



合肥大學  
HEFEI UNIVERSITY



# Datenbanken

## 42. Logisches Schema: 2. Normalform

Thomas Weise (汤卫思)  
[tweise@hfuu.edu.cn](mailto:tweise@hfuu.edu.cn)

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

应用优化研究所  
人工智能与大数据学院  
合肥大学  
中国安徽省合肥市

# 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. Verletzung der 2. Normalform
4. Lösung: Wiederherstellung der 2. Normalform
5. Zusammenfassung





# Einleitung



# Einleitung

- Wir haben schon die 1NF gelernt.



# Einleitung

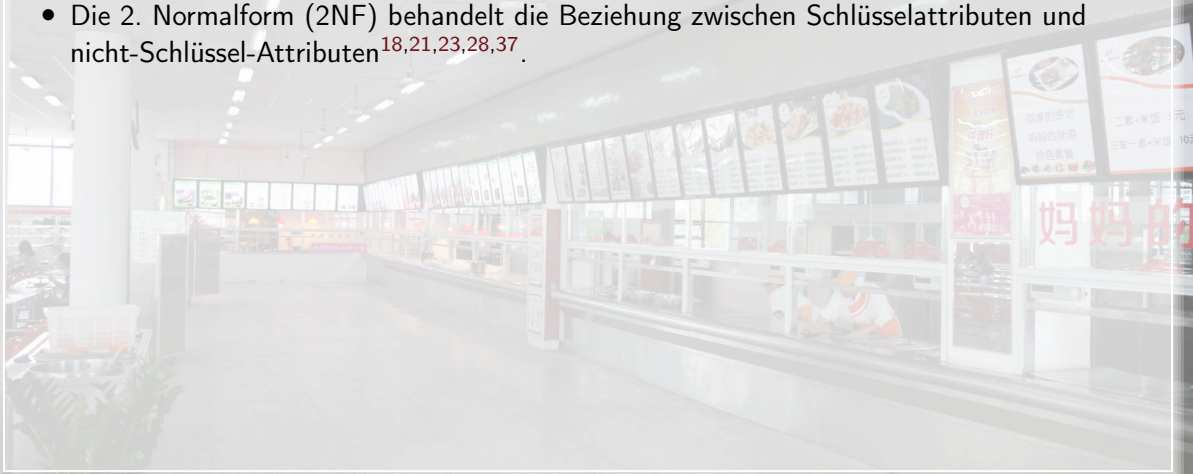
- Wir haben schon die 1NF gelernt.
- Es wird Sie nicht überraschen, dass das nicht die einzige Normalform ist.



# Einleitung



- Wir haben schon die 1NF gelernt.
- Es wird Sie nicht überraschen, dass das nicht die einzige Normalform ist.
- Die 2. Normalform (2NF) behandelt die Beziehung zwischen Schlüsselattributen und nicht-Schlüssel-Attributen<sup>18,21,23,28,37</sup>.



# Einleitung



- Wir haben schon die 1NF gelernt.
- Es wird Sie nicht überraschen, dass das nicht die einzige Normalform ist.
- Die 2. Normalform (2NF) behandelt die Beziehung zwischen Schlüsselattributen und nicht-Schlüssel-Attributen<sup>18,21,23,28,37</sup>.
- Sie gilt nur für zusammengesetzte Schlüssel, also Schlüssel, die aus mehreren Spalten bestehen.

# Einleitung



- Wir haben schon die 1NF gelernt.
- Es wird Sie nicht überraschen, dass das nicht die einzige Normalform ist.
- Die 2. Normalform (2NF) behandelt die Beziehung zwischen Schlüsselattributen und nicht-Schlüssel-Attributen<sup>18,21,23,28,37</sup>.
- Sie gilt nur für zusammengesetzte Schlüssel, also Schlüssel, die aus mehreren Spalten bestehen.
- Wir haben gelernt, dass ein Schlüssel Entitäten eindeutig identifizieren kann.

# Einleitung



- Wir haben schon die 1NF gelernt.
- Es wird Sie nicht überraschen, dass das nicht die einzige Normalform ist.
- Die 2. Normalform (2NF) behandelt die Beziehung zwischen Schlüsselattributen und nicht-Schlüssel-Attributen<sup>18,21,23,28,37</sup>.
- Sie gilt nur für zusammengesetzte Schlüssel, also Schlüssel, die aus mehreren Spalten bestehen.
- Wir haben gelernt, dass ein Schlüssel Entitäten eindeutig identifizieren kann.
- Wenn ein Schlüssel  $X$  eine Entität identifiziert, dann bedeutet das, dass alle anderen Attribute der Entität im Grunde zusätzliche Informationen über den Schlüssel bereitstellen.

# Einleitung



- Wir haben schon die 1NF gelernt.
- Es wird Sie nicht überraschen, dass das nicht die einzige Normalform ist.
- Die 2. Normalform (2NF) behandelt die Beziehung zwischen Schlüsselattributen und nicht-Schlüssel-Attributen<sup>18,21,23,28,37</sup>.
- Sie gilt nur für zusammengesetzte Schlüssel, also Schlüssel, die aus mehreren Spalten bestehen.
- Wir haben gelernt, dass ein Schlüssel Entitäten eindeutig identifizieren kann.
- Wenn ein Schlüssel  $X$  eine Entität identifiziert, dann bedeutet das, dass alle anderen Attribute der Entität im Grunde zusätzliche Informationen über den Schlüssel bereitstellen.
- Im relationalen Modell ist ein Schlüssel der eindeutige Identifikator einer Zeile.

# Einleitung



- Wir haben schon die 1NF gelernt.
- Es wird Sie nicht überraschen, dass das nicht die einzige Normalform ist.
- Die 2. Normalform (2NF) behandelt die Beziehung zwischen Schlüsselattributen und nicht-Schlüssel-Attributen<sup>18,21,23,28,37</sup>.
- Sie gilt nur für zusammengesetzte Schlüssel, also Schlüssel, die aus mehreren Spalten bestehen.
- Wir haben gelernt, dass ein Schlüssel Entitäten eindeutig identifizieren kann.
- Wenn ein Schlüssel  $X$  eine Entität identifiziert, dann bedeutet das, dass alle anderen Attribute der Entität im Grunde zusätzliche Informationen über den Schlüssel bereitstellen.
- Im relationalen Modell ist ein Schlüssel der eindeutige Identifikator einer Zeile.
- Es kann immer höchstens eine Zeile für jeden Wert von  $X$  geben.

# Einleitung



- Wir haben schon die 1NF gelernt.
- Es wird Sie nicht überraschen, dass das nicht die einzige Normalform ist.
- Die 2. Normalform (2NF) behandelt die Beziehung zwischen Schlüsselattributen und nicht-Schlüssel-Attributen<sup>18,21,23,28,37</sup>.
- Sie gilt nur für zusammengesetzte Schlüssel, also Schlüssel, die aus mehreren Spalten bestehen.
- Wir haben gelernt, dass ein Schlüssel Entitäten eindeutig identifizieren kann.
- Wenn ein Schlüssel  $X$  eine Entität identifiziert, dann bedeutet das, dass alle anderen Attribute der Entität im Grunde zusätzliche Informationen über den Schlüssel bereitstellen.
- Im relationalen Modell ist ein Schlüssel der eindeutige Identifikator einer Zeile.
- Es kann immer höchstens eine Zeile für jeden Wert von  $X$  geben.
- Alle anderen Spalten stellen weitere Informationen über das von  $X$  identifizierte Objekt bereit.

## 2NF

- Rufen wir uns nochmal die Definition von Schlüsseln in Erinnerung.



## 2NF

- Rufen wir uns nochmal die Definition von Schlüsseln in Erinnerung.



### Definition: Schlüssel

Eine Menge von Attributen  $K \subseteq \Sigma(R)$  gegeben als  $K = \{k_i : i \in 1..m \wedge k_i \in \Sigma(R)\}$  einer Relation  $R$  ist ein *Schlüssel*, wenn und nur wenn:

1.  $K$  identifizierend (EN: *identifying*) wirkt: Definieren wir  $v_{1,i}$  und  $v_{2,i}$  für alle  $i \in 1..m$  als die Werte der Attribute  $k_i$  für zwei Zeilen  $r_1$  und  $r_2$  in  $R$ . Wenn gilt das  $r_1 \neq r_2$ , dann gibt es mindestens ein  $j \in 1..m$  mit  $v_{1,j} \neq v_{2,j}$  **UND**
2. Es gibt keine Untermenge  $\{k_1, \dots, k_m\}$  mit diese Eigenschaft, also der Schlüssel ist minimal

### Definition: Schlüssel

Eine Menge von Attributen  $K \subseteq \Sigma(R)$  gegeben als  $K = \{k_i : i \in 1..m \wedge k_i \in \Sigma(R)\}$  einer Relation  $R$  ist ein *Schlüssel*, wenn und nur wenn:

1.  $K$  identifizierend (EN: *identifying*) wirkt UND
  2. Es gibt keine Untermenge  $\{k_1, \dots, k_m\}$  mit diese Eigenschaft, also der Schlüssel ist minimal
- Die 2NF ist verletzt, wenn ein nicht-Schlüssel-Attribut Informationen über eine **echte Untermenge** eines beliebigen Schlüssels darstellt<sup>37</sup>.

### Definition: Schlüssel

Eine Menge von Attributen  $K \subseteq \Sigma(R)$  gegeben als  $K = \{k_i : i \in 1..m \wedge k_i \in \Sigma(R)\}$  einer Relation  $R$  ist ein *Schlüssel*, wenn und nur wenn:

1.  $K$  identifizierend (EN: *identifying*) wirkt UND
  2. Es gibt keine Untermenge  $\{k_1, \dots, k_m\}$  mit diese Eigenschaft, also der Schlüssel ist minimal
- Die 2NF ist verletzt, wenn ein nicht-Schlüssel-Attribut Informationen über eine **echte Untermenge** eines beliebigen Schlüssels darstellt<sup>37</sup>.
  - Eine echte Untermenge kann nur existieren, wenn eine Menge aus mehr als einem Element besteht.

### Definition: Schlüssel

Eine Menge von Attributen  $K \subseteq \Sigma(R)$  gegeben als  $K = \{k_i : i \in 1..m \wedge k_i \in \Sigma(R)\}$  einer Relation  $R$  ist ein *Schlüssel*, wenn und nur wenn:

1.  $K$  identifizierend (EN: *identifying*) wirkt UND
  2. Es gibt keine Untermenge  $\{k_1, \dots, k_m\}$  mit diese Eigenschaft, also der Schlüssel ist minimal
- Die 2NF ist verletzt, wenn ein nicht-Schlüssel-Attribut Informationen über eine **echte Untermenge** eines beliebigen Schlüssels darstellt<sup>37</sup>.
  - Eine echte Untermenge kann nur existieren, wenn eine Menge aus mehr als einem Element besteht.
  - Die 2NF ist nur für Tabellen mit zusammengesetzten Schlüsseln relevant, also Schlüsseln, die aus mehr als einer Spalte bestehen.

### Definition: Schlüssel

Eine Menge von Attributen  $K \subseteq \Sigma(R)$  gegeben als  $K = \{k_i : i \in 1..m \wedge k_i \in \Sigma(R)\}$  einer Relation  $R$  ist ein *Schlüssel*, wenn und nur wenn:

1.  $K$  identifizierend (EN: *identifying*) wirkt UND
  2. Es gibt keine Untermenge  $\{k_1, \dots, k_m\}$  mit diese Eigenschaft, also der Schlüssel ist minimal
- Die 2NF ist verletzt, wenn ein nicht-Schlüssel-Attribut Informationen über eine **echte Untermenge** eines beliebigen Schlüssels darstellt<sup>37</sup>.
  - Eine echte Untermenge kann nur existieren, wenn eine Menge aus mehr als einem Element besteht.
  - Die 2NF ist nur für Tabellen mit zusammengesetzten Schlüsseln relevant, also Schlüsseln, die aus mehr als einer Spalte bestehen.
  - Eine Tabelle ist in er 2NF wenn sie in der 1NF ist *und* wenn alle Spalten, die nicht Teil eines Schlüssels sind, nur Informationen über die vollständigen Schlüssel bereitstellen.

# Funktionale Abhängigkeiten



- Eine andere Perspektive auf die 2NF bieten *funktionale Abhängigkeiten* (EN: *functional dependencies*) (FD).

# Funktionale Abhängigkeiten



- Eine andere Perspektive auf die 2NF bieten *funktionale Abhängigkeiten* (EN: *functional dependencies*) (FD).

## Definition: Funktionale Abhängigkeit

Eine funktionale Abhängigkeit (FD) ist eine Beziehung zwischen zwei *Gruppen* von Attributen  $X$  und  $Y$ , wobei gilt, dass für jede Instanz von  $X$  der Wert von  $X$  den Wert von  $Y$  bestimmt<sup>59</sup>. Das schreibt man als  $X \rightarrow Y$ .

# Funktionale Abhängigkeiten



- Eine andere Perspektive auf die 2NF bieten *funktionale Abhängigkeiten* (EN: *functional dependencies*) (FD).

## Definition: Funktionale Abhängigkeit

Eine funktionale Abhängigkeit (FD) ist eine Beziehung zwischen zwei *Gruppen* von Attributen  $X$  und  $Y$ , wobei gilt, dass für jede Instanz von  $X$  der Wert von  $X$  den Wert von  $Y$  bestimmt<sup>59</sup>. Das schreibt man als  $X \rightarrow Y$ .

- Mit anderen Worten, wenn  $X \rightarrow Y$  gilt und  $X$  ein Schlüssel ist, dann gibt es keine zwei Datensätze mit dem selben Wert für  $X$  aber anderen Werten von  $Y$ <sup>37</sup>.

# Funktionale Abhängigkeiten



- Eine andere Perspektive auf die 2NF bieten *funktionale Abhängigkeiten* (EN: *functional dependencies*) (FD).

## Definition: Funktionale Abhängigkeit

Eine funktionale Abhängigkeit (FD) ist eine Beziehung zwischen zwei *Gruppen* von Attributen  $X$  und  $Y$ , wobei gilt, dass für jede Instanz von  $X$  der Wert von  $X$  den Wert von  $Y$  bestimmt<sup>59</sup>. Das schreibt man als  $X \rightarrow Y$ .

- Mit anderen Worten, wenn  $X \rightarrow Y$  gilt und  $X$  ein Schlüssel ist, dann gibt es keine zwei Datensätze mit dem selben Wert für  $X$  aber anderen Werten von  $Y$ <sup>37</sup>.
- Jeder Wert von  $X$  taucht immer mit dem selben dazugehörigem Wert von  $Y$  auf.

# Funktionale Abhängigkeiten



- Eine andere Perspektive auf die 2NF bieten *funktionale Abhängigkeiten* (EN: *functional dependencies*) (FD).

## Definition: Funktionale Abhängigkeit

Eine funktionale Abhängigkeit (FD) ist eine Beziehung zwischen zwei *Gruppen* von Attributen  $X$  und  $Y$ , wobei gilt, dass für jede Instanz von  $X$  der Wert von  $X$  den Wert von  $Y$  bestimmt<sup>59</sup>. Das schreibt man als  $X \rightarrow Y$ .

