzen von Entitätstypen in Tabellen genannt hatten.

Lehre

- Wir haben gelernt: Wenn man irgendetwas Sinnvolles mit mehrwertigen Attributen, die als „wiederholte Gruppe“ dargestellt sind, machen will . . . dann muss man sie in die 1NF bringen.
- Das bedeutet, dass wir sie als separate Relation darstellen müssen.
- Also genau in der Form, die wir beim Übersetzen von Entitätstypen in Tabellen genannt hatten.
- Warum speichern wir sie also nicht gleich so?

Lehre

- Wir haben gelernt: Wenn man irgendetwas Sinnvolles mit mehrwertigen Attributen, die als „wiederholte Gruppe“ dargestellt sind, machen will … dann muss man sie in die 1NF bringen.
- Das bedeutet, dass wir sie als separate Relation darstellen müssen.
- Also genau in der Form, die wir beim Übersetzen von Entitätstypen in Tabellen genannt hatten.
- Warum speichern wir sie also nicht gleich so?
- Das Modell von vorhin ist einfach nur eine falsche Darstellung einer `student` ⋊○—⊣⋉ `address`-Beziehung.

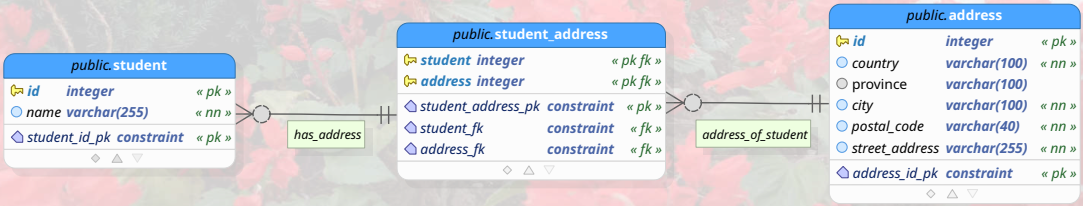
Lehre

- Wir haben gelernt: Wenn man irgendetwas Sinnvolles mit mehrwertigen Attributen, die als „wiederholte Gruppe“ dargestellt sind, machen will … dann muss man sie in die 1NF bringen.
- Das bedeutet, dass wir sie als separate Relation darstellen müssen.
- Also genau in der Form, die wir beim Übersetzen von Entitätstypen in Tabellen genannt hatten.
- Warum speichern wir sie also nicht gleich so?
- Das Modell von vorhin ist einfach nur eine falsche Darstellung einer `student` $\gg\!\circ\!\!-\!\!\dashv\!\ll$ `address`-Beziehung.
- Wir hatten diese korrekt als Q $\gg\!\circ\!\!-\!\!\dashv\!\ll$ R-Beziehung kennengelernt.

Mehrwertige Attribute: Lösung

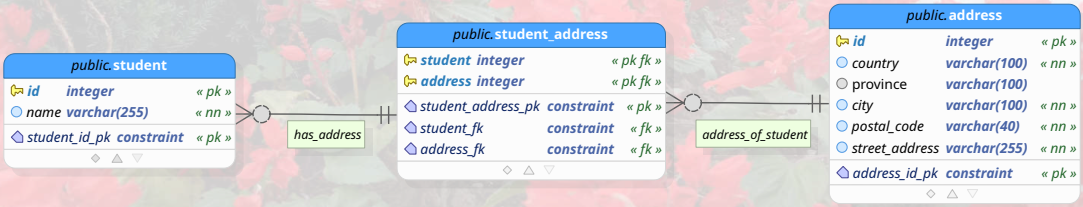
Mehrwertige Attribute: Lösung

- Lösen wir das Problem.



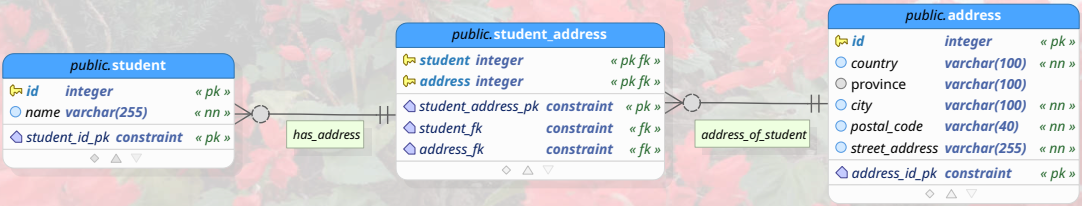
Mehrwertige Attribute: Lösung

- Lösen wir das Problem.
- Wir extrahieren das mehrwertige Attribut in eine weitere Tabelle.



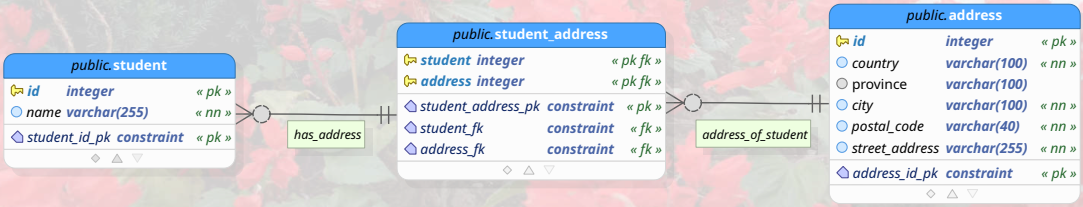
Mehrwertige Attribute: Lösung

- Lösen wir das Problem.
- Wir extrahieren das mehrwertige Attribut in eine weitere Tabelle.
- Um es einfacher zu machen, erzwingen wir nicht, dass jeder Student *mindestens* eine Adresse haben muss.



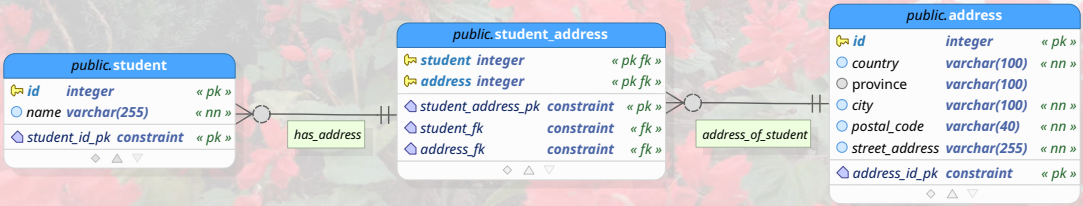
Mehrwertige Attribute: Lösung

- Lösen wir das Problem.
- Wir extrahieren das mehrwertige Attribut in eine weitere Tabelle.
- Um es einfacher zu machen, erzwingen wir nicht, dass jeder Student *mindestens* eine Adresse haben muss.
- Wir implementieren die einfachere Beziehung `student` >o—o< `address`.



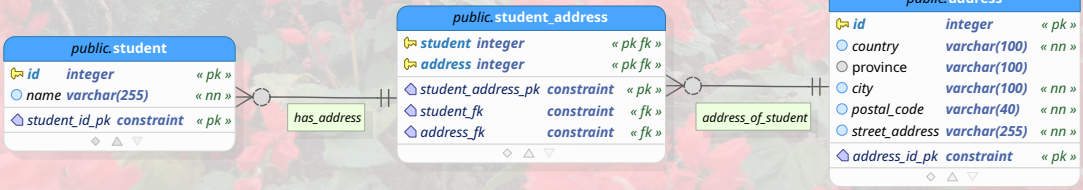
Mehrwertige Attribute: Lösung

- Lösen wir das Problem.
- Wir extrahieren das mehrwertige Attribut in eine weitere Tabelle.
- Um es einfacher zu machen, erzwingen wir nicht, dass jeder Student *mindestens* eine Adresse haben muss.
- Wir implementieren die einfachere Beziehung `student` >o—o< `address`.
- Sie haben ja gelernt, wie es hardcore gehen würden ... also denke ich, das ist OK hier.



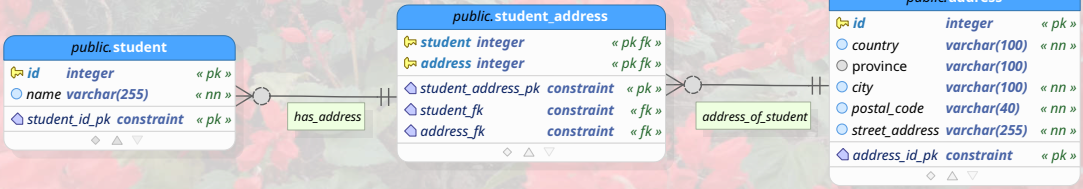
Mehrwertige Attribute: Lösung

- Lösen wir das Problem.
- Wir extrahieren das mehrwertige Attribut in eine weitere Tabelle.
- Um es einfacher zu machen, erzwingen wir nicht, dass jeder Student *mindestens* eine Adresse haben muss.
- Wir implementieren die einfachere Beziehung `student` >o—o< `address`.
- Sie haben ja gelernt, wie es hardcore gehen würden ... also denke ich, das ist OK hier.
- Wir brauchen nun eine dritte Tabelle, die wir `student_address` nennen.



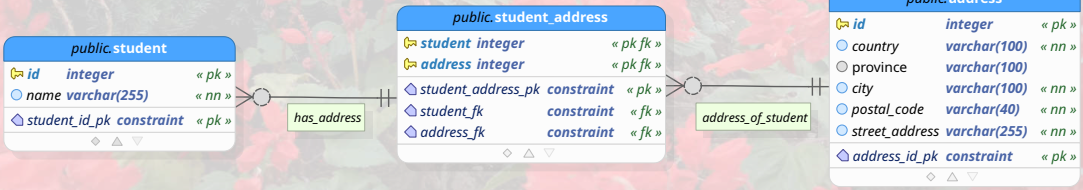
Mehrwertige Attribute: Lösung

- Wir extrahieren das mehrwertige Attribut in eine weitere Tabelle.
- Um es einfacher zu machen, erzwingen wir nicht, dass jeder Student *mindestens* eine Adresse haben muss.
- Wir implementieren die einfachere Beziehung `student` >o—o< `address`.
- Sie haben ja gelernt, wie es hardcore gehen würden ... also denke ich, das ist OK hier.
- Wir brauchen nun eine dritte Tabelle, die wir `student_address` nennen.
- Die Tabelle hat einen zusammengesetzten Primärschlüssel aus einem Fremdschlüssel auf Tabelle `student` und einem auf Tabelle `address`.



