



合肥大學
HEFEI UNIVERSITY



Datenbanken

39. Logisches Schema: Beziehungsattribute

Thomas Weise (汤卫思)
tweise@hfu.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

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-Code 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

1. Einleitung
2. Beispiel
3. Generiertes SQL
4. Zusammenfassung





Einleitung



Einleitung



- Beziehungen in konzeptuellen Modellen können Attribute haben, wie wir in Einheit 29 gelernt haben.

Einleitung



- Beziehungen in konzeptuellen Modellen können Attribute haben, wie wir in Einheit 29 gelernt haben.
- Weil es Beziehungen im relationalen Datenmodell nicht als einzelne Objekte gibt, müssen wir diese Attribute also irgendwo anders hintun.

Einleitung



- Beziehungen in konzeptuellen Modellen können Attribute haben, wie wir in Einheit 29 gelernt haben.
- Weil es Beziehungen im relationalen Datenmodell nicht als einzelne Objekte gibt, müssen wir diese Attribute also irgendwo anders hintun.
- Im relationalen Modell gibt es nur Relationen, die als Tabellen implementiert werden.

Einleitung



- Beziehungen in konzeptuellen Modellen können Attribute haben, wie wir in Einheit 29 gelernt haben.
- Weil es Beziehungen im relationalen Datenmodell nicht als einzelne Objekte gibt, müssen wir diese Attribute also irgendwo anders hintun.
- Im relationalen Modell gibt es nur Relationen, die als Tabellen implementiert werden.
- Also müssen die Attribute von Beziehungen Tabellenspalten werden.



Beispiel



Beispiel: Konzeptuelle Ebene



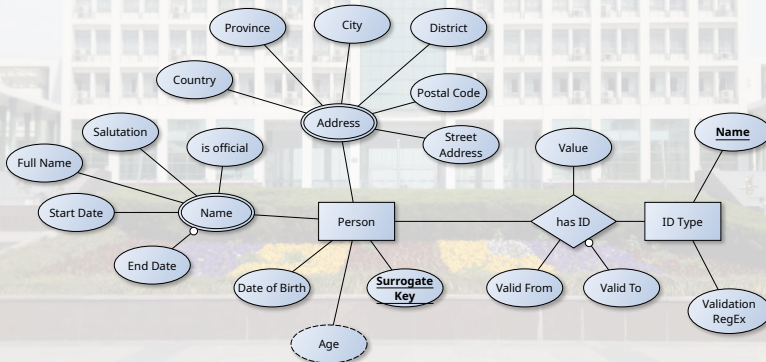
- In welche Tabelle die Beziehungsattribute kommen hängt von der Situation ab.



Beispiel: Konzeptuelle Ebene



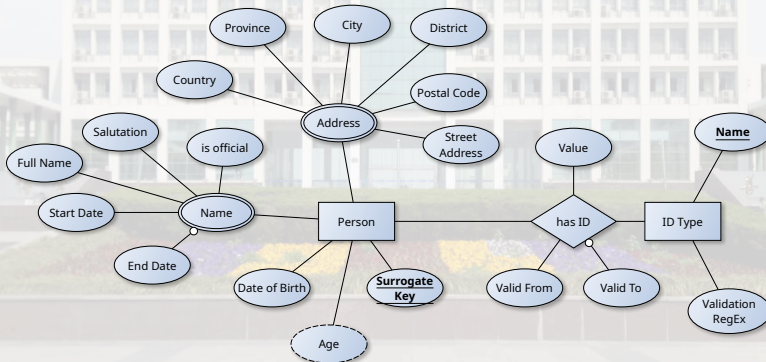
- In welche Tabelle die Beziehungsattribute kommen hängt von der Situation ab.
- Erinnern wir uns zurück an ein frühes Modell unserer *Person*-Entität.



Beispiel: Konzeptuelle Ebene



- In welche Tabelle die Beziehungsattribute kommen hängt von der Situation ab.
- Erinnern wir uns zurück an ein frühes Modell unserer *Person*-Entität.
- Hier gibt es eine Beziehung *has ID*, die die *Person*-Entitäten mit dem Entitätstyp *ID Type* verbindet.



Beispiel: Konzeptuelle Ebene



- In welche Tabelle die Beziehungsattribute kommen hängt von der Situation ab.
- Erinnern wir uns zurück an ein frühes Modell unserer *Person*-Entität.
- Hier gibt es eine Beziehung *has ID*, die die *Person*-Entitäten mit dem Entitätstyp *ID Type* verbindet.
- Wir hatten damals keine Kardinalitäten modelliert, weil wir damals noch nicht so weit waren.

- In welche Tabelle die Beziehungsattribute kommen hängt von der Situation ab.
- Erinnern wir uns zurück an ein frühes Modell unserer *Person*-Entität.
- Hier gibt es eine Beziehung *has ID*, die die *Person*-Entitäten mit dem Entitätstyp *ID Type* verbindet.
- Wir hatten damals keine Kardinalitäten modelliert, weil wir damals noch nicht so weit waren.
- Es ist aber ziemlich klar, dass das entweder eine *Person* $\times \bigcirc \text{---} \bigcirc \triangleleft ID\ Type$ oder eine *Person* $\times \text{---} \bigcirc \triangleleft ID\ Type$ -Beziehung wäre.

- In welche Tabelle die Beziehungsattribute kommen hängt von der Situation ab.
- Erinnern wir uns zurück an ein frühes Modell unserer *Person*-Entität.
- Hier gibt es eine Beziehung *has ID*, die die *Person*-Entitäten mit dem Entitätstyp *ID Type* verbindet.
- Wir hatten damals keine Kardinalitäten modelliert, weil wir damals noch nicht so weit waren.
- Es ist aber ziemlich klar, dass das entweder eine *Person* $\times \bigcirc \bigcirc \triangleleft ID\ Type$ oder eine *Person* $\times \text{---} \bigcirc \triangleleft ID\ Type$ -Beziehung wäre.
- Wir können beliebig viele (Arten von) IDs für jede Person speichern.

- In welche Tabelle die Beziehungsattribute kommen hängt von der Situation ab.
- Erinnern wir uns zurück an ein frühes Modell unserer *Person*-Entität.
- Hier gibt es eine Beziehung *has ID*, die die *Person*-Entitäten mit dem Entitätstyp *ID Type* verbindet.
- Wir hatten damals keine Kardinalitäten modelliert, weil wir damals noch nicht so weit waren.
- Es ist aber ziemlich klar, dass das entweder eine *Person* $\times \bigcirc \bigcirc \lrcorner ID\ Type$ oder eine *Person* $\times \text{---} \bigcirc \lrcorner ID\ Type$ -Beziehung wäre.
- Wir können beliebig viele (Arten von) IDs für jede Person speichern.
- Jede Art von ID kann von beliebig vielen Personen verwendet werden.

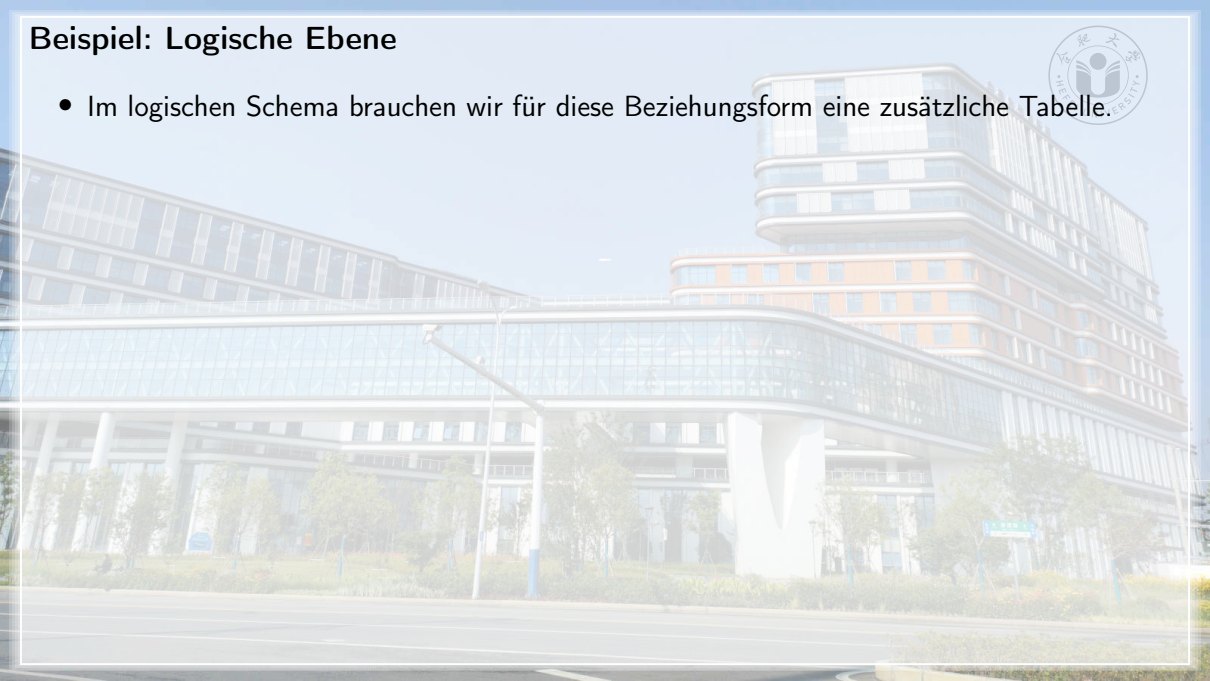
- In welche Tabelle die Beziehungsattribute kommen hängt von der Situation ab.
- Erinnern wir uns zurück an ein frühes Modell unserer *Person*-Entität.
- Hier gibt es eine Beziehung *has ID*, die die *Person*-Entitäten mit dem Entitätstyp *ID Type* verbindet.
- Wir hatten damals keine Kardinalitäten modelliert, weil wir damals noch nicht so weit waren.
- Es ist aber ziemlich klar, dass das entweder eine *Person* $\times \bigcirc \text{---} \bigcirc \lhd ID\ Type$ oder eine *Person* $\times \text{---} \bigcirc \lhd ID\ Type$ -Beziehung wäre.
- Wir können beliebig viele (Arten von) IDs für jede Person speichern.
- Jede Art von ID kann von beliebig vielen Personen verwendet werden.
- In den letzten Slides hatten wir die schwere Variante einer Beziehung modelliert...
... nahmen wir diesmal die leichte.

- In welche Tabelle die Beziehungsattribute kommen hängt von der Situation ab.
- Erinnern wir uns zurück an ein frühes Modell unserer *Person*-Entität.
- Hier gibt es eine Beziehung *has ID*, die die *Person*-Entitäten mit dem Entitätstyp *ID Type* verbindet.
- Wir hatten damals keine Kardinalitäten modelliert, weil wir damals noch nicht so weit waren.
- Es ist aber ziemlich klar, dass das entweder eine *Person* $\times \text{---} \circ \text{---} \circ \text{---} ID\ Type$ oder eine *Person* $\times \text{---} | \text{---} \circ \text{---} ID\ Type$ -Beziehung wäre.
- Wir können beliebig viele (Arten von) IDs für jede Person speichern.
- Jede Art von ID kann von beliebig vielen Personen verwendet werden.
- In den letzten Slides hatten wir die schwere Variante einer Beziehung modelliert...
... nahmen wir diesmal die leichte.
- Wir wählen *Person* $\times \text{---} \circ \text{---} \circ \text{---} ID\ Type$.

- In welche Tabelle die Beziehungsattribute kommen hängt von der Situation ab.
- Erinnern wir uns zurück an ein frühes Modell unserer *Person*-Entität.
- Hier gibt es eine Beziehung *has ID*, die die *Person*-Entitäten mit dem Entitätstyp *ID Type* verbindet.
- Wir hatten damals keine Kardinalitäten modelliert, weil wir damals noch nicht so weit waren.
- Es ist aber ziemlich klar, dass das entweder eine *Person* $\times \text{---} \circ \leftarrow ID\ Type$ oder eine *Person* $\times \text{---} | \leftarrow ID\ Type$ -Beziehung wäre.
- Wir können beliebig viele (Arten von) IDs für jede Person speichern.
- Jede Art von ID kann von beliebig vielen Personen verwendet werden.
- In den letzten Slides hatten wir die schwere Variante einer Beziehung modelliert...
... nahmen wir diesmal die leichte.
- Wir wählen *Person* $\times \text{---} \circ \leftarrow ID\ Type$.
- Wir folgen also dem Schema $O \times \text{---} \circ \leftarrow P$.

Beispiel: Logische Ebene

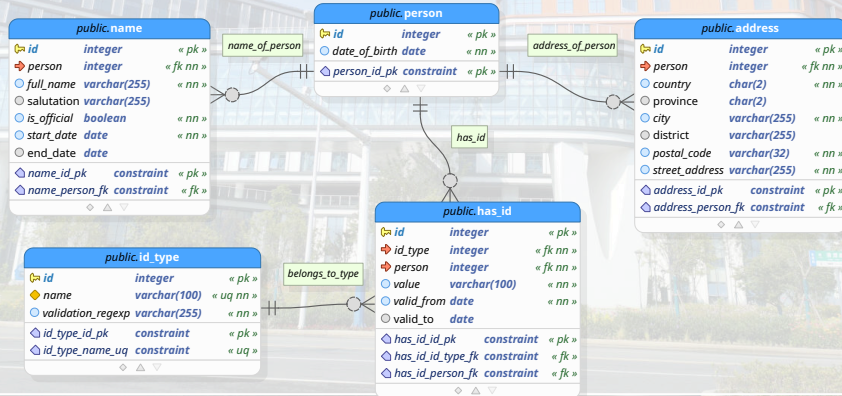
- Im logischen Schema brauchen wir für diese Beziehungsform eine zusätzliche Tabelle.



Beispiel: Logische Ebene



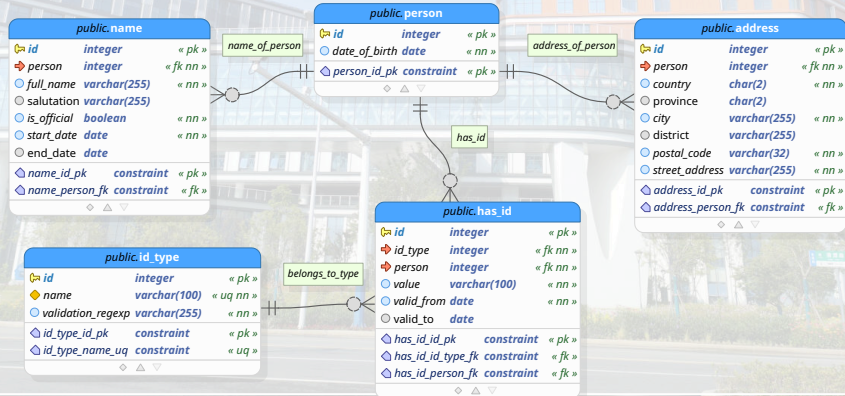
- Im logischen Schema brauchen wir für diese Beziehungsform eine zusätzliche Tabelle.
- Wir folgen der selben Methode wie damals bei $O \bowtie O \bowtie P$, benutzen aber natürlich andere Namen.



Beispiel: Logische Ebene



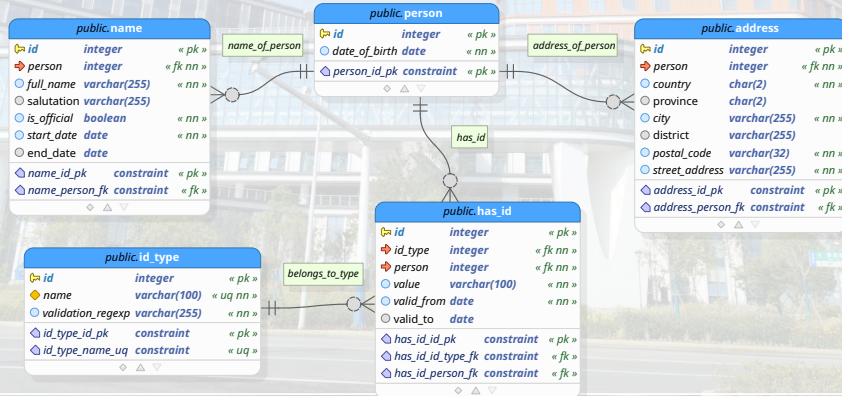
- Im logischen Schema brauchen wir für diese Beziehungsform eine zusätzliche Tabelle.
- Wir folgen der selben Methode wie damals bei $O \bowtie O \bowtie P$, benutzen aber natürlich andere Namen.
- Wir benutzen wieder den PgModeler.



Beispiel: Logische Ebene



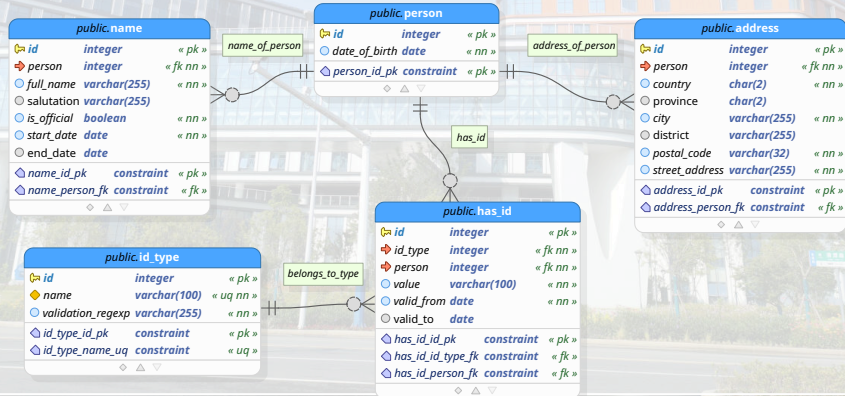
- Wir folgen der selben Methode wie damals bei $O \bowtie P$, benutzen aber natürlich andere Namen.
- Wir benutzen wieder den PgModeler.
- Wir nennen die zusätzliche Tabelle `has_id`.



Beispiel: Logische Ebene



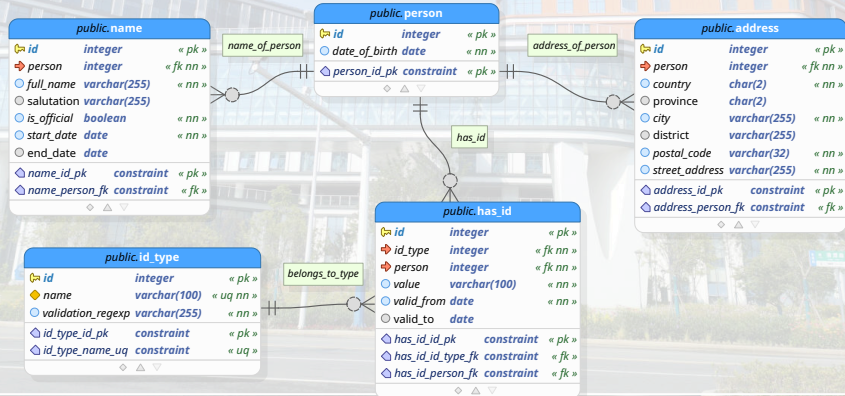
- Wir benutzen wieder den PgModeler.
- Wir nennen die zusätzliche Tabelle `has_id`.
- Die Beziehungsattribute werden in diese Tabelle gehen.