- Mit anderen Worten, wenn  $X \rightarrow Y$  gilt und  $X$  ein Schlüssel ist, dann gibt es keine zwei Datensätze mit dem selben Wert für  $X$  aber anderen Werten von  $Y$ <sup>37</sup>.
- Jeder Wert von  $X$  taucht immer mit dem selben dazugehörigem Wert von  $Y$  auf.
- Wenn  $X$  ein Schlüssel ist, dann hängen alle anderen Spalten per Definition von  $X$  ab, denn es gibt niemals zwei Zeilen mit dem selben Wert von  $X$ .

## 2NF (unvollständig)



- Das relationale Schema einer Relation  $R$  sei  $\Sigma(R)$  und  $X \subseteq \Sigma(R)$  ist ein Schlüssel.

## 2NF (unvollständig)



- Das relationale Schema einer Relation  $R$  sei  $\Sigma(R)$  und  $X \subseteq \Sigma(R)$  ist ein Schlüssel.
- Natürlich hängen dann alle Attribute  $a$  in  $\Sigma(R)$  von den Schlüsselattributen  $X$  (funktional) ab, also es gilt immer  $X \rightarrow a$ .

## 2NF (unvollständig)



- Das relationale Schema einer Relation  $R$  sei  $\Sigma(R)$  und  $X \subseteq \Sigma(R)$  ist ein Schlüssel.
- Natürlich hängen dann alle Attribute  $a$  in  $\Sigma(R)$  von den Schlüsselattributen  $X$  (funktional) ab, also es gilt immer  $X \rightarrow a$ .
- Betrachten wir die Definition der 2NF mit Hilfe von funktionalen Abhängigkeiten.

## 2NF (unvollständig)



- Das relationale Schema einer Relation  $R$  sei  $\Sigma(R)$  und  $X \subseteq \Sigma(R)$  ist ein Schlüssel.
- Natürlich hängen dann alle Attribute  $a$  in  $\Sigma(R)$  von den Schlüsselattributen  $X$  (funktional) ab, also es gilt immer  $X \rightarrow a$ .
- Betrachten wir die Definition der 2NF mit Hilfe von funktionalen Abhängigkeiten.
- Wenn die 2NF eingehalten wird, dann gilt für alle Attribute  $a \in \Sigma(R)$  die nicht Teil des Schlüssels  $X$  sind (also  $a \notin X$ ), dass es **keine echte Untermenge**  $X' \subset X$  mit  $X' \neq X$  gibt, so dass  $a$  funktional von  $X'$  abhängt (also  $X' \rightarrow a$ ).

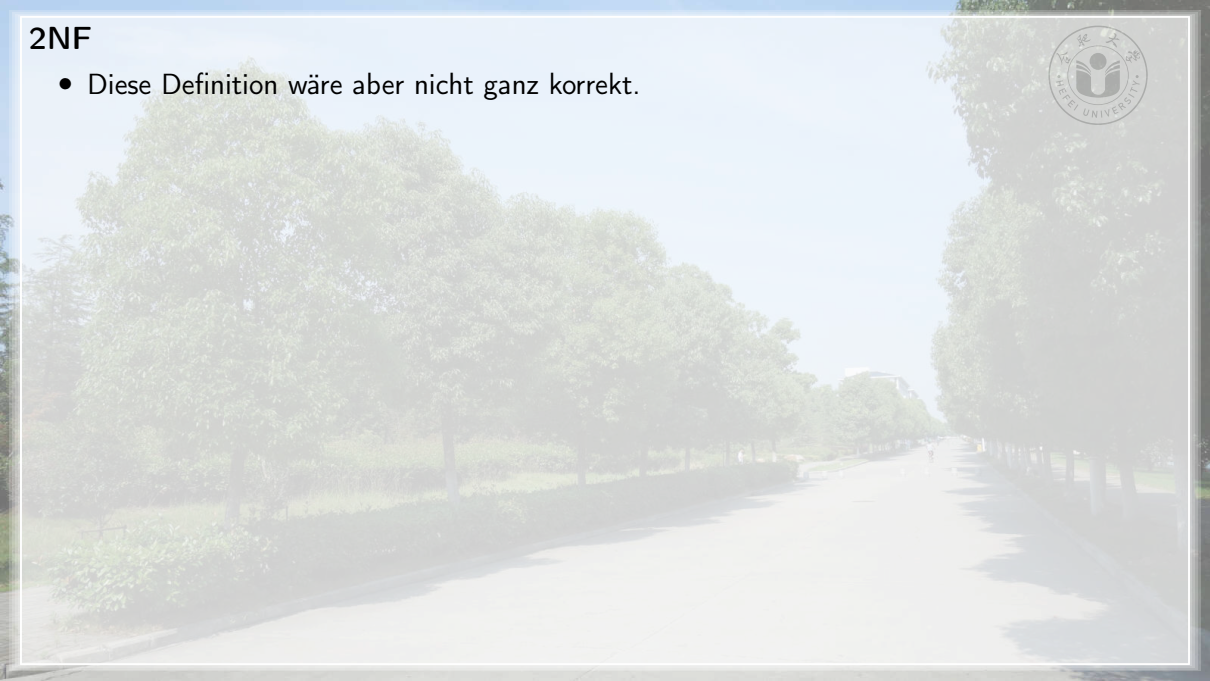
## 2NF (unvollständig)



- Das relationale Schema einer Relation  $R$  sei  $\Sigma(R)$  und  $X \subseteq \Sigma(R)$  ist ein Schlüssel.
- Natürlich hängen dann alle Attribute  $a$  in  $\Sigma(R)$  von den Schlüsselattributen  $X$  (funktional) ab, also es gilt immer  $X \rightarrow a$ .
- Betrachten wir die Definition der 2NF mit Hilfe von funktionalen Abhängigkeiten.
- Wenn die 2NF eingehalten wird, dann gilt für alle Attribute  $a \in \Sigma(R)$  die nicht Teil des Schlüssels  $X$  sind (also  $a \notin X$ ), dass es **keine echte Untermenge**  $X' \subset X$  mit  $X' \neq X$  gibt, so dass  $a$  funktional von  $X'$  abhängt (also  $X' \rightarrow a$ ).
- Unter der 2NF hängen alle Attribute vom vollständigen Schlüssel  $X$  ab.

## 2NF

- Diese Definition wäre aber nicht ganz korrekt.



## 2NF

- Diese Definition wäre aber nicht ganz korrekt.
- Wie wir bereits diskutiert haben, kann es mehrere Schlüssel, also mehrere Mengen von Attributen geben, die die Zeilen in der Relation  $R$  identifizieren.



## 2NF



- Diese Definition wäre aber nicht ganz korrekt.
- Wie wir bereits diskutiert haben, kann es mehrere Schlüssel, also mehrere Mengen von Attributen geben, die die Zeilen in der Relation  $R$  identifizieren.
- Ja, wir wählen eine davon als Primärschlüssel aus ... das ändert aber nichts daran, dass es mehrere Schlüssel geben kann.

## 2NF



- Diese Definition wäre aber nicht ganz korrekt.
- Wie wir bereits diskutiert haben, kann es mehrere Schlüssel, also mehrere Mengen von Attributen geben, die die Zeilen in der Relation  $R$  identifizieren.
- Ja, wir wählen eine davon als Primärschlüssel aus ... das ändert aber nichts daran, dass es mehrere Schlüssel geben kann.
- Das ist besonders dann wahr, wenn wir einen Ersatzschlüssel verwenden, weil die anderen Schlüsselkandidaten zu groß sind.

## 2NF



- Diese Definition wäre aber nicht ganz korrekt.
- Wie wir bereits diskutiert haben, kann es mehrere Schlüssel, also mehrere Mengen von Attributen geben, die die Zeilen in der Relation  $R$  identifizieren.
- Ja, wir wählen eine davon als Primärschlüssel aus ... das ändert aber nichts daran, dass es mehrere Schlüssel geben kann.
- Das ist besonders dann wahr, wenn wir einen Ersatzschlüssel verwenden, weil die anderen Schlüsselkandidaten zu groß sind.
- Alle nicht-Schlüssel-Attribute müssen von allen diesen Schlüsseln vollständig abhängen.

## 2NF



- Diese Definition wäre aber nicht ganz korrekt.
- Wie wir bereits diskutiert haben, kann es mehrere Schlüssel, also mehrere Mengen von Attributen geben, die die Zeilen in der Relation  $R$  identifizieren.
- Ja, wir wählen eine davon als Primärschlüssel aus ... das ändert aber nichts daran, dass es mehrere Schlüssel geben kann.
- Das ist besonders dann wahr, wenn wir einen Ersatzschlüssel verwenden, weil die anderen Schlüsselkandidaten zu groß sind.
- Alle nicht-Schlüssel-Attribute müssen von allen diesen Schlüsseln vollständig abhängen.
- Unter der 2NF ist es nicht zulässig, dass sie von einer Untermenge eines beliebigen Schlüssels abhängen.

## 2NF



- Wie wir bereits diskutiert haben, kann es mehrere Schlüssel, also mehrere Mengen von Attributen geben, die die Zeilen in der Relation  $R$  identifizieren.
- Ja, wir wählen eine davon als Primärschlüssel aus ... das ändert aber nichts daran, dass es mehrere Schlüssel geben kann.
- Das ist besonders dann wahr, wenn wir einen Ersatzschlüssel verwenden, weil die anderen Schlüsselkandidaten zu groß sind.
- Alle nicht-Schlüssel-Attribute müssen von allen diesen Schlüsseln vollständig abhängen.
- Unter der 2NF ist es nicht zulässig, dass sie von einer Untermenge eines beliebigen Schlüssels abhängen.
- Die korrekte Definition der 2NF ist also<sup>55</sup>:

## 2NF



- Ja, wir wählen eine davon als Primärschlüssel aus ... das ändert aber nichts daran, dass es mehrere Schlüssel geben kann.
- Das ist besonders dann wahr, wenn wir einen Ersatzschlüssel verwenden, weil die anderen Schlüsselkandidaten zu groß sind.
- Alle nicht-Schlüssel-Attribute müssen von allen diesen Schlüsseln vollständig abhängen.
- Unter der 2NF ist es nicht zulässig, dass sie von einer Untermenge eines beliebigen Schlüssels abhängen.
- Die korrekte Definition der 2NF ist also<sup>55</sup>:

### Definition: 2. Normalform

Eine Relation  $R$  mit dem Primärschlüssel  $P \subseteq \Sigma(R)$  ist in der 2. Normalform (2NF) wenn und nur wenn sie in der 1NF ist und für alle Mengen von Attributen  $X \subseteq \Sigma(R)$  und alle Attribute  $a \in \Sigma(R)$  mit  $a \notin X$ ,  $a \notin P$  und  $X \rightarrow a$ , gilt, dass  $X$  entweder ein Schlüssel oder Superschlüssel ist, aber keine echte Untermenge eines Schlüssels von  $R$ .



Beispiel



# Raum-Management System



- Wenn eine Tabelle nach der 2NF normalisiert ist, dann hängen alle Attribute vom gesamten Primärschlüssel ab.

# Raum-Management System



- Wenn eine Tabelle nach der 2NF normalisiert ist, dann hängen alle Attribute vom gesamten Primärschlüssel ab.
- Es ist egal, ob der einfach oder zusammengesetzt ist (also aus mehreren Spalten besteht).

# Raum-Management System



- Wenn eine Tabelle nach der 2NF normalisiert ist, dann hängen alle Attribute vom gesamten Primärschlüssel ab.
- Es ist egal, ob der einfach oder zusammengesetzt ist (also aus mehreren Spalten besteht).
- Nehmen wir der Einfachheit halber an, der Primärschlüssel wäre der einzige Schlüssel.

# Raum-Management System



- Wenn eine Tabelle nach der 2NF normalisiert ist, dann hängen alle Attribute vom gesamten Primärschlüssel ab.
- Es ist egal, ob der einfach oder zusammengesetzt ist (also aus mehreren Spalten besteht).
- Nehmen wir der Einfachheit halber an, der Primärschlüssel wäre der einzige Schlüssel.
- Wenn eine Tabelle die 2NF verletzt, dann sind einige der Spalten funktional nur von einem Teil (aber nicht dem ganzen) Primärschlüssel abhängig.

# Raum-Management System

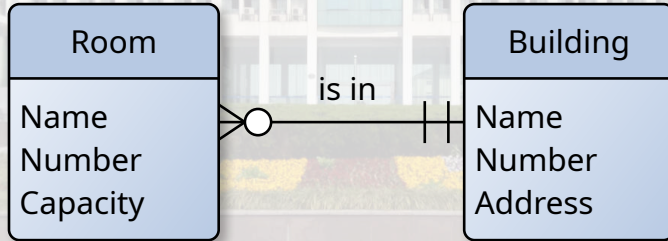


- Wenn eine Tabelle nach der 2NF normalisiert ist, dann hängen alle Attribute vom gesamten Primärschlüssel ab.
- Es ist egal, ob der einfach oder zusammengesetzt ist (also aus mehreren Spalten besteht).
- Nehmen wir der Einfachheit halber an, der Primärschlüssel wäre der einzige Schlüssel.
- Wenn eine Tabelle die 2NF verletzt, dann sind einige der Spalten funktional nur von einem Teil (aber nicht dem ganzen) Primärschlüssel abhängig.
- Schauen wir uns an, warum das schlecht ist.

# Raum-Management System



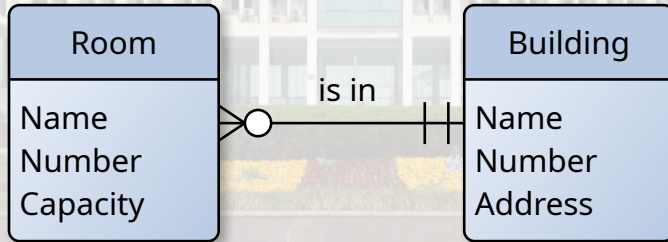
- Schauen wir uns an, warum das schlecht ist.
- Wir hatten vor einer Weile mal ein ERD für ein Raum-Management System als Teil unserer Lehre-Management-Plattform gezeichnet.



# Raum-Management System



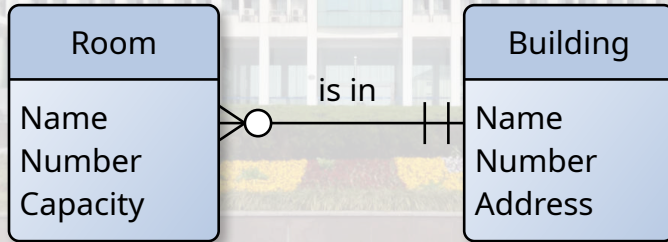
- Schauen wir uns an, warum das schlecht ist.
- Wir hatten vor einer Weile mal ein ERD für ein Raum-Management System als Teil unserer Lehre-Management-Plattform gezeichnet.
- Wir werden das zweimal implementieren.



# Raum-Management System



- Schauen wir uns an, warum das schlecht ist.
- Wir hatten vor einer Weile mal ein ERD für ein Raum-Management System als Teil unserer Lehre-Management-Plattform gezeichnet.
- Wir werden das zweimal implementieren.
- Die erste Implementierung wird die 2NF verletzen, die zweite nicht.