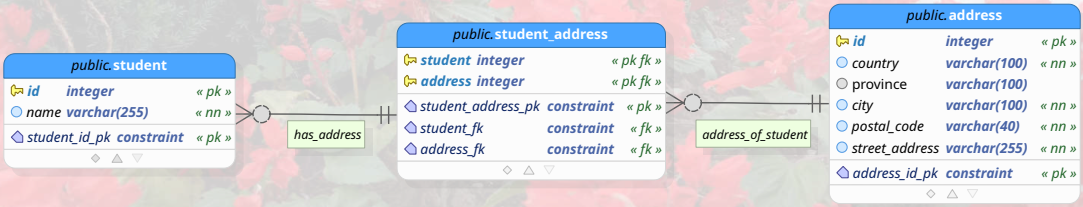
Mehrwertige Attribute: Lösung

- Um es einfacher zu machen, erzwingen wir nicht, dass jeder Student *mindestens* eine Adresse haben muss.
- Wir implementieren die einfachere Beziehung `student` >o—o< `address`.
- Sie haben ja gelernt, wie es hardcore gehen würden ... also denke ich, das ist OK hier.
- Wir brauchen nun eine dritte Tabelle, die wir `student_address` nennen.
- Die Tabelle hat einen zusammengesetzten Primärschlüssel aus einem Fremdschlüssel auf Tabelle `student` und einem auf Tabelle `address`.
- Die Adressspalten verschwinden aus Tabelle `student`.



Mehrwertige Attribute: Lösung

- Wir implementieren die einfachere Beziehung `student` >o—o< `address`.
- Sie haben ja gelernt, wie es hardcore gehen würden ... also denke ich, das ist OK hier.
- Wir brauchen nun eine dritte Tabelle, die wir `student_address` nennen.
- Die Tabelle hat einen zusammengesetzten Primärschlüssel aus einem Fremdschlüssel auf Tabelle `student` und einem auf Tabelle `address`.
- Die Adressspalten verschwinden aus Tabelle `student`.
- Sie hat jetzt nur noch den Primärschlüssel und das Attribut `name` überig.



Mehrwertige Attribute: Lösung

- Sie haben ja gelernt, wie es hardcore gehen würden ... also denke ich, das ist OK hier.
- Wir brauchen nun eine dritte Tabelle, die wir `student_address` nennen.
- Die Tabelle hat einen zusammengesetzten Primärschlüssel aus einem Fremdschlüssel auf Tabelle `student` und einem auf Tabelle `address`.
- Die Adressspalten verschwinden aus Tabelle `student`.
- Sie hat jetzt nur noch den Primärschlüssel und das Attribut `name` überig.
- Die Tabelle `address` verändert sich nicht.

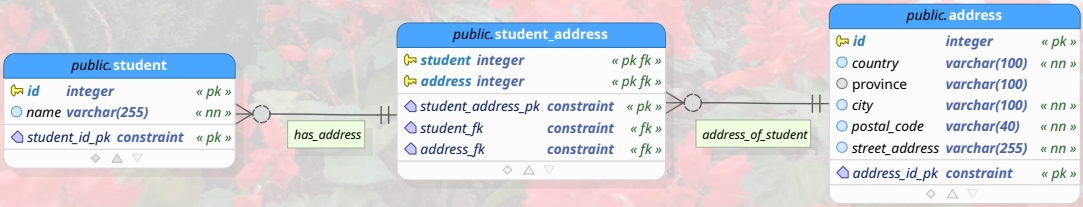


Tabelle student

- Hier ist das auto-generierte Skript zum Erstellen der Tabelle `student`.

```
-- object: public.student | type: TABLE --
-- DROP TABLE IF EXISTS public.student CASCADE;
CREATE TABLE public.student (
    id integer NOT NULL GENERATED BY DEFAULT AS IDENTITY ,
    name varchar(255) NOT NULL,
    CONSTRAINT student_id_pk PRIMARY KEY (id)
);
-- ddl-end --
ALTER TABLE public.student OWNER TO postgres;
-- ddl-end --
```

```
$ psql "postgres://postgres:XXX@localhost/fixed" -v ON_ERROR_STOP=1 -ebf
    ↪ 04_public_student_table_5079.sql
CREATE TABLE
ALTER TABLE
# psql 16.11 succeeded with exit code 0.
```

Tabelle student

- Hier ist das auto-generierte Skript zum Erstellen der Tabelle `student`.
- Sie hat nur noch die beiden Spalten `id` und `name`.

```
-- object: public.student | type: TABLE --
-- DROP TABLE IF EXISTS public.student CASCADE;
CREATE TABLE public.student (
    id integer NOT NULL GENERATED BY DEFAULT AS IDENTITY ,
    name varchar(255) NOT NULL,
    CONSTRAINT student_id_pk PRIMARY KEY (id)
);
-- ddl-end --
ALTER TABLE public.student OWNER TO postgres;
-- ddl-end --
```

```
$ psql "postgres://postgres:XXX@localhost/fixed" -v ON_ERROR_STOP=1 -ebf
    ↪ 04_public_student_table_5079.sql
CREATE TABLE
ALTER TABLE
# psql 16.11 succeeded with exit code 0.
```

Tabelle student_address

- Die Tabelle `student_address` verbindet die Studenten-Datensätze mit den Address-Datensätzen.

```
-- object: public.student_address | type: TABLE --
-- DROP TABLE IF EXISTS public.student_address CASCADE;
CREATE TABLE public.student_address (
    student integer NOT NULL,
    address integer NOT NULL,
    CONSTRAINT student_address_pk PRIMARY KEY (student,address)
);
-- ddl-end --
ALTER TABLE public.student_address OWNER TO postgres;
-- ddl-end --
```

```
$ psql "postgres://postgres:XXX@localhost/fixed" -v ON_ERROR_STOP=1 -ebf
    ↪ 05_public_student_address_table_5083.sql
CREATE TABLE
ALTER TABLE
# psql 16.11 succeeded with exit code 0.
```

Tabelle student_address

- Die Tabelle `student_address` verbindet die Studenten-Datensätze mit den Address-Datensätzen.
- Durch den zusammengesetzten Primärschlüssel stellen wir sicher, dass jeder Student mit jeder Address höchstens einmal verbunden sein kann und umgekehrt.

```
-- object: public.student_address | type: TABLE --
-- DROP TABLE IF EXISTS public.student_address CASCADE;
CREATE TABLE public.student_address (
    student integer NOT NULL,
    address integer NOT NULL,
    CONSTRAINT student_address_pk PRIMARY KEY (student,address)
);
-- ddl-end --
ALTER TABLE public.student_address OWNER TO postgres;
-- ddl-end --
```

```
$ psql "postgres://postgres:XXX@localhost/fixed" -v ON_ERROR_STOP=1 -ebf
    ↪ 05_public_student_address_table_5083.sql
CREATE TABLE
ALTER TABLE
# psql 16.11 succeeded with exit code 0.
```

Tabelle student_address

- Die Tabelle `student_address` verbindet die Studenten-Datensätze mit den Address-Datensätzen.
- Durch den zusammengesetzten Primärschlüssel stellen wir sicher, dass jeder Student mit jeder Address höchstens einmal verbunden sein kann und umgekehrt.
- Das ist so, weil `PRIMARY KEY`-Einschränkungen `UNIQUE` implizieren.

```
1  -- object: public.student_address | type: TABLE --
2  -- DROP TABLE IF EXISTS public.student_address CASCADE;
3  CREATE TABLE public.student_address (
4      student integer NOT NULL,
5      address integer NOT NULL,
6      CONSTRAINT student_address_pk PRIMARY KEY (student,address)
7  );
8  -- ddl-end --
9  ALTER TABLE public.student_address OWNER TO postgres;
10 -- ddl-end --
```

```
1 $ psql "postgres://postgres:XXX@localhost/fixed" -v ON_ERROR_STOP=1 -ebf
    ↪ 05_public_student_address_table_5083.sql
2 CREATE TABLE
3 ALTER TABLE
4 # psql 16.11 succeeded with exit code 0.
```

Tabelle student_address

- Die Tabelle `student_address` verbindet die Studenten-Datensätze mit den Address-Datensätzen.
- Durch den zusammengesetzten Primärschlüssel stellen wir sicher, dass jeder Student mit jeder Address höchstens einmal verbunden sein kann und umgekehrt.
- Das ist so, weil `PRIMARY KEY`-Einschränkungen `UNIQUE` implizieren.
- Wir lassen hier die Skripte für die `REFERENCES`-Einschränkungen aus Platzgründen weg.

```
-- object: public.student_address | type: TABLE --
-- DROP TABLE IF EXISTS public.student_address CASCADE;
CREATE TABLE public.student_address (
    student integer NOT NULL,
    address integer NOT NULL,
    CONSTRAINT student_address_pk PRIMARY KEY (student,address)
);
-- ddl-end --
ALTER TABLE public.student_address OWNER TO postgres;
-- ddl-end --
```

```
$ psql "postgres://postgres:XXX@localhost/fixed" -v ON_ERROR_STOP=1 -ebf
    ↪ 05_public_student_address_table_5083.sql
CREATE TABLE
ALTER TABLE
# psql 16.11 succeeded with exit code 0.
```