Beispiel: Logische Ebene



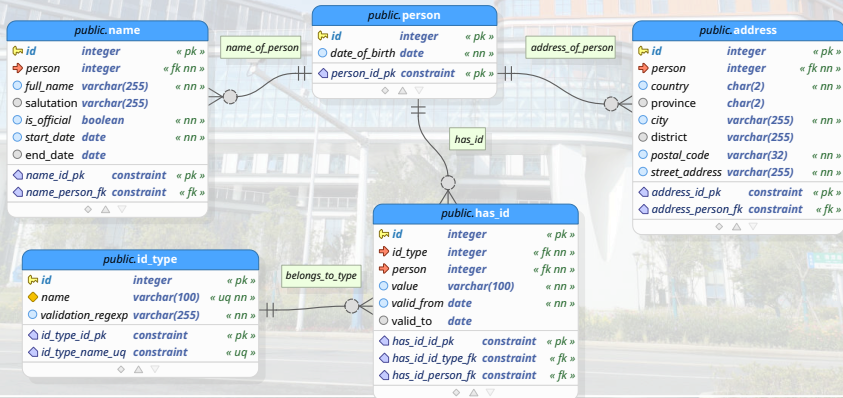
- Wir nennen die zusätzliche Tabelle `has_id`.
- Die Beziehungsattribute werden in diese Tabelle gehen.
- Und wieder bekommen wir eine neue Perspektive auf Beziehungen.



Beispiel: Logische Ebene



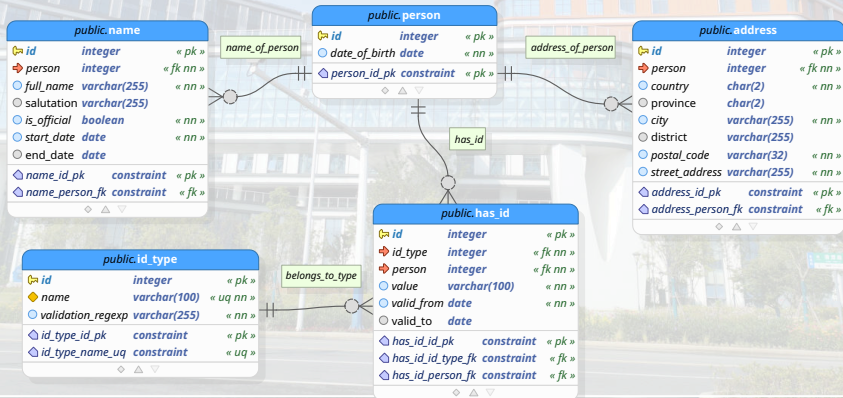
- Die Beziehungsattribute werden in diese Tabelle gehen.
- Und wieder bekommen wir eine neue Perspektive auf Beziehungen:
- Seinerzeit hatten wir $O \supset \bigcirc \text{---} \bigcirc \subset P$ mit der zusätzlichen Tabelle `relate_o_and_p` implementiert.



Beispiel: Logische Ebene



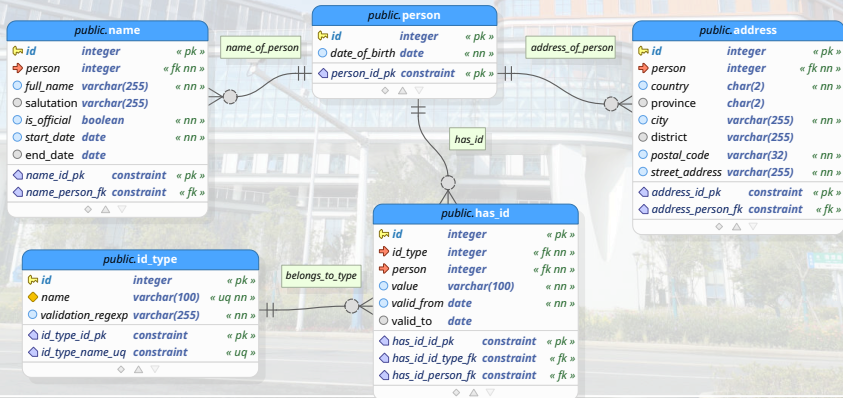
- Und wieder bekommen wir eine neue Perspektive auf Beziehungen:
- Seinerzeit hatten wir $O \supset \bigcirc - \bigcirc \supset P$ mit der zusätzlichen Tabelle `relate_o_and_p` implementiert.
- Eine andere Perspektive wäre, das wir eigentlich zwei Beziehungen implementiert haben.



Beispiel: Logische Ebene



- Seinerzeit hatten wir $O \supset \bigcirc \text{---} \bigcirc \leftarrow P$ mit der zusätzlichen Tabelle `relate_o_and_p` implementiert.
- Eine andere Perspektive wäre, das wir eigentlich zwei Beziehungen implementiert haben:
- $o \text{++} \text{---} \bigcirc \leftarrow \text{relate_o_and_p}$ und $p \text{++} \text{---} \bigcirc \leftarrow \text{relate_o_and_p}$.



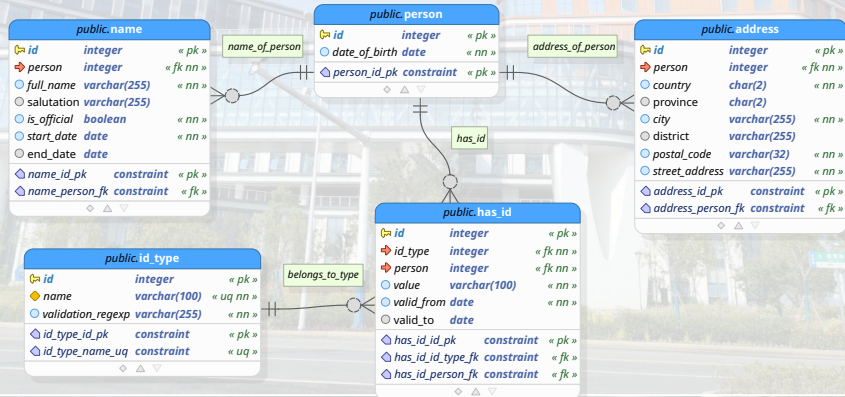
-
- The diagram illustrates the relationships between five tables: **public.name**, **public.person**, **public.address**, **public.has_id**, and **public.id_type**.
- public.name** and **public.person** are connected by a one-to-one relationship labeled **name_of_person**. The **name** table has attributes: **id** (integer, pk), **person** (integer, fk nn), **full_name** (varchar(255), nn), **salutation** (varchar(255), nn), **is_official** (boolean, nn), **start_date** (date, nn), **end_date** (date, nn), **name_id_pk** (constraint, pk), and **name_person_fk** (constraint, fk).
 - public.person** and **public.address** are connected by a one-to-one relationship labeled **address_of_person**. The **person** table has attributes: **id** (integer, pk), **date_of_birth** (date, nn), **person_id_pk** (constraint, pk), **has_id** (relationship), **id_type** (integer, fk nn), **person** (integer, fk nn), **value** (varchar(100), nn), **valid_from** (date, nn), **valid_to** (date, nn), **has_id_id_pk** (constraint, pk), **has_id_id_type_fk** (constraint, fk), and **has_id_person_fk** (constraint, fk).
 - public.address** and **public.has_id** are connected by a one-to-one relationship labeled **has_id**. The **address** table has attributes: **id** (integer, pk), **person** (integer, fk nn), **country** (char(2), nn), **province** (char(2), nn), **city** (varchar(255), nn), **district** (varchar(255), nn), **postal_code** (varchar(32), nn), **street_address** (varchar(255), nn), **address_id_pk** (constraint, pk), and **address_person_fk** (constraint, fk).
 - public.has_id** and **public.id_type** are connected by a one-to-one relationship labeled **belongs_to_type**. The **has_id** table has attributes: **id** (integer, pk), **id_type** (integer, fk nn), **person** (integer, fk nn), **value** (varchar(100), nn), **valid_from** (date, nn), **valid_to** (date, nn), **has_id_id_pk** (constraint, pk), **has_id_id_type_fk** (constraint, fk), and **has_id_person_fk** (constraint, fk).
 - public.id_type** has attributes: **id** (integer, pk), **name** (varchar(100), uq nn), **validation_regexp** (varchar(255), nn), **id_type_id_pk** (constraint, pk), and **id_type_name_uq** (constraint, uq).

-
- The diagram illustrates the decomposition of a 'public.person' table into three tables: 'public.name', 'public.address', and 'public.has_id'. The relationships are defined by the following constraints:
- public.name** and **public.address** are connected by a relationship labeled **name_of_person** and **address_of_person**, which are represented by green boxes. This relationship is a 1:1 relationship.
 - public.name** and **public.has_id** are connected by a relationship labeled **has_id**, which is represented by a green box. This relationship is a 1:1 relationship.
 - public.address** and **public.has_id** are connected by a relationship labeled **belongs_to_type**, which is represented by a green box. This relationship is a 1:1 relationship.
- The attributes and constraints for each table are as follows:
- public.name**:
 - Attributes: **id** (integer, pk), **person** (integer, fk nn), **full_name** (varchar(255), nn), **salutation** (varchar(255), nn), **is_official** (boolean, nn), **start_date** (date, nn), **end_date** (date, nn).
 - Constraints: **name_id_pk** (constraint, pk), **name_person_fk** (constraint, fk).
 - public.address**:
 - Attributes: **id** (integer, pk), **person** (integer, fk nn), **country** (char(2), nn), **province** (char(2), nn), **city** (varchar(255), nn), **district** (varchar(255), nn), **postal_code** (varchar(32), nn), **street_address** (varchar(255), nn).
 - Constraints: **address_id_pk** (constraint, pk), **address_person_fk** (constraint, fk).
 - public.has_id**:
 - Attributes: **id** (integer, pk), **id_type** (integer, fk nn), **person** (integer, fk nn), **value** (varchar(100), nn), **valid_from** (date, nn), **valid_to** (date, nn).
 - Constraints: **has_id_id_pk** (constraint, pk), **has_id_id_type_fk** (constraint, fk), **has_id_person_fk** (constraint, fk).
 - public.id_type**:
 - Attributes: **id** (integer, pk), **name** (varchar(100), uq nn), **validation_regexp** (varchar(255), nn).
 - Constraints: **id_type_id_pk** (constraint, pk), **id_type_name_uq** (constraint, uq).

Beispiel: Logische Ebene



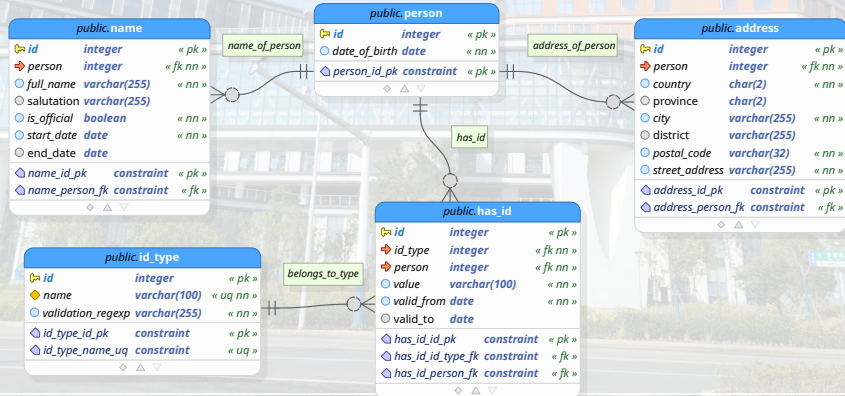
- Jede Zeile in Tabelle **o** kann mit beliebig vielen Zeilen in Tabelle **relate_o_and_p** in Beziehung stehen.
- Jede Zeile in Tabelle **p** kann mit beliebig vielen Zeilen in Tabelle **relate_o_and_p** in Beziehung stehen.



Beispiel: Logische Ebene



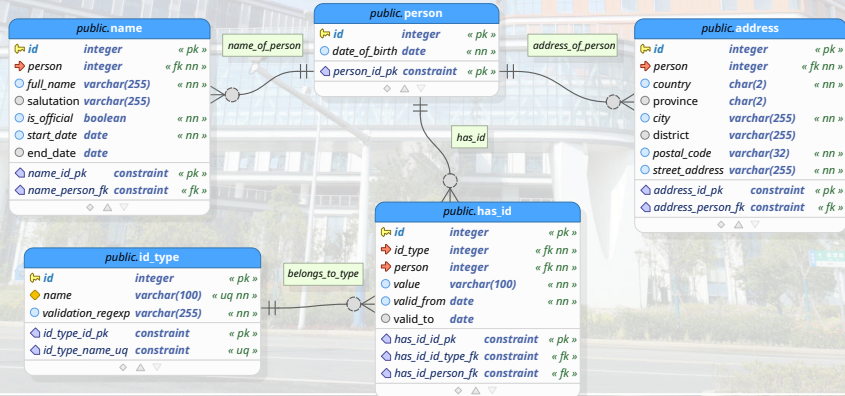
- Jede Zeile in Tabelle `p` kann mit beliebig vielen Zeilen in Tabelle `relate_o_and_p` in Beziehung stehen.
- Fürwahr, das erzeugt eine $O \times O \times P$ -Beziehung.



Beispiel: Logische Ebene



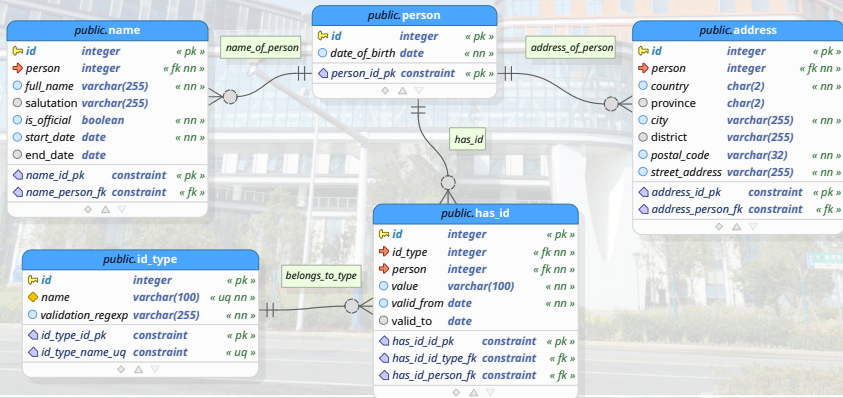
- Jede Zeile in Tabelle `p` kann mit beliebig vielen Zeilen in Tabelle `relate_o_and_p` in Beziehung stehen.
- Fürwahr, das erzeugt eine $O \supset \bigcirc - \bigcirc \subset P$ -Beziehung.
- Und das ist auch in dem Diagramm für das logische Modell hier reflektiert.



Beispiel: Logische Ebene



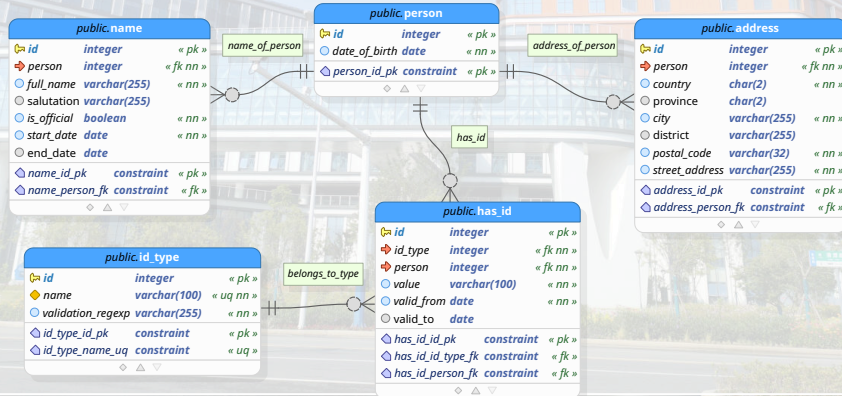
- Fürwahr, das erzeugt eine $O \supset \bigcirc \text{---} \bigcirc \supset P$ -Beziehung.
- Und das ist auch in dem Diagramm für das logische Modell hier reflektiert.
- Auch begegnen uns hier wieder mehrwertige Attribute: *Name* und *Address*.



Beispiel: Logische Ebene



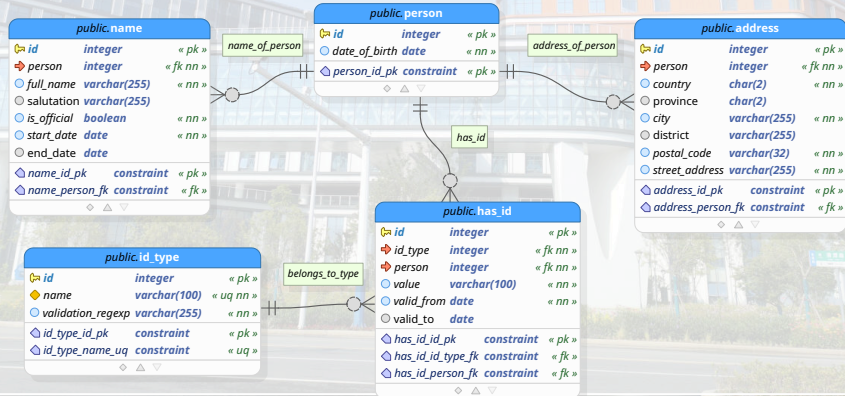
- Und das ist auch in dem Diagramm für das logische Modell hier reflektiert.
- Auch begegnen uns hier wieder mehrwertige Attribute: *Name* und *Address*.
- Wie wir schon wissen, gehen die in zusätzliche Tabellen, die wir hier `name` und `address` nennen.



Beispiel: Logische Ebene



- Auch begegnen uns hier wieder mehrwertige Attribute: *Name* und *Address*.
- Wie wir schon wissen, gehen die in zusätzliche Tabellen, die wir hier `name` und `address` nennen.
- Natürlich erzeugen wir wieder passende Fremdschlüssel-**REFERENCES**-Einschränkungen.





Generiertes SQL



Datenbank erstellen

- Erstellen wir nun die Datenbank und Tabellen mit den Skripten, die der PgModeler für uns generiert hat.



Datenbank erstellen



- Erstellen wir nun die Datenbank und Tabellen mit den Skripten, die der PgModeler für uns generiert hat.
- Fangen wir mit der Datenbank an.

```
1  -- object: person_database | type: DATABASE --
2  -- DROP DATABASE IF EXISTS person_database;
3  CREATE DATABASE person_database;
4  -- ddl-end --
```

```
1  $ psql "postgres://postgres:XXX@localhost" -v ON_ERROR_STOP=1 -ebf 01
   ↪ _person_database_database_2001.sql
2  CREATE DATABASE
3  # psql 16.11 succeeded with exit code 0.
```

Tabelle person



- Hier sehen wir das auto-generierte Skript, um die Tabelle `person` zu erstellen.