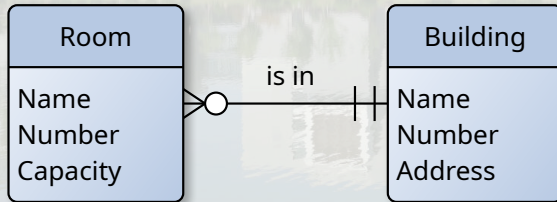
## Verletzung der 2. Normalform



# Konzeptuelles und Logisches Modell



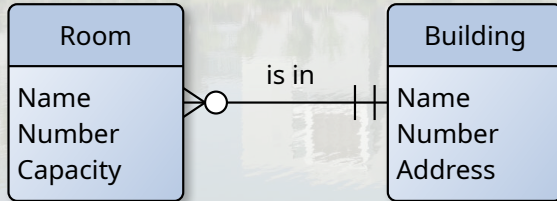
- In dem ERD sehen wir einen Teil des Raum-Management Systems.



# Konzeptuelles und Logisches Modell



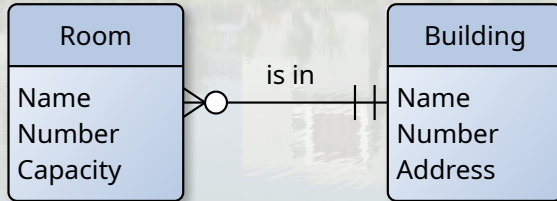
- In dem ERD sehen wir einen Teil des Raum-Management Systems.
- Wir haben nur zwei Entitätstypen dargestellt, *Room* und *Building*.



# Konzeptuelles und Logisches Modell



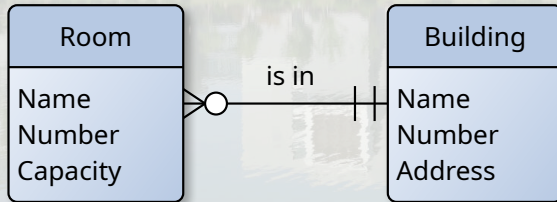
- In dem ERD sehen wir einen Teil des Raum-Management Systems.
- Wir haben nur zwei Entitätstypen dargestellt, *Room* und *Building*.
- Wie Sie sehen können, sind die beiden mit einer Room  $\supset \bigcirc \text{---} \text{||}$  Building-Beziehung verbunden.



# Konzeptuelles und Logisches Modell



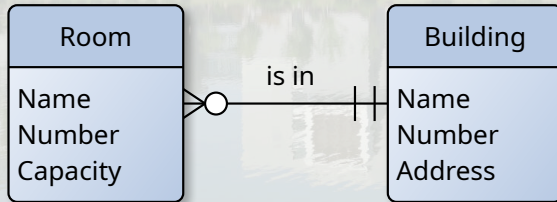
- In dem ERD sehen wir einen Teil des Raum-Management Systems.
- Wir haben nur zwei Entitätstypen dargestellt, *Room* und *Building*.
- Wie Sie sehen können, sind die beiden mit einer Room  $\supset \bigcirc \text{---} \text{||}$  Building-Beziehung verbunden.
- Wir hatten dieses Muster als K  $\supset \bigcirc \text{---} \text{||}$  L diskutiert.



# Konzeptuelles und Logisches Modell



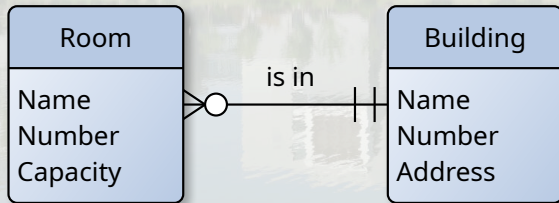
- In dem ERD sehen wir einen Teil des Raum-Management Systems.
- Wir haben nur zwei Entitätstypen dargestellt, *Room* und *Building*.
- Wie Sie sehen können, sind die beiden mit einer Room  $\supset \bigcirc \text{---} \text{||}$  Building-Beziehung verbunden.
- Wir hatten dieses Muster als K  $\supset \bigcirc \text{---} \text{||}$  L diskutiert.
- Nehmen wir an, der Datenbankdesigner hat sich für eine einfachere Implementierung entschieden.



# Konzeptuelles und Logisches Modell



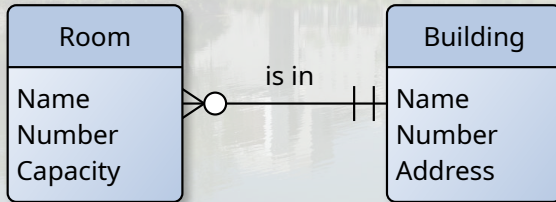
- Wir haben nur zwei Entitätstypen dargestellt, *Room* und *Building*.
- Wie Sie sehen können, sind die beiden mit einer Room  $\supset \circ \text{---} \parallel \parallel$  Building-Beziehung verbunden.
- Wir hatten dieses Muster als K  $\supset \circ \text{---} \parallel \parallel$  L diskutiert.
- Nehmen wir an, der Datenbankdesigner hat sich für eine einfachere Implementierung entschieden.
- Er wird alle Raume- und Gebäudedaten in eine einzelne Tabelle tun.



# Konzeptuelles und Logisches Modell



- Wie Sie sehen können, sind die beiden mit einer Room  $\bowtie$ —|| Building-Beziehung verbunden.
- Wir hatten dieses Muster als K  $\bowtie$ —|| L diskutiert.
- Nehmen wir an, der Datenbankdesigner hat sich für eine einfachere Implementierung entschieden.
- Er wird alle Raume- und Gebäudedaten in eine einzelne Tabelle tun.
- Das Ergebnis sehen wir unten rechts.

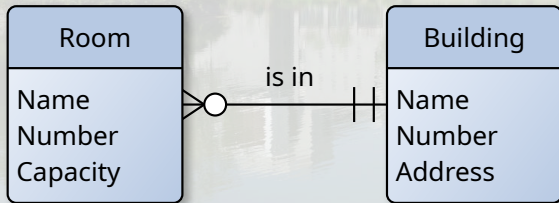


| public.building_room                                                                                       |                     |        |
|------------------------------------------------------------------------------------------------------------|---------------------|--------|
|  <b>building_number</b> | <b>varchar(4)</b>   | « pk » |
|  <b>room_number</b>     | <b>varchar(4)</b>   | « pk » |
|  <b>building_name</b>   | <b>varchar(100)</b> | « nn » |
|  <b>room_name</b>       | <b>varchar(100)</b> | « nn » |
|  <b>capacity</b>        | <b>integer</b>      | « nn » |
|  <b>address</b>         | <b>varchar(100)</b> | « nn » |
|  <b>br_pk</b>           | <b>constraint</b>   | « pk » |

# Konzeptuelles und Logisches Modell



- Wir hatten dieses Muster als  $K \supset \bigcirc \text{---} \parallel L$  diskutiert.
- Nehmen wir an, der Datenbankdesigner hat sich für eine einfachere Implementierung entschieden.
- Er wird alle Raume- und Gebäudedaten in eine einzelne Tabelle tun.
- Das Ergebnis sehen wir unten rechts.
- Die Tabelle `building_room` wurde entworfen.

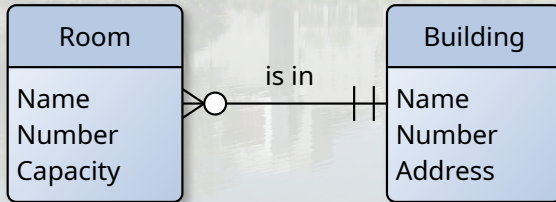


| public.building_room                                                                                       |                     |        |
|------------------------------------------------------------------------------------------------------------|---------------------|--------|
|  <b>building_number</b> | <b>varchar(4)</b>   | « pk » |
|  <b>room_number</b>     | <b>varchar(4)</b>   | « pk » |
|  <b>building_name</b>   | <b>varchar(100)</b> | « nn » |
|  <b>room_name</b>       | <b>varchar(100)</b> | « nn » |
|  <b>capacity</b>        | <b>integer</b>      | « nn » |
|  <b>address</b>         | <b>varchar(100)</b> | « nn » |
|  <b>br_pk</b>           | <b>constraint</b>   | « pk » |

# Konzeptuelles und Logisches Modell



- Nehmen wir an, der Datenbankdesigner hat sich für eine einfachere Implementierung entschieden.
- Er wird alle Raume- und Gebäudedaten in eine einzelne Tabelle tun.
- Das Ergebnis sehen wir unten rechts.
- Die Tabelle `building_room` wurde entworfen.
- Sie hat die Spalten `building_number`, `room_number`, `building_name`, `room_name`, `capacity` und `address`.

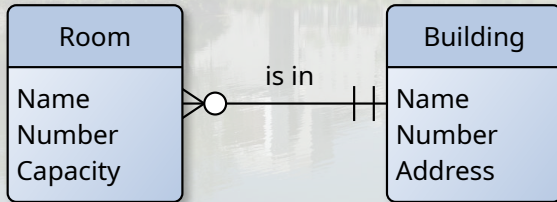


| public.building_room                                                                                       |                     |        |
|------------------------------------------------------------------------------------------------------------|---------------------|--------|
|  <b>building_number</b> | <b>varchar(4)</b>   | « pk » |
|  <b>room_number</b>     | <b>varchar(4)</b>   | « pk » |
|  <b>building_name</b>   | <b>varchar(100)</b> | « nn » |
|  <b>room_name</b>       | <b>varchar(100)</b> | « nn » |
|  <b>capacity</b>        | <b>integer</b>      | « nn » |
|  <b>address</b>         | <b>varchar(100)</b> | « nn » |
|  <b>br_pk</b>           | <b>constraint</b>   | « pk » |

# Konzeptuelles und Logisches Modell



- Er wird alle Raume- und Gebäudedaten in eine einzelne Tabelle tun.
- Das Ergebnis sehen wir unten rechts.
- Die Tabelle `building_room` wurde entworfen.
- Sie hat die Spalten `building_number`, `room_number`, `building_name`, `room_name`, `capacity` und `address`.
- Die Gebäudenummer und die Raumnummer sind beides `VARCHAR` mit Maximallänge 4.

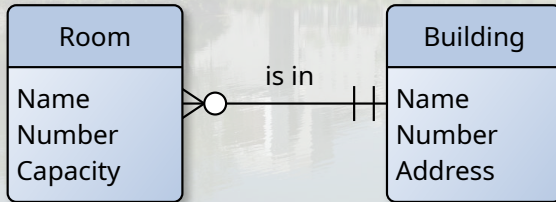


| public.building_room                                                                                       |                     |        |
|------------------------------------------------------------------------------------------------------------|---------------------|--------|
|  <b>building_number</b> | <b>varchar(4)</b>   | « pk » |
|  <b>room_number</b>     | <b>varchar(4)</b>   | « pk » |
|  <b>building_name</b>   | <b>varchar(100)</b> | « nn » |
|  <b>room_name</b>       | <b>varchar(100)</b> | « nn » |
|  <b>capacity</b>        | <b>integer</b>      | « nn » |
|  <b>address</b>         | <b>varchar(100)</b> | « nn » |
|  <b>br_pk</b>           | <b>constraint</b>   | « pk » |

# Konzeptuelles und Logisches Modell



- Das Ergebnis sehen wir unten rechts.
- Die Tabelle `building_room` wurde entworfen.
- Sie hat die Spalten `building_number`, `room_number`, `building_name`, `room_name`, `capacity` und `address`.
- Die Gebäudenummer und die Raumnummer sind beides `VARCHAR` mit Maximallänge 4.
- Sie sind nicht notwendigerweise Nummern, sondern könnten auch so etwas wie `'105b'` sein.

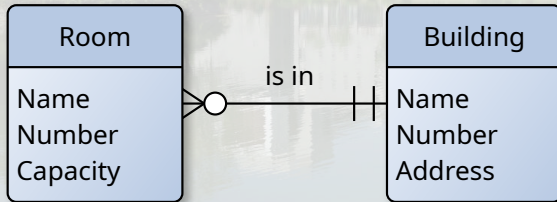


| public.building_room                                                                                       |                     |        |
|------------------------------------------------------------------------------------------------------------|---------------------|--------|
|  <b>building_number</b> | <b>varchar(4)</b>   | « pk » |
|  <b>room_number</b>     | <b>varchar(4)</b>   | « pk » |
|  <b>building_name</b>   | <b>varchar(100)</b> | « nn » |
|  <b>room_name</b>       | <b>varchar(100)</b> | « nn » |
|  <b>capacity</b>        | <b>integer</b>      | « nn » |
|  <b>address</b>         | <b>varchar(100)</b> | « nn » |
|  <b>br_pk</b>           | <b>constraint</b>   | « pk » |

# Konzeptuelles und Logisches Modell



- Sie hat die Spalten `building_number`, `room_number`, `building_name`, `room_name`, `capacity` und `address`.
- Die Gebäudenummer und die Raumnummer sind beides `VARCHAR` mit Maximallänge 4.
- Sie sind nicht notwendigerweise Nummern, sondern könnten auch so etwas wie `'105b'` sein.
- Zusammen formen diese beiden Spalten den Primärschlüssel.

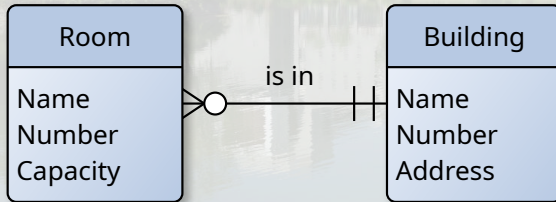


| public.building_room           |                           |        |
|--------------------------------|---------------------------|--------|
| 🔑 <code>building_number</code> | <code>varchar(4)</code>   | « pk » |
| 🔑 <code>room_number</code>     | <code>varchar(4)</code>   | « pk » |
| 🔵 <code>building_name</code>   | <code>varchar(100)</code> | « nn » |
| 🔵 <code>room_name</code>       | <code>varchar(100)</code> | « nn » |
| 🔵 <code>capacity</code>        | <code>integer</code>      | « nn » |
| 🔵 <code>address</code>         | <code>varchar(100)</code> | « nn » |
| 🔷 <code>br_pk</code>           | <code>constraint</code>   | « pk » |

# Konzeptuelles und Logisches Modell



- Die Gebäudenummer und die Raumnummer sind beides `VARCHAR` mit Maximallänge 4.
- Sie sind nicht notwendigerweise Nummern, sondern könnten auch so etwas wie `'105b'` sein.
- Zusammen formen diese beiden Spalten den Primärschlüssel.
- Jeder Raum in unserer Uni wird eindeutig durch die Gebäude- und Raumnummer identifiziert.