Daten Einfügen

- Fügen wir nun Daten in unsere normalisierten Tabellen ein.

```sql
/** Insert data into the database. */

-- Insert several address records.
INSERT INTO address (
        country, province, city, postal_code, street_address) VALUES
    ('China', 'Anhui', 'Hefei', '230601', 'Jinkaiqu, Hefei University'),
    ('China', 'Anhui', 'Hefei', '230026', 'USTC'),
    ('Deutschland', 'Sachsen', 'Chemnitz', '09111', 'Am Rathaus 1'),
    ('USA', 'NY', 'New York', '10013', 'Canal Street 4, Chinatown'),
    ('Deutschland', 'Sachsen', 'Chemnitz', '09111', 'TU Chemnitz'),
    ('PRC', 'Fujian', 'Quanzhou', '362002', 'West Street'),
    ('P.R.C.', NULL, 'Beijing', '100084', 'Tsinghua University'),
    ('Spain', 'Andalusia', 'Granada', '18009', 'Alhambra de Granada');

-- Create the five student records.
INSERT INTO student (name) VALUES
    ('Bibbo'), ('Bebbo'), ('Bibbi'), ('Babbo'), ('Bebbe');

-- Establish the relationship to the addresses.
INSERT INTO student_address (student, address) VALUES
    (1, 1), (1, 2), (2, 3), (3, 4), (4, 5), (4, 6), (5, 7), (5, 8);
```

```
$ psql "postgres://postgres:XXX@localhost/fixed" -v ON_ERROR_STOP=1 -ebf
    ↪ insert.sql
INSERT 0 8
INSERT 0 5
INSERT 0 8
# psql 16.11 succeeded with exit code 0.
```

Daten Einfügen

- Fügen wir nun Daten in unsere normalisierten Tabellen ein.
- Wir benutzen die selben Daten wie vorhin.

```
/** Insert data into the database. */

-- Insert several address records.
INSERT INTO address (
        country, province, city, postal_code, street_address) VALUES
    ('China', 'Anhui', 'Hefei', '230601', 'Jinkaiqu, Hefei University'),
    ('China', 'Anhui', 'Hefei', '230026', 'USTC'),
    ('Deutschland', 'Sachsen', 'Chemnitz', '09111', 'Am Rathaus 1'),
    ('USA', 'NY', 'New York', '10013', 'Canal Street 4, Chinatown'),
    ('Deutschland', 'Sachsen', 'Chemnitz', '09111', 'TU Chemnitz'),
    ('PRC', 'Fujian', 'Quanzhou', '362002', 'West Street'),
    ('P.R.C.', NULL, 'Beijing', '100084', 'Tsinghua University'),
    ('Spain', 'Andalusia', 'Granada', '18009', 'Alhambra de Granada');

-- Create the five student records.
INSERT INTO student (name) VALUES
    ('Bibbo'), ('Bebbo'), ('Bibbi'), ('Babbo'), ('Bebbe');

-- Establish the relationship to the addresses.
INSERT INTO student_address (student, address) VALUES
    (1, 1), (1, 2), (2, 3), (3, 4), (4, 5), (4, 6), (5, 7), (5, 8);
```

```
$ psql "postgres://postgres:XXX@localhost/fixed" -v ON_ERROR_STOP=1 -ebf
    ↪ insert.sql
INSERT 0 8
INSERT 0 5
INSERT 0 8
# psql 16.11 succeeded with exit code 0.
```

Daten Einfügen

- Fügen wir nun Daten in unsere normalisierten Tabellen ein.
- Wir benutzen die selben Daten wie vorhin.
- Wir erstellen zuerst die Adress-Datensätze.

```
/** Insert data into the database. */

-- Insert several address records.
INSERT INTO address (
        country, province, city, postal_code, street_address) VALUES
    ('China', 'Anhui', 'Hefei', '230601', 'Jinkaiqu, Hefei University'),
    ('China', 'Anhui', 'Hefei', '230026', 'USTC'),
    ('Deutschland', 'Sachsen', 'Chemnitz', '09111', 'Am Rathaus 1'),
    ('USA', 'NY', 'New York', '10013', 'Canal Street 4, Chinatown'),
    ('Deutschland', 'Sachsen', 'Chemnitz', '09111', 'TU Chemnitz'),
    ('PRC', 'Fujian', 'Quanzhou', '362002', 'West Street'),
    ('P.R.C.', NULL, 'Beijing', '100084', 'Tsinghua University'),
    ('Spain', 'Andalusia', 'Granada', '18009', 'Alhambra de Granada');

-- Create the five student records.
INSERT INTO student (name) VALUES
    ('Bibbo'), ('Bebbo'), ('Bibbi'), ('Babbo'), ('Bebbe');

-- Establish the relationship to the addresses.
INSERT INTO student_address (student, address) VALUES
    (1, 1), (1, 2), (2, 3), (3, 4), (4, 5), (4, 6), (5, 7), (5, 8);
```

```
$ psql "postgres://postgres:XXX@localhost/fixed" -v ON_ERROR_STOP=1 -ebf
    ↪ insert.sql
INSERT 0 8
INSERT 0 5
INSERT 0 8
# psql 16.11 succeeded with exit code 0.
```

Daten Einfügen

- Fügen wir nun Daten in unsere normalisierten Tabellen ein.
- Wir benutzen die selben Daten wie vorhin.
- Wir erstellen zuerst die Adress-Datensätze.
- Dann erstellen wir die Studenten-Datensätze.

```sql
/** Insert data into the database. */

-- Insert several address records.
INSERT INTO address (
        country, province, city, postal_code, street_address) VALUES
    ('China', 'Anhui', 'Hefei', '230601', 'Jinkaiqu, Hefei University'),
    ('China', 'Anhui', 'Hefei', '230026', 'USTC'),
    ('Deutschland', 'Sachsen', 'Chemnitz', '09111', 'Am Rathaus 1'),
    ('USA', 'NY', 'New York', '10013', 'Canal Street 4, Chinatown'),
    ('Deutschland', 'Sachsen', 'Chemnitz', '09111', 'TU Chemnitz'),
    ('PRC', 'Fujian', 'Quanzhou', '362002', 'West Street'),
    ('P.R.C.', NULL, 'Beijing', '100084', 'Tsinghua University'),
    ('Spain', 'Andalusia', 'Granada', '18009', 'Alhambra de Granada');

-- Create the five student records.
INSERT INTO student (name) VALUES
    ('Bibbo'), ('Bebbo'), ('Bibbi'), ('Babbo'), ('Bebbe');

-- Establish the relationship to the addresses.
INSERT INTO student_address (student, address) VALUES
    (1, 1), (1, 2), (2, 3), (3, 4), (4, 5), (4, 6), (5, 7), (5, 8);
```

```
$ psql "postgres://postgres:XXX@localhost/fixed" -v ON_ERROR_STOP=1 -ebf
    ↪ insert.sql
INSERT 0 8
INSERT 0 5
INSERT 0 8
# psql 16.11 succeeded with exit code 0.
```

Daten Einfügen

- Fügen wir nun Daten in unsere normalisierten Tabellen ein.
- Wir benutzen die selben Daten wie vorhin.
- Wir erstellen zuerst die Adress-Datensätze.
- Dann erstellen wir die Studenten-Datensätze.
- Dann fügen wir die Beziehungen zwischen den Studenten- und den Adress-Datensätzen in die Tabelle `student_address` ein.

```sql
/** Insert data into the database. */

-- Insert several address records.
INSERT INTO address (
        country, province, city, postal_code, street_address) VALUES
    ('China', 'Anhui', 'Hefei', '230601', 'Jinkaiqu, Hefei University'),
    ('China', 'Anhui', 'Hefei', '230026', 'USTC'),
    ('Deutschland', 'Sachsen', 'Chemnitz', '09111', 'Am Rathaus 1'),
    ('USA', 'NY', 'New York', '10013', 'Canal Street 4, Chinatown'),
    ('Deutschland', 'Sachsen', 'Chemnitz', '09111', 'TU Chemnitz'),
    ('PRC', 'Fujian', 'Quanzhou', '362002', 'West Street'),
    ('P.R.C.', NULL, 'Beijing', '100084', 'Tsinghua University'),
    ('Spain', 'Andalusia', 'Granada', '18009', 'Alhambra de Granada');

-- Create the five student records.
INSERT INTO student (name) VALUES
    ('Bibbo'), ('Bebbo'), ('Bibbi'), ('Babbo'), ('Bebbe');

-- Establish the relationship to the addresses.
INSERT INTO student_address (student, address) VALUES
    (1, 1), (1, 2), (2, 3), (3, 4), (4, 5), (4, 6), (5, 7), (5, 8);
```

```
$ psql "postgres://postgres:XXX@localhost/fixed" -v ON_ERROR_STOP=1 -ebf
    ↪ insert.sql
INSERT 0 8
INSERT 0 5
INSERT 0 8
# psql 16.11 succeeded with exit code 0.
```

Daten Einfügen

- Fügen wir nun Daten in unsere normalisierten Tabellen ein.
- Wir benutzen die selben Daten wie vorhin.
- Wir erstellen zuerst die Adress-Datensätze.
- Dann erstellen wir die Studenten-Datensätze.
- Dann fügen wir die Beziehungen zwischen den Studenten- und den Adress-Datensätzen in die Tabelle `student_address` ein.
- Beachten Sie, dass dieser Schritt jetzt notwendig ist.

```sql
/** Insert data into the database. */

-- Insert several address records.
INSERT INTO address (
        country, province, city, postal_code, street_address) VALUES
    ('China', 'Anhui', 'Hefei', '230601', 'Jinkaiqu, Hefei University'),
    ('China', 'Anhui', 'Hefei', '230026', 'USTC'),
    ('Deutschland', 'Sachsen', 'Chemnitz', '09111', 'Am Rathaus 1'),
    ('USA', 'NY', 'New York', '10013', 'Canal Street 4, Chinatown'),
    ('Deutschland', 'Sachsen', 'Chemnitz', '09111', 'TU Chemnitz'),
    ('PRC', 'Fujian', 'Quanzhou', '362002', 'West Street'),
    ('P.R.C.', NULL, 'Beijing', '100084', 'Tsinghua University'),
    ('Spain', 'Andalusia', 'Granada', '18009', 'Alhambra de Granada');

-- Create the five student records.
INSERT INTO student (name) VALUES
    ('Bibbo'), ('Bebbo'), ('Bibbi'), ('Babbo'), ('Bebbe');

-- Establish the relationship to the addresses.
INSERT INTO student_address (student, address) VALUES
    (1, 1), (1, 2), (2, 3), (3, 4), (4, 5), (4, 6), (5, 7), (5, 8);
```

```
$ psql "postgres://postgres:XXX@localhost/fixed" -v ON_ERROR_STOP=1 -ebf
    ↪ insert.sql
INSERT 0 8
INSERT 0 5
INSERT 0 8
# psql 16.11 succeeded with exit code 0.
```