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

1  $ psql "postgres://postgres:XXX@localhost/person_database" -v
   ↪ ON_ERROR_STOP=1 -ebf 03_public_person_table_5071.sql
2  CREATE TABLE
3  ALTER TABLE
4  # psql 16.11 succeeded with exit code 0.
```

Tabelle person



- Hier sehen wir das auto-generierte Skript, um die Tabelle `person` zu erstellen.
- Wir verwenden einen Ersatz-Primärschlüssel.

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

```
1 $ psql "postgres://postgres:XXX@localhost/person_database" -v
   ↳ ON_ERROR_STOP=1 -ebf 03_public_person_table_5071.sql
2 CREATE TABLE
3 ALTER TABLE
4 # psql 16.11 succeeded with exit code 0.
```

Tabelle person



- Hier sehen wir das auto-generierte Skript, um die Tabelle `person` zu erstellen.
- Wir verwenden einen Ersatz-Primärschlüssel.
- Dazu gibt es ein Attribut `date_of_birth` für das Geburtsdatum.

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

1 $ psql "postgres://postgres:XXX@localhost/person_database" -v
   ↳ ON_ERROR_STOP=1 -ebf 03_public_person_table_5071.sql
2 CREATE TABLE
3 ALTER TABLE
4 # psql 16.11 succeeded with exit code 0.
```

Tabelle name



- Hier sehen wir das auto-generierte Skript, um die Tabelle `name` zu erstellen.

```
1  -- object: public.name | type: TABLE --
2  -- DROP TABLE IF EXISTS public.name CASCADE;
3  CREATE TABLE public.name (
4      id integer NOT NULL GENERATED BY DEFAULT AS IDENTITY ,
5      person integer NOT NULL,
6      full_name varchar(255) NOT NULL,
7      salutation varchar(255),
8      is_official boolean NOT NULL DEFAULT True,
9      start_date date NOT NULL DEFAULT CURRENT_DATE,
10     end_date date,
11     CONSTRAINT name_id_pk PRIMARY KEY (id)
12 );
13 -- ddl-end --
14 ALTER TABLE public.name OWNER TO postgres;
15 -- ddl-end --
```

```
1  $ psql "postgres://postgres:XXX@localhost/person_database" -v
   ↪ ON_ERROR_STOP=1 -ebf 04_public_name_table_5075.sql
2  CREATE TABLE
3  ALTER TABLE
4  # psql 16.11 succeeded with exit code 0.
```

Tabelle name



- Hier sehen wir das auto-generierte Skript, um die Tabelle `name` zu erstellen.
- Sie ist für das zusammengesetzte, mehrwertige Attribut *Name*.

```
1  -- object: public.name | type: TABLE --
2  -- DROP TABLE IF EXISTS public.name CASCADE;
3  CREATE TABLE public.name (
4      id integer NOT NULL GENERATED BY DEFAULT AS IDENTITY ,
5      person integer NOT NULL,
6      full_name varchar(255) NOT NULL,
7      salutation varchar(255),
8      is_official boolean NOT NULL DEFAULT True,
9      start_date date NOT NULL DEFAULT CURRENT_DATE,
10     end_date date,
11     CONSTRAINT name_id_pk PRIMARY KEY (id)
12 );
13 -- ddl-end --
14 ALTER TABLE public.name OWNER TO postgres;
15 -- ddl-end --
```

```
1  $ psql "postgres://postgres:XXX@localhost/person_database" -v
   ↳ ON_ERROR_STOP=1 -ebf 04_public_name_table_5075.sql
2  CREATE TABLE
3  ALTER TABLE
4  # psql 16.11 succeeded with exit code 0.
```

Tabelle name



- Hier sehen wir das auto-generierte Skript, um die Tabelle `name` zu erstellen.
- Sie ist für das zusammengesetzte, mehrwertige Attribut *Name*.
- Jede Zeile speichert einen Fremdschlüssel `person` der mit der Tabelle `person` über eine **REFERENCES**-Einschränkung verbunden ist.

```
1  -- object: public.name | type: TABLE --
2  -- DROP TABLE IF EXISTS public.name CASCADE;
3  CREATE TABLE public.name (
4      id integer NOT NULL GENERATED BY DEFAULT AS IDENTITY ,
5      person integer NOT NULL,
6      full_name varchar(255) NOT NULL,
7      salutation varchar(255),
8      is_official boolean NOT NULL DEFAULT True,
9      start_date date NOT NULL DEFAULT CURRENT_DATE,
10     end_date date,
11     CONSTRAINT name_id_pk PRIMARY KEY (id)
12 );
13 -- ddl-end --
14 ALTER TABLE public.name OWNER TO postgres;
15 -- ddl-end --
```

```
1  $ psql "postgres://postgres:XXX@localhost/person_database" -v
   ↳ ON_ERROR_STOP=1 -ebf 04_public_name_table_5075.sql
2  CREATE TABLE
3  ALTER TABLE
4  # psql 16.11 succeeded with exit code 0.
```



Tabelle name

- Hier sehen wir das auto-generierte Skript, um die Tabelle `name` zu erstellen.
- Sie ist für das zusammengesetzte, mehrwertige Attribut *Name*.
- Jede Zeile speichert einen Fremdschlüssel `person` der mit der Tabelle `person` über eine **REFERENCES**-Einschränkung verbunden ist.
- Diese wird ein paar Slides weiter über **ALTER TABLE** erstellt.

```
1  -- object: public.name | type: TABLE --
2  -- DROP TABLE IF EXISTS public.name CASCADE;
3  CREATE TABLE public.name (
4      id integer NOT NULL GENERATED BY DEFAULT AS IDENTITY ,
5      person integer NOT NULL,
6      full_name varchar(255) NOT NULL,
7      salutation varchar(255),
8      is_official boolean NOT NULL DEFAULT True,
9      start_date date NOT NULL DEFAULT CURRENT_DATE,
10     end_date date,
11     CONSTRAINT name_id_pk PRIMARY KEY (id)
12 );
13 -- ddl-end --
14 ALTER TABLE public.name OWNER TO postgres;
15 -- ddl-end --
```

```
1  $ psql "postgres://postgres:XXX@localhost/person_database" -v
   ↳ ON_ERROR_STOP=1 -ebf 04_public_name_table_5075.sql
2  CREATE TABLE
3  ALTER TABLE
4  # psql 16.11 succeeded with exit code 0.
```

Tabelle name



- Hier sehen wir das auto-generierte Skript, um die Tabelle `name` zu erstellen.
- Sie ist für das zusammengesetzte, mehrwertige Attribut *Name*.
- Jede Zeile speichert einen Fremdschlüssel `person` der mit der Tabelle `person` über eine **REFERENCES**-Einschränkung verbunden ist.
- Diese wird ein paar Slides weiter über **ALTER TABLE** erstellt.
- Davon abgesehen fügen wir einen kleinen Gimmick aus Spaß ein.

```
1  -- object: public.name | type: TABLE --
2  -- DROP TABLE IF EXISTS public.name CASCADE;
3  CREATE TABLE public.name (
4      id integer NOT NULL GENERATED BY DEFAULT AS IDENTITY ,
5      person integer NOT NULL,
6      full_name varchar(255) NOT NULL,
7      salutation varchar(255),
8      is_official boolean NOT NULL DEFAULT True,
9      start_date date NOT NULL DEFAULT CURRENT_DATE,
10     end_date date,
11     CONSTRAINT name_id_pk PRIMARY KEY (id)
12 );
13 -- ddl-end --
14 ALTER TABLE public.name OWNER TO postgres;
15 -- ddl-end --
```

```
1  $ psql "postgres://postgres:XXX@localhost/person_database" -v
   ↳ ON_ERROR_STOP=1 -ebf 04_public_name_table_5075.sql
2  CREATE TABLE
3  ALTER TABLE
4  # psql 16.11 succeeded with exit code 0.
```



Tabelle name

- Sie ist für das zusammengesetzte, mehrwertige Attribut *Name*.
- Jede Zeile speichert einen Fremdschlüssel `person` der mit der Tabelle `person` über eine `REFERENCES`-Einschränkung verbunden ist.
- Diese wird ein paar Slides weiter über `ALTER TABLE` erstellt.
- Davon abgesehen fügen wir einen kleinen Gimmick aus Spaß ein:
- Die Spalte `start_date` wird generiert als `NOT NULL DEFAULT CURRENT_DATE`.

```
1  -- object: public.name | type: TABLE --
2  -- DROP TABLE IF EXISTS public.name CASCADE;
3  CREATE TABLE public.name (
4      id integer NOT NULL GENERATED BY DEFAULT AS IDENTITY ,
5      person integer NOT NULL,
6      full_name varchar(255) NOT NULL,
7      salutation varchar(255),
8      is_official boolean NOT NULL DEFAULT True,
9      start_date date NOT NULL DEFAULT CURRENT_DATE,
10     end_date date,
11     CONSTRAINT name_id_pk PRIMARY KEY (id)
12 );
13 -- ddl-end --
14 ALTER TABLE public.name OWNER TO postgres;
15 -- ddl-end --
```

```
1  $ psql "postgres://postgres:XXX@localhost/person_database" -v
   ↳ ON_ERROR_STOP=1 -ebf 04_public_name_table_5075.sql
2  CREATE TABLE
3  ALTER TABLE
4  # psql 16.11 succeeded with exit code 0.
```

Tabelle name



- Jede Zeile speichert einen Fremdschlüssel `person` der mit der Tabelle `person` über eine `REFERENCES`-Einschränkung verbunden ist.
- Diese wird ein paar Slides weiter über `ALTER TABLE` erstellt.
- Davon abgesehen fügen wir einen kleinen Gimmick aus Spaß ein:
- Die Spalte `start_date` wird generiert als `NOT NULL DEFAULT CURRENT_DATE`.
- Das bedeutet, dass für jede Zeile in Tabelle `name` ein vernünftiges Start-Datum gespeichert werden muss (wegen dem `NOT NULL`).

```
1  -- object: public.name | type: TABLE --
2  -- DROP TABLE IF EXISTS public.name CASCADE;
3  CREATE TABLE public.name (
4      id integer NOT NULL GENERATED BY DEFAULT AS IDENTITY ,
5      person integer NOT NULL,
6      full_name varchar(255) NOT NULL,
7      salutation varchar(255),
8      is_official boolean NOT NULL DEFAULT True,
9      start_date date NOT NULL DEFAULT CURRENT_DATE,
10     end_date date,
11     CONSTRAINT name_id_pk PRIMARY KEY (id)
12 );
13 -- ddl-end --
14 ALTER TABLE public.name OWNER TO postgres;
15 -- ddl-end --
```

```
1  $ psql "postgres://postgres:XXX@localhost/person_database" -v
   ↳ ON_ERROR_STOP=1 -ebf 04_public_name_table_5075.sql
2  CREATE TABLE
3  ALTER TABLE
4  # psql 16.11 succeeded with exit code 0.
```

Tabelle name



- Diese wird ein paar Slides weiter über `ALTER TABLE` erstellt.
- Davon abgesehen fügen wir einen kleinen Gimmick aus Spaß ein:
- Die Spalte `start_date` wird generiert als `NOT NULL DEFAULT CURRENT_DATE`.
- Das bedeutet, das für jede Zeile in Tabelle `name` ein vernünftiges Start-Datum gespeichert werden muss (wegen dem `NOT NULL`).
- Wenn wir es beim Erstellen der Zeile nicht angeben, dann wird stattdessen ein `DEFAULT`-Wert gespeichert.

```
1  -- object: public.name | type: TABLE --
2  -- DROP TABLE IF EXISTS public.name CASCADE;
3  CREATE TABLE public.name (
4      id integer NOT NULL GENERATED BY DEFAULT AS IDENTITY ,
5      person integer NOT NULL,
6      full_name varchar(255) NOT NULL,
7      salutation varchar(255),
8      is_official boolean NOT NULL DEFAULT True,
9      start_date date NOT NULL DEFAULT CURRENT_DATE,
10     end_date date,
11     CONSTRAINT name_id_pk PRIMARY KEY (id)
12 );
13 -- ddl-end --
14 ALTER TABLE public.name OWNER TO postgres;
15 -- ddl-end --
```

```
1  $ psql "postgres://postgres:XXX@localhost/person_database" -v
   ↳ ON_ERROR_STOP=1 -ebf 04_public_name_table_5075.sql
2  CREATE TABLE
3  ALTER TABLE
4  # psql 16.11 succeeded with exit code 0.
```

Tabelle name



- Davon abgesehen fügen wir einen kleinen Gimmick aus Spaß ein:
- Die Spalte `start_date` wird generiert als `NOT NULL` `DEFAULT CURRENT_DATE`.
- Das bedeutet, das für jede Zeile in Tabelle `name` ein vernünftiges Start-Datum gespeichert werden muss (wegen dem `NOT NULL`).
- Wenn wir es beim Erstellen der Zeile nicht angeben, dann wird stattdessen ein `DEFAULT`-Wert gespeichert.
- Dieses `DEFAULT` ist `CURRENT_DATE`, welches, wie der Name schon erklärt, das aktuelle Datum in genau dem Moment ist, wo wir die Zeile erstellen²¹.

```
1  -- object: public.name | type: TABLE --
2  -- DROP TABLE IF EXISTS public.name CASCADE;
3  CREATE TABLE public.name (
4      id integer NOT NULL GENERATED BY DEFAULT AS IDENTITY ,
5      person integer NOT NULL,
6      full_name varchar(255) NOT NULL,
7      salutation varchar(255),
8      is_official boolean NOT NULL DEFAULT True,
9      start_date date NOT NULL DEFAULT CURRENT_DATE,
10     end_date date,
11     CONSTRAINT name_id_pk PRIMARY KEY (id)
12 );
13 -- ddl-end --
14 ALTER TABLE public.name OWNER TO postgres;
15 -- ddl-end --
```

```
1  $ psql "postgres://postgres:XXX@localhost/person_database" -v
   ↳ ON_ERROR_STOP=1 -ebf 04_public_name_table_5075.sql
2  CREATE TABLE
3  ALTER TABLE
4  # psql 16.11 succeeded with exit code 0.
```

Tabelle name



- Das bedeutet, dass für jede Zeile in Tabelle `name` ein vernünftiges Start-Datum gespeichert werden muss (wegen dem `NOT NULL`).
- Wenn wir es beim Erstellen der Zeile nicht angeben, dann wird stattdessen ein `DEFAULT`-Wert gespeichert.
- Dieses `DEFAULT` ist `CURRENT_DATE`, welches, wie der Name schon erklärt, das aktuelle Datum in genau dem Moment ist, wo wir die Zeile erstellen²¹.
- Es gibt noch ähnliche Funktionen wie `CURRENT_TIME` und, besonders wichtig, `CURRENT_TIMESTAMP`²¹, der oft für Timestamping verwendet wird.

```
1  -- object: public.name | type: TABLE --
2  -- DROP TABLE IF EXISTS public.name CASCADE;
3  CREATE TABLE public.name (
4      id integer NOT NULL GENERATED BY DEFAULT AS IDENTITY ,
5      person integer NOT NULL,
6      full_name varchar(255) NOT NULL,
7      salutation varchar(255),
8      is_official boolean NOT NULL DEFAULT True,
9      start_date date NOT NULL DEFAULT CURRENT_DATE,
10     end_date date,
11     CONSTRAINT name_id_pk PRIMARY KEY (id)
12 );
13 -- ddl-end --
14 ALTER TABLE public.name OWNER TO postgres;
15 -- ddl-end --
```

```
1  $ psql "postgres://postgres:XXX@localhost/person_database" -v
2      ↪ ON_ERROR_STOP=1 -ebf 04_public_name_table_5075.sql
3  CREATE TABLE
4  ALTER TABLE
# psql 16.11 succeeded with exit code 0.
```

Tabelle address



- Hier sehen wir das auto-generierte Skript, um die Tabelle `address` zu erstellen.

```
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      person integer NOT NULL,
6      country char(2) NOT NULL DEFAULT 'CN',
7      province char(2) DEFAULT 'AH',
8      city varchar(255) NOT NULL,
9      district varchar(255),
10     postal_code varchar(32) NOT NULL,
11     street_address varchar(255) NOT NULL,
12     CONSTRAINT address_id_pk PRIMARY KEY (id)
13 );
14 -- ddl-end --
15 ALTER TABLE public.address OWNER TO postgres;
16 -- ddl-end --
```

```
1  $ psql "postgres://postgres:XXX@localhost/person_database" -v
   ↳ ON_ERROR_STOP=1 -ebf 05_public_address_table_5084.sql
2  CREATE TABLE
3  ALTER TABLE
4  # psql 16.11 succeeded with exit code 0.
```

Tabelle address



- Hier sehen wir das auto-generierte Skript, um die Tabelle `address` zu erstellen.
- Da gibt es nicht viel Besonderes zu sehen.

```
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      person integer NOT NULL,
6      country char(2) NOT NULL DEFAULT 'CN',
7      province char(2) DEFAULT 'AH',
8      city varchar(255) NOT NULL,
9      district varchar(255),
10     postal_code varchar(32) NOT NULL,
11     street_address varchar(255) NOT NULL,
12     CONSTRAINT address_id_pk PRIMARY KEY (id)
13 );
14 -- ddl-end --
15 ALTER TABLE public.address OWNER TO postgres;
16 -- ddl-end --
```

```
1  $ psql "postgres://postgres:XXX@localhost/person_database" -v
   ↳ ON_ERROR_STOP=1 -ebf 05_public_address_table_5084.sql
2  CREATE TABLE
3  ALTER TABLE
4  # psql 16.11 succeeded with exit code 0.
```

Tabelle address



- Hier sehen wir das auto-generierte Skript, um die Tabelle `address` zu erstellen.
- Da gibt es nicht viel Besonderes zu sehen.
- Wir verwenden `DEFAULT`-Werte auch für `country` und `province`.

```
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      person integer NOT NULL,
6      country char(2) NOT NULL DEFAULT 'CN',
7      province char(2) DEFAULT 'AH',
8      city varchar(255) NOT NULL,
9      district varchar(255),
10     postal_code varchar(32) NOT NULL,
11     street_address varchar(255) NOT NULL,
12     CONSTRAINT address_id_pk PRIMARY KEY (id)
13 );
14 -- ddl-end --
15 ALTER TABLE public.address OWNER TO postgres;
16 -- ddl-end --
```

```
1  $ psql "postgres://postgres:XXX@localhost/person_database" -v
   ↪ ON_ERROR_STOP=1 -ebf 05_public_address_table_5084.sql
2  CREATE TABLE
3  ALTER TABLE
4  # psql 16.11 succeeded with exit code 0.
```

Tabelle address



- Hier sehen wir das auto-generierte Skript, um die Tabelle `address` zu erstellen.
- Da gibt es nicht viel Besonderes zu sehen.
- Wir verwenden `DEFAULT`-Werte auch für `country` und `province`.
- Wenn man diese beim Eingeben der Daten nicht angibt, werden sie auf `'CN'` und `'AH'`, gesetzt.

```
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     person integer NOT NULL,
6     country char(2) NOT NULL DEFAULT 'CN',
7     province char(2) DEFAULT 'AH',
8     city varchar(255) NOT NULL,
9     district varchar(255),
10    postal_code varchar(32) NOT NULL,
11    street_address varchar(255) NOT NULL,
12    CONSTRAINT address_id_pk PRIMARY KEY (id)
13 );
14 -- ddl-end --
15 ALTER TABLE public.address OWNER TO postgres;
16 -- ddl-end --
```

```
1 $ psql "postgres://postgres:XXX@localhost/person_database" -v
   ↳ ON_ERROR_STOP=1 -ebf 05_public_address_table_5084.sql
2 CREATE TABLE
3 ALTER TABLE
4 # psql 16.11 succeeded with exit code 0.
```