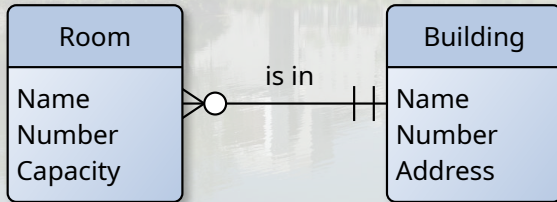


| public.building_room                                                                                       |                     |        |
|------------------------------------------------------------------------------------------------------------|---------------------|--------|
|  <b>building_number</b> | <b>varchar(4)</b>   | « pk » |
|  <b>room_number</b>     | <b>varchar(4)</b>   | « pk » |
|  <b>building_name</b>   | <b>varchar(100)</b> | « nn » |
|  <b>room_name</b>       | <b>varchar(100)</b> | « nn » |
|  <b>capacity</b>        | <b>integer</b>      | « nn » |
|  <b>address</b>         | <b>varchar(100)</b> | « nn » |
|  <b>br_pk</b>           | <b>constraint</b>   | « pk » |

# Konzeptuelles und Logisches Modell



- Sie sind nicht notwendigerweise Nummern, sondern könnten auch so etwas wie '105b' sein.
- Zusammen formen diese beiden Spalten den Primärschlüssel.
- Jeder Raum in unserer Uni wird eindeutig durch die Gebäude- und Raumnummer identifiziert.
- Es kann nie zwei verschiedene Räume mit der selben Gebäude- und Raumnummer geben.

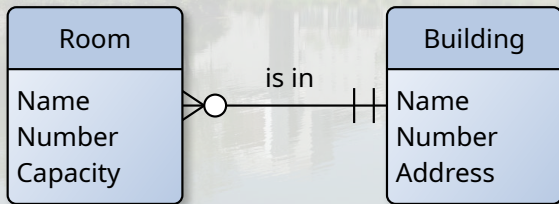


| public.building_room                                                                                       |                     |        |
|------------------------------------------------------------------------------------------------------------|---------------------|--------|
|  <b>building_number</b> | <b>varchar(4)</b>   | « pk » |
|  <b>room_number</b>     | <b>varchar(4)</b>   | « pk » |
|  building_name          | <b>varchar(100)</b> | « nn » |
|  room_name              | <b>varchar(100)</b> | « nn » |
|  capacity               | <b>integer</b>      | « nn » |
|  address                | <b>varchar(100)</b> | « nn » |
|  br_pk                  | <b>constraint</b>   | « pk » |

# Konzeptuelles und Logisches Modell



- Zusammen formen diese beiden Spalten den Primärschlüssel.
- Jeder Raum in unserer Uni wird eindeutig durch die Gebäude- und Raumnummer identifiziert.
- Es kann nie zwei verschiedene Räume mit der selben Gebäude- und Raumnummer geben.
- Dieses Design verletzt die 2NF, denn es gibt einige Spalten, die nur von einem Teil des Primärschlüssels abhängen.

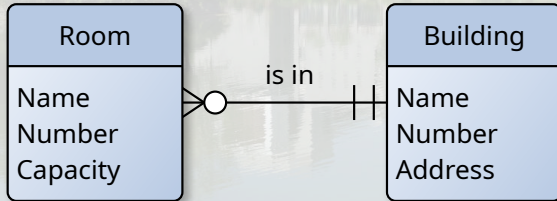


| public.building_room                                                                                       |                     |        |
|------------------------------------------------------------------------------------------------------------|---------------------|--------|
|  <b>building_number</b> | <b>varchar(4)</b>   | « pk » |
|  <b>room_number</b>     | <b>varchar(4)</b>   | « pk » |
|  <b>building_name</b>   | <b>varchar(100)</b> | « nn » |
|  <b>room_name</b>       | <b>varchar(100)</b> | « nn » |
|  <b>capacity</b>        | <b>integer</b>      | « nn » |
|  <b>address</b>         | <b>varchar(100)</b> | « nn » |
|  <b>br_pk</b>           | <b>constraint</b>   | « pk » |

# Konzeptuelles und Logisches Modell



- Jeder Raum in unserer Uni wird eindeutig durch die Gebäude- und Raumnummer identifiziert.
- Es kann nie zwei verschiedene Räume mit der selben Gebäude- und Raumnummer geben.
- Dieses Design verletzt die 2NF, denn es gibt einige Spalten, die nur von einem Teil des Primärschlüssels abhängen.
- Die Spalten `building_name` und `address` repräsentieren Fakten *nur* über `building_number`.

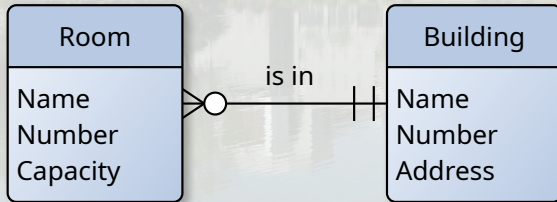


| public.building_room                                                                                             |                           |        |
|------------------------------------------------------------------------------------------------------------------|---------------------------|--------|
|  <code>building_number</code> | <code>varchar(4)</code>   | « pk » |
|  <code>room_number</code>     | <code>varchar(4)</code>   | « pk » |
|  <code>building_name</code>   | <code>varchar(100)</code> | « nn » |
|  <code>room_name</code>       | <code>varchar(100)</code> | « nn » |
|  <code>capacity</code>        | <code>integer</code>      | « nn » |
|  <code>address</code>         | <code>varchar(100)</code> | « nn » |
|  <code>br_pk</code>           | <code>constraint</code>   | « pk » |

# Konzeptuelles und Logisches Modell



- Es kann nie zwei verschiedene Räume mit der selben Gebäude- und Raumnummer geben.
- Dieses Design verletzt die 2NF, denn es gibt einige Spalten, die nur von einem Teil des Primärschlüssels abhängen.
- Die Spalten `building_name` und `address` repräsentieren Fakten *nur* über `building_number`.
- Sie sind nicht funktional von `room_number` abhängig.



| public.building_room         |                           |        |
|------------------------------|---------------------------|--------|
| <code>building_number</code> | <code>varchar(4)</code>   | « pk » |
| <code>room_number</code>     | <code>varchar(4)</code>   | « pk » |
| <code>building_name</code>   | <code>varchar(100)</code> | « nn » |
| <code>room_name</code>       | <code>varchar(100)</code> | « nn » |
| <code>capacity</code>        | <code>integer</code>      | « nn » |
| <code>address</code>         | <code>varchar(100)</code> | « nn » |
| <code>br_pk</code>           | <code>constraint</code>   | « pk » |

# Tabelle building\_room



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

```
1  -- object: public.building_room | type: TABLE --
2  -- DROP TABLE IF EXISTS public.building_room CASCADE;
3  CREATE TABLE public.building_room (
4      building_number varchar(4) NOT NULL,
5      room_number varchar(4) NOT NULL,
6      building_name varchar(100) NOT NULL,
7      room_name varchar(100) NOT NULL,
8      capacity integer NOT NULL,
9      address varchar(100) NOT NULL,
10     CONSTRAINT br_pk PRIMARY KEY (building_number,room_number)
11 );
12 -- ddl-end --
13 ALTER TABLE public.building_room OWNER TO postgres;
14 -- ddl-end --
```

```
1  $ psql "postgres://postgres:XXX@localhost/violation" -v ON_ERROR_STOP=1
2  ↪ -ebf 03_public_building_room_table_5071.sql
3  CREATE TABLE
4  ALTER TABLE
5  # psql 16.11 succeeded with exit code 0.
```

# Tabelle building\_room



- Hier ist das auto-generierte Skript zum Erstellen der Tabelle `building_room`.
- `building_number` und `room_number` bilden zusammen den Primärschlüssel.

```
1  -- object: public.building_room | type: TABLE --
2  -- DROP TABLE IF EXISTS public.building_room CASCADE;
3  CREATE TABLE public.building_room (
4      building_number varchar(4) NOT NULL,
5      room_number varchar(4) NOT NULL,
6      building_name varchar(100) NOT NULL,
7      room_name varchar(100) NOT NULL,
8      capacity integer NOT NULL,
9      address varchar(100) NOT NULL,
10     CONSTRAINT br_pk PRIMARY KEY (building_number,room_number)
11 );
12 -- ddl-end --
13 ALTER TABLE public.building_room OWNER TO postgres;
14 -- ddl-end --
```

```
1  $ psql "postgres://postgres:XXX@localhost/violation" -v ON_ERROR_STOP=1
2  ↪ -ebf 03_public_building_room_table_5071.sql
3  CREATE TABLE
4  ALTER TABLE
5  # psql 16.11 succeeded with exit code 0.
```

# Tabelle building\_room



- Hier ist das auto-generierte Skript zum Erstellen der Tabelle `building_room`.
- `building_number` und `room_number` bilden zusammen den Primärschlüssel.
- Zusätzlich gibt es Spalten für die Raum- und Gebäudenamen sowie die Adresse als `VARCHAR`s mit der Maximallänge 100.

```
1  -- object: public.building_room | type: TABLE --
2  -- DROP TABLE IF EXISTS public.building_room CASCADE;
3  CREATE TABLE public.building_room (
4      building_number varchar(4) NOT NULL,
5      room_number varchar(4) NOT NULL,
6      building_name varchar(100) NOT NULL,
7      room_name varchar(100) NOT NULL,
8      capacity integer NOT NULL,
9      address varchar(100) NOT NULL,
10     CONSTRAINT br_pk PRIMARY KEY (building_number,room_number)
11 );
12 -- ddl-end --
13 ALTER TABLE public.building_room OWNER TO postgres;
14 -- ddl-end --
```

```
1  $ psql "postgres://postgres:XXX@localhost/violation" -v ON_ERROR_STOP=1
2  ↪ -ebf 03_public_building_room_table_5071.sql
3  CREATE TABLE
4  ALTER TABLE
5  # psql 16.11 succeeded with exit code 0.
```

# Tabelle building\_room



- Hier ist das auto-generierte Skript zum Erstellen der Tabelle `building_room`.
- `building_number` und `room_number` bilden zusammen den Primärschlüssel.
- Zusätzlich gibt es Spalten für die Raum- und Gebäudenamen sowie die Adresse als `VARCHAR`s mit der Maximallänge 100.
- Die Kapazität für Studenten eines Raumes ist eine `INTEGER`-Zahl.

```
1  -- object: public.building_room | type: TABLE --
2  -- DROP TABLE IF EXISTS public.building_room CASCADE;
3  CREATE TABLE public.building_room (
4      building_number varchar(4) NOT NULL,
5      room_number varchar(4) NOT NULL,
6      building_name varchar(100) NOT NULL,
7      room_name varchar(100) NOT NULL,
8      capacity integer NOT NULL,
9      address varchar(100) NOT NULL,
10     CONSTRAINT br_pk PRIMARY KEY (building_number,room_number)
11 );
12 -- ddl-end --
13 ALTER TABLE public.building_room OWNER TO postgres;
14 -- ddl-end --
```

```
1  $ psql "postgres://postgres:XXX@localhost/violation" -v ON_ERROR_STOP=1
2  ↪ -ebf 03_public_building_room_table_5071.sql
3  CREATE TABLE
4  ALTER TABLE
5  # psql 16.11 succeeded with exit code 0.
```

# Daten Einfügen

- Befüllen wir die Tabelle mit Daten.



```
1  /** Insert data into the database. */
2
3  -- Insert several room records.
4  INSERT INTO building_room (building_number, room_number, building_name,
5                             room_name, capacity, address) VALUES
6      ('36', '305', 'CS Teaching Building', 'Meeting Room', 40,
7       'South Campus II'),
8      ('36', '105', 'CS Teaching Building', 'Lecture Room 1', 80,
9       'South Campus II'),
10     ('10', '100', 'Language Teaching Building', 'Teaching Room A', 30,
11      'South Campus 1'),
12     ('10', '102', 'Language Teaching Building', 'Teaching Room B', 30,
13      'South Campus 1'),
14     ('53', '904a', 'Comprehensive Experimental Building',
15      'Office 1', 10, 'South Campus 2'),
16     ('53', '904b', 'Comprehensive Experimental Building',
17      'Cluster Room', 3, 'South Campus 2'),
18     ('7', '200', 'Main Teaching Building', 'Auditorium', 120,
19      'South Campus 1'),
20     ('36', '106', 'CS Teaching Building', 'Lecture Room 2', 80,
21      'South Campus 2');
```

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

# Daten Einfügen



- Befüllen wir die Tabelle mit Daten.
- Z. B. speichern wir den Meeting-Room (Room 305) des Building 36, also des *Computer Science Teaching Building*, in South Campus 2.

```
1  /** Insert data into the database. */
2
3  -- Insert several room records.
4  INSERT INTO building_room (building_number, room_number, building_name,
5                             room_name, capacity, address) VALUES
6      ('36', '305', 'CS Teaching Building', 'Meeting Room', 40,
7       'South Campus II'),
8      ('36', '105', 'CS Teaching Building', 'Lecture Room 1', 80,
9       'South Campus II'),
10     ('10', '100', 'Language Teaching Building', 'Teaching Room A', 30,
11      'South Campus 1'),
12     ('10', '102', 'Language Teaching Building', 'Teaching Room B', 30,
13      'South Campus 1'),
14     ('53', '904a', 'Comprehensive Experimental Building',
15      'Office 1', 10, 'South Campus 2'),
16     ('53', '904b', 'Comprehensive Experimental Building',
17      'Cluster Room', 3, 'South Campus 2'),
18     ('7', '200', 'Main Teaching Building', 'Auditorium', 120,
19      'South Campus 1'),
20     ('36', '106', 'CS Teaching Building', 'Lecture Room 2', 80,
21      'South Campus 2');
```

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

# Daten Einfügen



- Befüllen wir die Tabelle mit Daten.
- Z. B. speichern wir den Meeting-Room (Room 305) des Building 36, also des *Computer Science Teaching Building*, in South Campus 2.
- Wir speichern auch einen Vorlesungsraum im selben Gebäude.

```
1  /** Insert data into the database. */
2
3  -- Insert several room records.
4  INSERT INTO building_room (building_number, room_number, building_name,
5                             room_name, capacity, address) VALUES
6      ('36', '305', 'CS Teaching Building', 'Meeting Room', 40,
7       'South Campus II'),
8      ('36', '105', 'CS Teaching Building', 'Lecture Room 1', 80,
9       'South Campus II'),
10     ('10', '100', 'Language Teaching Building', 'Teaching Room A', 30,
11      'South Campus 1'),
12     ('10', '102', 'Language Teaching Building', 'Teaching Room B', 30,
13      'South Campus 1'),
14     ('53', '904a', 'Comprehensive Experimental Building',
15      'Office 1', 10, 'South Campus 2'),
16     ('53', '904b', 'Comprehensive Experimental Building',
17      'Cluster Room', 3, 'South Campus 2'),
18     ('7', '200', 'Main Teaching Building', 'Auditorium', 120,
19      'South Campus 1'),
20     ('36', '106', 'CS Teaching Building', 'Lecture Room 2', 80,
21      'South Campus 2');
```