Daten Einfügen

- Wir erstellen zuerst die Adress-Datensätze.
- Dann erstellen wir die Studenten-Datensätze.
- Dann fügen wir die Beziehungen zwischen den Studenten- und den Adress-Datensätzen in die Tabelle `student_address` ein.
- Beachten Sie, dass dieser Schritt jetzt notwendig ist.
- Als unsere Tabellen nicht in der 1NF waren, konnten wir von den Studenten-Datensätzen aus direkt auf die Adress-Datensätze verweisen.

```sql
/** Insert data into the database. */

-- Insert several address records.
INSERT INTO address (
        country, province, city, postal_code, street_address) VALUES
    ('China', 'Anhui', 'Hefei', '230601', 'Jinkaiqu, Hefei University'),
    ('China', 'Anhui', 'Hefei', '230026', 'USTC'),
    ('Deutschland', 'Sachsen', 'Chemnitz', '09111', 'Am Rathaus 1'),
    ('USA', 'NY', 'New York', '10013', 'Canal Street 4, Chinatown'),
    ('Deutschland', 'Sachsen', 'Chemnitz', '09111', 'TU Chemnitz'),
    ('PRC', 'Fujian', 'Quanzhou', '362002', 'West Street'),
    ('P.R.C.', NULL, 'Beijing', '100084', 'Tsinghua University'),
    ('Spain', 'Andalusia', 'Granada', '18009', 'Alhambra de Granada');

-- Create the five student records.
INSERT INTO student (name) VALUES
    ('Bibbo'), ('Bebbo'), ('Bibbi'), ('Babbo'), ('Bebbe');

-- Establish the relationship to the addresses.
INSERT INTO student_address (student, address) VALUES
    (1, 1), (1, 2), (2, 3), (3, 4), (4, 5), (4, 6), (5, 7), (5, 8);
```

```
$ psql "postgres://postgres:XXX@localhost/fixed" -v ON_ERROR_STOP=1 -ebf
    ↪ insert.sql
INSERT 0 8
INSERT 0 5
INSERT 0 8
# psql 16.11 succeeded with exit code 0.
```

Daten Einfügen

- Dann erstellen wir die Studenten-Datensätze.
- Dann fügen wir die Beziehungen zwischen den Studenten- und den Adress-Datensätzen in die Tabelle `student_address` ein.
- Beachten Sie, dass dieser Schritt jetzt notwendig ist.
- Als unsere Tabellen nicht in der 1NF waren, konnten wir von den Studenten-Datensätzen aus direkt auf die Adress-Datensätze verweisen.
- Jetzt müssen wir den Extra-Schritt über eine neue Tabelle gehen.

```sql
/** Insert data into the database. */

-- Insert several address records.
INSERT INTO address (
        country, province, city, postal_code, street_address) VALUES
    ('China', 'Anhui', 'Hefei', '230601', 'Jinkaiqu, Hefei University'),
    ('China', 'Anhui', 'Hefei', '230026', 'USTC'),
    ('Deutschland', 'Sachsen', 'Chemnitz', '09111', 'Am Rathaus 1'),
    ('USA', 'NY', 'New York', '10013', 'Canal Street 4, Chinatown'),
    ('Deutschland', 'Sachsen', 'Chemnitz', '09111', 'TU Chemnitz'),
    ('PRC', 'Fujian', 'Quanzhou', '362002', 'West Street'),
    ('P.R.C.', NULL, 'Beijing', '100084', 'Tsinghua University'),
    ('Spain', 'Andalusia', 'Granada', '18009', 'Alhambra de Granada');

-- Create the five student records.
INSERT INTO student (name) VALUES
    ('Bibbo'), ('Bebbo'), ('Bibbi'), ('Babbo'), ('Bebbe');

-- Establish the relationship to the addresses.
INSERT INTO student_address (student, address) VALUES
    (1, 1), (1, 2), (2, 3), (3, 4), (4, 5), (4, 6), (5, 7), (5, 8);
```

```
$ psql "postgres://postgres:XXX@localhost/fixed" -v ON_ERROR_STOP=1 -ebf
    ↪ insert.sql
INSERT 0 8
INSERT 0 5
INSERT 0 8
# psql 16.11 succeeded with exit code 0.
```

Daten Einfügen

- Dann erstellen wir die Studenten-Datensätze.
- Dann fügen wir die Beziehungen zwischen den Studenten- und den Adress-Datensätzen in die Tabelle `student_address` ein.
- Beachten Sie, dass dieser Schritt jetzt notwendig ist.
- Als unsere Tabellen nicht in der 1NF waren, konnten wir von den Studenten-Datensätzen aus direkt auf die Adress-Datensätze verweisen.
- Jetzt müssen wir den Extra-Schritt über eine neue Tabelle gehen.
- Das sind die Kosten der 1NF.

```sql
/** Insert data into the database. */

-- Insert several address records.
INSERT INTO address (
        country, province, city, postal_code, street_address) VALUES
    ('China', 'Anhui', 'Hefei', '230601', 'Jinkaiqu, Hefei University'),
    ('China', 'Anhui', 'Hefei', '230026', 'USTC'),
    ('Deutschland', 'Sachsen', 'Chemnitz', '09111', 'Am Rathaus 1'),
    ('USA', 'NY', 'New York', '10013', 'Canal Street 4, Chinatown'),
    ('Deutschland', 'Sachsen', 'Chemnitz', '09111', 'TU Chemnitz'),
    ('PRC', 'Fujian', 'Quanzhou', '362002', 'West Street'),
    ('P.R.C.', NULL, 'Beijing', '100084', 'Tsinghua University'),
    ('Spain', 'Andalusia', 'Granada', '18009', 'Alhambra de Granada');

-- Create the five student records.
INSERT INTO student (name) VALUES
    ('Bibbo'), ('Bebbo'), ('Bibbi'), ('Babbo'), ('Bebbe');

-- Establish the relationship to the addresses.
INSERT INTO student_address (student, address) VALUES
    (1, 1), (1, 2), (2, 3), (3, 4), (4, 5), (4, 6), (5, 7), (5, 8);
```

```
$ psql "postgres://postgres:XXX@localhost/fixed" -v ON_ERROR_STOP=1 -ebf
    ↪ insert.sql
INSERT 0 8
INSERT 0 5
INSERT 0 8
# psql 16.11 succeeded with exit code 0.
```

Daten auslesen

- Die Studenten mit einer Adresse in China zu finden ist nun einfacher.

```sql
/** Get a list of students with at least one address in China. */

SELECT DISTINCT ON (student)
    name, city || ', ' || street_address AS address FROM student_address
    INNER JOIN address ON address = address.id
    INNER JOIN student ON student = student.id
    WHERE country ILIKE ANY(ARRAY['%china%', '%PRC%', '%P.R.C.%']);
```

```
$ psql "postgres://postgres:XXX@localhost/fixed" -v ON_ERROR_STOP=1 -ebf
    ↪ select.sql
 name  |               address
-------+-----------------------------------
 Bibbo | Hefei, Jinkaiqu, Hefei University
 Babbo | Quanzhou, West Street
 Bebbe | Beijing, Tsinghua University
(3 rows)

# psql 16.11 succeeded with exit code 0.
```

Daten auslesen

- Die Studenten mit einer Adresse in China zu finden ist nun einfacher.
- Wir brauchen kein `UNION`-Statement mehr.

```sql
/** Get a list of students with at least one address in China. */

SELECT DISTINCT ON (student)
    name, city || ', ' || street_address AS address FROM student_address
    INNER JOIN address ON address = address.id
    INNER JOIN student ON student = student.id
    WHERE country ILIKE ANY(ARRAY['%china%', '%PRC%', '%P.R.C.%']);
```

```
$ psql "postgres://postgres:XXX@localhost/fixed" -v ON_ERROR_STOP=1 -ebf
    ↪ select.sql
 name  |               address
-------+-----------------------------------
 Bibbo | Hefei, Jinkaiqu, Hefei University
 Babbo | Quanzhou, West Street
 Bebbe | Beijing, Tsinghua University
(3 rows)

# psql 16.11 succeeded with exit code 0.
```

Daten auslesen

- Die Studenten mit einer Adresse in China zu finden ist nun einfacher.
- Wir brauchen kein `UNION`-Statement mehr.
- Wir haben die Student-Addresse-Beziehung ja schon in perfekter Tabellenform.

```
/** Get a list of students with at least one address in China. */

SELECT DISTINCT ON (student)
    name, city || ', ' || street_address AS address FROM student_address
    INNER JOIN address ON address = address.id
    INNER JOIN student ON student = student.id
    WHERE country ILIKE ANY(ARRAY['%china%', '%PRC%', '%P.R.C.%']);
```

```
$ psql "postgres://postgres:XXX@localhost/fixed" -v ON_ERROR_STOP=1 -ebf
    ↪ select.sql
 name  |               address
-------+-----------------------------------
 Bibbo | Hefei, Jinkaiqu, Hefei University
 Babbo | Quanzhou, West Street
 Bebbe | Beijing, Tsinghua University
(3 rows)

# psql 16.11 succeeded with exit code 0.
```