Tabelle address



- Hier sehen wir das auto-generierte Skript, um die Tabelle `address` zu erstellen.
- Da gibt es nicht viel Besonderes zu sehen.
- Wir verwenden `DEAULT`-Werte auch für `country` und `province`.
- Wenn man diese beim Eingeben der Daten nicht angibt, werden sie auf `'CN'` und `'AH'`, gesetzt.
- Der Fremdschlüssel `person` zeigt auf Tabelle `person` über eine `REFERENCES`-Einschränkung, die wir in paar Slides später zeigen.

```
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      person integer NOT NULL,
6      country char(2) NOT NULL DEFAULT 'CN',
7      province char(2) DEFAULT 'AH',
8      city varchar(255) NOT NULL,
9      district varchar(255),
10     postal_code varchar(32) NOT NULL,
11     street_address varchar(255) NOT NULL,
12     CONSTRAINT address_id_pk PRIMARY KEY (id)
13 );
14 -- ddl-end --
15 ALTER TABLE public.address OWNER TO postgres;
16 -- ddl-end --
```

```
1  $ psql "postgres://postgres:XXX@localhost/person_database" -v
   ↳ ON_ERROR_STOP=1 -ebf 05_public_address_table_5084.sql
2  CREATE TABLE
3  ALTER TABLE
4  # psql 16.11 succeeded with exit code 0.
```

Tabelle id_type



- Hier sehen wir das auto-generierte Skript, um die Tabelle `id_type` zu erstellen.

```
1  -- object: public.id_type | type: TABLE --
2  -- DROP TABLE IF EXISTS public.id_type CASCADE;
3  CREATE TABLE public.id_type (
4      id integer NOT NULL GENERATED ALWAYS AS IDENTITY ,
5      name varchar(100) NOT NULL,
6      validation_regexp varchar(255) NOT NULL DEFAULT '.*+',
7      CONSTRAINT id_type_id_pk PRIMARY KEY (id),
8      CONSTRAINT id_type_name_uq UNIQUE (name)
9  );
10 -- ddl-end --
11 ALTER TABLE public.id_type OWNER TO postgres;
12 -- ddl-end --
```

```
1  $ psql "postgres://postgres:XXX@localhost/person_database" -v
   ↪ ON_ERROR_STOP=1 -ebf 06_public_id_type_table_5094.sql
2  CREATE TABLE
3  ALTER TABLE
4  # psql 16.11 succeeded with exit code 0.
```

Tabelle id_type



- Hier sehen wir das auto-generierte Skript, um die Tabelle `id_type` zu erstellen.
- Das machen wir im Grunde genauso wie in der letzten Einheit.

```
1 -- object: public.id_type | type: TABLE --
2 -- DROP TABLE IF EXISTS public.id_type CASCADE;
3 CREATE TABLE public.id_type (
4     id integer NOT NULL GENERATED ALWAYS AS IDENTITY ,
5     name varchar(100) NOT NULL,
6     validation_regexp varchar(255) NOT NULL DEFAULT '.*+',
7     CONSTRAINT id_type_id_pk PRIMARY KEY (id),
8     CONSTRAINT id_type_name_uq UNIQUE (name)
9 );
10 -- ddl-end --
11 ALTER TABLE public.id_type OWNER TO postgres;
12 -- ddl-end --
```

```
1 $ psql "postgres://postgres:XXX@localhost/person_database" -v
   ↳ ON_ERROR_STOP=1 -ebf 06_public_id_type_table_5094.sql
2 CREATE TABLE
3 ALTER TABLE
4 # psql 16.11 succeeded with exit code 0.
```

Tabelle has_id



- Hier sehen wir das auto-generierte Skript, um die Tabelle `has_id` zu erstellen.

```
1  -- object: public.has_id | type: TABLE --
2  -- DROP TABLE IF EXISTS public.has_id CASCADE;
3  CREATE TABLE public.has_id (
4      id integer NOT NULL GENERATED BY DEFAULT AS IDENTITY ,
5      id_type integer NOT NULL,
6      person integer NOT NULL,
7      value varchar(100) NOT NULL,
8      valid_from date NOT NULL,
9      valid_to date,
10     CONSTRAINT has_id_id_pk PRIMARY KEY (id)
11 );
12 -- ddl-end --
13 ALTER TABLE public.has_id OWNER TO postgres;
14 -- ddl-end --
```

```
1  $ psql "postgres://postgres:XXX@localhost/person_database" -v
2      ↪ ON_ERROR_STOP=1 -ebf 07_public_has_id_table_5100.sql
3  CREATE TABLE
4  ALTER TABLE
5  # psql 16.11 succeeded with exit code 0.
```

Tabelle has_id



- Hier sehen wir das auto-generierte Skript, um die Tabelle `has_id` zu erstellen.
- Sie folgt im Grunde dem selben Design als Tabelle `personal_id` in der vorigen Einheit.

```
1 -- object: public.has_id | type: TABLE --
2 -- DROP TABLE IF EXISTS public.has_id CASCADE;
3 CREATE TABLE public.has_id (
4     id integer NOT NULL GENERATED BY DEFAULT AS IDENTITY ,
5     id_type integer NOT NULL,
6     person integer NOT NULL,
7     value varchar(100) NOT NULL,
8     valid_from date NOT NULL,
9     valid_to date,
10    CONSTRAINT has_id_id_pk PRIMARY KEY (id)
11 );
12 -- ddl-end --
13 ALTER TABLE public.has_id OWNER TO postgres;
14 -- ddl-end --
```

```
1 $ psql "postgres://postgres:XXX@localhost/person_database" -v
2 ↪ ON_ERROR_STOP=1 -ebf 07_public_has_id_table_5100.sql
3 CREATE TABLE
4 ALTER TABLE
# psql 16.11 succeeded with exit code 0.
```

Tabelle has_id



- Hier sehen wir das auto-generierte Skript, um die Tabelle `has_id` zu erstellen.
- Sie folgt im Grunde dem selben Design als Tabelle `personal_id` in der vorigen Einheit.
- Sie beinhaltet die Beziehungsattribute, die wir im konzeptuellen Modell definiert haben.

```
1  -- object: public.has_id | type: TABLE --
2  -- DROP TABLE IF EXISTS public.has_id CASCADE;
3  CREATE TABLE public.has_id (
4      id integer NOT NULL GENERATED BY DEFAULT AS IDENTITY ,
5      id_type integer NOT NULL,
6      person integer NOT NULL,
7      value varchar(100) NOT NULL,
8      valid_from date NOT NULL,
9      valid_to date,
10     CONSTRAINT has_id_id_pk PRIMARY KEY (id)
11 );
12 -- ddl-end --
13 ALTER TABLE public.has_id OWNER TO postgres;
14 -- ddl-end --
```

```
1  $ psql "postgres://postgres:XXX@localhost/person_database" -v
2      ↳ ON_ERROR_STOP=1 -ebf 07_public_has_id_table_5100.sql
3  CREATE TABLE
4  ALTER TABLE
5  # psql 16.11 succeeded with exit code 0.
```

Einschränkung (1)



- Hier sehen wir das auto-generierte Skript, um dir Fremdschlüssel-Einschränkung zu erstellen, die sicherstellt, dass jede Zeile in Tabelle `name` mit einer Zeile in Tabelle `person` verbunden ist.

```
1 -- object: name_person_fk | type: CONSTRAINT --
2 -- ALTER TABLE public.name DROP CONSTRAINT IF EXISTS name_person_fk
  ↳ CASCADE;
3 ALTER TABLE public.name ADD CONSTRAINT name_person_fk FOREIGN KEY (
  ↳ person)
4 REFERENCES public.person (id) MATCH SIMPLE
5 ON DELETE NO ACTION ON UPDATE NO ACTION;
6 -- ddl-end --
```

```
1 $ psql "postgres://postgres:XXX@localhost/person_database" -v
  ↳ ON_ERROR_STOP=1 -ebf 08_public_name_name_person_fk_constraint_5108
  ↳ .sql
2 ALTER TABLE
3 # psql 16.11 succeeded with exit code 0.
```

Einschränkung (2)

- Hier sehen wir das auto-generierte Skript, um dir Fremdschlüssel-Einschränkung zu erstellen, die sicherstellt, dass jede Zeile in Tabelle `address` mit einer Zeile in Tabelle `person` verbunden ist.

```
1 -- object: address_person_fk | type: CONSTRAINT --
2 -- ALTER TABLE public.address DROP CONSTRAINT IF EXISTS
  ↳ address_person_fk CASCADE;
3 ALTER TABLE public.address ADD CONSTRAINT address_person_fk FOREIGN KEY
  ↳ (person)
4 REFERENCES public.person (id) MATCH SIMPLE
5 ON DELETE NO ACTION ON UPDATE NO ACTION;
6 -- ddl-end --
```

```
1 $ psql "postgres://postgres:XXX@localhost/person_database" -v
  ↳ ON_ERROR_STOP=1 -ebf 09
  ↳ _public_address_address_person_fk_constraint_5109.sql
2 ALTER TABLE
3 # psql 16.11 succeeded with exit code 0.
```



Einschränkung (3)



- Hier sehen wir das auto-generierte Skript, um dir Fremdschlüssel-Einschränkung zu erstellen, die sicherstellt, dass jede Zeile in Tabelle `has_id` mit einer Zeile in Tabelle `id_type` verbunden ist.

```
1 -- object: has_id_id_type_fk | type: CONSTRAINT --
2 -- ALTER TABLE public.has_id DROP CONSTRAINT IF EXISTS has_id_id_type_fk
  ↳ CASCADE;
3 ALTER TABLE public.has_id ADD CONSTRAINT has_id_id_type_fk FOREIGN KEY (
  ↳ id_type)
4 REFERENCES public.id_type (id) MATCH SIMPLE
5 ON DELETE NO ACTION ON UPDATE NO ACTION;
6 -- ddl-end --

1 $ psql "postgres://postgres:XXX@localhost/person_database" -v
  ↳ ON_ERROR_STOP=1 -ebf 10
  ↳ _public_has_id_has_id_id_type_fk_constraint_5110.sql
2 ALTER TABLE
3 # psql 16.11 succeeded with exit code 0.
```

Einschränkung (4)



- Hier sehen wir das auto-generierte Skript, um dir Fremdschlüssel-Einschränkung zu erstellen, die sicherstellt, dass jede Zeile in Tabelle `has_id` mit einer Zeile in Tabelle `person` verbunden ist.

```
1 -- object: has_id_person_fk | type: CONSTRAINT --
2 -- ALTER TABLE public.has_id DROP CONSTRAINT IF EXISTS has_id_person_fk
  ↳ CASCADE;
3 ALTER TABLE public.has_id ADD CONSTRAINT has_id_person_fk FOREIGN KEY (
  ↳ person)
4 REFERENCES public.person (id) MATCH SIMPLE
5 ON DELETE NO ACTION ON UPDATE NO ACTION;
6 -- ddl-end --
```

```
1 $ psql "postgres://postgres:XXX@localhost/person_database" -v
  ↳ ON_ERROR_STOP=1 -ebf 11
  ↳ _public_has_id_has_id_person_fk_constraint_5111.sql
2 ALTER TABLE
3 # psql 16.11 succeeded with exit code 0.
```

Daten Einfügen

- Fügen wir nun Daten in die Tabellen ein.

```
1  /** Insert some data into the tables of our person database. */
2
3  -- Create the persons Bebbo, Bibbo, and Bebbä.
4  INSERT INTO person (date_of_birth) VALUES
5      ('2005-08-07'), ('1995-01-02'), ('1963-11-13');
6
7  -- Create the names of Bebbo, Bibbo, and Bebbä.
8  INSERT INTO name (person, full_name, salutation, is_official,
9                  start_date, end_date) VALUES
10     (1, 'Bebbo', 'Bebbo Machine', TRUE, '2005-08-07', NULL),
11     (2, 'Bibbo', 'The Bib-Man', TRUE, '1995-01-02', NULL),
12     (3, 'Bibbi', 'Ms. Bibbi', FALSE, '1963-11-13', '1989-02-12'),
13     (3, 'Bebba', 'Mrs. Bebbä', TRUE, '1989-02-12', NULL);
14
15  -- Create two ID types: Chinese national ID and mobile phone numbers.
16  INSERT INTO id_type (name, validation_regexp) VALUES
17     ('national ID', '~\d{6}((19)|(20))\d{9}[0-9X]$'),
18     ('mobile phone number', '~\d{11}$');
19
20  -- Insert the personal IDs and mobile phone numbers of the people.
21  INSERT INTO has_id (id_type, person, value, valid_from) VALUES
22     (1, 1, '123456200508071234', '2021-09-21'),
23     (1, 2, '123456199501021234', '2024-12-01'),
24     (1, 3, '123456196311131234', '1983-10-03'),
25     (2, 1, '2222222222', '2020-09-12'),
26     (2, 2, '1111111111', '2012-07-30'),
27     (2, 3, '4444444444', '2012-07-30');
28
29  -- Provide an address for each person.
30  INSERT INTO address (person, country, province, city,
31                      district, postal_code, street_address) VALUES
32     (1, 'DE', 'SN', 'Chemnitz', 'Zentrum', '09111', 'Rathaus'),
33     (2, 'CN', 'AH', 'Hefei', 'Jinkaiqu', '230601', 'Hefei University'),
34     (3, 'US', 'NY', 'New York', 'Manhattan', '10036', 'Times Square');
```

```
1  $ psql "postgres://postgres:XXX@localhost/person_database" -v
2  ↳ ON_ERROR_STOP=1 -ebf insert.sql
```

```
3  INSERT 0 3
4  INSERT 0 4
5  INSERT 0 2
6  INSERT 0 6
7  INSERT 0 3
```

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

Daten Einfügen

- Fügen wir nun Daten in die Tabellen ein.
- Wir fangen mit drei Datensätzen für Tabelle `person` an.

```
1  /** Insert some data into the tables of our person database. */
2
3  -- Create the persons Bebbo, Bibbo, and Bebbu.
4  INSERT INTO person (date_of_birth) VALUES
5      ('2005-08-07'), ('1995-01-02'), ('1963-11-13');
6
7
8  -- Create the names of Bebbo, Bibbo, and Bebbu.
9  INSERT INTO name (person, full_name, salutation, is_official,
10                  start_date, end_date) VALUES
11      (1, 'Bebbo', 'Bebbo Machine', TRUE, '2005-08-07', NULL),
12      (2, 'Bibbo', 'The Bib-Man', TRUE, '1995-01-02', NULL),
13      (3, 'Bibbi', 'Ms. Bibbi', FALSE, '1963-11-13', '1989-02-12'),
14      (3, 'Bebba', 'Mrs. Bebbu', TRUE, '1989-02-12', NULL);
15
16 -- Create two ID types: Chinese national ID and mobile phone numbers.
17 INSERT INTO id_type (name, validation_regexp) VALUES
18     ('national ID', '~\d{6}((19)|(20))\d{9}[0-9X]$'),
19     ('mobile phone number', '~\d{11}$');
20
21 -- Insert the personal IDs and mobile phone numbers of the people.
22 INSERT INTO has_id (id_type, person, value, valid_from) VALUES
23     (1, 1, '123456200508071234', '2021-09-21'),
24     (1, 2, '123456199501021234', '2024-12-01'),
25     (1, 3, '123456196311131234', '1983-10-03'),
26     (2, 1, '22222222222', '2020-09-12'),
27     (2, 2, '11111111111', '2012-07-30'),
28     (2, 3, '44444444444', '2012-07-30');
29
30 -- Provide an address for each person.
31 INSERT INTO address (person, country, province, city,
32                     district, postal_code, street_address) VALUES
33     (1, 'DE', 'SN', 'Chemnitz', 'Zentrum', '09111', 'Rathaus'),
34     (2, 'CN', 'AH', 'Hefei', 'Jinkaiqu', '230601', 'Hefei University'),
35     (3, 'US', 'NY', 'New York', 'Manhattan', '10036', 'Times Square');
```

```
1  $ psql "postgres://postgres:XXX@localhost/person_database" -v
2  ↳ ON_ERROR_STOP=1 -ebf insert.sql
```

```
3  INSERT 0 3
4  INSERT 0 4
5  INSERT 0 2
6  INSERT 0 6
7  INSERT 0 3
```

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

Daten Einfügen

- Fügen wir nun Daten in die Tabellen ein.
- Wir fangen mit drei Datensätzen für Tabelle `person` an.
- Dabei müssen wir nur jeweils das Geburtsdatum angeben.

```
1  /** Insert some data into the tables of our person database. */
2
3  -- Create the persons Bebbo, Bibbo, and Bebbi.
4  INSERT INTO person (date_of_birth) VALUES
5      ('2005-08-07'), ('1995-01-02'), ('1963-11-13');
6
7
8  -- Create the names of Bebbo, Bibbo, and Bebbi.
9  INSERT INTO name (person, full_name, salutation, is_official,
10                  start_date, end_date) VALUES
11      (1, 'Bebbo', 'Bebbo Machine', TRUE, '2005-08-07', NULL),
12      (2, 'Bibbo', 'The Bib-Man', TRUE, '1995-01-02', NULL),
13      (3, 'Bibbi', 'Ms. Bibbi', FALSE, '1963-11-13', '1989-02-12'),
14      (3, 'Bebba', 'Mrs. Bebbi', TRUE, '1989-02-12', NULL);
15
16 -- Create two ID types: Chinese national ID and mobile phone numbers.
17 INSERT INTO id_type (name, validation_regexp) VALUES
18     ('national ID', '~\d{6}((19)|(20))\d{9}[0-9X]$'),
19     ('mobile phone number', '~\d{11}$');
20
21 -- Insert the personal IDs and mobile phone numbers of the people.
22 INSERT INTO has_id (id_type, person, value, valid_from) VALUES
23     (1, 1, '123456200508071234', '2021-09-21'),
24     (1, 2, '123456199501021234', '2024-12-01'),
25     (1, 3, '123456196311131234', '1983-10-03'),
26     (2, 1, '2222222222', '2020-09-12'),
27     (2, 2, '1111111111', '2012-07-30'),
28     (2, 3, '4444444444', '2012-07-30');
29
30 -- Provide an address for each person.
31 INSERT INTO address (person, country, province, city,
32                    district, postal_code, street_address) VALUES
33     (1, 'DE', 'SN', 'Chemnitz', 'Zentrum', '09111', 'Rathaus'),
34     (2, 'CN', 'AH', 'Hefei', 'Jinkaiqu', '230601', 'Hefei University'),
35     (3, 'US', 'NY', 'New York', 'Manhattan', '10036', 'Times Square');
```

```
1  $ psql "postgres://postgres:XXX@localhost/person_database" -v
2  ↳ ON_ERROR_STOP=1 -ebf insert.sql
```

```
3  INSERT 0 3
4  INSERT 0 4
5  INSERT 0 2
6  INSERT 0 6
7  INSERT 0 3
```