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

# Daten Einfügen



- Befüllen wir die Tabelle mit Daten.
- Z. B. speichern wir den Meeting-Room (Room 305) des Building 36, also des *Computer Science Teaching Building*, in South Campus 2.
- Wir speichern auch einen Vorlesungsraum im selben Gebäude.
- Wir speichern Informationen über zwei Vorlesungsräume des *Language Teaching Building* in South Campus 1, über ein paar Büros in Gebäude 53 vom South Campus 2 und ein paar andere Räume.

```
1  /** Insert data into the database. */
2
3  -- Insert several room records.
4  INSERT INTO building_room (building_number, room_number, building_name,
5                             room_name, capacity, address) VALUES
6      ('36', '305', 'CS Teaching Building', 'Meeting Room', 40,
7       'South Campus II'),
8      ('36', '105', 'CS Teaching Building', 'Lecture Room 1', 80,
9       'South Campus II'),
10     ('10', '100', 'Language Teaching Building', 'Teaching Room A', 30,
11      'South Campus 1'),
12     ('10', '102', 'Language Teaching Building', 'Teaching Room B', 30,
13      'South Campus 1'),
14     ('53', '904a', 'Comprehensive Experimental Building',
15      'Office 1', 10, 'South Campus 2'),
16     ('53', '904b', 'Comprehensive Experimental Building',
17      'Cluster Room', 3, 'South Campus 2'),
18     ('7', '200', 'Main Teaching Building', 'Auditorium', 120,
19      'South Campus 1'),
20     ('36', '106', 'CS Teaching Building', 'Lecture Room 2', 80,
21      'South Campus 2');
```

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

# Daten Einfügen



- Z. B. speichern wir den Meeting-Room (Room 305) des Building 36, also des *Computer Science Teaching Building*, in South Campus 2.
- Wir speichern auch einen Vorlesungsraum im selben Gebäude.
- Wir speichern Informationen über zwei Vorlesungsräume des *Language Teaching Building* in South Campus 1, über ein paar Büros in Gebäude 53 vom South Campus 2 und ein paar andere Räume.
- Wenn wir uns diese **INSERT**-Anweisungen angucken wir eins klar: Diese Struktur erzeugt eine Menge Redundanz.

```
1  /** Insert data into the database. */
2
3  -- Insert several room records.
4  INSERT INTO building_room (building_number, room_number, building_name,
5                             room_name, capacity, address) VALUES
6      ('36', '305', 'CS Teaching Building', 'Meeting Room', 40,
7       'South Campus II'),
8      ('36', '105', 'CS Teaching Building', 'Lecture Room 1', 80,
9       'South Campus II'),
10     ('10', '100', 'Language Teaching Building', 'Teaching Room A', 30,
11      'South Campus 1'),
12     ('10', '102', 'Language Teaching Building', 'Teaching Room B', 30,
13      'South Campus 1'),
14     ('53', '904a', 'Comprehensive Experimental Building',
15      'Office 1', 10, 'South Campus 2'),
16     ('53', '904b', 'Comprehensive Experimental Building',
17      'Cluster Room', 3, 'South Campus 2'),
18     ('7', '200', 'Main Teaching Building', 'Auditorium', 120,
19      'South Campus 1'),
20     ('36', '106', 'CS Teaching Building', 'Lecture Room 2', 80,
21      'South Campus 2');
```

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

# Daten Einfügen



- Wir speichern auch einen Vorlesungsraum im selben Gebäude.
- Wir speichern Informationen über zwei Vorlesungsräume des *Language Teaching Building* in South Campus 1, über ein paar Büros in Gebäude 53 vom South Campus 2 und ein paar andere Räume.
- Wenn wir uns diese INSERT-Anweisungen angucken wir eins klar: Diese Struktur erzeugt eine Menge Redundanz.
- Die Adresse von Gebäude 36 wird dreimal gespeichert.

```
1  /** Insert data into the database. */
2
3  -- Insert several room records.
4  INSERT INTO building_room (building_number, room_number, building_name,
5                             room_name, capacity, address) VALUES
6      ('36', '305', 'CS Teaching Building', 'Meeting Room', 40,
7       'South Campus II'),
8      ('36', '105', 'CS Teaching Building', 'Lecture Room 1', 80,
9       'South Campus II'),
10     ('10', '100', 'Language Teaching Building', 'Teaching Room A', 30,
11      'South Campus 1'),
12     ('10', '102', 'Language Teaching Building', 'Teaching Room B', 30,
13      'South Campus 1'),
14     ('53', '904a', 'Comprehensive Experimental Building',
15      'Office 1', 10, 'South Campus 2'),
16     ('53', '904b', 'Comprehensive Experimental Building',
17      'Cluster Room', 3, 'South Campus 2'),
18     ('7', '200', 'Main Teaching Building', 'Auditorium', 120,
19      'South Campus 1'),
20     ('36', '106', 'CS Teaching Building', 'Lecture Room 2', 80,
21      'South Campus 2');
```

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

# Daten Einfügen



- Wir speichern Informationen über zwei Vorlesungsräume des *Language Teaching Building* in South Campus 1, über ein paar Büros in Gebäude 53 vom South Campus 2 und ein paar andere Räume.
- Wenn wir uns diese `INSERT`-Anweisungen angucken wir eins klar: Diese Struktur erzeugt eine Menge Redundanz.
- Die Adresse von Gebäude 36 wird dreimal gespeichert.
- Genaugenommen wird auch der Name von Gebäude 36 dreimal gespeichert.

```
1  /** Insert data into the database. */
2
3  -- Insert several room records.
4  INSERT INTO building_room (building_number, room_number, building_name,
5                             room_name, capacity, address) VALUES
6      ('36', '305', 'CS Teaching Building', 'Meeting Room', 40,
7       'South Campus II'),
8      ('36', '105', 'CS Teaching Building', 'Lecture Room 1', 80,
9       'South Campus II'),
10     ('10', '100', 'Language Teaching Building', 'Teaching Room A', 30,
11      'South Campus 1'),
12     ('10', '102', 'Language Teaching Building', 'Teaching Room B', 30,
13      'South Campus 1'),
14     ('53', '904a', 'Comprehensive Experimental Building',
15      'Office 1', 10, 'South Campus 2'),
16     ('53', '904b', 'Comprehensive Experimental Building',
17      'Cluster Room', 3, 'South Campus 2'),
18     ('7', '200', 'Main Teaching Building', 'Auditorium', 120,
19      'South Campus 1'),
20     ('36', '106', 'CS Teaching Building', 'Lecture Room 2', 80,
21      'South Campus 2');
```

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

# Daten Einfügen



- Wenn wir uns diese **INSERT**-Anweisungen angucken wir eins klar: Diese Struktur erzeugt eine Menge Redundanz.
- Die Adresse von Gebäude 36 wird dreimal gespeichert.
- Genaugenommen wird auch der Name von Gebäude 36 dreimal gespeichert.
- Das fühlt sich falsch an.

```
1  /** Insert data into the database. */
2
3  -- Insert several room records.
4  INSERT INTO building_room (building_number, room_number, building_name,
5                             room_name, capacity, address) VALUES
6      ('36', '305', 'CS Teaching Building', 'Meeting Room', 40,
7       'South Campus II'),
8      ('36', '105', 'CS Teaching Building', 'Lecture Room 1', 80,
9       'South Campus II'),
10     ('10', '100', 'Language Teaching Building', 'Teaching Room A', 30,
11      'South Campus 1'),
12     ('10', '102', 'Language Teaching Building', 'Teaching Room B', 30,
13      'South Campus 1'),
14     ('53', '904a', 'Comprehensive Experimental Building',
15      'Office 1', 10, 'South Campus 2'),
16     ('53', '904b', 'Comprehensive Experimental Building',
17      'Cluster Room', 3, 'South Campus 2'),
18     ('7', '200', 'Main Teaching Building', 'Auditorium', 120,
19      'South Campus 1'),
20     ('36', '106', 'CS Teaching Building', 'Lecture Room 2', 80,
21      'South Campus 2');
```

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

# Daten Löschen

- Neben der Redundanz hat diese Struktur aber noch weitere Probleme.

```
1  /** Delete room 904a and 904b, which doesn't delete the building 53. */
2
3  -- Get the list of all buildings.
4  SELECT DISTINCT building_number, building_name FROM building_room;
5
6  -- Delete rooms 904a and 904b in building 53.
7  DELETE FROM building_room WHERE building_number = '53'
8      AND room_number LIKE '904%'
9      RETURNING building_number, room_number;
10
11 -- Get the list of all buildings again: Building 53 is still there.
12 SELECT DISTINCT building_number, building_name FROM building_room;
```

```
1  $ psql "postgres://postgres:XXX@localhost/violation" -v ON_ERROR_STOP=1
   ↪ -ebf delete.sql
2  building_number |          building_name
3  -----+-----
4  36              | CS Teaching Building
5  53              | Comprehensive Experimental Building
6  10              | Language Teaching Building
7  7              | Main Teaching Building
8  (4 rows)
9
10 building_number | room_number
11 -----+-----
12 53              | 904a
13 53              | 904b
14 (2 rows)
15
16 DELETE 2
17 building_number |          building_name
18 -----+-----
19 36              | CS Teaching Building
20 10              | Language Teaching Building
21 7              | Main Teaching Building
22 (3 rows)
23
24 # psql 16.11 succeeded with exit code 0.
```

# Daten Löschen

- Neben der Redundanz hat diese Struktur aber noch weitere Probleme.
- Diese werden sichtbar, wenn wir die Daten ändern.

```
1  /** Delete room 904a and 904b, which doesn't delete the building 53. */
2
3  -- Get the list of all buildings.
4  SELECT DISTINCT building_number, building_name FROM building_room;
5
6  -- Delete rooms 904a and 904b in building 53.
7  DELETE FROM building_room WHERE building_number = '53'
8      AND room_number LIKE '904%'
9      RETURNING building_number, room_number;
10
11 -- Get the list of all buildings again: Building 53 is still there.
12 SELECT DISTINCT building_number, building_name FROM building_room;
```

```
1  $ psql "postgres://postgres:XXX@localhost/violation" -v ON_ERROR_STOP=1
   ↪ -ebf delete.sql
2  building_number |          building_name
3  -----+-----
4  36              | CS Teaching Building
5  53              | Comprehensive Experimental Building
6  10              | Language Teaching Building
7  7              | Main Teaching Building
8  (4 rows)
9
10 building_number | room_number
11 -----+-----
12 53              | 904a
13 53              | 904b
14 (2 rows)
15
16 DELETE 2
17 building_number |          building_name
18 -----+-----
19 36              | CS Teaching Building
20 10              | Language Teaching Building
21 7              | Main Teaching Building
22 (3 rows)
23
24 # psql 16.11 succeeded with exit code 0.
```

# Daten Löschen

- Neben der Redundanz hat diese Struktur aber noch weitere Probleme.
- Diese werden sichtbar, wenn wir die Daten ändern.
- Mit der ersten Anfrage rechts wollen wir uns zuerst eine Liste der Gebäude aus unserer Datenbank holen.

```
1  /** Delete room 904a and 904b, which doesn't delete the building 53. */
2
3  -- Get the list of all buildings.
4  SELECT DISTINCT building_number, building_name FROM building_room;
5
6  -- Delete rooms 904a and 904b in building 53.
7  DELETE FROM building_room WHERE building_number = '53'
8      AND room_number LIKE '904%'
9      RETURNING building_number, room_number;
10
11 -- Get the list of all buildings again: Building 53 is still there.
12 SELECT DISTINCT building_number, building_name FROM building_room;
```

```
1  $ psql "postgres://postgres:XXX@localhost/violation" -v ON_ERROR_STOP=1
   ↪ -ebf delete.sql
2  building_number |          building_name
3  -----+-----
4  36              | CS Teaching Building
5  53              | Comprehensive Experimental Building
6  10              | Language Teaching Building
7  7               | Main Teaching Building
8  (4 rows)
9
10 building_number | room_number
11 -----+-----
12 53              | 904a
13 53              | 904b
14 (2 rows)
15
16 DELETE 2
17 building_number |          building_name
18 -----+-----
19 36              | CS Teaching Building
20 10              | Language Teaching Building
21 7               | Main Teaching Building
22 (3 rows)
23
24 # psql 16.11 succeeded with exit code 0.
```

# Daten Löschen

- Neben der Redundanz hat diese Struktur aber noch weitere Probleme.
- Diese werden sichtbar, wenn wir die Daten ändern.
- Mit der ersten Anfrage rechts wollen wir uns zuerst eine Liste der Gebäude aus unserer Datenbank holen.
- Wir machen ein `SELECT` der Gebäudenummer und des Gebäudenamens aus der Tabelle `building_rooms`.

```
1  /** Delete room 904a and 904b, which doesn't delete the building 53. */
2
3  -- Get the list of all buildings.
4  SELECT DISTINCT building_number, building_name FROM building_room;
5
6  -- Delete rooms 904a and 904b in building 53.
7  DELETE FROM building_room WHERE building_number = '53'
8      AND room_number LIKE '904%'
9      RETURNING building_number, room_number;
10
11 -- Get the list of all buildings again: Building 53 is still there.
12 SELECT DISTINCT building_number, building_name FROM building_room;
```

```
1  $ psql "postgres://postgres:XXX@localhost/violation" -v ON_ERROR_STOP=1
   ↪ -ebf delete.sql
2  building_number |          building_name
3  -----+-----
4  36              | CS Teaching Building
5  53              | Comprehensive Experimental Building
6  10              | Language Teaching Building
7  7               | Main Teaching Building
8  (4 rows)
9
10 building_number | room_number
11 -----+-----
12 53              | 904a
13 53              | 904b
14 (2 rows)
15
16 DELETE 2
17 building_number |          building_name
18 -----+-----
19 36              | CS Teaching Building
20 10              | Language Teaching Building
21 7               | Main Teaching Building
22 (3 rows)
23
24 # psql 16.11 succeeded with exit code 0.
```

# Daten Löschen

- Neben der Redundanz hat diese Struktur aber noch weitere Probleme.
- Diese werden sichtbar, wenn wir die Daten ändern.
- Mit der ersten Anfrage rechts wollen wir uns zuerst eine Liste der Gebäude aus unserer Datenbank holen.
- Wir machen ein `SELECT` der Gebäudenummer und des Gebäudenamens aus der Tabelle `building_rooms`.
- Weil diese mehrfach auftreten könnten, machen ein `SELECT DISTINCT` anstatt nur eines `SELECT`.

```
1  /** Delete room 904a and 904b, which doesn't delete the building 53. */
2
3  -- Get the list of all buildings.
4  SELECT DISTINCT building_number, building_name FROM building_room;
5
6  -- Delete rooms 904a and 904b in building 53.
7  DELETE FROM building_room WHERE building_number = '53'
8      AND room_number LIKE '904%'
9      RETURNING building_number, room_number;
10
11 -- Get the list of all buildings again: Building 53 is still there.
12 SELECT DISTINCT building_number, building_name FROM building_room;
```

```
1  $ psql "postgres://postgres:XXX@localhost/violation" -v ON_ERROR_STOP=1
   ↪ -ebf delete.sql
2  building_number |          building_name
3  -----+-----
4  36              | CS Teaching Building
5  53              | Comprehensive Experimental Building
6  10              | Language Teaching Building
7  7              | Main Teaching Building
8  (4 rows)
9
10 building_number | room_number
11 -----+-----
12 53              | 904a
13 53              | 904b
14 (2 rows)
15
16 DELETE 2
17 building_number |          building_name
18 -----+-----
19 36              | CS Teaching Building
20 10              | Language Teaching Building
21 7              | Main Teaching Building
22 (3 rows)
23
24 # psql 16.11 succeeded with exit code 0.
```