Daten auslesen

- Die Studenten mit einer Adresse in China zu finden ist nun einfacher.
- Wir brauchen kein `UNION`-Statement mehr.
- Wir haben die Student-Addresse-Beziehung ja schon in perfekter Tabellenform.
- Mit nur zwei `INNER JOIN`s bauen wir alle Daten zusammen.

```
/** Get a list of students with at least one address in China. */

SELECT DISTINCT ON (student)
    name, city || ', ' || street_address AS address FROM student_address
    INNER JOIN address ON address = address.id
    INNER JOIN student ON student = student.id
    WHERE country ILIKE ANY(ARRAY['%china%', '%PRC%', '%P.R.C.%']);
```

```
$ psql "postgres://postgres:XXX@localhost/fixed" -v ON_ERROR_STOP=1 -ebf
    ↪ select.sql
 name  |               address
-------+-----------------------------------
 Bibbo | Hefei, Jinkaiqu, Hefei University
 Babbo | Quanzhou, West Street
 Bebbe | Beijing, Tsinghua University
(3 rows)

# psql 16.11 succeeded with exit code 0.
```

Daten auslesen

- Die Studenten mit einer Adresse in China zu finden ist nun einfacher.
- Wir brauchen kein `UNION`-Statement mehr.
- Wir haben die Student-Addresse-Beziehung ja schon in perfekter Tabellenform.
- Mit nur zwei `INNER JOIN`s bauen wir alle Daten zusammen.
- Die Befehle `ILIKE`, `ANY`, `ARRAY` und `DISTINCT ON` können wir unverändert wiederverwenden.

```
/** Get a list of students with at least one address in China. */

SELECT DISTINCT ON (student)
    name, city || ', ' || street_address AS address FROM student_address
    INNER JOIN address ON address = address.id
    INNER JOIN student ON student = student.id
    WHERE country ILIKE ANY(ARRAY['%china%', '%PRC%', '%P.R.C.%']);
```

```
$ psql "postgres://postgres:XXX@localhost/fixed" -v ON_ERROR_STOP=1 -ebf
    ↪ select.sql
 name  |               address
-------+-----------------------------------
 Bibbo | Hefei, Jinkaiqu, Hefei University
 Babbo | Quanzhou, West Street
 Bebbe | Beijing, Tsinghua University
(3 rows)

# psql 16.11 succeeded with exit code 0.
```

Zusammenfassung

Zusammenfassung

- Wenn unsere Daten der 1NF gehorchen, dann bedeutet das, das wir das relationale Datenmodell konsistent auf der logischen Ebene umgesetzt haben.

Zusammenfassung

- Wenn unsere Daten der 1NF gehorchen, dann bedeutet das, das wir das relationale Datenmodell konsistent auf der logischen Ebene umgesetzt haben.
- Oftmals produzieren wir sowieso Modelle, der der 1NF gehorchen, wenn wir das konzeptuelle in das logische Modell übersetzen.

Zusammenfassung

- Wenn unsere Daten der 1NF gehorchen, dann bedeutet das, das wir das relationale Datenmodell konsistent auf der logischen Ebene umgesetzt haben.
- Oftmals produzieren wir sowieso Modelle, der der 1NF gehorchen, wenn wir das konzeptuelle in das logische Modell übersetzen.
- So haben wir das ja gelernt.

Zusammenfassung

- Wenn unsere Daten der 1NF gehorchen, dann bedeutet das, das wir das relationale Datenmodell konsistent auf der logischen Ebene umgesetzt haben.
- Oftmals produzieren wir sowieso Modelle, der der 1NF gehorchen, wenn wir das konzeptuelle in das logische Modell übersetzen.
- So haben wir das ja gelernt.
- Wenn aber das konzeptuelle Modell nicht zur 1NF passt, dann kann es sein, dass wir wiederholende Gruppen von Attributen bekommen, oder ein Attribut, das eigentlich aus zwei Attributen bestehen sollte.

Zusammenfassung

- Wenn unsere Daten der 1NF gehorchen, dann bedeutet das, das wir das relationale Datenmodell konsistent auf der logischen Ebene umgesetzt haben.
- Oftmals produzieren wir sowieso Modelle, der der 1NF gehorchen, wenn wir das konzeptuelle in das logische Modell übersetzen.
- So haben wir das ja gelernt.
- Wenn aber das konzeptuelle Modell nicht zur 1NF passt, dann kann es sein, dass wir wiederholende Gruppen von Attributen bekommen, oder ein Attribut, das eigentlich aus zwei Attributen bestehen sollte.
- Das kann passieren.

Zusammenfassung

- Wenn unsere Daten der 1NF gehorchen, dann bedeutet das, das wir das relationale Datenmodell konsistent auf der logischen Ebene umgesetzt haben.
- Oftmals produzieren wir sowieso Modelle, der der 1NF gehorchen, wenn wir das konzeptuelle in das logische Modell übersetzen.
- So haben wir das ja gelernt.
- Wenn aber das konzeptuelle Modell nicht zur 1NF passt, dann kann es sein, dass wir wiederholende Gruppen von Attributen bekommen, oder ein Attribut, das eigentlich aus zwei Attributen bestehen sollte.
- Das kann passieren.
- Wir haben ja oft gesagt, dass das konzeptuelle Schema technologieunabhängig sein soll.

Zusammenfassung

- Wenn unsere Daten der 1NF gehorchen, dann bedeutet das, das wir das relationale Datenmodell konsistent auf der logischen Ebene umgesetzt haben.
- Oftmals produzieren wir sowieso Modelle, der der 1NF gehorchen, wenn wir das konzeptuelle in das logische Modell übersetzen.
- So haben wir das ja gelernt.
- Wenn aber das konzeptuelle Modell nicht zur 1NF passt, dann kann es sein, dass wir wiederholende Gruppen von Attributen bekommen, oder ein Attribut, das eigentlich aus zwei Attributen bestehen sollte.
- Das kann passieren.
- Wir haben ja oft gesagt, dass das konzeptuelle Schema technologieunabhängig sein soll.
- Dann können wir uns nicht beschweren, wenn es nicht zum relationalen Datenmodell passt.

Zusammenfassung

- Wenn unsere Daten der 1NF gehorchen, dann bedeutet das, das wir das relationale Datenmodell konsistent auf der logischen Ebene umgesetzt haben.
- Oftmals produzieren wir sowieso Modelle, der der 1NF gehorchen, wenn wir das konzeptuelle in das logische Modell übersetzen.
- So haben wir das ja gelernt.
- Wenn aber das konzeptuelle Modell nicht zur 1NF passt, dann kann es sein, dass wir wiederholende Gruppen von Attributen bekommen, oder ein Attribut, das eigentlich aus zwei Attributen bestehen sollte.
- Das kann passieren.
- Wir haben ja oft gesagt, dass das konzeptuelle Schema technologieunabhängig sein soll.
- Dann können wir uns nicht beschweren, wenn es nicht zum relationalen Datenmodell passt.
- Deshalb muss man immer vorsichtig sein während dem Design des logischen Modells.

Zusammenfassung

- Wenn unsere Daten der 1NF gehorchen, dann bedeutet das, das wir das relationale Datenmodell konsistent auf der logischen Ebene umgesetzt haben.
- Oftmals produzieren wir sowieso Modelle, der der 1NF gehorchen, wenn wir das konzeptuelle in das logische Modell übersetzen.
- So haben wir das ja gelernt.
- Wenn aber das konzeptuelle Modell nicht zur 1NF passt, dann kann es sein, dass wir wiederholende Gruppen von Attributen bekommen, oder ein Attribut, das eigentlich aus zwei Attributen bestehen sollte.
- Das kann passieren.
- Wir haben ja oft gesagt, dass das konzeptuelle Schema technologieunabhängig sein soll.
- Dann können wir uns nicht beschweren, wenn es nicht zum relationalen Datenmodell passt.
- Deshalb muss man immer vorsichtig sein während dem Design des logischen Modells.
- Wenn wir wirklich ein logisches Modell bekommen, das vom konzeptuellen Modell abweicht, um Normalisierung zu erreichen ... dann sollten wir nochmal zurückgehen, und auch das konzeptuelle Modell entsprechend anpassen.

Zusammenfassung

- Oftmals produzieren wir sowieso Modelle, der der 1NF gehorchen, wenn wir das konzeptuelle in das logische Modell übersetzen.
- So haben wir das ja gelernt.
- Wenn aber das konzeptuelle Modell nicht zur 1NF passt, dann kann es sein, dass wir wiederholende Gruppen von Attributen bekommen, oder ein Attribut, das eigentlich aus zwei Attributen bestehen sollte.
- Das kann passieren.
- Wir haben ja oft gesagt, dass das konzeptuelle Schema technologieunabhängig sein soll.
- Dann können wir uns nicht beschweren, wenn es nicht zum relationalen Datenmodell passt.
- Deshalb muss man immer vorsichtig sein während dem Design des logischen Modells.
- Wenn wir wirklich ein logisches Modell bekommen, das vom konzeptuellen Modell abweicht, um Normalisierung zu erreichen ... dann sollten wir nochmal zurückgehen, und auch das konzeptuelle Modell entsprechend anpassen.
- Wir wollen ja nicht am Ende verschiedene Designdokumente haben, die sich gegenseitig widersprechen.

谢谢您们！

Thank you!

Vielen Dank!

References I

[1] Raphael „rkhaotix“ Araújo e Silva. *pgModeler – PostgreSQL Database Modeler*. Palmas, Tocantins, Brazil, 2006–2025. URL: https://pgmodeler.io (besucht am 2025-04-12) (siehe S. 265).