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

Daten Einfügen

- Fügen wir nun Daten in die Tabellen ein.
- Wir fangen mit drei Datensätzen für Tabelle `person` an.
- Dabei müssen wir nur jeweils das Geburtsdatum angeben.
- Danach speichern wir die Namen der drei Leute in Tabelle `name`.

```
/** Insert some data into the tables of our person database. */
2
3 -- Create the persons Bebbo, Bibbo, and Bebbi.
4 INSERT INTO person (date_of_birth) VALUES
5     ('2005-08-07'), ('1995-01-02'), ('1963-11-13');
6
7
8 -- Create the names of Bebbo, Bibbo, and Bebbi.
9 INSERT INTO name (person, full_name, salutation, is_official,
10     start_date, end_date) VALUES
11     (1, 'Bebbo', 'Bebbo Machine', TRUE, '2005-08-07', NULL),
12     (2, 'Bibbo', 'The Bib-Man', TRUE, '1995-01-02', NULL),
13     (3, 'Bibbi', 'Ms. Bibbi', FALSE, '1963-11-13', '1989-02-12'),
14     (3, 'Bebba', 'Mrs. Bebbi', TRUE, '1989-02-12', NULL);
15
16 -- Create two ID types: Chinese national ID and mobile phone numbers.
17 INSERT INTO id_type (name, validation_regexp) VALUES
18     ('national ID', '~\d{6}((19)|(20))\d{9}[0-9X]$'),
19     ('mobile phone number', '~\d{11}$');
20
21 -- Insert the personal IDs and mobile phone numbers of the people.
22 INSERT INTO has_id (id_type, person, value, valid_from) VALUES
23     (1, 1, '123456200508071234', '2021-09-21'),
24     (1, 2, '123456199501021234', '2024-12-01'),
25     (1, 3, '123456196311131234', '1983-10-03'),
26     (2, 1, '22222222222', '2020-09-12'),
27     (2, 2, '11111111111', '2012-07-30'),
28     (2, 3, '44444444444', '2012-07-30');
29
30 -- Provide an address for each person.
31 INSERT INTO address (person, country, province, city,
32     district, postal_code, street_address) VALUES
33     (1, 'DE', 'SN', 'Chemnitz', 'Zentrum', '09111', 'Rathaus'),
34     (2, 'CN', 'AH', 'Hefei', 'Jinkaiqu', '230601', 'Hefei University'),
35     (3, 'US', 'NY', 'New York', 'Manhattan', '10036', 'Times Square');
```

```
1 $ psql "postgres://postgres:XXX@localhost/person_database" -v
2   ↳ ON_ERROR_STOP=1 -ebf insert.sql
```

```
3 INSERT 0 3
4 INSERT 0 4
5 INSERT 0 2
6 INSERT 0 6
7 INSERT 0 3
```

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

Daten Einfügen

- Fügen wir nun Daten in die Tabellen ein.
- Wir fangen mit drei Datensätzen für Tabelle `person` an.
- Dabei müssen wir nur jeweils das Geburtsdatum angeben.
- Danach speichern wir die Namen der drei Leute in Tabelle `name`.
- Interessanterweise speichern wir vier Namenseinträge.

```
/** Insert some data into the tables of our person database. */  
2  
3 -- Create the persons Bebbo, Bibbo, and Bebbi.  
4 INSERT INTO person (date_of_birth) VALUES  
5 ('2005-08-07'), ('1995-01-02'), ('1963-11-13');  
6  
7  
8 -- Create the names of Bebbo, Bibbo, and Bebbi.  
9 INSERT INTO name (person, full_name, salutation, is_official,  
10 start_date, end_date) VALUES  
11 (1, 'Bebbo', 'Bebbo Machine', TRUE, '2005-08-07', NULL),  
12 (2, 'Bibbo', 'The Bib-Man', TRUE, '1995-01-02', NULL),  
13 (3, 'Bibbi', 'Ms. Bibbi', FALSE, '1963-11-13', '1989-02-12'),  
14 (3, 'Bebba', 'Mrs. Bebbi', TRUE, '1989-02-12', NULL);  
15  
16 -- Create two ID types: Chinese national ID and mobile phone numbers.  
17 INSERT INTO id_type (name, validation_regexp) VALUES  
18 ('national ID', '~\d{6}((19)|(20))\d{9}[0-9X]$'),  
19 ('mobile phone number', '~\d{11}$');  
20  
21 -- Insert the personal IDs and mobile phone numbers of the people.  
22 INSERT INTO has_id (id_type, person, value, valid_from) VALUES  
23 (1, 1, '123456200508071234', '2021-09-21'),  
24 (1, 2, '123456199501021234', '2024-12-01'),  
25 (1, 3, '123456196311131234', '1983-10-03'),  
26 (2, 1, '22222222222', '2020-09-12'),  
27 (2, 2, '11111111111', '2012-07-30'),  
28 (2, 3, '44444444444', '2012-07-30');  
29  
30 -- Provide an address for each person.  
31 INSERT INTO address (person, country, province, city,  
32 district, postal_code, street_address) VALUES  
33 (1, 'DE', 'SN', 'Chemnitz', 'Zentrum', '09111', 'Rathaus'),  
34 (2, 'CN', 'AH', 'Hefei', 'Jinkaiqu', '230601', 'Hefei University'),  
35 (3, 'US', 'NY', 'New York', 'Manhattan', '10036', 'Times Square');
```

```
1 $ psql "postgres://postgres:XXX@localhost/person_database" -v  
2 ↪ ON_ERROR_STOP=1 -ebf insert.sql
```

```
3 INSERT 0 3  
4 INSERT 0 4  
5 INSERT 0 2  
6 INSERT 0 6  
7 INSERT 0 3
```

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

Daten Einfügen

- Wir fangen mit drei Datensätzen für Tabelle `person` an.
- Dabei müssen wir nur jeweils das Geburtsdatum angeben.
- Danach speichern wir die Namen der drei Leute in Tabelle `name`.
- Interessanterweise speichern wir vier Namenseinträge.
- Der Grund ist, dass am 2. February 1989 die 26-jährige *Ms. Bibbi* geheiratet hat und sich entschlossen hat, den Namen ihres Ehemannes zu übernehmen.

```
/** Insert some data into the tables of our person database. */
2
3 -- Create the persons Bebbo, Bibbo, and Bebbi.
4 INSERT INTO person (date_of_birth) VALUES
5     ('2005-08-07'), ('1995-01-02'), ('1963-11-13');
6
7
8 -- Create the names of Bebbo, Bibbo, and Bebbi.
9 INSERT INTO name (person, full_name, salutation, is_official,
10     start_date, end_date) VALUES
11     (1, 'Bebbo', 'Bebbo Machine', TRUE, '2005-08-07', NULL),
12     (2, 'Bibbo', 'The Bib-Man', TRUE, '1995-01-02', NULL),
13     (3, 'Bibbi', 'Ms. Bibbi', FALSE, '1963-11-13', '1989-02-12'),
14     (3, 'Bebba', 'Mrs. Bebbi', TRUE, '1989-02-12', NULL);
15
16 -- Create two ID types: Chinese national ID and mobile phone numbers.
17 INSERT INTO id_type (name, validation_regexp) VALUES
18     ('national ID', '~\d{6}((19)|(20))\d{9}[0-9X]$'),
19     ('mobile phone number', '~\d{11}$');
20
21 -- Insert the personal IDs and mobile phone numbers of the people.
22 INSERT INTO has_id (id_type, person, value, valid_from) VALUES
23     (1, 1, '123456200508071234', '2021-09-21'),
24     (1, 2, '123456199501021234', '2024-12-01'),
25     (1, 3, '123456196311131234', '1983-10-03'),
26     (2, 1, '22222222222', '2020-09-12'),
27     (2, 2, '11111111111', '2012-07-30'),
28     (2, 3, '44444444444', '2012-07-30');
29
30 -- Provide an address for each person.
31 INSERT INTO address (person, country, province, city,
32     district, postal_code, street_address) VALUES
33     (1, 'DE', 'SN', 'Chemnitz', 'Zentrum', '09111', 'Rathaus'),
34     (2, 'CN', 'AH', 'Hefei', 'Jinkaiqu', '230601', 'Hefei University'),
35     (3, 'US', 'NY', 'New York', 'Manhattan', '10036', 'Times Square');
36
37
38 $ psql "postgres://postgres:XXX@localhost/person_database" -v
39     ↪ ON_ERROR_STOP=1 -ebf insert.sql
40
41 INSERT 0 3
42 INSERT 0 4
43 INSERT 0 2
44 INSERT 0 6
45 INSERT 0 3
46
47 # psql 16.11 succeeded with exit code 0.
```

Daten Einfügen

- Dabei müssen wir nur jeweils das Geburtsdatum angeben.
- Danach speichern wir die Namen der drei Leute in Tabelle `name`.
- Interessanterweise speichern wir vier Namenseinträge.
- Der Grund ist, dass am 2. February 1989 die 26-jährige *Ms. Bibbi* geheiratet hat und sich entschlossen hat, den Namen ihres Ehemannes zu übernehmen.
- Ihr Name hat sich also zu *Mrs. Bebbä* geändert.

```
/** Insert some data into the tables of our person database. */
2
3 -- Create the persons Bebbö, Bibbo, and Bebbä.
4 INSERT INTO person (date_of_birth) VALUES
5     ('2005-08-07'), ('1995-01-02'), ('1963-11-13');
6
7 -- Create the names of Bebbö, Bibbo, and Bebbä.
8 INSERT INTO name (person, full_name, salutation, is_official,
9     start_date, end_date) VALUES
10     (1, 'Bebbo', 'Bebbo Machine', TRUE, '2005-08-07', NULL),
11     (2, 'Bibbo', 'The Bib-Man', TRUE, '1995-01-02', NULL),
12     (3, 'Bibbi', 'Ms. Bibbi', FALSE, '1963-11-13', '1989-02-12'),
13     (3, 'Bebba', 'Mrs. Bebbä', TRUE, '1989-02-12', NULL);
14
15 -- Create two ID types: Chinese national ID and mobile phone numbers.
16 INSERT INTO id_type (name, validation_regexp) VALUES
17     ('national ID', '~\d{6}((19)|(20))\d{9}[0-9X]$'),
18     ('mobile phone number', '~\d{11}$');
19
20 -- Insert the personal IDs and mobile phone numbers of the people.
21 INSERT INTO has_id (id_type, person, value, valid_from) VALUES
22     (1, 1, '123456200508071234', '2021-09-21'),
23     (1, 2, '123456199501021234', '2024-12-01'),
24     (1, 3, '123456196311131234', '1983-10-03'),
25     (2, 1, '22222222222', '2020-09-12'),
26     (2, 2, '11111111111', '2012-07-30'),
27     (2, 3, '44444444444', '2012-07-30');
28
29 -- Provide an address for each person.
30 INSERT INTO address (person, country, province, city,
31     district, postal_code, street_address) VALUES
32     (1, 'DE', 'SN', 'Chemnitz', 'Zentrum', '09111', 'Rathaus'),
33     (2, 'CN', 'AH', 'Hefei', 'Jinkaiqu', '230601', 'Hefei University'),
34     (3, 'US', 'NY', 'New York', 'Manhattan', '10036', 'Times Square');
35
36 $ psql "postgres://postgres:XXX@localhost/person_database" -v
37   ↳ ON_ERROR_STOP=1 -ebf insert.sql
38
39 INSERT 0 3
40 INSERT 0 4
41 INSERT 0 2
42 INSERT 0 6
43 INSERT 0 3
44
45 # psql 16.11 succeeded with exit code 0.
```

Daten Einfügen

- Danach speichern wir die Namen der drei Leute in Tabelle `name`.
- Interessanterweise speichern wir vier Namenseinträge.
- Der Grund ist, dass am 2. February 1989 die 26-jährige *Ms. Bibbi* geheiratet hat und sich entschlossen hat, den Namen ihres Ehemannes zu übernehmen.
- Ihr Name hat sich also zu *Mrs. Bebbä* geändert.
- Das sehen wir an den letzten beiden Zeilen, die in Tabelle `name` eingefügt wurden.

```
/** Insert some data into the tables of our person database. */
2
3 -- Create the persons Bebbo, Bibbo, and Bebbä.
4 INSERT INTO person (date_of_birth) VALUES
5     ('2005-08-07'), ('1995-01-02'), ('1963-11-13');
6
7 -- Create the names of Bebbo, Bibbo, and Bebbä.
8 INSERT INTO name (person, full_name, salutation, is_official,
9     start_date, end_date) VALUES
10     (1, 'Bebbo', 'Bebbo Machine', TRUE, '2005-08-07', NULL),
11     (2, 'Bibbo', 'The Bib-Man', TRUE, '1995-01-02', NULL),
12     (3, 'Bibbi', 'Ms. Bibbi', FALSE, '1963-11-13', '1989-02-12'),
13     (3, 'Bebba', 'Mrs. Bebbä', TRUE, '1989-02-12', NULL);
14
15 -- Create two ID types: Chinese national ID and mobile phone numbers.
16 INSERT INTO id_type (name, validation_regexp) VALUES
17     ('national ID', '~\d{6}((19)|(20))\d{9}[0-9X]$'),
18     ('mobile phone number', '~\d{11}$');
19
20 -- Insert the personal IDs and mobile phone numbers of the people.
21 INSERT INTO has_id (id_type, person, value, valid_from) VALUES
22     (1, 1, '123456200508071234', '2021-09-21'),
23     (1, 2, '123456199501021234', '2024-12-01'),
24     (1, 3, '123456196311131234', '1983-10-03'),
25     (2, 1, '2222222222', '2020-09-12'),
26     (2, 2, '1111111111', '2012-07-30'),
27     (2, 3, '4444444444', '2012-07-30');
28
29 -- Provide an address for each person.
30 INSERT INTO address (person, country, province, city,
31     district, postal_code, street_address) VALUES
32     (1, 'DE', 'SN', 'Chemnitz', 'Zentrum', '09111', 'Rathaus'),
33     (2, 'CN', 'AH', 'Hefei', 'Jinkaiqu', '230601', 'Hefei University'),
34     (3, 'US', 'NY', 'New York', 'Manhattan', '10036', 'Times Square');
35
36 $ psql "postgres://postgres:XXX@localhost/person_database" -v
37   ↳ ON_ERROR_STOP=1 -ebf insert.sql
38
39 INSERT 0 3
40 INSERT 0 4
41 INSERT 0 2
42 INSERT 0 6
43 INSERT 0 3
44
45 # psql 16.11 succeeded with exit code 0.
```

Daten Einfügen

- Der Grund ist, dass am 2. February 1989 die 26-jährige *Ms. Bibbi* geheiratet hat und sich entschlossen hat, den Namen ihres Ehemannes zu übernehmen.
- Ihr Name hat sich also zu *Mrs. Bebbä* geändert.
- Das sehen wir an den letzten beiden Zeilen, die in Tabelle `name` eingefügt wurden.
- Die selben beiden ID-Typen wie in den vergangenen Beispielen – chinesische Ausweisnummer und Mobiltelefonnummer – werden in Tabelle `id_type` erstellt.

```
1  /** Insert some data into the tables of our person database. */
2
3  -- Create the persons Bebbö, Bibbo, and Bebbä.
4  INSERT INTO person (date_of_birth) VALUES
5      ('2005-08-07'), ('1995-01-02'), ('1963-11-13');
6
7
8  -- Create the names of Bebbö, Bibbo, and Bebbä.
9  INSERT INTO name (person, salutation, is_official,
10                  start_date, end_date) VALUES
11      (1, 'Bebbo', 'Bebbo Machine', TRUE, '2005-08-07', NULL),
12      (2, 'Bibbo', 'The Bib-Man', TRUE, '1995-01-02', NULL),
13      (3, 'Bibbi', 'Ms. Bibbi', FALSE, '1963-11-13', '1989-02-12'),
14      (3, 'Bebba', 'Mrs. Bebbä', TRUE, '1989-02-12', NULL);
15
16 -- Create two ID types: Chinese national ID and mobile phone numbers.
17 INSERT INTO id_type (name, validation_regexp) VALUES
18     ('national ID', '~\d{6}((19)|(20))\d{9}[0-9X]$'),
19     ('mobile phone number', '~\d{11}$');
20
21 -- Insert the personal IDs and mobile phone numbers of the people.
22 INSERT INTO has_id (id_type, person, value, valid_from) VALUES
23     (1, 1, '123456200508071234', '2021-09-21'),
24     (1, 2, '123456199501021234', '2024-12-01'),
25     (1, 3, '123456196311131234', '1983-10-03'),
26     (2, 1, '2222222222', '2020-09-12'),
27     (2, 2, '1111111111', '2012-07-30'),
28     (2, 3, '4444444444', '2012-07-30');
29
30 -- Provide an address for each person.
31 INSERT INTO address (person, country, province, city,
32                    district, postal_code, street_address) VALUES
33     (1, 'DE', 'SN', 'Chemnitz', 'Zentrum', '09111', 'Rathaus'),
34     (2, 'CN', 'AH', 'Hefei', 'Jinkaiqu', '230601', 'Hefei University'),
35     (3, 'US', 'NY', 'New York', 'Manhattan', '10036', 'Times Square');
```

```
1  $ psql "postgres://postgres:XXX@localhost/person_database" -v
2  ↳ ON_ERROR_STOP=1 -ebf insert.sql
```

```
3  INSERT 0 3
4  INSERT 0 4
5  INSERT 0 2
6  INSERT 0 6
7  INSERT 0 3
```