# Daten Löschen

- Diese werden sichtbar, wenn wir die Daten ändern.
- Mit der ersten Anfrage rechts wollen wir uns zuerst eine Liste der Gebäude aus unserer Datenbank holen.
- Wir machen ein `SELECT` der Gebäudenummer und des Gebäudenamens aus der Tabelle `building_rooms`.
- Weil diese mehrfach auftreten könnten, machen ein `SELECT DISTINCT` anstatt nur eines `SELECT`.
- So bekommen wir jede Zeile nur einmal.

```
1  /** Delete room 904a and 904b, which doesn't delete the building 53. */
2
3  -- Get the list of all buildings.
4  SELECT DISTINCT building_number, building_name FROM building_room;
5
6  -- Delete rooms 904a and 904b in building 53.
7  DELETE FROM building_room WHERE building_number = '53'
8      AND room_number LIKE '904%'
9      RETURNING building_number, room_number;
10
11 -- Get the list of all buildings again: Building 53 is still there.
12 SELECT DISTINCT building_number, building_name FROM building_room;
```

```
1  $ psql "postgres://postgres:XXX@localhost/violation" -v ON_ERROR_STOP=1
   ↪ -ebf delete.sql
2  building_number |          building_name
3  -----+-----
4  36              | CS Teaching Building
5  53              | Comprehensive Experimental Building
6  10              | Language Teaching Building
7  7              | Main Teaching Building
8  (4 rows)
9
10 building_number | room_number
11 -----+-----
12 53              | 904a
13 53              | 904b
14 (2 rows)
15
16 DELETE 2
17 building_number |          building_name
18 -----+-----
19 36              | CS Teaching Building
20 10              | Language Teaching Building
21 7              | Main Teaching Building
22 (3 rows)
23
24 # psql 16.11 succeeded with exit code 0.
```

# Daten Löschen

- Mit der ersten Anfrage rechts wollen wir uns zuerst eine Liste der Gebäude aus unserer Datenbank holen.
- Wir machen ein `SELECT` der Gebäudenummer und des Gebäudenamens aus der Tabelle `building_rooms`.
- Weil diese mehrfach auftreten könnten, machen ein `SELECT DISTINCT` anstatt nur eines `SELECT`.
- So bekommen wir jede Zeile nur einmal.
- Das Ergebnis zeigt, dass wir vier Gebäude gespeichert haben.

```
1  /** Delete room 904a and 904b, which doesn't delete the building 53. */
2
3  -- Get the list of all buildings.
4  SELECT DISTINCT building_number, building_name FROM building_room;
5
6  -- Delete rooms 904a and 904b in building 53.
7  DELETE FROM building_room WHERE building_number = '53'
8      AND room_number LIKE '904%'
9      RETURNING building_number, room_number;
10
11 -- Get the list of all buildings again: Building 53 is still there.
12 SELECT DISTINCT building_number, building_name FROM building_room;
```

```
1  $ psql "postgres://postgres:XXX@localhost/violation" -v ON_ERROR_STOP=1
   ↪ -ebf delete.sql
2  building_number |          building_name
3  -----+-----
4  36              | CS Teaching Building
5  53              | Comprehensive Experimental Building
6  10              | Language Teaching Building
7  7              | Main Teaching Building
8  (4 rows)
9
10 building_number | room_number
11 -----+-----
12 53              | 904a
13 53              | 904b
14 (2 rows)
15
16 DELETE 2
17 building_number |          building_name
18 -----+-----
19 36              | CS Teaching Building
20 10              | Language Teaching Building
21 7              | Main Teaching Building
22 (3 rows)
23
24 # psql 16.11 succeeded with exit code 0.
```

# Daten Löschen

- Wir machen ein `SELECT` der Gebäudenummer und des Gebäudenamens aus der Tabelle `building_rooms`.
- Weil diese mehrfach auftreten könnten, machen ein `SELECT DISTINCT` anstatt nur eines `SELECT`.
- So bekommen wir jede Zeile nur einmal.
- Das Ergebnis zeigt, dass wir vier Gebäude gespeichert haben.
- Stellen Sie sich vor, dass einige Zeit vergangen ist und wir uns entschieden haben, dass wir die Räume 904a und 904b nicht mehr für die Lehre verwenden wollen.

```
1  /** Delete room 904a and 904b, which doesn't delete the building 53. */
2
3  -- Get the list of all buildings.
4  SELECT DISTINCT building_number, building_name FROM building_room;
5
6  -- Delete rooms 904a and 904b in building 53.
7  DELETE FROM building_room WHERE building_number = '53'
8      AND room_number LIKE '904%'
9      RETURNING building_number, room_number;
10
11 -- Get the list of all buildings again: Building 53 is still there.
12 SELECT DISTINCT building_number, building_name FROM building_room;
```

```
1  $ psql "postgres://postgres:XXX@localhost/violation" -v ON_ERROR_STOP=1
   ↪ -ebf delete.sql
2  building_number |          building_name
3  -----+-----
4  36              | CS Teaching Building
5  53              | Comprehensive Experimental Building
6  10              | Language Teaching Building
7  7              | Main Teaching Building
8  (4 rows)
9
10 building_number | room_number
11 -----+-----
12 53              | 904a
13 53              | 904b
14 (2 rows)
15
16 DELETE 2
17 building_number |          building_name
18 -----+-----
19 36              | CS Teaching Building
20 10              | Language Teaching Building
21 7              | Main Teaching Building
22 (3 rows)
23
24 # psql 16.11 succeeded with exit code 0.
```

# Daten Löschen

- Weil diese mehrfach auftreten könnten, machen ein **SELECT DISTINCT** anstatt nur eines **SELECT**.
- So bekommen wir jede Zeile nur einmal.
- Das Ergebnis zeigt, dass wir vier Gebäude gespeichert haben.
- Stellen Sie sich vor, dass einige Zeit vergangen ist und wir uns entschieden haben, dass wir die Räume 904a und 904b nicht mehr für die Lehre verwenden wollen.
- Mit der zweiten Anfrage rechts löschen wir sie aus unserer Datenbank.

```
1  /** Delete room 904a and 904b, which doesn't delete the building 53. */
2
3  -- Get the list of all buildings.
4  SELECT DISTINCT building_number, building_name FROM building_room;
5
6  -- Delete rooms 904a and 904b in building 53.
7  DELETE FROM building_room WHERE building_number = '53'
8      AND room_number LIKE '904%'
9      RETURNING building_number, room_number;
10
11 -- Get the list of all buildings again: Building 53 is still there.
12 SELECT DISTINCT building_number, building_name FROM building_room;
```

```
1  $ psql "postgres://postgres:XXX@localhost/violation" -v ON_ERROR_STOP=1
   ↪ -ebf delete.sql
2  building_number |          building_name
3  -----+-----
4  36              | CS Teaching Building
5  53              | Comprehensive Experimental Building
6  10              | Language Teaching Building
7  7              | Main Teaching Building
8  (4 rows)
9
10 building_number | room_number
11 -----+-----
12 53              | 904a
13 53              | 904b
14 (2 rows)
15
16 DELETE 2
17 building_number |          building_name
18 -----+-----
19 36              | CS Teaching Building
20 10              | Language Teaching Building
21 7              | Main Teaching Building
22 (3 rows)
23
24 # psql 16.11 succeeded with exit code 0.
```

# Daten Löschen

- So bekommen wir jede Zeile nur einmal.
- Das Ergebnis zeigt, dass wir vier Gebäude gespeichert haben.
- Stellen Sie sich vor, dass einige Zeit vergangen ist und wir uns entschieden haben, dass wir die Räume 904a und 904b nicht mehr für die Lehre verwenden wollen.
- Mit der zweiten Anfrage rechts löschen wir sie aus unserer Datenbank.
- Das geht mit `DELETE FROM`, wobei das `WHERE` erzwingt, dass die Gebäudenummer 53 ist und dass die zu löschenden Zimmer mit 904 anfangen (`LIKE '904%'`).

```
1  /** Delete room 904a and 904b, which doesn't delete the building 53. */
2
3  -- Get the list of all buildings.
4  SELECT DISTINCT building_number, building_name FROM building_room;
5
6  -- Delete rooms 904a and 904b in building 53.
7  DELETE FROM building_room WHERE building_number = '53'
8      AND room_number LIKE '904%'
9      RETURNING building_number, room_number;
10
11 -- Get the list of all buildings again: Building 53 is still there.
12 SELECT DISTINCT building_number, building_name FROM building_room;
```

```
1  $ psql "postgres://postgres:XXX@localhost/violation" -v ON_ERROR_STOP=1
   ↪ -ebf delete.sql
2  building_number |          building_name
3  -----+-----
4  36              | CS Teaching Building
5  53              | Comprehensive Experimental Building
6  10              | Language Teaching Building
7  7              | Main Teaching Building
8  (4 rows)
9
10 building_number | room_number
11 -----+-----
12 53              | 904a
13 53              | 904b
14 (2 rows)
15
16 DELETE 2
17 building_number |          building_name
18 -----+-----
19 36              | CS Teaching Building
20 10              | Language Teaching Building
21 7              | Main Teaching Building
22 (3 rows)
23
24 # psql 16.11 succeeded with exit code 0.
```

# Daten Löschen

- Stellen Sie sich vor, dass einige Zeit vergangen ist und wir uns entschieden haben, dass wir die Räume 904a und 904b nicht mehr für die Lehre verwenden wollen.
- Mit der zweiten Anfrage rechts löschen wir sie aus unserer Datenbank.
- Das geht mit `DELETE FROM`, wobei das `WHERE` erzwingt, dass die Gebäudenummer 53 ist und dass die zu löschenden Zimmer mit 904 anfangen (`LIKE '904%'`).
- PostgreSQL erlaubt es uns, die gelöschten Zeilen über das `RETURNING`-Statement<sup>52</sup> zurückzuliefern.

```
1  /** Delete room 904a and 904b, which doesn't delete the building 53. */
2
3  -- Get the list of all buildings.
4  SELECT DISTINCT building_number, building_name FROM building_room;
5
6  -- Delete rooms 904a and 904b in building 53.
7  DELETE FROM building_room WHERE building_number = '53'
8      AND room_number LIKE '904%'
9      RETURNING building_number, room_number;
10
11 -- Get the list of all buildings again: Building 53 is still there.
12 SELECT DISTINCT building_number, building_name FROM building_room;
```

```
1  $ psql "postgres://postgres:XXX@localhost/violation" -v ON_ERROR_STOP=1
   ↪ -ebf delete.sql
2  building_number |          building_name
3  -----+-----
4  36              | CS Teaching Building
5  53              | Comprehensive Experimental Building
6  10              | Language Teaching Building
7  7               | Main Teaching Building
8  (4 rows)
9
10 building_number | room_number
11 -----+-----
12 53              | 904a
13 53              | 904b
14 (2 rows)
15
16 DELETE 2
17 building_number |          building_name
18 -----+-----
19 36              | CS Teaching Building
20 10              | Language Teaching Building
21 7               | Main Teaching Building
22 (3 rows)
23
24 # psql 16.11 succeeded with exit code 0.
```

# Daten Löschen

- Mit der zweiten Anfrage rechts löschen wir sie aus unserer Datenbank.
- Das geht mit `DELETE FORM`, wobei das `WHERE` erzwingt, dass die Gebäudenummer 53 ist und dass die zu löschenden Zimmer mit 904 anfangen (`LIKE '904%'`).
- PostgreSQL erlaubt es uns, die gelöschten Zeilen über das `RETURNING`-Statement<sup>52</sup> zurückzuliefern.
- Die Ausgabe zeigt die Gebäudenummer und Raumnummern der gelöschten Zimmer.

```
1  /** Delete room 904a and 904b, which doesn't delete the building 53. */
2
3  -- Get the list of all buildings.
4  SELECT DISTINCT building_number, building_name FROM building_room;
5
6  -- Delete rooms 904a and 904b in building 53.
7  DELETE FROM building_room WHERE building_number = '53'
8      AND room_number LIKE '904%'
9      RETURNING building_number, room_number;
10
11 -- Get the list of all buildings again: Building 53 is still there.
12 SELECT DISTINCT building_number, building_name FROM building_room;
```

```
1  $ psql "postgres://postgres:XXX@localhost/violation" -v ON_ERROR_STOP=1
   ↪ -ebf delete.sql
2  building_number |          building_name
3  -----+-----
4  36              | CS Teaching Building
5  53              | Comprehensive Experimental Building
6  10              | Language Teaching Building
7  7              | Main Teaching Building
8  (4 rows)
9
10 building_number | room_number
11 -----+-----
12 53              | 904a
13 53              | 904b
14 (2 rows)
15
16 DELETE 2
17 building_number |          building_name
18 -----+-----
19 36              | CS Teaching Building
20 10              | Language Teaching Building
21 7              | Main Teaching Building
22 (3 rows)
23
24 # psql 16.11 succeeded with exit code 0.
```

# Daten Löschen

- Mit der zweiten Anfrage rechts löschen wir sie aus unserer Datenbank.
- Das geht mit **DELETE FORM**, wobei das **WHERE** erzwingt, dass die Gebäudenummer 53 ist und dass die zu löschenden Zimmer mit 904 anfangen (**LIKE '904%'**).
- PostgreSQL erlaubt es uns, die gelöschten Zeilen über das **RETURNING**-Statement<sup>52</sup> zurückzuliefern.
- Die Ausgabe zeigt die Gebäudenummer und Raumnummern der gelöschten Zimmer.
- Wir haben wirklich Räume 904a und 904b von Gebäude 53 gelöscht.

```
1  /** Delete room 904a and 904b, which doesn't delete the building 53. */
2
3  -- Get the list of all buildings.
4  SELECT DISTINCT building_number, building_name FROM building_room;
5
6  -- Delete rooms 904a and 904b in building 53.
7  DELETE FROM building_room WHERE building_number = '53'
8      AND room_number LIKE '904%'
9      RETURNING building_number, room_number;
10
11 -- Get the list of all buildings again: Building 53 is still there.
12 SELECT DISTINCT building_number, building_name FROM building_room;
```

```
1  $ psql "postgres://postgres:XXX@localhost/violation" -v ON_ERROR_STOP=1
   ↪ -ebf delete.sql
2  building_number |          building_name
3  -----+-----
4  36              | CS Teaching Building
5  53              | Comprehensive Experimental Building
6  10              | Language Teaching Building
7  7              | Main Teaching Building
8  (4 rows)
9
10 building_number | room_number
11 -----+-----
12 53              | 904a
13 53              | 904b
14 (2 rows)
15
16 DELETE 2
17 building_number |          building_name
18 -----+-----
19 36              | CS Teaching Building
20 10              | Language Teaching Building
21 7              | Main Teaching Building
22 (3 rows)
23
24 # psql 16.11 succeeded with exit code 0.
```