[2] “Arrays”. In: *PostgreSQL Documentation*. 17.4. The PostgreSQL Global Development Group (PGDG), 20. Feb. 2025. Kap. 8.15. URL: https://www.postgresql.org/docs/17/arrays.html (besucht am 2025-05-08) (siehe S. 109–127).

[3] Adam Aspin und Karine Aspin. *Query Answers with MariaDB – Volume I: Introduction to SQL Queries*. Tetras Publishing, Okt. 2018. ISBN: 978-1-9996172-4-0. See also[4] (siehe S. 255, 265).

[4] Adam Aspin und Karine Aspin. *Query Answers with MariaDB – Volume II: In-Depth Querying*. Tetras Publishing, Okt. 2018. ISBN: 978-1-9996172-5-7. See also[3] (siehe S. 255, 265).

[5] Richard Barker. *Case*Method: Entity Relationship Modelling (Oracle)*. 1. Aufl. Redwood City, CA, USA: Addison Wesley Longman Publishing Co., Inc., Jan. 1990. ISBN: 978-0-201-41696-1 (siehe S. 264).

[6] Daniel J. Barrett. *Efficient Linux at the Command Line*. Sebastopol, CA, USA: O'Reilly Media, Inc., Feb. 2022. ISBN: 978-1-0981-1340-7 (siehe S. 265, 266).

[7] Daniel Bartholomew. *Learning the MariaDB Ecosystem: Enterprise-level Features for Scalability and Availability*. New York, NY, USA: Apress Media, LLC, Okt. 2019. ISBN: 978-1-4842-5514-8 (siehe S. 265).

[8] Tim Berners-Lee. *Re: Qualifiers on Hypertext links...* Geneva, Switzerland: World Wide Web project, European Organization for Nuclear Research (CERN) und Newsgroups: alt.hypertext, 6. Aug. 1991. URL: https://www.w3.org/People/Berners-Lee/1991/08/art-6484.txt (besucht am 2025-02-05) (siehe S. 266).

[9] Alex Berson. *Client/Server Architecture*. 2. Aufl. Computer Communications Series. New York, NY, USA: McGraw-Hill, 29. März 1996. ISBN: 978-0-07-005664-0 (siehe S. 264).

[10] Silvia Botros und Jeremy Tinley. *High Performance MySQL*. 4. Aufl. Sebastopol, CA, USA: O'Reilly Media, Inc., Nov. 2021. ISBN: 978-1-4920-8051-0 (siehe S. 265).

[11] Ed Bott. *Windows 11 Inside Out*. Hoboken, NJ, USA: Microsoft Press, Pearson Education, Inc., Feb. 2023. ISBN: 978-0-13-769132-6 (siehe S. 265).

References II

[12] Ron Brash und Ganesh Naik. *Bash Cookbook*. Birmingham, England, UK: Packt Publishing Ltd, Juli 2018. ISBN: 978-1-78862-936-2 (siehe S. 264).

[13] Ben Brumm. "A Guide to the Entity Relationship Diagram (ERD)". In: *Database Star*. Armadale, VIC, Australia: Elevated Online Services PTY Ltd., 30. Juli 2019–23. Dez. 2023. URL: https://www.databasestar.com/entity-relationship-diagram (besucht am 2025-03-29) (siehe S. 264).

[14] Jason Cannon. *High Availability for the LAMP Stack*. Shelter Island, NY, USA: Manning Publications, Juni 2022 (siehe S. 265, 266).

[15] Donald D. Chamberlin. "50 Years of Queries". *Communications of the ACM (CACM)* 67(8):110–121, Aug. 2024. New York, NY, USA: Association for Computing Machinery (ACM). ISSN: 0001-0782. doi:10.1145/3649887. URL: https://cacm.acm.org/research/50-years-of-queries (besucht am 2025-01-09) (siehe S. 266).

[16] Peter Pin-Shan Chen. "Entity-Relationship Modeling: Historical Events, Future Trends, and Lessons Learned". In: *Software Pioneers: Contributions to Software Engineering*. Hrsg. von Manfred Broy und Ernst Denert. Berlin/Heidelberg, Germany: Springer-Verlag GmbH Germany, Feb. 2002, S. 296–310. doi:10.1007/978-3-642-59412-0_17. URL: http://bit.csc.lsu.edu/%7Echen/pdf/Chen_Pioneers.pdf (besucht am 2025-03-06) (siehe S. 264).

[17] Peter Pin-Shan Chen. "The Entity-Relationship Model – Toward a Unified View of Data". *ACM Transactions on Database Systems (TODS)* 1(1):9–36, März 1976. New York, NY, USA: Association for Computing Machinery (ACM). ISSN: 0362-5915. doi:10.1145/320434.320440 (siehe S. 256, 264).

[18] Peter Pin-Shan Chen. "The Entity-Relationship Model: Toward a Unified View of Data". In: *1st International Conference on Very Large Data Bases (VLDB'1975)*. 22.–24. Sep. 1975, Framingham, MA, USA. Hrsg. von Douglas S. Kerr. New York, NY, USA: Association for Computing Machinery (ACM), 1975, S. 173. ISBN: 978-1-4503-3920-9. doi:10.1145/1282480.1282492. See[17] for a more comprehensive introduction. (Siehe S. 264).

[19] David Clinton und Christopher Negus. *Ubuntu Linux Bible*. 10. Aufl. Bible Series. Chichester, West Sussex, England, UK: John Wiley and Sons Ltd., 10. Nov. 2020. ISBN: 978-1-119-72233-5 (siehe S. 266).

References III

[20] Edgar Frank „Ted“ Codd. *Normalized Data Base Structure: A Brief Tutorial*. IBM Research Report RJ935. San Jose, CA, USA: IBM Research Laboratory, 1971. URL: https://www.fsmwarden.com/Codd/Normalized%20data%20base%20structure_%20a%20brief%20tutorial(1971,%20nov).pdf (besucht am 2025-05-04) (siehe S. 257).

[21] Edgar Frank „Ted“ Codd. “Normalized Data Base Structure: A Brief Tutorial”. In: *ACM SIGFIDET Workshop on Data Description, Access, and Control*. 11.–12. Nov. 1971, San Diego, CA, USA. Hrsg. von Edgar Frank „Ted“ Codd und Albert L. Dean Jr. New York, NY, USA: Association for Computing Machinery (ACM), 1971, S. 1–17. ISBN: 978-1-4503-7300-5. doi:10.1145/1734714.1734716. See also[20] (siehe S. 264).

[22] Edgar Frank „Ted“ Codd. “A Relational Model of Data for Large Shared Data Banks”. *Communications of the ACM (CACM)* 13(6):377–387, Juni 1970. New York, NY, USA: Association for Computing Machinery (ACM). ISSN: 0001-0782. doi:10.1145/362384.362685. URL: https://www.seas.upenn.edu/~zives/03f/cis550/codd.pdf (besucht am 2025-01-05) (siehe S. 18–24, 264, 266).

[23] Edgar Frank „Ted“ Codd. *Further Normalization of the Data Base Relational Model*. IBM Research Report RJ909. San Jose, CA, USA: IBM Research Laboratory, 31. Aug. 1971. URL: https://forum.thethirdmanifesto.com/wp-content/uploads/asgarosforum/987737/00-efc-further-normalization.pdf (besucht am 2025-05-04). Reprinted in and presented at[58] (siehe S. 264).

[24] “Conditional Expressions”. In: *PostgreSQL Documentation*. 17.4. The PostgreSQL Global Development Group (PGDG), 20. Feb. 2025. Kap. 9.18. URL: https://www.postgresql.org/docs/17/functions-conditional.html (besucht am 2025-05-06) (siehe S. 109–123).

[25] *Database Language SQL*. Techn. Ber. ANSI X3.135-1986. Washington, D.C., USA: American National Standards Institute (ANSI), 1986 (siehe S. 266).

[26] Christopher J. Date. *An Introduction to Database Systems*. 8. Aufl. Hoboken, NJ, USA: Pearson Education, Inc., Juli 2003. ISBN: 978-0-321-19784-9 (siehe S. 5–8, 264, 265).

[27] Matt David und Blake Barnhill. *How to Teach People SQL*. San Francisco, CA, USA: The Data School, Chart.io, Inc., 10. Dez. 2019–10. Apr. 2023. URL: https://dataschool.com/how-to-teach-people-sql (besucht am 2025-02-27) (siehe S. 266).

References IV

[28] *Database Language SQL*. International Standard ISO 9075-1987. Geneva, Switzerland: International Organization for Standardization (ISO), 1987 (siehe S. 266).

[29] Paul Deitel, Harvey Deitel und Abbey Deitel. *Internet & World Wide WebW[: How to Program*. 5. Aufl. Hoboken, NJ, USA: Pearson Education, Inc., Nov. 2011. ISBN: 978-0-13-299045-5 (siehe S. 266).

[30] Russell J.T. Dyer. *Learning MySQL and MariaDB*. Sebastopol, CA, USA: O'Reilly Media, Inc., März 2015. ISBN: 978-1-4493-6290-4 (siehe S. 265).

[31] Ramez Elmasri und Shamkant Navathe. *Fundamentals of Database Systems*. 7. Aufl. Hoboken, NJ, USA: Pearson Education, Inc., Juni 2015. ISBN: 978-0-13-397077-7 (siehe S. 5–8, 264, 265).

[32] Luca Ferrari und Enrico Pirozzi. *Learn PostgreSQL*. 2. Aufl. Birmingham, England, UK: Packt Publishing Ltd, Okt. 2023. ISBN: 978-1-83763-564-1 (siehe S. 265).

[33] Terry Halpin und Tony Morgan. *Information Modeling and Relational Databases*. 3. Aufl. Burlington, MA, USA/San Mateo, CA, USA: Morgan Kaufmann Publishers, Juli 2024. ISBN: 978-0-443-23791-1 (siehe S. 266).