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

Daten Einfügen

- Ihr Name hat sich also zu *Mrs. Bebba* geändert.
- Das sehen wir an den letzten beiden Zeilen, die in Tabelle `name` eingefügt wurden.
- Die selben beiden ID-Typen wie in den vergangenen Beispielen – chinesische Ausweisnummer und Mobiltelefonnummer – werden in Tabelle `id_type` erstellt.
- Für jeden ID-Typ und jede Person speichern wir dann einen Wert in Tabelle `has_id`.

```
/** Insert some data into the tables of our person database. */
2
3 -- Create the persons Bebbo, Bibbo, and Bebba.
4 INSERT INTO person (date_of_birth) VALUES
5     ('2005-08-07'), ('1995-01-02'), ('1963-11-13');
6
7 -- Create the names of Bebbo, Bibbo, and Bebba.
8 INSERT INTO name (person, full_name, salutation, is_official,
9     start_date, end_date) VALUES
10     (1, 'Bebbo', 'Bebbo Machine', TRUE, '2005-08-07', NULL),
11     (2, 'Bibbo', 'The Bib-Man', TRUE, '1995-01-02', NULL),
12     (3, 'Bibbi', 'Ms. Bibbi', FALSE, '1963-11-13', '1989-02-12'),
13     (3, 'Bebba', 'Mrs. Bebba', TRUE, '1989-02-12', NULL);
14
15 -- Create two ID types: Chinese national ID and mobile phone numbers.
16 INSERT INTO id_type (name, validation_regexp) VALUES
17     ('national ID', '~\d{6}((19)|(20))\d{9}[0-9X]$'),
18     ('mobile phone number', '~\d{11}$');
19
20 -- Insert the personal IDs and mobile phone numbers of the people.
21 INSERT INTO has_id (id_type, person, value, valid_from) VALUES
22     (1, 1, '123456200508071234', '2021-09-21'),
23     (1, 2, '123456199501021234', '2024-12-01'),
24     (1, 3, '123456196311131234', '1983-10-03'),
25     (2, 1, '22222222222', '2020-09-12'),
26     (2, 2, '11111111111', '2012-07-30'),
27     (2, 3, '44444444444', '2012-07-30');
28
29 -- Provide an address for each person.
30 INSERT INTO address (person, country, province, city,
31     district, postal_code, street_address) VALUES
32     (1, 'DE', 'SN', 'Chemnitz', 'Zentrum', '09111', 'Rathaus'),
33     (2, 'CN', 'AH', 'Hefei', 'Jinkaiqu', '230601', 'Hefei University'),
34     (3, 'US', 'NY', 'New York', 'Manhattan', '10036', 'Times Square');
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100
101
102
103
104
105
106
107
108
109
110
111
112
113
114
115
116
117
118
119
120
121
122
123
124
125
126
127
128
129
130
131
132
133
134
135
136
137
138
139
140
141
142
143
144
145
146
147
148
149
150
151
152
153
154
155
156
157
158
159
160
161
162
163
164
165
166
167
168
169
170
171
172
173
174
175
176
177
178
179
180
181
182
183
184
185
186
187
188
189
190
191
192
193
194
195
196
197
198
199
200
201
202
203
204
205
206
207
208
209
210
211
212
213
214
215
216
217
218
219
220
221
222
223
224
225
226
227
228
229
230
231
232
233
234
235
236
237
238
239
240
241
242
243
244
245
246
247
248
249
250
251
252
253
254
255
256
257
258
259
260
261
262
263
264
265
266
267
268
269
270
271
272
273
274
275
276
277
278
279
280
281
282
283
284
285
286
287
288
289
290
291
292
293
294
295
296
297
298
299
300
301
302
303
304
305
306
307
308
309
310
311
312
313
314
315
316
317
318
319
320
321
322
323
324
325
326
327
328
329
330
331
332
333
334
335
336
337
338
339
340
341
342
343
344
345
346
347
348
349
350
351
352
353
354
355
356
357
358
359
360
361
362
363
364
365
366
367
368
369
370
371
372
373
374
375
376
377
378
379
380
381
382
383
384
385
386
387
388
389
390
391
392
393
394
395
396
397
398
399
400
401
402
403
404
405
406
407
408
409
410
411
412
413
414
415
416
417
418
419
420
421
422
423
424
425
426
427
428
429
430
431
432
433
434
435
436
437
438
439
440
441
442
443
444
445
446
447
448
449
450
451
452
453
454
455
456
457
458
459
460
461
462
463
464
465
466
467
468
469
470
471
472
473
474
475
476
477
478
479
480
481
482
483
484
485
486
487
488
489
490
491
492
493
494
495
496
497
498
499
500
501
502
503
504
505
506
507
508
509
510
511
512
513
514
515
516
517
518
519
520
521
522
523
524
525
526
527
528
529
530
531
532
533
534
535
536
537
538
539
540
541
542
543
544
545
546
547
548
549
550
551
552
553
554
555
556
557
558
559
560
561
562
563
564
565
566
567
568
569
570
571
572
573
574
575
576
577
578
579
580
581
582
583
584
585
586
587
588
589
590
591
592
593
594
595
596
597
598
599
600
601
602
603
604
605
606
607
608
609
610
611
612
613
614
615
616
617
618
619
620
621
622
623
624
625
626
627
628
629
630
631
632
633
634
635
636
637
638
639
640
641
642
643
644
645
646
647
648
649
650
651
652
653
654
655
656
657
658
659
660
661
662
663
664
665
666
667
668
669
670
671
672
673
674
675
676
677
678
679
680
681
682
683
684
685
686
687
688
689
690
691
692
693
694
695
696
697
698
699
700
701
702
703
704
705
706
707
708
709
710
711
712
713
714
715
716
717
718
719
720
721
722
723
724
725
726
727
728
729
730
731
732
733
734
735
736
737
738
739
740
741
742
743
744
745
746
747
748
749
750
751
752
753
754
755
756
757
758
759
760
761
762
763
764
765
766
767
768
769
770
771
772
773
774
775
776
777
778
779
780
781
782
783
784
785
786
787
788
789
790
791
792
793
794
795
796
797
798
799
800
801
802
803
804
805
806
807
808
809
810
811
812
813
814
815
816
817
818
819
820
821
822
823
824
825
826
827
828
829
830
831
832
833
834
835
836
837
838
839
840
841
842
843
844
845
846
847
848
849
850
851
852
853
854
855
856
857
858
859
860
861
862
863
864
865
866
867
868
869
870
871
872
873
874
875
876
877
878
879
880
881
882
883
884
885
886
887
888
889
890
891
892
893
894
895
896
897
898
899
900
901
902
903
904
905
906
907
908
909
910
911
912
913
914
915
916
917
918
919
920
921
922
923
924
925
926
927
928
929
930
931
932
933
934
935
936
937
938
939
940
941
942
943
944
945
946
947
948
949
950
951
952
953
954
955
956
957
958
959
960
961
962
963
964
965
966
967
968
969
970
971
972
973
974
975
976
977
978
979
980
981
982
983
984
985
986
987
988
989
990
991
992
993
994
995
996
997
998
999
1000
1001
1002
1003
1004
1005
1006
1007
1008
1009
1010
1011
1012
1013
1014
1015
1016
1017
1018
1019
1020
1021
1022
1023
1024
1025
1026
1027
1028
1029
1030
1031
1032
1033
1034
1035
1036
1037
1038
1039
1040
1041
1042
1043
1044
1045
1046
1047
1048
1049
1050
1051
1052
1053
1054
1055
1056
1057
1058
1059
1060
1061
1062
1063
1064
1065
1066
1067
1068
1069
1070
1071
1072
1073
1074
1075
1076
1077
1078
1079
1080
1081
1082
1083
1084
1085
1086
1087
1088
1089
1090
1091
1092
1093
1094
1095
1096
1097
1098
1099
1100
1101
1102
1103
1104
1105
1106
1107
1108
1109
1110
1111
1112
1113
1114
1115
1116
1117
1118
1119
1120
1121
1122
1123
1124
1125
1126
1127
1128
1129
1130
1131
1132
1133
1134
1135
1136
1137
1138
1139
1140
1141
1142
1143
1144
1145
1146
1147
1148
1149
1150
1151
1152
1153
1154
1155
1156
1157
1158
1159
1160
1161
1162
1163
1164
1165
1166
1167
1168
1169
1170
1171
1172
1173
1174
1175
1176
1177
1178
1179
1180
1181
1182
1183
1184
1185
1186
1187
1188
1189
1190
1191
1192
1193
1194
1195
1196
1197
1198
1199
1200
1201
1202
1203
1204
1205
1206
1207
1208
1209
1210
1211
1212
1213
1214
1215
1216
1217
1218
1219
1220
1221
1222
1223
1224
1225
1226
1227
1228
1229
1230
1231
1232
1233
1234
1235
1236
1237
1238
1239
1240
1241
1242
1243
1244
1245
1246
1247
1248
1249
1250
1251
1252
1253
1254
1255
1256
1257
1258
1259
1260
1261
1262
1263
1264
1265
1266
1267
1268
1269
1270
1271
1272
1273
1274
1275
1276
1277
1278
1279
1280
1281
1282
1283
1284
1285
1286
1287
1288
1289
1290
1291
1292
1293
1294
1295
1296
1297
1298
1299
1300
1301
1302
1303
1304
1305
1306
1307
1308
1309
1310
1311
1312
1313
1314
1315
1316
1317
1318
1319
1320
1321
1322
1323
1324
1325
1326
1327
1328
1329
1330
1331
1332
1333
1334
1335
1336
1337
1338
1339
1340
1341
1342
1343
1344
1345
1346
1347
1348
1349
1350
1351
1352
1353
1354
1355
1356
1357
1358
1359
1360
1361
1362
1363
1364
1365
1366
1367
1368
1369
1370
1371
1372
1373
1374
1375
1376
1377
1378
1379
1380
1381
1382
1383
1384
1385
1386
1387
1388
1389
1390
1391
1392
1393
1394
1395
1396
1397
1398
1399
1400
1401
1402
1403
1404
1405
1406
1407
1408
1409
1410
1411
1412
1413
1414
1415
1416
1417
1418
1419
1420
1421
1422
1423
1424
1425
1426
1427
1428
1429
1430
1431
1432
1433
1434
1435
1436
1437
1438
1439
1440
1441
1442
1443
1444
1445
1446
1447
1448
1449
1450
1451
1452
1453
1454
1455
1456
1457
1458
1459
1460
1461
1462
1463
1464
1465
1466
1467
1468
1469
1470
1471
1472
1473
1474
1475
1476
1477
1478
1479
1480
1481
1482
1483
1484
1485
1486
1487
1488
1489
1490
1491
1492
1493
1494
1495
1496
1497
1498
1499
1500
1501
1502
1503
1504
1505
1506
1507
1508
1509
1510
1511
1512
1513
1514
1515
1516
1517
1518
1519
1520
1521
1522
1523
1524
1525
1526
1527
1528
1529
1530
1531
1532
1533
1534
1535
1536
1537
1538
1539
1540
1541
1542
1543
1544
1545
1546
1547
1548
1549
1550
1551
1552
1553
1554
1555
1556
1557
1558
1559
1560
1561
1562
1563
1564
1565
1566
1567
1568
1569
1570
1571
1572
1573
1574
1575
1576
1577
1578
1579
1580
1581
1582
1583
1584
1585
1586
1587
1588
1589
1590
1591
1592
1593
1594
1595
1596
1597
1598
1599
1600
1601
1602
1603
1604
1605
1606
1607
1608
1609
1610
1611
1612
1613
1614
1615
1616
1617
1618
1619
1620
1621
1622
1623
1624
1625
1626
1627
1628
1629
1630
1631
1632
1633
1634
1635
1636
1637
1638
1639
1640
1641
1642
1643
1644
1645
1646
1647
1648
1649
1650
1651
1652
1653
1654
1655
1656
1657
1658
1659
1660
1661
1662
1663
1664
1665
1666
1667
1668
1669
1670
1671
1672
1673
1674
1675
1676
1677
1678
1679
1680
1681
1682
1683
1684
1685
1686
1687
1688
1689
1690
1691
1692
1693
1694
1695
1696
1697
1698
1699
1700
1701
1702
1703
1704
1705
1706
1707
1708
1709
1710
1711
1712
1713
1714
1715
1716
1717
1718
1719
1720
1721
1722
1723
1724
1725
1726
1727
1728
1729
1730
1731
1732
1733
1734
1735
1736
1737
1738
1739
1740
1741
1742
1743
1744
1745
1746
1747
1748
1749
1750
1751
1752
1753
1754
1755
1756
1757
1758
1759
1760
1761
1762
1763
1764
1765
1766
1767
1768
1769
1770
1771
1772
1773
1774
1775
1776
1777
1778
1779
1780
1781
1782
1783
1784
1785
1786
1787
1788
1789
1790
1791
1792
1793
1794
1795
1796
1797
1798
1799
1800
1801
1802
1803
1804
1805
1806
1807
1808
1809
1810
1811
1812
1813
1814
1815
1816
1817
1818
1819
1820
1821
1822
1823
1824
1825
1826
1827
1828
1829
1830
1831
1832
1833
1834
1835
1836
1837
1838
1839
1840
1841
1842
1843
1844
1845
1846
1847
1848
1849
1850
1851
1852
1853
1854
1855
1856
1857
1858
1859
1860
1861
1862
1863
1864
1865
1866
1867
1868
1869
1870
1871
1872
1873
1874
1875
1876
1877
1878
1879
1880
1881
1882
1883
1884
1885
1886
1887
1888
1889
1890
1891
1892
1893
1894
1895
1896
1897
1898
1899
1900
1901
1902
1903
1904
1905
1906
1907
1908
1909
1910
1911
1912
1913
1914
1915
1916
1917
1918
1919
1920
1921
1922
1923
1924
1925
1926
1927
1928
1929
1930
1931
1932
1933
1934
1935
1936
1937
1938
1939
1940
1941
1942
1943
1944
1945
1946
1947
1948
1949
1950
1951
1952
1953
1954
1955
1956
1957
1958
1959
1960
1961
1962
1963
1964
1965
1966
1967
1968
1969
1970
1971
1972
1973
1974
1975
1976
1977
1978
1979
1980
1981
1982
1983
1984
1985
1986
1987
1988
1989
1990
1991
1992
1993
1994
1995
1996
1997
1998
1999
2000
2001
2002
2003
2004
2005
2006
2007
2008
2009
2010
2011
2012
2013
2014
2015
2016
2017
2018
2019
2020
2021
2022
2023
2024
2025
2026
2027
2028
2029
2030
2031
2032
2033
2034
2035
2036
2037
2038
2039
2040
2041
2042
2043
2044
2045
2046
2047
2048
2049
2050
2051
2052
2053
2054
2055
2056
2057
2058
2059
2060
2061
2062
2063
2064
2065
2066
2067
2068
2069
2070
2071
2072
2073
2074
2075
2076
2077
2078
2079
2080
2081
2082
2083
2084
2085
2086
2087
2088
2089
2090
2091
2092
2093
2094
2095
2096
2097
2098
2099
2100
2101
2102
2103
2104
2105
2106
2107
2108
2109
2110
2111
2112
2113
2114
2115
2116
2117
2118
2119
2120
2121
2122
2123
2124
2125
2126
2127
2128
2129
2130
2131
2132
2133
2134
2135
2136
2137
2138
2139
2140
2141
2142
2143
2144
2145
2146
2147
2148
2149
2150
2151
2152
2153
2154
2155
2156
2157
2158
2159
2160
2161
2162
2163
2164
2165
2166
2167
2168
2169
2170
2171
2172
2173
2174
2175
2176
2177
2178
2179
2180
2181
2182
2183
2184
2185
2186
2187
2188
2189
2190
2191
2192
2193
2194
2195
2196
2197
2198
2199
2200
2201
2202
2203
2204
2205
2206
2207
2208
2209
2210
2211
2212
2213
2214
2215
2216
2217
2218
2219
2220
2221
2222
2223
2224
2225
2226
2227
2228
2229
2230
2231
2232
2233
2234
2235
2236
2237
2238
2239
2240
2241
2242
2243
2244
2245
2246
2247
2248
2249
2250
2251
2252
2253
2254
2255
2256
2257
2258
2259
2260
2261
2262
2263
2264
2265
2266
2267
2268
2269
2270
2271
2272
2273
2274
2275
2276
2277
2278
2279
2280
2281
2282
2283
2284
2285
2286
2287
2288
2289
2290
2291
2292
2293
2294
2295
2296
2297
2298
2299
2300
2301
2302
2303
2304
2305
2306
2307
2308
2309
2310
2311
2312
2313
2314
2315
2316
2317
2318
2319
2320
2321
2322
2323
2324
2325
2326
2327
2328
2329
2330
2331
2332
2333
2334
2335
2336
2337
2338
2339
2340
2341
2342
2343
2344
2345
2346
2347
2348
2349
2350
2351
2352
2353
2354
2355
2356
2357
2358
2359
2360
2361
2362
2363
2364
2365
2366
2367
2368
2369
2370
2371
2372
2373
2374
2375
2376
2377
2378
2379
2380
2381
2382
2383
2384
2385
2386
2387
2388
2389
2390
2391
2392
2393
2394
2395
2396
2397
2398
2399
2400
2401
2402
2403
2404
2405
2406
2407
2408
2409
2410
2411
2412
2413
2414
2415
2416
2417
2418
2419
2420
2421
2422
2423
2424
2425
2426
2427
2428
2429
2430
2431
2432
2433
2434
2435
2436
2437
2438
2439
2440
2441
2442
2443
2444
2445
2446
2447
2448
2449
2450
2451
2452
2453
2454
2455
2456
2457
2458
2459
2460
2461
2462
2463
2464
2465
2466
2467
2468
2469
2470
2471
2472
2473
2474
2475
2476
2477
2478
2479
2480
2481
2482
2483
2484
2485
2486
2487
2488
2489
2490
2491
2492
2493
2494
2495
2496
2497
2498
2499
2500
2501
2502
2503
2504
2505
2506
2507
2508
2509
2510
2511
2512
2513
2514
2515
2516
2517
2518
2
```

Daten Einfügen

- Ihr Name hat sich also zu *Mrs. Bebba* geändert.
- Das sehen wir an den letzten beiden Zeilen, die in Tabelle `name` eingefügt wurden.
- Die selben beiden ID-Typen wie in den vergangenen Beispielen – chinesische Ausweisnummer und Mobiltelefonnummer – werden in Tabelle `id_type` erstellt.
- Für jeden ID-Typ und jede Person speichern wir dann einen Wert in Tabelle `has_id`.
- Wir benutzen dabei die Beziehungsattribute, die dort gespeichert werden.

```
1  /** Insert some data into the tables of our person database. */
2
3  -- Create the persons Bebbo, Bibbo, and Bebba.
4  INSERT INTO person (date_of_birth) VALUES
5      ('2005-08-07'), ('1995-01-02'), ('1963-11-13');
6
7
8  -- Create the names of Bebbo, Bibbo, and Bebba.
9  INSERT INTO name (person, full_name, salutation, is_official,
10                  start_date, end_date) VALUES
11      (1, 'Bebbo', 'Bebbo Machine', TRUE, '2005-08-07', NULL),
12      (2, 'Bibbo', 'The Bib-Man', TRUE, '1995-01-02', NULL),
13      (3, 'Bibbi', 'Ms. Bibbi', FALSE, '1963-11-13', '1989-02-12'),
14      (3, 'Bebba', 'Mrs. Bebba', TRUE, '1989-02-12', NULL);
15
16 -- Create two ID types: Chinese national ID and mobile phone numbers.
17 INSERT INTO id_type (name, validation_regexp) VALUES
18     ('national ID', '~\d{6}((19)|(20))\d{9}[0-9X]$'),
19     ('mobile phone number', '~\d{11}$');
20
21 -- Insert the personal IDs and mobile phone numbers of the people.
22 INSERT INTO has_id (id_type, person, value, valid_from) VALUES
23     (1, 1, '123456200508071234', '2021-09-21'),
24     (1, 2, '123456199501021234', '2024-12-01'),
25     (1, 3, '123456196311131234', '1983-10-03'),
26     (2, 1, '22222222222', '2020-09-12'),
27     (2, 2, '11111111111', '2012-07-30'),
28     (2, 3, '44444444444', '2012-07-30');
29
30 -- Provide an address for each person.
31 INSERT INTO address (person, country, province, city,
32                     district, postal_code, street_address) VALUES
33     (1, 'DE', 'SN', 'Chemnitz', 'Zentrum', '09111', 'Rathaus'),
34     (2, 'CN', 'AH', 'Hefei', 'Jinkaiqu', '230601', 'Hefei University'),
35     (3, 'US', 'NY', 'New York', 'Manhattan', '10036', 'Times Square');
36
37
38 1  $ psql "postgres://postgres:XXX@localhost/person_database" -v
39    ↪ ON_ERROR_STOP=1 -ebf insert.sql
40
41 2  INSERT 0 3
42 3  INSERT 0 4
43 4  INSERT 0 2
44 5  INSERT 0 6
45 6  INSERT 0 3
46
47 7  # psql 16.11 succeeded with exit code 0.
```