# Daten Löschen

- PostgreSQL erlaubt es uns, die gelöschten Zeilen über das **RETURNING**-Statement<sup>52</sup> zurückzuliefern.
- Die Ausgabe zeigt die Gebäudenummer und Raumnummern der gelöschten Zimmer.
- Wir haben wirklich Räume 904a und 904b von Gebäude 53 gelöscht.
- Wir führen nun die selbe Anfrage für die Liste der Gebäude noch einmal (als dritte Anfrage) aus.

```
1  /** Delete room 904a and 904b, which doesn't delete the building 53. */
2
3  -- Get the list of all buildings.
4  SELECT DISTINCT building_number, building_name FROM building_room;
5
6  -- Delete rooms 904a and 904b in building 53.
7  DELETE FROM building_room WHERE building_number = '53'
8      AND room_number LIKE '904%'
9      RETURNING building_number, room_number;
10
11 -- Get the list of all buildings again: Building 53 is still there.
12 SELECT DISTINCT building_number, building_name FROM building_room;
```

```
1  $ psql "postgres://postgres:XXX@localhost/violation" -v ON_ERROR_STOP=1
   ↪ -ebf delete.sql
2
3  building_number |          building_name
4  -----+-----
5  36              | CS Teaching Building
6  53              | Comprehensive Experimental Building
7  10              | Language Teaching Building
8  7              | Main Teaching Building
9  (4 rows)
10
11 building_number | room_number
12 -----+-----
13  53              | 904a
14  53              | 904b
15  (2 rows)
16
17 DELETE 2
18
19 building_number |          building_name
20 -----+-----
21  36              | CS Teaching Building
22  10              | Language Teaching Building
23  7              | Main Teaching Building
24  (3 rows)
25
26 # psql 16.11 succeeded with exit code 0.
```

# Daten Löschen

- Die Ausgabe zeigt die Gebäudenummer und Raumnummern der gelöschten Zimmer.
- Wir haben wirklich Räume 904a und 904b von Gebäude 53 gelöscht.
- Wir führen nun die selbe Anfrage für die Liste der Gebäude noch einmal (als dritte Anfrage) aus.
- Wir stellen fest, dass wir durch das Löschen der beiden Zimmer von Gebäude 53 auch gleich alle Informationen über das Gebäude selber mit gelöscht haben.

```
1  /** Delete room 904a and 904b, which doesn't delete the building 53. */
2
3  -- Get the list of all buildings.
4  SELECT DISTINCT building_number, building_name FROM building_room;
5
6  -- Delete rooms 904a and 904b in building 53.
7  DELETE FROM building_room WHERE building_number = '53'
8      AND room_number LIKE '904%'
9      RETURNING building_number, room_number;
10
11 -- Get the list of all buildings again: Building 53 is still there.
12 SELECT DISTINCT building_number, building_name FROM building_room;
```

```
1  $ psql "postgres://postgres:XXX@localhost/violation" -v ON_ERROR_STOP=1
   ↪ -ebf delete.sql
2
3  building_number |          building_name
4  -----+-----
5  36              | CS Teaching Building
6  53              | Comprehensive Experimental Building
7  10              | Language Teaching Building
8  7              | Main Teaching Building
9  (4 rows)
10
11 building_number | room_number
12 -----+-----
13  53              | 904a
14  53              | 904b
15  (2 rows)
16
17 DELETE 2
18
19 building_number |          building_name
20 -----+-----
21  36              | CS Teaching Building
22  10              | Language Teaching Building
23  7              | Main Teaching Building
24  (3 rows)
25
26 # psql 16.11 succeeded with exit code 0.
```

# Daten Löschen

- Die Ausgabe zeigt die Gebäudenummer und Raumnummern der gelöschten Zimmer.
- Wir haben wirklich Räume 904a und 904b von Gebäude 53 gelöscht.
- Wir führen nun die selbe Anfrage für die Liste der Gebäude noch einmal (als dritte Anfrage) aus.
- Wir stellen fest, dass wir durch das Löschen der beiden Zimmer von Gebäude 53 auch gleich alle Informationen über das Gebäude selber mit gelöscht haben.
- Seine Gebäudenummer, Gebäudename und Adresse sind alle verloren.

```
1  /** Delete room 904a and 904b, which doesn't delete the building 53. */
2
3  -- Get the list of all buildings.
4  SELECT DISTINCT building_number, building_name FROM building_room;
5
6  -- Delete rooms 904a and 904b in building 53.
7  DELETE FROM building_room WHERE building_number = '53'
8      AND room_number LIKE '904%'
9      RETURNING building_number, room_number;
10
11 -- Get the list of all buildings again: Building 53 is still there.
12 SELECT DISTINCT building_number, building_name FROM building_room;
```

```
1  $ psql "postgres://postgres:XXX@localhost/violation" -v ON_ERROR_STOP=1
   ↪ -ebf delete.sql
2  building_number |          building_name
3  -----+-----
4  36              | CS Teaching Building
5  53              | Comprehensive Experimental Building
6  10              | Language Teaching Building
7  7               | Main Teaching Building
8  (4 rows)
9
10 building_number | room_number
11 -----+-----
12 53              | 904a
13 53              | 904b
14 (2 rows)
15
16 DELETE 2
17 building_number |          building_name
18 -----+-----
19 36              | CS Teaching Building
20 10              | Language Teaching Building
21 7               | Main Teaching Building
22 (3 rows)
23
24 # psql 16.11 succeeded with exit code 0.
```

# Daten Löschen

- Wir haben wirklich Räume 904a und 904b von Gebäude 53 gelöscht.
- Wir führen nun die selbe Anfrage für die Liste der Gebäude noch einmal (als dritte Anfrage) aus.
- Wir stellen fest, dass wir durch das Löschen der beiden Zimmer von Gebäude 53 auch gleich alle Informationen über das Gebäude selber mit gelöscht haben.
- Seine Gebäudenummer, Gebäudename und Adresse sind alle verloren.
- Sie waren ja zusammen mit den Zimmer-Daten gespeichert.

```
1  /** Delete room 904a and 904b, which doesn't delete the building 53. */
2
3  -- Get the list of all buildings.
4  SELECT DISTINCT building_number, building_name FROM building_room;
5
6  -- Delete rooms 904a and 904b in building 53.
7  DELETE FROM building_room WHERE building_number = '53'
8      AND room_number LIKE '904%'
9      RETURNING building_number, room_number;
10
11 -- Get the list of all buildings again: Building 53 is still there.
12 SELECT DISTINCT building_number, building_name FROM building_room;
```

```
1  $ psql "postgres://postgres:XXX@localhost/violation" -v ON_ERROR_STOP=1
   ↪ -ebf delete.sql
2  building_number |          building_name
3  -----+-----
4  36              | CS Teaching Building
5  53              | Comprehensive Experimental Building
6  10              | Language Teaching Building
7  7              | Main Teaching Building
8  (4 rows)
9
10 building_number | room_number
11 -----+-----
12 53              | 904a
13 53              | 904b
14 (2 rows)
15
16 DELETE 2
17 building_number |          building_name
18 -----+-----
19 36              | CS Teaching Building
20 10              | Language Teaching Building
21 7              | Main Teaching Building
22 (3 rows)
23
24 # psql 16.11 succeeded with exit code 0.
```

# Daten Löschen

- Wir führen nun die selbe Anfrage für die Liste der Gebäude noch einmal (als dritte Anfrage) aus.
- Wir stellen fest, dass wir durch das Löschen der beiden Zimmer von Gebäude 53 auch gleich alle Informationen über das Gebäude selber mit gelöscht haben.
- Seine Gebäudenummer, Gebäudename und Adresse sind alle verloren.
- Sie waren ja zusammen mit den Zimmer-Daten gespeichert.
- Das nennt man Löschanomalie.

```
1  /** Delete room 904a and 904b, which doesn't delete the building 53. */
2
3  -- Get the list of all buildings.
4  SELECT DISTINCT building_number, building_name FROM building_room;
5
6  -- Delete rooms 904a and 904b in building 53.
7  DELETE FROM building_room WHERE building_number = '53'
8      AND room_number LIKE '904%'
9      RETURNING building_number, room_number;
10
11 -- Get the list of all buildings again: Building 53 is still there.
12 SELECT DISTINCT building_number, building_name FROM building_room;
```

```
1  $ psql "postgres://postgres:XXX@localhost/violation" -v ON_ERROR_STOP=1
   ↪ -ebf delete.sql
2  building_number |          building_name
3  -----+-----
4  36              | CS Teaching Building
5  53              | Comprehensive Experimental Building
6  10              | Language Teaching Building
7  7               | Main Teaching Building
8  (4 rows)
9
10 building_number | room_number
11 -----+-----
12 53              | 904a
13 53              | 904b
14 (2 rows)
15
16 DELETE 2
17 building_number |          building_name
18 -----+-----
19 36              | CS Teaching Building
20 10              | Language Teaching Building
21 7               | Main Teaching Building
22 (3 rows)
23
24 # psql 16.11 succeeded with exit code 0.
```

# Daten Löschen



- Seine Gebäudenummer, Gebäudename und Adresse sind alle verloren.
- Sie waren ja zusammen mit den Zimmer-Daten gespeichert.
- Das nennt man Löschanomalie.

## Definition: Löschanomalie

Eine Situation wo das Löschen von unbenötigten Daten auch das Löschen von noch benötigten Daten bewirkt, nennt man *Löschanomalie* (EN: *deletion anomaly*)<sup>59</sup>.

# Daten Löschen

- Sie waren ja zusammen mit den Zimmer-Daten gespeichert.
- Das nennt man Löschanomalie.
- Na gut, Sie könnten ja argumentieren, dass das so gewünscht war.

```
1  /** Delete room 904a and 904b, which doesn't delete the building 53. */
2
3  -- Get the list of all buildings.
4  SELECT DISTINCT building_number, building_name FROM building_room;
5
6  -- Delete rooms 904a and 904b in building 53.
7  DELETE FROM building_room WHERE building_number = '53'
8      AND room_number LIKE '904%'
9      RETURNING building_number, room_number;
10
11 -- Get the list of all buildings again: Building 53 is still there.
12 SELECT DISTINCT building_number, building_name FROM building_room;
```

```
1  $ psql "postgres://postgres:XXX@localhost/violation" -v ON_ERROR_STOP=1
   ↪ -ebf delete.sql
2  building_number |          building_name
3  -----+-----
4  36              | CS Teaching Building
5  53              | Comprehensive Experimental Building
6  10              | Language Teaching Building
7  7               | Main Teaching Building
8  (4 rows)
9
10 building_number | room_number
11 -----+-----
12 53              | 904a
13 53              | 904b
14 (2 rows)
15
16 DELETE 2
17 building_number |          building_name
18 -----+-----
19 36              | CS Teaching Building
20 10              | Language Teaching Building
21 7               | Main Teaching Building
22 (3 rows)
23
24 # psql 16.11 succeeded with exit code 0.
```

# Daten Löschen

- Sie waren ja zusammen mit den Zimmer-Daten gespeichert.
- Das nennt man Löschanomalie.
- Na gut, Sie könnten ja argumentieren, dass das so gewünscht war.
- Im Originalmodell war die Beziehung zwischen Raum und Gebäude ja als Room  $\rhd$   $\text{---}$   $\text{---}$  Building angegeben.

```
1  /** Delete room 904a and 904b, which doesn't delete the building 53. */
2
3  -- Get the list of all buildings.
4  SELECT DISTINCT building_number, building_name FROM building_room;
5
6  -- Delete rooms 904a and 904b in building 53.
7  DELETE FROM building_room WHERE building_number = '53'
8      AND room_number LIKE '904%'
9      RETURNING building_number, room_number;
10
11 -- Get the list of all buildings again: Building 53 is still there.
12 SELECT DISTINCT building_number, building_name FROM building_room;
```

```
1  $ psql "postgres://postgres:XXX@localhost/violation" -v ON_ERROR_STOP=1
   ↪ -ebf delete.sql
2  building_number |          building_name
3  -----+-----
4  36              | CS Teaching Building
5  53              | Comprehensive Experimental Building
6  10              | Language Teaching Building
7  7              | Main Teaching Building
8  (4 rows)
9
10 building_number | room_number
11 -----+-----
12 53              | 904a
13 53              | 904b
14 (2 rows)
15
16 DELETE 2
17 building_number |          building_name
18 -----+-----
19 36              | CS Teaching Building
20 10              | Language Teaching Building
21 7              | Main Teaching Building
22 (3 rows)
23
24 # psql 16.11 succeeded with exit code 0.
```

## Daten Löschen

- Das nennt man Löschanomalie.
- Na gut, Sie könnten ja argumentieren, dass das so gewünscht war.
- Im Originalmodell war die Beziehung zwischen Raum und Gebäude ja als  $\text{Room} \rhd \text{Building}$  angegeben.
- In dieser Beziehungsart wäre es ja sowieso *erforderlich* die Gebäude zu löschen, wenn alle ihre Räume gelöscht werden.

```

1  /** Delete room 904a and 904b, which doesn't delete the building 53. */
2
3  -- Get the list of all buildings.
4  SELECT DISTINCT building_number, building_name FROM building_room;
5
6  -- Delete rooms 904a and 904b in building 53.
7  DELETE FROM building_room WHERE building_number = '53'
8      AND room_number LIKE '904%'
9      RETURNING building_number, room_number;
10
11 -- Get the list of all buildings again: Building 53 is still there.
12 SELECT DISTINCT building_number, building_name FROM building_room;

```

```

1  $ psql "postgres://postgres:XXX@localhost/violation" -v ON_ERROR_STOP=1
   ↪ -ebf delete.sql

```

| building_number | building_name                       |
|-----------------|-------------------------------------|
| 36              | CS Teaching Building                |
| 53              | Comprehensive Experimental Building |
| 10              | Language Teaching Building          |
| 7               | Main Teaching Building              |

(4 rows)

| building_number | room_number |
|-----------------|-------------|
| 53              | 904a        |
| 53              | 904b        |

(2 rows)

```

16 DELETE 2

```

| building_number | building_name              |
|-----------------|----------------------------|
| 36              | CS Teaching Building       |
| 10              | Language Teaching Building |
| 7               | Main Teaching Building     |

(3 rows)

```

24 # psql 16.11 succeeded with exit code 0.

```

## Daten Löschen

- Na gut, Sie könnten ja argumentieren, dass das so gewünscht war.
- Im Originalmodell war die Beziehung zwischen Raum und Gebäude ja als  $\text{Room} \succ \text{Building}$  angegeben.
- In dieser Beziehungsart wäre es ja sowieso *erforderlich* die Gebäude zu löschen, wenn alle ihre Räume gelöscht werden.
- Diese Anomalie tritt aber auch dann auf, wenn wir dem  $\text{K} \text{--} \text{L}$  Schema folgen, wo so eine Löschung nicht notwendig ist.

```

1  /** Delete room 904a and 904b, which doesn't delete the building 53. */
2
3  -- Get the list of all buildings.
4  SELECT DISTINCT building_number, building_name FROM building_room;
5
6  -- Delete rooms 904a and 904b in building 53.
7  DELETE FROM building_room WHERE building_number = '53'
8      AND room_number LIKE '904%'
9      RETURNING building_number, room_number;
10
11 -- Get the list of all buildings again: Building 53 is still there.
12 SELECT DISTINCT building_number, building_name FROM building_room;

```

---

```

1  $ psql "postgres://postgres:XXX@localhost/violation" -v ON_ERROR_STOP=1
   ↪ -ebf delete.sql

```

| building_number | building_name                       |
|-----------------|-------------------------------------|
| 36              | CS Teaching Building                |
| 53              | Comprehensive Experimental Building |
| 10              | Language Teaching Building          |
| 7               | Main Teaching Building              |

(4 rows)

| building_number | room_number |
|-----------------|-------------|
| 53              | 904a        |
| 53              | 904b        |

(2 rows)

```

16 DELETE 2

```

| building_number | building_name              |
|-----------------|----------------------------|
| 36              | CS Teaching Building       |
| 10              | Language Teaching Building |
| 7               | Main Teaching Building     |

(3 rows)

```

24 # psql 16.11 succeeded with exit code 0.