[34] Jan L. Harrington. *Relational Database Design and Implementation*. 4. Aufl. Burlington, MA, USA/San Mateo, CA, USA: Morgan Kaufmann Publishers, Apr. 2016. ISBN: 978-0-12-849902-3 (siehe S. 266).

[35] Michael Hausenblas. *Learning Modern Linux*. Sebastopol, CA, USA: O'Reilly Media, Inc., Apr. 2022. ISBN: 978-1-0981-0894-6 (siehe S. 265).

[36] Matthew Helmke. *Ubuntu Linux Unleashed 2021 Edition*. 14. Aufl. Reading, MA, USA: Addison-Wesley Professional, Aug. 2020. ISBN: 978-0-13-668539-5 (siehe S. 265, 266).

[37] John Hunt. *A Beginners Guide to Python 3 Programming*. 2. Aufl. Undergraduate Topics in Computer Science (UTICS). Cham, Switzerland: Springer, 2023. ISBN: 978-3-031-35121-1. doi:10.1007/978-3-031-35122-8 (siehe S. 265).

References V

[38] *Information Technology – Database Languages – SQL – Part 1: Framework (SQL/Framework), Part 1*. International Standard ISO/IEC 9075-1:2023(E), Sixth Edition, (ANSI X3.135). Geneva, Switzerland: International Organization for Standardization (ISO) und International Electrotechnical Commission (IEC), Juni 2023. URL: https://standards.iso.org/ittf/PubliclyAvailableStandards/ISO_IEC_9075-1_2023_ed_6_-_id_76583_Publication_PDF_(en).zip (besucht am 2025-01-08). Consists of several parts, see https://modern-sql.com/standard for information where to obtain them. (Siehe S. 266).

[39] Shannon Kempe und Paul Williams. *A Short History of the ER Diagram and Information Modeling*. Studio City, CA, USA: Dataversity Digital LLC, 25. Sep. 2012. URL: https://www.dataversity.net/a-short-history-of-the-er-diagram-and-information-modeling (besucht am 2025-03-06) (siehe S. 264).

[40] William (Bill) Kent. "A Simple Guide to Five Normal Forms in Relational Database Theory". *Communications of the ACM (CACM)* 26(2):120–125, Sep. 1982–Feb. 1983. New York, NY, USA: Association for Computing Machinery (ACM). ISSN: 0001-0782. doi:10.1145/358024.358054. URL: https://www.cs.dartmouth.edu/~cs61/Resources/Papers/CACM%20Kent%20Five%20Normal%20Forms.pdf (besucht am 2025-05-03) (siehe S. 9–16, 264, 265).

[41] Jay LaCroix. *Mastering Ubuntu Server*. 4. Aufl. Birmingham, England, UK: Packt Publishing Ltd, Sep. 2022. ISBN: 978-1-80323-424-3 (siehe S. 266).

[42] Kent D. Lee und Steve Hubbard. *Data Structures and Algorithms with Python*. Undergraduate Topics in Computer Science (UTICS). Cham, Switzerland: Springer, 2015. ISBN: 978-3-319-13071-2. doi:10.1007/978-3-319-13072-9 (siehe S. 265).

[43] Gloria Lotha, Aakanksha Gaur, Erik Gregersen, Swati Chopra und William L. Hosch. "Client-Server Architecture". In: *Encyclopaedia Britannica*. Hrsg. von The Editors of Encyclopaedia Britannica. Chicago, IL, USA: Encyclopædia Britannica, Inc., 3. Jan. 2025. URL: https://www.britannica.com/technology/client-server-architecture (besucht am 2025-01-20) (siehe S. 264).

[44] Mark Lutz. *Learning Python*. 6. Aufl. Sebastopol, CA, USA: O'Reilly Media, Inc., März 2025. ISBN: 978-1-0981-7130-8 (siehe S. 265).

[45] *MariaDB Server Documentation*. Milpitas, CA, USA: MariaDB, 2025. URL: https://mariadb.com/kb/en/documentation (besucht am 2025-04-24) (siehe S. 265).

References VI

[46] Jim Melton und Alan R. Simon. *SQL: 1999 – Understanding Relational Language Components*. The Morgan Kaufmann Series in Data Management Systems. Burlington, MA, USA/San Mateo, CA, USA: Morgan Kaufmann Publishers, Juni 2001. ISBN: 978-1-55860-456-8 (siehe S. 266).

[47] Cameron Newham und Bill Rosenblatt. *Learning the Bash Shell – Unix Shell Programming: Covers Bash 3.0*. 3. Aufl. Sebastopol, CA, USA: O'Reilly Media, Inc., 2005. ISBN: 978-0-596-00965-6 (siehe S. 264).

[48] Regina O. Obe und Leo S. Hsu. *PostgreSQL: Up and Running*. 3. Aufl. Sebastopol, CA, USA: O'Reilly Media, Inc., Okt. 2017. ISBN: 978-1-4919-6336-4 (siehe S. 265).

[49] Robert Orfali, Dan Harkey und Jeri Edwards. *Client/Server Survival Guide*. 3. Aufl. Chichester, West Sussex, England, UK: John Wiley and Sons Ltd., 25. Jan. 1999. ISBN: 978-0-471-31615-2 (siehe S. 264).

[50] "Pattern Matching". In: *PostgreSQL Documentation*. 17.4. The PostgreSQL Global Development Group (PGDG), 20. Feb. 2025. Kap. 9.7. URL: https://www.postgresql.org/docs/17/functions-matching.html (besucht am 2025-02-27) (siehe S. 64–69).

[51] *PostgreSQL Documentation*. 17.4. The PostgreSQL Global Development Group (PGDG), Feb. 2025. URL: https://www.postgresql.org/docs/17/index.html (besucht am 2025-02-25).

[52] *PostgreSQL Essentials: Leveling Up Your Data Work*. Sebastopol, CA, USA: O'Reilly Media, Inc., März 2024 (siehe S. 265).

[53] Abhishek Ratan, Eric Chou, Pradeeban Kathiravelu und Dr. M.O. Faruque Sarker. *Python Network Programming*. Birmingham, England, UK: Packt Publishing Ltd, Jan. 2019. ISBN: 978-1-78883-546-6 (siehe S. 264).

[54] Federico Razzoli. *Mastering MariaDB*. Birmingham, England, UK: Packt Publishing Ltd, Sep. 2014. ISBN: 978-1-78398-154-0 (siehe S. 265).

[55] Mike Reichardt, Michael Gundall und Hans D. Schotten. "Benchmarking the Operation Times of NoSQL and MySQL Databases for Python Clients". In: *47th Annual Conference of the IEEE Industrial Electronics Society (IECON'2021*. 13.–15. Okt. 2021, Toronto, ON, Canada. Piscataway, NJ, USA: Institute of Electrical and Electronics Engineers (IEEE), 2021, S. 1–8. ISSN: 2577-1647. ISBN: 978-1-6654-3554-3. doi:10.1109/IECON48115.2021.9589382 (siehe S. 265).

References VII

[56] Mark Richards und Neal Ford. *Fundamentals of Software Architecture: An Engineering Approach*. Sebastopol, CA, USA: O'Reilly Media, Inc., Jan. 2020. ISBN: 978-1-4920-4345-4 (siehe S. 264).

[57] "Row and Array Comparisons". In: *PostgreSQL Documentation*. 17.4. The PostgreSQL Global Development Group (PGDG), 20. Feb. 2025. Kap. 9.25. URL: https://www.postgresql.org/docs/current/functions-comparisons.html (besucht am 2025-05-08) (siehe S. 109–129).

[58] Randall Rustin, Hrsg. *Data Base Systems: Courant Computer Science Symposium 6*. 24.–25. Mai 1971, New York, NY, USA. Englewood Cliffs, NJ, USA: Prentice-Hall, 1972. ISBN: 978-0-13-196741-0 (siehe S. 257).

[59] "SELECT". In: *PostgreSQL Documentation*. 17.4. The PostgreSQL Global Development Group (PGDG), 20. Feb. 2025. Kap. Part VI. Reference. URL: https://www.postgresql.org/docs/17/sql-select.html (besucht am 2025-05-08) (siehe S. 179–201).

[60] Yuriy Shamshin. "Conceptual Database Model. Entity Relationship Diagram (ERD)". In: *Databases*. Riga, Latvia: ISMA University of Applied Sciences, Mai 2024. Kap. 04. URL: https://dbs.academy.lv/lection/dbs_LS04EN_erd.pdf (besucht am 2025-03-29) (siehe S. 264).

[61] Yuriy Shamshin. *Databases*. Riga, Latvia: ISMA University of Applied Sciences, Mai 2024. URL: https://dbs.academy.lv (besucht am 2025-01-11).

[62] Yuriy Shamshin. "Normalization". In: *Databases*. Riga, Latvia: ISMA University of Applied Sciences, Mai 2024. Kap. 07a. URL: https://dbs.academy.lv/lection/dbs_LS07ENa_normalization.pdf (besucht am 2025-05-03) (siehe S. 9–16, 265).

[63] Yuriy Shamshin. "RDM Normalization. Data Anomalies. Functional Dependency. Normal Forms.". In: *Databases*. Riga, Latvia: ISMA University of Applied Sciences, Mai 2024. Kap. 07. URL: https://dbs.academy.lv/lection/dbs_LS07EN_normalization.pdf (besucht am 2025-05-03) (siehe S. 9–16, 265).

[64] Ellen Siever, Stephen Figgins, Robert Love und Arnold Robbins. *Linux in a Nutshell*. 6. Aufl. Sebastopol, CA, USA: O'Reilly Media, Inc., Sep. 2009. ISBN: 978-0-596-15448-6 (siehe S. 265).

[65] John Miles Smith und Philip Yen-Tang Chang. "Optimizing the Performance of a Relational Algebra Database Interface". *Communications of the ACM (CACM)* 18(10):568–579, Okt. 1975. New York, NY, USA: Association for Computing Machinery (ACM). ISSN: 0001-0782. doi:10.1145/361020.361025 (siehe S. 266).