Daten Einfügen

- Das sehen wir an den letzten beiden Zeilen, die in Tabelle `name` eingefügt wurden.
- Die selben beiden ID-Typen wie in den vergangenen Beispielen – chinesische Ausweisnummer und Mobiltelefonnummer – werden in Tabelle `id_type` erstellt.
- Für jeden ID-Typ und jede Person speichern wir dann einen Wert in Tabelle `has_id`.
- Wir benutzen dabei die Beziehungsattribute, die dort gespeichert werden.
- Wir geben dann noch für jede Person einen `address`-Datensatz ein.

```
/** Insert some data into the tables of our person database. */
2
3 -- Create the persons Bebbo, Bibbo, and Bebbu.
4 INSERT INTO person (date_of_birth) VALUES
5     ('2005-08-07'), ('1995-01-02'), ('1963-11-13');
6
7
8 -- Create the names of Bebbo, Bibbo, and Bebbu.
9 INSERT INTO name (person, full_name, salutation, is_official,
10     start_date, end_date) VALUES
11     (1, 'Bebbo', 'Bebbo Machine', TRUE, '2005-08-07', NULL),
12     (2, 'Bibbo', 'The Bib-Man', TRUE, '1995-01-02', NULL),
13     (3, 'Bibbi', 'Ms. Bibbi', FALSE, '1963-11-13', '1989-02-12'),
14     (3, 'Bebba', 'Mrs. Bebbu', TRUE, '1989-02-12', NULL);
15
16 -- Create two ID types: Chinese national ID and mobile phone numbers.
17 INSERT INTO id_type (name, validation_regexp) VALUES
18     ('national ID', '~\d{6}((19)|(20))\d{9}[0-9X]$'),
19     ('mobile phone number', '~\d{11}$');
20
21 -- Insert the personal IDs and mobile phone numbers of the people.
22 INSERT INTO has_id (id_type, person, value, valid_from) VALUES
23     (1, 1, '123456200508071234', '2021-09-21'),
24     (1, 2, '123456199501021234', '2024-12-01'),
25     (1, 3, '123456196311131234', '1983-10-03'),
26     (2, 1, '22222222222', '2020-09-12'),
27     (2, 2, '11111111111', '2012-07-30'),
28     (2, 3, '44444444444', '2012-07-30');
29
30 -- Provide an address for each person.
31 INSERT INTO address (person, country, province, city,
32     district, postal_code, street_address) VALUES
33     (1, 'DE', 'SN', 'Chemnitz', 'Zentrum', '09111', 'Rathaus'),
34     (2, 'CN', 'AH', 'Hefei', 'Jinkaiqu', '230601', 'Hefei University'),
35     (3, 'US', 'NY', 'New York', 'Manhattan', '10036', 'Times Square');
36
37
38 $ psql "postgres://postgres:XXX@localhost/person_database" -v
39     ↳ ON_ERROR_STOP=1 -ebf insert.sql
40
41 INSERT 0 3
42 INSERT 0 4
43 INSERT 0 2
44 INSERT 0 6
45 INSERT 0 3
46
47 # psql 16.11 succeeded with exit code 0.
```

Daten Einfügen

- Die selben beiden ID-Typen wie in den vergangenen Beispielen – chinesische Ausweisnummer und Mobiltelefonnummer – werden in Tabelle `id_type` erstellt.
- Für jeden ID-Typ und jede Person speichern wir dann einen Wert in Tabelle `has_id`.
- Wir benutzen dabei die Beziehungsattribute, die dort gespeichert werden.
- Wir geben dann noch für jede Person einen `address`-Datensatz ein.
- Damit haben wir alle Daten eingegeben.

```
1  /** Insert some data into the tables of our person database. */
2
3  -- Create the persons Bebbio, Bibbo, and Bebbia.
4  INSERT INTO person (date_of_birth) VALUES
5      ('2005-08-07'), ('1995-01-02'), ('1963-11-13');
6
7  -- Create the names of Bebbio, Bibbo, and Bebbia.
8  INSERT INTO name (person, salutation, is_official,
9                  start_date, end_date) VALUES
10     (1, 'Bebbo', 'Bebbo Machine', TRUE, '2005-08-07', NULL),
11     (2, 'Bibbo', 'The Bib-Man', TRUE, '1995-01-02', NULL),
12     (3, 'Bibbi', 'Ms. Bibbi', FALSE, '1963-11-13', '1989-02-12'),
13     (3, 'Bebba', 'Mrs. Bebbia', TRUE, '1989-02-12', NULL);
14
15  -- Create two ID types: Chinese national ID and mobile phone numbers.
16  INSERT INTO id_type (name, validation_regexp) VALUES
17     ('national ID', '~\d{6}((19)|(20))\d{9}[0-9X]$'),
18     ('mobile phone number', '~\d{11}$');
19
20  -- Insert the personal IDs and mobile phone numbers of the people.
21  INSERT INTO has_id (id_type, person, value, valid_from) VALUES
22     (1, 1, '123456200508071234', '2021-09-21'),
23     (1, 2, '123456199501021234', '2024-12-01'),
24     (1, 3, '123456196311131234', '1983-10-03'),
25     (2, 1, '22222222222', '2020-09-12'),
26     (2, 2, '11111111111', '2012-07-30'),
27     (2, 3, '44444444444', '2012-07-30');
28
29  -- Provide an address for each person.
30  INSERT INTO address (person, country, province, city,
31                      district, postal_code, street_address) VALUES
32     (1, 'DE', 'SN', 'Chemnitz', 'Zentrum', '09111', 'Rathaus'),
33     (2, 'CN', 'AH', 'Hefei', 'Jinkaiqu', '230601', 'Hefei University'),
34     (3, 'US', 'NY', 'New York', 'Manhattan', '10036', 'Times Square');
35
36
37  1  $ psql "postgres://postgres:XXX@localhost/person_database" -v
38     ↳ ON_ERROR_STOP=1 -ebf insert.sql
39
40  2  INSERT 0 3
41  3  INSERT 0 4
42  4  INSERT 0 2
43  5  INSERT 0 6
44  6  INSERT 0 3
45
46  7  # psql 16.11 succeeded with exit code 0.
```

Daten Auslesen

- Und jetzt lesen wir die Daten wieder aus.



```
1  /** Combine people, names, addresses, and phone numbers. */
2
3  SELECT full_name || ': ' || country || '-' || province || ', ' ||
4         city || ', ' || street_address ||
5         ' (call ' || value || ')' AS contact FROM name
6         INNER JOIN person ON person.id = name.person
7         INNER JOIN address ON person.id = address.person
8         INNER JOIN has_id ON person.id = has_id.person
9         WHERE name.is_official = TRUE AND -- use official name only
10              has_id.id_type = 2          -- use ID type for mobile phone
11         ORDER BY person.date_of_birth; -- sort by age, oldest people first

$ psql "postgres://postgres:XXX@localhost/person_database" -v
↪ ON_ERROR_STOP=1 -ebf select.sql
      contact
-----
Bebba: US-NY, New York, Times Square (call 444444444444)
Bibbo: CN-AH, Hefei, Hefei University (call 111111111111)
Bebbo: DE-SN, Chemnitz, Rathaus (call 222222222222)
(3 rows)

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

Daten Auslesen

- Und jetzt lesen wir die Daten wieder aus.
- Wir verwenden dafür das vielleicht größte **INNER JOIN**, das wir bisher verwendet haben.



```
1  /** Combine people, names, addresses, and phone numbers. */
2
3  SELECT full_name || ': ' || country || '-' || province || ', ' ||
4         city || ', ' || street_address ||
5         ' (call ' || value || ')' AS contact FROM name
6         INNER JOIN person ON person.id = name.person
7         INNER JOIN address ON person.id = address.person
8         INNER JOIN has_id ON person.id = has_id.person
9         WHERE name.is_official = TRUE AND -- use official name only
10              has_id.id_type = 2 -- use ID type for mobile phone
11         ORDER BY person.date_of_birth; -- sort by age, oldest people first
```

```
1  $ psql "postgres://postgres:XXX@localhost/person_database" -v
2  ↪ ON_ERROR_STOP=1 -ebf select.sql
3         contact
4  -----
5  Bebba: US-NY, New York, Times Square (call 444444444444)
6  Bibbo: CN-AH, Hefei, Hefei University (call 111111111111)
7  Bebbbo: DE-SN, Chemnitz, Rathaus (call 222222222222)
8  (3 rows)
9
10 # psql 16.11 succeeded with exit code 0.
```

Daten Auslesen



- Und jetzt lesen wir die Daten wieder aus.
- Wir verwenden dafür das vielleicht größte **INNER JOIN**, das wir bisher verwendet haben.
- Wir wollen die Namen der Personen mit ihren Adressen und Mobiltelefonnummern kombinieren.

```
1  /** Combine people, names, addresses, and phone numbers. */
2
3  SELECT full_name || ': ' || country || '-' || province || ', ' ||
4         city || ', ' || street_address ||
5         ' (call ' || value || ')' AS contact FROM name
6         INNER JOIN person ON person.id = name.person
7         INNER JOIN address ON person.id = address.person
8         INNER JOIN has_id ON person.id = has_id.person
9         WHERE name.is_official = TRUE AND -- use official name only
10              has_id.id_type = 2 -- use ID type for mobile phone
11         ORDER BY person.date_of_birth; -- sort by age, oldest people first
```



```
1  $ psql "postgres://postgres:XXX@localhost/person_database" -v
   ↳ ON_ERROR_STOP=1 -ebf select.sql
   contact
2
3  -----
4  Bebba: US-NY, New York, Times Square (call 444444444444)
5  Bibbo: CN-AH, Hefei, Hefei University (call 111111111111)
6  Bebbbo: DE-SN, Chemnitz, Rathaus (call 222222222222)
7  (3 rows)
8
9  # psql 16.11 succeeded with exit code 0.
```

Daten Auslesen



- Und jetzt lesen wir die Daten wieder aus.
- Wir verwenden dafür das vielleicht größte **INNER JOIN**, das wir bisher verwendet haben.
- Wir wollen die Namen der Personen mit ihren Adressen und Mobiltelefonnummern kombinieren.
- Das geht, in dem wir den Zeichenketten-Verbinden-Operator **||** verwenden.

```
1  /** Combine people, names, addresses, and phone numbers. */
2
3  SELECT full_name || ': ' || country || '-' || province || ', ' ||
4         city || ', ' || street_address ||
5         ' (call ' || value || ')' AS contact FROM name
6
7  INNER JOIN person ON person.id = name.person
8  INNER JOIN address ON person.id = address.person
9  INNER JOIN has_id ON person.id = has_id.person
10 WHERE name.is_official = TRUE AND -- use official name only
11        has_id.id_type = 2         -- use ID type for mobile phone
12
13 ORDER BY person.date_of_birth; -- sort by age, oldest people first
```



```
1  $ psql "postgres://postgres:XXX@localhost/person_database" -v
2  ↪ ON_ERROR_STOP=1 -ebf select.sql
3
4  -----
5  Bebba: US-NY, New York, Times Square (call 444444444444)
6  Bibbo: CN-AH, Hefei, Hefei University (call 111111111111)
7  Bebbbo: DE-SN, Chemnitz, Rathaus (call 222222222222)
8  (3 rows)
9
10 # psql 16.11 succeeded with exit code 0.
```

Daten Auslesen



- Und jetzt lesen wir die Daten wieder aus.
- Wir verwenden dafür das vielleicht größte **INNER JOIN**, das wir bisher verwendet haben.
- Wir wollen die Namen der Personen mit ihren Adressen und Mobiltelefonnummern kombinieren.
- Das geht, in dem wir den Zeichenketten-Verbinden-Operator **||** verwenden.
- Um die benötigten Daten zu bekommen, gehen wir zeilenweise durch die Tabelle **name**.

```
1  /** Combine people, names, addresses, and phone numbers. */
2
3  SELECT full_name || ': ' || country || '-' || province || ', ' ||
4         city || ', ' || street_address ||
5         ' (call ' || value || ')' AS contact FROM name
6
7  INNER JOIN person ON person.id = name.person
8  INNER JOIN address ON person.id = address.person
9  INNER JOIN has_id ON person.id = has_id.person
10 WHERE name.is_official = TRUE AND -- use official name only
11        has_id.id_type = 2         -- use ID type for mobile phone
12
13 ORDER BY person.date_of_birth; -- sort by age, oldest people first
14
15 $ psql "postgres://postgres:XXX@localhost/person_database" -v
16 ↪ ON_ERROR_STOP=1 -ebf select.sql
17
18 -----
19 4  Bebba: US-NY, New York, Times Square (call 444444444444)
20 5  Bibbo: CN-AH, Hefei, Hefei University (call 111111111111)
21 6  Bebbo: DE-SN, Chemnitz, Rathaus (call 222222222222)
22 7  (3 rows)
23
24 # psql 16.11 succeeded with exit code 0.
```

Daten Auslesen



- Wir wollen die Namen der Personen mit ihren Adressen und Mobiltelefonnummern kombinieren.
- Das geht, in dem wir den Zeichenketten-Verbinden-Operator `||` verwenden.
- Um die benötigten Daten zu bekommen, gehen wir zeilenweise durch die Tabelle `name`.
- Später, in der `WHERE`-Klausel, stellen wir sicher, dass wir nur die aktuell gültigen Namen verwenden, in dem wir mit `name.is_official = TRUE` selektieren.

```
1  /** Combine people, names, addresses, and phone numbers. */
2
3  SELECT full_name || ': ' || country || '-' || province || ', ' ||
4         city || ', ' || street_address ||
5         ' (call ' || value || ')' AS contact FROM name
6
7  INNER JOIN person ON person.id = name.person
8  INNER JOIN address ON person.id = address.person
9  INNER JOIN has_id ON person.id = has_id.person
10 WHERE name.is_official = TRUE AND -- use official name only
11        has_id.id_type = 2         -- use ID type for mobile phone
12
13 ORDER BY person.date_of_birth; -- sort by age, oldest people first
```



```
1  $ psql "postgres://postgres:XXX@localhost/person_database" -v
2    ↳ ON_ERROR_STOP=1 -ebf select.sql
3                                     contact
4  -----
5  Bebba: US-NY, New York, Times Square (call 444444444444)
6  Bibbo: CN-AH, Hefei, Hefei University (call 111111111111)
7  Bebbo: DE-SN, Chemnitz, Rathaus (call 222222222222)
8  (3 rows)
9
10 # psql 16.11 succeeded with exit code 0.
```

Daten Auslesen



- Das geht, in dem wir den Zeichenketten-Verbinden-Operator `||` verwenden.
- Um die benötigten Daten zu bekommen, gehen wir zeilenweise durch die Tabelle `name`.
- Später, in der `WHERE`-Klausel, stellen wir sicher, das wir nur die aktuell gültigen Namen verwenden, in dem wir mit `name.is_official = TRUE` selektieren.
- Für jede solche Zeile in Tabelle `name` holen wir uns die passende Zeile in Tabelle `person` über den Fremdschlüssel.

```
1  /** Combine people, names, addresses, and phone numbers. */
2
3  SELECT full_name || ': ' || country || '-' || province || ', ' ||
4         city || ', ' || street_address ||
5         ' (call ' || value || ')' AS contact FROM name
6
7  INNER JOIN person ON person.id = name.person
8  INNER JOIN address ON person.id = address.person
9  INNER JOIN has_id ON person.id = has_id.person
10 WHERE name.is_official = TRUE AND -- use official name only
11        has_id.id_type = 2         -- use ID type for mobile phone
12
13 ORDER BY person.date_of_birth; -- sort by age, oldest people first
14
15
16 $ psql "postgres://postgres:XXX@localhost/person_database" -v
17 ↪ ON_ERROR_STOP=1 -ebf select.sql
18
19 -----
20
21 Bebba: US-NY, New York, Times Square (call 444444444444)
22 Bibbo: CN-AH, Hefei, Hefei University (call 111111111111)
23 Bebbo: DE-SN, Chemnitz, Rathaus (call 222222222222)
24 (3 rows)
25
26 # psql 16.11 succeeded with exit code 0.
```

Daten Auslesen



- Um die benötigten Daten zu bekommen, gehen wir zeilenweise durch die Tabelle `name`.
- Später, in der `WHERE`-Klausel, stellen wir sicher, das wir nur die aktuell gültigen Namen verwenden, in dem wir mit `name.is_official = TRUE` selektieren.
- Für jede solche Zeile in Tabelle `name` holen wir uns die passende Zeile in Tabelle `person` über den Fremdschlüssel.
- Mit dem Primärschlüssel dieses Datensatzes können wir uns dann die passenden Zeilen aus den Tabellen `address` und `has_id` holen.

```
1  /** Combine people, names, addresses, and phone numbers. */
2
3  SELECT full_name || ': ' || country || '-' || province || ', ' ||
4         city || ', ' || street_address ||
5         ' (call ' || value || ')' AS contact FROM name
6         INNER JOIN person ON person.id = name.person
7         INNER JOIN address ON person.id = address.person
8         INNER JOIN has_id ON person.id = has_id.person
9  WHERE name.is_official = TRUE AND -- use official name only
10        has_id.id_type = 2         -- use ID type for mobile phone
11  ORDER BY person.date_of_birth; -- sort by age, oldest people first

1  $ psql "postgres://postgres:XXX@localhost/person_database" -v
   ↳ ON_ERROR_STOP=1 -ebf select.sql
2                                     contact
3  -----
4  Bebba: US-NY, New York, Times Square (call 444444444444)
5  Bibbo: CN-AH, Hefei, Hefei University (call 111111111111)
6  Bebbo: DE-SN, Chemnitz, Rathaus (call 222222222222)
7  (3 rows)
8
9  # psql 16.11 succeeded with exit code 0.
```

Daten Auslesen



- Für jede solche Zeile in Tabelle `name` holen wir uns die passende Zeile in Tabelle `person` über den Fremdschlüssel.
- Mit dem Primärschlüssel dieses Datensatzes können wir uns dann die passenden Zeilen aus den Tabellen `address` und `has_id` holen.
- Wir behalten nur die IDs vom Typ 2 über `has_id.id_type = 2` in der `WHERE`-Klausel, also die Mobiltelefonnummern.

```
1  /** Combine people, names, addresses, and phone numbers. */
2
3  SELECT full_name || ': ' || country || '-' || province || ', ' ||
4         city || ', ' || street_address ||
5         ' (call ' || value || ')' AS contact FROM name
6         INNER JOIN person ON person.id = name.person
7         INNER JOIN address ON person.id = address.person
8         INNER JOIN has_id ON person.id = has_id.person
9         WHERE name.is_official = TRUE AND -- use official name only
10              has_id.id_type = 2          -- use ID type for mobile phone
11         ORDER BY person.date_of_birth; -- sort by age, oldest people first
```



```
1  $ psql "postgres://postgres:XXX@localhost/person_database" -v
2     ↳ ON_ERROR_STOP=1 -ebf select.sql
3                                     contact
4  -----
5  Bebba: US-NY, New York, Times Square (call 444444444444)
6  Bibbo: CN-AH, Hefei, Hefei University (call 111111111111)
7  Bebbbo: DE-SN, Chemnitz, Rathaus (call 222222222222)
8  (3 rows)
9
10 # psql 16.11 succeeded with exit code 0.
```

Daten Auslesen



- Für jede solche Zeile in Tabelle `name` holen wir uns die passende Zeile in Tabelle `person` über den Fremdschlüssel.
- Mit dem Primärschlüssel dieses Datensatzes können wir uns dann die passenden Zeilen aus den Tabellen `address` und `has_id` holen.
- Wir behalten nur die IDs vom Typ 2 über `has_id.id_type = 2` in der `WHERE`-Klausel, also die Mobiltelefonnummern.
- Wenn eine Person keine Mobiltelefonnummer oder Adresse hätte, dann würde sie nicht in der Liste auftauchen.