References VIII

[66] "SQL Commands". In: *PostgreSQL Documentation*. 17.4. The PostgreSQL Global Development Group (PGDG), 20. Feb. 2025. Kap. Part VI. Reference. URL: https://www.postgresql.org/docs/17/sql-commands.html (besucht am 2025-02-25) (siehe S. 266).

[67] Ryan K. Stephens und Ronald R. Plew. *Sams Teach Yourself SQL in 21 Days*. 4. Aufl. Sams Tech Yourself. Indianapolis, IN, USA: SAMS Technical Publishing und Hoboken, NJ, USA: Pearson Education, Inc., Okt. 2002. ISBN: 978-0-672-32451-2 (siehe S. 262, 266).

[68] Ryan K. Stephens, Ronald R. Plew, Bryan Morgan und Jeff Perkins. *SQL in 21 Tagen. Die Datenbank-Abfragesprache SQL vollständig erklärt (in 14/21 Tagen)*. 6. Aufl. Burgthann, Bayern, Germany: Markt+Technik Verlag GmbH, Feb. 1998. ISBN: 978-3-8272-2020-2. Translation of[67] (siehe S. 266).

[69] "String Functions and Operators". In: *PostgreSQL Documentation*. 17.4. The PostgreSQL Global Development Group (PGDG), 20. Feb. 2025. Kap. 9.4. URL: https://www.postgresql.org/docs/17/functions-string.html (besucht am 2025-05-06) (siehe S. 105–108).

[70] Allen Taylor. *Introducing SQL and Relational Databases*. New York, NY, USA: Apress Media, LLC, Sep. 2018. ISBN: 978-1-4842-3841-7 (siehe S. 266).

[71] Alkin Tezuysal und Ibrar Ahmed. *Database Design and Modeling with PostgreSQL and MySQL*. Birmingham, England, UK: Packt Publishing Ltd, Juli 2024. ISBN: 978-1-80323-347-5 (siehe S. 265).

[72] Linus Torvalds. "The Linux Edge". *Communications of the ACM (CACM)* 42(4):38–39, Apr. 1999. New York, NY, USA: Association for Computing Machinery (ACM). ISSN: 0001-0782. doi:10.1145/299157.299165 (siehe S. 265).

[73] Sander van Vugt. *Linux Fundamentals*. 2. Aufl. Hoboken, NJ, USA: Pearson IT Certification, Juni 2022. ISBN: 978-0-13-792931-3 (siehe S. 265).

[74] Thomas Weise (汤卫思). *Databases*. Hefei, Anhui, China (中国安徽省合肥市): Hefei University (合肥大学), School of Artificial Intelligence and Big Data (人工智能与大数据学院), Institute of Applied Optimization (应用优化研究所, IAO), 2025. URL: https://thomasweise.github.io/databases (besucht am 2025-01-05) (siehe S. 264, 266).

References IX

[75] Thomas Weise (汤卫思). *Programming with Python*. Hefei, Anhui, China (中国安徽省合肥市): Hefei University (合肥大学), School of Artificial Intelligence and Big Data (人工智能与大数据学院), Institute of Applied Optimization (应用优化研究所, IAO), 2024–2025. URL: https://thomasweise.github.io/programmingWithPython (besucht am 2025-01-05) (siehe S. 265).

[76] Matthew West. *Developing High Quality Data Models*. Version: 2.0, Issue: 2.1. London, England, UK: Shell International Limited und European Process Industries STEP Technical Liaison Executive (EPISTLE); Burlington, MA, USA/San Mateo, CA, USA: Morgan Kaufmann Publishers, 8. Dez. 1995–Dez. 2010. ISBN: 978-0-12-375107-2. URL: https://www.researchgate.net/publication/286610894 (besucht am 2025-03-24). Edited by Julian Fowler (siehe S. 264).

[77] *What is a Relational Database?* Armonk, NY, USA: International Business Machines Corporation (IBM), 20. Okt. 2021–12. Dez. 2024. URL: https://www.ibm.com/think/topics/relational-databases (besucht am 2025-01-05) (siehe S. 266).

[78] Ulf Michael „Monty“ Widenius, David Axmark und Uppsala, Sweden: MySQL AB. *MySQL Reference Manual – Documentation from the Source*. Sebastopol, CA, USA: O'Reilly Media, Inc., 9. Juli 2002. ISBN: 978-0-596-00265-7 (siehe S. 265).

[79] Kinza Yasar und Craig S. Mullins. *Definition: Database Management System (DBMS)*. Newton, MA, USA: TechTarget, Inc., Juni 2024. URL: https://www.techtarget.com/searchdatamanagement/definition/database-management-system (besucht am 2025-01-11) (siehe S. 264).

[80] Giorgio Zarrelli. *Mastering Bash*. Birmingham, England, UK: Packt Publishing Ltd, Juni 2017. ISBN: 978-1-78439-687-9 (siehe S. 264).

Glossary (in English) I

1NF The first normal form (NF) in relational databases (DBs)[22,26,31,40].

2NF The second normal form (NF) in relational DBs[21,23,26,31,40].

3NF The third normal form (NF) in relational DBs[21,23,26,31,40].

Bash is a the shell used under Ubuntu Linux, i.e., the program that „runs“ in the terminal and interprets your commands, allowing you to start and interact with other programs[12,47,80]. Learn more at https://www.gnu.org/software/bash.

client In a client-server architecture, the client is a device or process that requests a service from the server. It initiates the communication with the server, sends a request, and receives the response with the result of the request. Typical examples for clients are web browsers in the internet as well as clients for database management systems (DBMSes), such as psql.

client-server architecture is a system design where a central server receives requests from one or multiple clients[9,43,49,53,56]. These requests and responses are usually sent over network connections. A typical example for such a system is the World Wide Web (WWW), where web servers host websites and make them available to web browsers, the clients. Another typical example is the structure of DB software, where a central server, the DBMS, offers access to the DB to the different clients. Here, the client can be some terminal software shipping with the DBMS, such as psql, or the different applications that access the DBs.

DB A *database* is an organized collection of structured information or data, typically stored electronically in a computer system. Databases are discussed in our book *Databases*[74].

DBMS A *database management system* is the software layer located between the user or application and the DB. The DBMS allows the user/application to create, read, write, update, delete, and otherwise manipulate the data in the DB[79].

DBS A *database system* is the combination of a DB and a the corresponding DBMS, i.e., basically, an installation of a DBMS on a computer together with one or multiple DBs. DBS = DB + DBMS.

ERD Entity relationship diagrams show the relationships between objects, e.g., between the tables in a DB and how they reference each other[5,13,16–18,39,60,76]

Glossary (in English) II

IT	information technology
LAMP Stack	A system setup for web applications: Linux, Apache (a web server), MySQL, and the server-side scripting language PHP[14,36].
Linux	is the leading open source operating system, i.e., a free alternative for Microsoft Windows[6,35,64,72,73]. We recommend using it for this course, for software development, and for research. Learn more at https://www.linux.org. Its variant Ubuntu is particularly easy to use and install.
MariaDB	An open source relational database management system that has forked off from MySQL[3,4,7,30,45,54]. See https://mariadb.org for more information.
Microsoft Windows	is a commercial proprietary operating system[11]. It is widely spread, but we recommend using a Linux variant such as Ubuntu for software development and for our course. Learn more at https://www.microsoft.com/windows.
MySQL	An open source relational database management system[10,30,55,71,78]. MySQL is famous for its use in the LAMP Stack. See https://www.mysql.com for more information.
NF	The *normal forms* define guidelines for the design of relational DBs with the goal to avoid redundancy and to prevent inconsistencies and anomalies[26,31,40,62,63]. There are several normal forms, first normal form (1NF), second normal form (2NF), third normal form (3NF), and so on, each more restrictive than the other.
PgModeler	the PostgreSQL DB modeler is a tool that allows for graphical modeling of logical schemas for DBs using an entity relationship diagram (ERD)-like notation[1]. Learn more at https://pgmodeler.io.
PostgreSQL	An open source object-relational DBMS[32,48,52,71]. See https://postgresql.org for more information.
psql	is the client program used to access the PostgreSQL DBMS server.
Python	The Python programming language[37,42,44,75], i.e., what you will learn about in our book[75]. Learn more at https://python.org.

Glossary (in English) III

relational database
: A relational DB is a database that organizes data into rows (tuples, records) and columns (attributes), which collectively form tables (relations) where the data points are related to each other[22,33,34,65,70,74,77].

server
: In a client-server architecture, the server is a process that fulfills the requests of the clients. It usually waits for incoming communication carring the requests from the clients. For each request, it takes the necessary actions, performs the required computations, and then sends a response with the result of the request. Typical examples for servers are web servers[14] in the internet as well as DBMSes. It is also common to refer to the computer running the server processes as server as well, i.e., to call it the „server computer“[41].

SQL
: The *Structured Query Language* is basically a programming language for querying and manipulating relational databases[15,25,27,28,38,46,66–68,70]. It is understood by many DBMSes. You find the Structured Query Language (SQL) commands supported by PostgreSQL in the reference[66].

terminal
: A terminal is a text-based window where you can enter commands and execute them[6,19]. Knowing what a terminal is and how to use it is very essential in any programming- or system administration-related task. If you want to open a terminal under Microsoft Windows, you can Druck auf ⊞+R, dann Schreiben von `cmd`, dann Druck auf ↵. Under Ubuntu Linux, Ctrl+Alt+T opens a terminal, which then runs a Bash shell inside.

Ubuntu
: is a variant of the open source operating system Linux[19,36]. We recommend that you use this operating system to follow this class, for software development, and for research. Learn more at https://ubuntu.com. If you are in China, you can download it from https://mirrors.ustc.edu.cn/ubuntu-releases.

WWW
: World Wide Web[8,29]