```
1  /** Combine people, names, addresses, and phone numbers. */
2
3  SELECT full_name || ': ' || country || '-' || province || ', ' ||
4         city || ', ' || street_address ||
5         ' (call ' || value || ')' AS contact FROM name
6         INNER JOIN person ON person.id = name.person
7         INNER JOIN address ON person.id = address.person
8         INNER JOIN has_id ON person.id = has_id.person
9         WHERE name.is_official = TRUE AND -- use official name only
10              has_id.id_type = 2          -- use ID type for mobile phone
11         ORDER BY person.date_of_birth; -- sort by age, oldest people first

1  $ psql "postgres://postgres:XXX@localhost/person_database" -v
2  ↪ ON_ERROR_STOP=1 -ebf select.sql
3         contact
4  -----
5  Bebba: US-NY, New York, Times Square (call 444444444444)
6  Bibbo: CN-AH, Hefei, Hefei University (call 111111111111)
7  Bebbbo: DE-SN, Chemnitz, Rathaus (call 222222222222)
8  (3 rows)
9
10 # psql 16.11 succeeded with exit code 0.
```

Daten Auslesen



- Mit dem Primärschlüssel dieses Datensatzes können wir uns dann die passenden Zeilen aus den Tabellen `address` und `has_id` holen.
- Wir behalten nur die IDs vom Typ 2 über `has_id.id_type = 2` in der `WHERE`-Klausel, also die Mobiltelefonnummern.
- Wenn eine Person keine Mobiltelefonnummer oder Adresse hätte, dann würde sie nicht in der Liste auftauchen.
- Was würde passieren, wenn eine Person mehrere Mobiltelefonnummern und/oder Adressen hat?

```
1  /** Combine people, names, addresses, and phone numbers. */
2
3  SELECT full_name || ': ' || country || '-' || province || ', ' ||
4         city || ', ' || street_address ||
5         ' (call ' || value || ')' AS contact FROM name
6         INNER JOIN person ON person.id = name.person
7         INNER JOIN address ON person.id = address.person
8         INNER JOIN has_id ON person.id = has_id.person
9         WHERE name.is_official = TRUE AND -- use official name only
10              has_id.id_type = 2          -- use ID type for mobile phone
11         ORDER BY person.date_of_birth; -- sort by age, oldest people first
```

```
1  $ psql "postgres://postgres:XXX@localhost/person_database" -v
2     ↳ ON_ERROR_STOP=1 -ebf select.sql
3         contact
4  -----
5  Bebba: US-NY, New York, Times Square (call 444444444444)
6  Bibbo: CN-AH, Hefei, Hefei University (call 111111111111)
7  Bebbo: DE-SN, Chemnitz, Rathaus (call 222222222222)
8  (3 rows)
9
10 # psql 16.11 succeeded with exit code 0.
```

Daten Auslesen



- Mit dem Primärschlüssel dieses Datensatzes können wir uns dann die passenden Zeilen aus den Tabellen `address` und `has_id` holen.
- Wir behalten nur die IDs vom Typ 2 über `has_id.id_type = 2` in der `WHERE`-Klausel, also die Mobiltelefonnummern.
- Wenn eine Person keine Mobiltelefonnummer oder Adresse hätte, dann würde sie nicht in der Liste auftauchen.
- Was würde passieren, wenn eine Person mehrere Mobiltelefonnummern und/oder Adressen hat?
- Probieren Sie es doch selbst aus.

```
1  /** Combine people, names, addresses, and phone numbers. */
2
3  SELECT full_name || ': ' || country || '-' || province || ', ' ||
4         city || ', ' || street_address ||
5         ' (call ' || value || ')' AS contact FROM name
6         INNER JOIN person ON person.id = name.person
7         INNER JOIN address ON person.id = address.person
8         INNER JOIN has_id ON person.id = has_id.person
9  WHERE name.is_official = TRUE AND -- use official name only
10        has_id.id_type = 2         -- use ID type for mobile phone
11  ORDER BY person.date_of_birth; -- sort by age, oldest people first
```



```
1  $ psql "postgres://postgres:XXX@localhost/person_database" -v
2  ↪ ON_ERROR_STOP=1 -ebf select.sql
3                                     contact
4  -----
5  Bebba: US-NY, New York, Times Square (call 444444444444)
6  Bibbo: CN-AH, Hefei, Hefei University (call 111111111111)
7  Bebbo: DE-SN, Chemnitz, Rathaus (call 222222222222)
8  (3 rows)
9
10 # psql 16.11 succeeded with exit code 0.
```

Daten Auslesen



- Wir behalten nur die IDs vom Typ 2 über `has_id.id_type = 2` in der `WHERE`-Klausel, also die Mobiltelefonnummern.
- Wenn eine Person keine Mobiltelefonnummer oder Adresse hätte, dann würde sie nicht in der Liste auftauchen.
- Was würde passieren, wenn eine Person mehrere Mobiltelefonnummern und/oder Adressen hat?
- Probieren Sie es doch selbst aus.
- Egal. Wir sortieren die Zeilen im Ergebnis nach dem Geburtsdatum.

```
1  /** Combine people, names, addresses, and phone numbers. */
2
3  SELECT full_name || ': ' || country || '-' || province || ', ' ||
4         city || ', ' || street_address ||
5         ' (call ' || value || ')' AS contact FROM name
6         INNER JOIN person ON person.id = name.person
7         INNER JOIN address ON person.id = address.person
8         INNER JOIN has_id ON person.id = has_id.person
9         WHERE name.is_official = TRUE AND -- use official name only
10              has_id.id_type = 2 -- use ID type for mobile phone
11         ORDER BY person.date_of_birth; -- sort by age, oldest people first

1  $ psql "postgres://postgres:XXX@localhost/person_database" -v
2  ↪ ON_ERROR_STOP=1 -ebf select.sql
3         contact
4  -----
5  Bebba: US-NY, New York, Times Square (call 444444444444)
6  Bibbo: CN-AH, Hefei, Hefei University (call 111111111111)
7  Bebbbo: DE-SN, Chemnitz, Rathaus (call 222222222222)
8  (3 rows)
9  # psql 16.11 succeeded with exit code 0.
```

Daten Auslesen



- Wenn eine Person keine Mobiltelefonnummer oder Adresse hätte, dann würde sie nicht in der Liste auftauchen.
- Was würde passieren, wenn eine Person mehrere Mobiltelefonnummern und/oder Adressen hat?
- Probieren Sie es doch selbst aus.
- Egal. Wir sortieren die Zeilen im Ergebnis nach dem Geburtsdatum.
- Die älteste Person kommt also zuerst.

```
1  /** Combine people, names, addresses, and phone numbers. */
2
3  SELECT full_name || ': ' || country || '-' || province || ', ' ||
4         city || ', ' || street_address ||
5         ' (call ' || value || ')' AS contact FROM name
6
7  INNER JOIN person ON person.id = name.person
8  INNER JOIN address ON person.id = address.person
9  INNER JOIN has_id ON person.id = has_id.person
10 WHERE name.is_official = TRUE AND -- use official name only
11        has_id.id_type = 2         -- use ID type for mobile phone
12
13 ORDER BY person.date_of_birth; -- sort by age, oldest people first
```



```
1  $ psql "postgres://postgres:XXX@localhost/person_database" -v
2  ↪ ON_ERROR_STOP=1 -ebf select.sql
3
4  -----
5  Bebba: US-NY, New York, Times Square (call 444444444444)
6  Bibbo: CN-AH, Hefei, Hefei University (call 111111111111)
7  Bebbo: DE-SN, Chemnitz, Rathaus (call 222222222222)
8  (3 rows)
9
10 # psql 16.11 succeeded with exit code 0.
```

Daten Auslesen



- Was würde passieren, wenn eine Person mehrere Mobiltelefonnummern und/oder Adressen hat?
- Probieren Sie es doch selbst aus.
- Egal. Wir sortieren die Zeilen im Ergebnis nach dem Geburtsdatum.
- Die älteste Person kommt also zuerst.
- Wir bekommen das erwartete Ergebnis.

```
1  /** Combine people, names, addresses, and phone numbers. */
2
3  SELECT full_name || ': ' || country || '-' || province || ', ' ||
4         city || ', ' || street_address ||
5         ' (call ' || value || ')' AS contact FROM name
6         INNER JOIN person ON person.id = name.person
7         INNER JOIN address ON person.id = address.person
8         INNER JOIN has_id ON person.id = has_id.person
9  WHERE name.is_official = TRUE AND -- use official name only
10        has_id.id_type = 2         -- use ID type for mobile phone
11  ORDER BY person.date_of_birth; -- sort by age, oldest people first

1  $ psql "postgres://postgres:XXX@localhost/person_database" -v
   ↳ ON_ERROR_STOP=1 -ebf select.sql
   contact
2
3  -----
4  Bebba: US-NY, New York, Times Square (call 444444444444)
5  Bibbo: CN-AH, Hefei, Hefei University (call 111111111111)
6  Bebbo: DE-SN, Chemnitz, Rathaus (call 222222222222)
7  (3 rows)
8
9  # psql 16.11 succeeded with exit code 0.
```

Aufräumen



- Räumen wir nun auf.

```
1  /* Cleanup after the example: Delete the person database. */
2
3  DROP DATABASE IF EXISTS person_database;
```

```
1  $ psql "postgres://postgres:XXX@localhost" -v ON_ERROR_STOP=1 -ebf
   ↪ cleanup.sql
2  DROP DATABASE
3  # psql 16.11 succeeded with exit code 0.
```



Zusammenfassung



Zusammenfassung



- Was haben wir also gelernt?

Zusammenfassung



- Was haben wir also gelernt?
- Auf der konzeptuellen Ebene gibt es Beziehungsattribute.

Zusammenfassung



- Was haben wir also gelernt?
- Auf der konzeptuellen Ebene gibt es Beziehungsattribute.
- In logischen Schemas basierend auf dem relationalen Datenmodell gibt es Beziehungen nicht als eigenständige Objekte.

Zusammenfassung



- Was haben wir also gelernt?
- Auf der konzeptuellen Ebene gibt es Beziehungsattribute.
- In logischen Schemas basierend auf dem relationalen Datenmodell gibt es Beziehungen nicht als eigenständige Objekte.
- Deshalb haben sie auch keine Attribute.

Zusammenfassung



- Was haben wir also gelernt?
- Auf der konzeptuellen Ebene gibt es Beziehungsattribute.
- In logischen Schemas basierend auf dem relationalen Datenmodell gibt es Beziehungen nicht als eigenständige Objekte.
- Deshalb haben sie auch keine Attribute.
- Deshalb werden die Beziehungsattribute von der konzeptuellen Ebene zu Spalten in Tabellen im logischen Schema.

Zusammenfassung



- Was haben wir also gelernt?
- Auf der konzeptuellen Ebene gibt es Beziehungsattribute.
- In logischen Schemas basierend auf dem relationalen Datenmodell gibt es Beziehungen nicht als eigenständige Objekte.
- Deshalb haben sie auch keine Attribute.
- Deshalb werden die Beziehungsattribute von der konzeptuellen Ebene zu Spalten in Tabellen im logischen Schema.
- Und diese Spalten tun wir dann in die Tabellen, die die jeweiligen Beziehungen repräsentieren.



谢谢你们！
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. 112).
- [2] 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 also³ (siehe S. 104, 112).
- [3] 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 also² (siehe S. 104, 112).
- [4] 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. 111).
- [5] 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. 112, 113).
- [6] 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. 112).
- [7] 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. 113).
- [8] 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. 111).
- [9] 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. 112).
- [10] Ed Bott. *Windows 11 Inside Out*. Hoboken, NJ, USA: Microsoft Press, Pearson Education, Inc., Feb. 2023. ISBN: 978-0-13-769132-6 (siehe S. 112).
- [11] Ron Brash und Ganesh Naik. *Bash Cookbook*. Birmingham, England, UK: Packt Publishing Ltd, Juli 2018. ISBN: 978-1-78862-936-2 (siehe S. 111).

References II



- [12] 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. 111).
- [13] Jason Cannon. *High Availability for the LAMP Stack*. Shelter Island, NY, USA: Manning Publications, Juni 2022 (siehe S. 111, 113).
- [14] 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. 113).
- [15] 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. 111).
- [16] 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. 105, 111).
- [17] 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. See¹⁶ for a more comprehensive introduction. (Siehe S. 111).
- [18] 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. 113).
- [19] 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. 112).

References III



- [20] *Database Language SQL*. Techn. Ber. ANSI X3.135-1986. Washington, D.C., USA: American National Standards Institute (ANSI), 1986 (siehe S. 113).
- [21] "Date/Time Functions and Operators". In: *PostgreSQL Documentation*. 17.4. The PostgreSQL Global Development Group (PGDG), 20. Feb. 2025. Kap. 9.9. URL: <https://www.postgresql.org/docs/17/functions-datetime.html> (besucht am 2025-05-01) (siehe S. 43–52, 113).
- [22] "Date/Time Types". In: *PostgreSQL Documentation*. 17.4. The PostgreSQL Global Development Group (PGDG), 20. Feb. 2025. Kap. 8.5. URL: <https://www.postgresql.org/docs/17/datatype-datetime.html> (besucht am 2025-02-28) (siehe S. 113).
- [23] 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. 113).
- [24] *Database Language SQL*. International Standard ISO 9075-1987. Geneva, Switzerland: International Organization for Standardization (ISO), 1987 (siehe S. 113).
- [25] Paul Deitel, Harvey Deitel und Abbey Deitel. *Internet & World Wide WebWJ: How to Program*. 5. Aufl. Hoboken, NJ, USA: Pearson Education, Inc., Nov. 2011. ISBN: 978-0-13-299045-5 (siehe S. 113).
- [26] 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. 112).
- [27] 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. 112).
- [28] 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. 112).
- [29] 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. 112).
- [30] Michael Hausenblas. *Learning Modern Linux*. Sebastopol, CA, USA: O'Reilly Media, Inc., Apr. 2022. ISBN: 978-1-0981-0894-6 (siehe S. 112).

References IV



- [31] 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. 111, 113).
- [32] „When should a database table use timestamps?“ In: *Software Engineering*. Hrsg. von GWed. New York, NY, USA: Stack Exchange Inc., 29. Jan. 2014–16. Sep. 2016. URL: <https://softwareengineering.stackexchange.com/questions/225903> (besucht am 2025-02-27) (siehe S. 113).
- [33] 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. 112).
- [34] *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](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. 113).
- [35] 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. 111).
- [36] Jay LaCroix. *Mastering Ubuntu Server*. 4. Aufl. Birmingham, England, UK: Packt Publishing Ltd, Sep. 2022. ISBN: **978-1-80323-424-3** (siehe S. 113).
- [37] 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. 112).
- [38] 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. 111).
- [39] Mark Lutz. *Learning Python*. 6. Aufl. Sebastopol, CA, USA: O'Reilly Media, Inc., März 2025. ISBN: **978-1-0981-7130-8** (siehe S. 112).

References V



- [40] *MariaDB Server Documentation*. Milpitas, CA, USA: MariaDB, 2025. URL: <https://mariadb.com/kb/en/documentation> (besucht am 2025-04-24) (siehe S. 112).
- [41] 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. 113).
- [42] 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. 111).
- [43] 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. 112).
- [44] 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. 111).
- [45] *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).
- [46] *PostgreSQL Essentials: Leveling Up Your Data Work*. Sebastopol, CA, USA: O'Reilly Media, Inc., März 2024 (siehe S. 112).
- [47] 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. 111).
- [48] Federico Razzoli. *Mastering MariaDB*. Birmingham, England, UK: Packt Publishing Ltd, Sep. 2014. ISBN: 978-1-78398-154-0 (siehe S. 112).
- [49] 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. 112).

References VI



- [50] 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. **111**).
- [51] 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. **111**).
- [52] 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. **112**).
- [53] 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](https://doi.org/10.1145/361020.361025) (siehe S. **112**).
- [54] "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. **113**).
- [55] 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. **109**, **113**).
- [56] 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. Burghann, Bayern, Germany: Markt+Technik Verlag GmbH, Feb. 1998. ISBN: **978-3-8272-2020-2**. Translation of ⁵⁵ (siehe S. **113**).
- [57] Allen Taylor. *Introducing SQL and Relational Databases*. New York, NY, USA: Apress Media, LLC, Sep. 2018. ISBN: **978-1-4842-3841-7** (siehe S. **112**, **113**).
- [58] 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. **112**).
- [59] Kristian Torp, Christian S. Jensen und Richard T. Snodgrass. "Effective Timestamping in Databases". 8(3-4):267–288, Feb. 2000. doi:[10.1007/S007780050008](https://doi.org/10.1007/S007780050008). URL: <https://people.cs.aau.dk/~csj/Thesis/pdf/chapter40.pdf> (besucht am 2025-05-01) (siehe S. **113**).

References VII



- [60] 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. 112).
- [61] Sander van Vugt. *Linux Fundamentals*. 2. Aufl. Hoboken, NJ, USA: Pearson IT Certification, Juni 2022. ISBN: 978-0-13-792931-3 (siehe S. 112).
- [62] 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. 111, 112).
- [63] 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. 112).
- [64] 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. 111).
- [65] *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. 112).
- [66] 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. 112).
- [67] 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. 111).
- [68] Giorgio Zarrelli. *Mastering Bash*. Birmingham, England, UK: Packt Publishing Ltd, Juni 2017. ISBN: 978-1-78439-687-9 (siehe S. 111).

Glossary (in English) I



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^{11,42,68}. 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^{8,38,44,47,50}. 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 database (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*⁶².

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 DB⁶⁷.

DOB Date of Birth

ERD Entity relationship diagrams show the relationships between objects, e.g., between the tables in a DB and how they reference each other^{4,12,15–17,35,51,64}

IT information technology

LAMP Stack A system setup for web applications: Linux, Apache (a web server), MySQL, and the server-side scripting language PHP^{13,31}.

Glossary (in English) II



Linux is the leading open source operating system, i.e., a free alternative for Microsoft Windows^{5,30,52,60,61}. 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^{2,3,6,26,40,48}. See <https://mariadb.org> for more information.

Microsoft Windows is a commercial proprietary operating system¹⁰. 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^{9,26,49,58,66}. MySQL is famous for its use in the LAMP Stack. See <https://www.mysql.com> for more information.

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 notation¹. Learn more at <https://pgmodeler.io>.

PostgreSQL An open source object-relational DBMS^{27,43,46,58}. See <https://postgresql.org> for more information.
psql is the client program used to access the PostgreSQL DBMS server.

Python The Python programming language^{33,37,39,63}, i.e., what you will learn about in our book⁶³. Learn more at <https://python.org>.

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^{19,28,29,53,57,62,65}.

Glossary (in English) III



server In a client-server architecture, the server is a process that fulfills the requests of the clients. It usually waits for incoming communication carrying 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 servers¹³ 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“³⁶.

SQL The *Structured Query Language* is basically a programming language for querying and manipulating relational databases^{14,20,23,24,34,41,54–57}. It is understood by many DBMSes. You find the Structured Query Language (SQL) commands supported by PostgreSQL in the reference⁵⁴.

terminal A terminal is a text-based window where you can enter commands and execute them^{5,18}. 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  + , dann Schreiben von `cmd`, dann Druck auf . Under Ubuntu Linux,  +  +  opens a terminal, which then runs a Bash shell inside.

timestamping is a technique used in DBs to keep track of the creation and/or modification time of data^{32,59}. For each table in a relational database to which it is applied, an additional column with datatype `TIMESTAMP`²² is added for the creation timestamp. This column is usually annotated as `NOT NULL DEFAULT CURRENT_TIMESTAMP`²¹, meaning that the DBMS will automatically store the current date and time when a row is created. If data can be changed, then another column for the last update time can be added to the table as well.

Ubuntu is a variant of the open source operating system Linux^{18,31}. 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^{7,25}