```

# Daten Ändern (1)



- Das Verletzen der 2NF macht es auch leichter, inkonsistente Daten zu produzieren.

```
1  /** Find all the rooms available in South Campus 2. */
2
3  -- Several rooms are missing, because the data is inconsistent:
4  -- Their address was 'South Campus II'.
5  SELECT building_number, room_number FROM building_room
6         WHERE address = 'South Campus 2';
```

```
1  $ psql "postgres://postgres:XXX@localhost/violation" -v ON_ERROR_STOP=1
   ↳ -ebf select_a_1.sql
2  building_number | room_number
3  -----+-----
4  36              | 106
5  (1 row)
6
7  # psql 16.11 succeeded with exit code 0.
```

# Daten Ändern (1)



- Das Verletzen der 2NF macht es auch leichter, inkonsistente Daten zu produzieren.
- Nehmen Sie an, wir wollten eine Liste von allen Räumen in South Campus 2 erstellen.

```
1  /** Find all the rooms available in South Campus 2. */
2
3  -- Several rooms are missing, because the data is inconsistent:
4  -- Their address was 'South Campus II'.
5  SELECT building_number, room_number FROM building_room
6         WHERE address = 'South Campus 2';
```

```
1  $ psql "postgres://postgres:XXX@localhost/violation" -v ON_ERROR_STOP=1
   ↪ -ebf select_a_1.sql
2  building_number | room_number
3  -----+-----
4  36              | 106
5  (1 row)
6
7  # psql 16.11 succeeded with exit code 0.
```

# Daten Ändern (1)



- Das Verletzen der 2NF macht es auch leichter, inkonsistente Daten zu produzieren.
- Nehmen Sie an, wir wollten eine Liste von allen Räumen in South Campus 2 erstellen.
- Dazu führen wir die Anfrage rechts aus.

```
1  /** Find all the rooms available in South Campus 2. */
2
3  -- Several rooms are missing, because the data is inconsistent:
4  -- Their address was 'South Campus II'.
5  SELECT building_number, room_number FROM building_room
6         WHERE address = 'South Campus 2';
```

```
1  $ psql "postgres://postgres:XXX@localhost/violation" -v ON_ERROR_STOP=1
   ↪ -ebf select_a_1.sql
2  building_number | room_number
3  -----+-----
4  36               | 106
5  (1 row)
6
7  # psql 16.11 succeeded with exit code 0.
```

# Daten Ändern (1)



- Das Verletzen der 2NF macht es auch leichter, inkonsistente Daten zu produzieren.
- Nehmen Sie an, wir wollten eine Liste von allen Räumen in South Campus 2 erstellen.
- Dazu führen wir die Anfrage rechts aus.
- Wir könnten einfach `SELECT` auf die Gebäude- und Raumnummer machen für alle Zeilen mit `address = 'South Campus 2'`.

```
1  /** Find all the rooms available in South Campus 2. */
2
3  -- Several rooms are missing, because the data is inconsistent:
4  -- Their address was 'South Campus II'.
5  SELECT building_number, room_number FROM building_room
6         WHERE address = 'South Campus 2';

```

```
1  $ psql "postgres://postgres:XXX@localhost/violation" -v ON_ERROR_STOP=1
   ↳ -ebf select_a_1.sql
2  building_number | room_number
3  -----+-----
4  36              | 106
5  (1 row)
6
7  # psql 16.11 succeeded with exit code 0.
```

# Daten Ändern (1)



- Das Verletzen der 2NF macht es auch leichter, inkonsistente Daten zu produzieren.
- Nehmen Sie an, wir wollten eine Liste von allen Räumen in South Campus 2 erstellen.
- Dazu führen wir die Anfrage rechts aus.
- Wir könnten einfach `SELECT` auf die Gebäude- und Raumnummer machen für alle Zeilen mit `address = 'South Campus 2'`.
- Diese Anfrage produziert nur genau ein Ergebnis, nämlich Room 106 in Building 36.

```
1  /** Find all the rooms available in South Campus 2. */
2
3  -- Several rooms are missing, because the data is inconsistent:
4  -- Their address was 'South Campus II'.
5  SELECT building_number, room_number FROM building_room
6         WHERE address = 'South Campus 2';

```

```
1  $ psql "postgres://postgres:XXX@localhost/violation" -v ON_ERROR_STOP=1
   ↪ -ebf select_a_1.sql
2  building_number | room_number
3  -----+-----
4  36              | 106
5  (1 row)
6
7  # psql 16.11 succeeded with exit code 0.

```

# Daten Ändern (1)



- Nehmen Sie an, wir wollten eine Liste von allen Räumen in South Campus 2 erstellen.
- Dazu führen wir die Anfrage rechts aus.
- Wir könnten einfach `SELECT` auf die Gebäude- und Raumnummer machen für alle Zeilen mit `address = 'South Campus 2'`.
- Diese Anfrage produziert nur genau ein Ergebnis, nämlich Room 106 in Building 36.
- Wir wissen, dass das falsch ist.

```
1  /** Find all the rooms available in South Campus 2. */
2
3  -- Several rooms are missing, because the data is inconsistent:
4  -- Their address was 'South Campus II'.
5  SELECT building_number, room_number FROM building_room
6         WHERE address = 'South Campus 2';
```

```
1  $ psql "postgres://postgres:XXX@localhost/violation" -v ON_ERROR_STOP=1
   ↪ -ebf select_a_1.sql
2  building_number | room_number
3  -----+-----
4  36              | 106
5  (1 row)
6
7  # psql 16.11 succeeded with exit code 0.
```

# Daten Ändern (1)



- Dazu führen wir die Anfrage rechts aus.
- Wir könnten einfach `SELECT` auf die Gebäude- und Raumnummer machen für alle Zeilen mit `address = 'South Campus 2'`.
- Diese Anfrage produziert nur genau ein Ergebnis, nämlich Room 106 in Building 36.
- Wir wissen, dass das falsch ist.
- Wir haben definitiv mehr als einen Raum für Building 36 gespeichert.

```
1  /** Find all the rooms available in South Campus 2. */
2
3  -- Several rooms are missing, because the data is inconsistent:
4  -- Their address was 'South Campus II'.
5  SELECT building_number, room_number FROM building_room
6         WHERE address = 'South Campus 2';
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
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
```

```
1  $ psql "postgres://postgres:XXX@localhost/violation" -v ON_ERROR_STOP=1
2  ↵ -ebf select_a_1.sql
3  building_number | room_number
4  -----+-----
5  36              | 106
6  (1 row)
7
8  # psql 16.11 succeeded with exit code 0.
```

# Daten Ändern (1)



- Wir könnten einfach `SELECT` auf die Gebäude- und Raumnummer machen für alle Zeilen mit `address = 'South Campus 2'`.
- Diese Anfrage produziert nur genau ein Ergebnis, nämlich Room 106 in Building 36.
- Wir wissen, dass das falsch ist.
- Wir haben definitiv mehr als einen Raum für Building 36 gespeichert.
- Schauen wir uns nochmal unser `INSERT`-Statement von vorhin an.

```
1  /** Insert data into the database. */
2
3  -- Insert several room records.
4  INSERT INTO building_room (building_number, room_number, building_name,
5                             room_name, capacity, address) VALUES
6
7      ('36', '305', 'CS Teaching Building', 'Meeting Room', 40,
8       'South Campus II'),
9      ('36', '105', 'CS Teaching Building', 'Lecture Room 1', 80,
10       'South Campus II'),
11      ('10', '100', 'Language Teaching Building', 'Teaching Room A', 30,
12       'South Campus 1'),
13      ('10', '102', 'Language Teaching Building', 'Teaching Room B', 30,
14       'South Campus 1'),
15      ('53', '904a', 'Comprehensive Experimental Building',
16       'Office 1', 10, 'South Campus 2'),
17      ('53', '904b', 'Comprehensive Experimental Building',
18       'Cluster Room', 3, 'South Campus 2'),
19      ('7', '200', 'Main Teaching Building', 'Auditorium', 120,
20       'South Campus 1'),
21      ('36', '106', 'CS Teaching Building', 'Lecture Room 2', 80,
22       'South Campus 2');
```

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

# Daten Ändern (1)



- Diese Anfrage produziert nur genau ein Ergebnis, nämlich Room 106 in Building 36.
- Wir wissen, dass das falsch ist.
- Wir haben definitiv mehr als einen Raum für Building 36 gespeichert.
- Schauen wir uns nochmal unser `INSERT`-Statement von vorhin an.
- Ah. Wir haben manchmal `'South Campus II'` anstatt von `'South Campus 2'` geschrieben.

```
1  /** Insert data into the database. */
2
3  -- Insert several room records.
4  INSERT INTO building_room (building_number, room_number, building_name,
5                             room_name, capacity, address) VALUES
6      ('36', '305', 'CS Teaching Building', 'Meeting Room', 40,
7       'South Campus II'),
8      ('36', '105', 'CS Teaching Building', 'Lecture Room 1', 80,
9       'South Campus II'),
10     ('10', '100', 'Language Teaching Building', 'Teaching Room A', 30,
11      'South Campus 1'),
12     ('10', '102', 'Language Teaching Building', 'Teaching Room B', 30,
13      'South Campus 1'),
14     ('53', '904a', 'Comprehensive Experimental Building',
15      'Office 1', 10, 'South Campus 2'),
16     ('53', '904b', 'Comprehensive Experimental Building',
17      'Cluster Room', 3, 'South Campus 2'),
18     ('7', '200', 'Main Teaching Building', 'Auditorium', 120,
19      'South Campus 1'),
20     ('36', '106', 'CS Teaching Building', 'Lecture Room 2', 80,
21      'South Campus 2');
```

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

# Daten Ändern (1)



- Wir wissen, dass das falsch ist.
- Wir haben definitiv mehr als einen Raum für Building 36 gespeichert.
- Schauen wir uns nochmal unser `INSERT`-Statement von vorhin an.
- Ah. Wir haben manchmal `'South Campus II'` anstatt von `'South Campus 2'` geschrieben.
- Für einen Computer sind das natürlich zwei verschiedene Dinge.

```
1  /** Insert data into the database. */
2
3  -- Insert several room records.
4  INSERT INTO building_room (building_number, room_number, building_name,
5                             room_name, capacity, address) VALUES
6      ('36', '305', 'CS Teaching Building', 'Meeting Room', 40,
7       'South Campus II'),
8      ('36', '105', 'CS Teaching Building', 'Lecture Room 1', 80,
9       'South Campus II'),
10     ('10', '100', 'Language Teaching Building', 'Teaching Room A', 30,
11      'South Campus 1'),
12     ('10', '102', 'Language Teaching Building', 'Teaching Room B', 30,
13      'South Campus 1'),
14     ('53', '904a', 'Comprehensive Experimental Building',
15      'Office 1', 10, 'South Campus 2'),
16     ('53', '904b', 'Comprehensive Experimental Building',
17      'Cluster Room', 3, 'South Campus 2'),
18     ('7', '200', 'Main Teaching Building', 'Auditorium', 120,
19      'South Campus 1'),
20     ('36', '106', 'CS Teaching Building', 'Lecture Room 2', 80,
21      'South Campus 2');
```

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

# Daten Ändern (1)



- Wir haben definitiv mehr als einen Raum für Building 36 gespeichert.
- Schauen wir uns nochmal unser `INSERT`-Statement von vorhin an.
- Ah. Wir haben manchmal `'South Campus II'` anstatt von `'South Campus 2'` geschrieben.
- Für einen Computer sind das natürlich zwei verschiedene Dinge.
- Dadurch, dass wir die selben Daten wieder und wieder eingeben müssen, werden solche Fehler wahrscheinlicher.

```
1  /** Insert data into the database. */
2
3  -- Insert several room records.
4  INSERT INTO building_room (building_number, room_number, building_name,
5                             room_name, capacity, address) VALUES
6      ('36', '305', 'CS Teaching Building', 'Meeting Room', 40,
7       'South Campus II'),
8      ('36', '105', 'CS Teaching Building', 'Lecture Room 1', 80,
9       'South Campus II'),
10     ('10', '100', 'Language Teaching Building', 'Teaching Room A', 30,
11      'South Campus 1'),
12     ('10', '102', 'Language Teaching Building', 'Teaching Room B', 30,
13      'South Campus 1'),
14     ('53', '904a', 'Comprehensive Experimental Building',
15      'Office 1', 10, 'South Campus 2'),
16     ('53', '904b', 'Comprehensive Experimental Building',
17      'Cluster Room', 3, 'South Campus 2'),
18     ('7', '200', 'Main Teaching Building', 'Auditorium', 120,
19      'South Campus 1'),
20     ('36', '106', 'CS Teaching Building', 'Lecture Room 2', 80,
21      'South Campus 2');
```

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

# Daten Ändern (1)



- Schauen wir uns nochmal unser `INSERT`-Statement von vorhin an.
- Ah. Wir haben manchmal `'South Campus II'` anstatt von `'South Campus 2'` geschrieben.
- Für einen Computer sind das natürlich zwei verschiedene Dinge.
- Dadurch, dass wir die selben Daten wieder und wieder eingeben müssen, werden solche Fehler wahrscheinlicher.
- Immer wenn jemand Daten eingeben muss gibt es eine gewisse Wahrscheinlichkeit, dass die Person einen Fehler macht.

```
1  /** Insert data into the database. */
2
3  -- Insert several room records.
4  INSERT INTO building_room (building_number, room_number, building_name,
5                             room_name, capacity, address) VALUES
6      ('36', '305', 'CS Teaching Building', 'Meeting Room', 40,
7       'South Campus II'),
8      ('36', '105', 'CS Teaching Building', 'Lecture Room 1', 80,
9       'South Campus II'),
10     ('10', '100', 'Language Teaching Building', 'Teaching Room A', 30,
11      'South Campus 1'),
12     ('10', '102', 'Language Teaching Building', 'Teaching Room B', 30,
13      'South Campus 1'),
14     ('53', '904a', 'Comprehensive Experimental Building',
15      'Office 1', 10, 'South Campus 2'),
16     ('53', '904b', 'Comprehensive Experimental Building',
17      'Cluster Room', 3, 'South Campus 2'),
18     ('7', '200', 'Main Teaching Building', 'Auditorium', 120,
19      'South Campus 1'),
20     ('36', '106', 'CS Teaching Building', 'Lecture Room 2', 80,
21      'South Campus 2');
```

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

# Daten Ändern (1)



- Ah. Wir haben manchmal `'South Campus II'` anstatt von `'South Campus 2'` geschrieben.
- Für einen Computer sind das natürlich zwei verschiedene Dinge.
- Dadurch, dass wir die selben Daten wieder und wieder eingeben müssen, werden solche Fehler wahrscheinlicher.
- Immer wenn jemand Daten eingeben muss gibt es eine gewisse Wahrscheinlichkeit, dass die Person einen Fehler macht.
- Je öfter wir Daten eingeben müssen, desto größer die Chance, Fehler zu machen.

```
1  /** Insert data into the database. */
2
3  -- Insert several room records.
4  INSERT INTO building_room (building_number, room_number, building_name,
5                             room_name, capacity, address) VALUES
6      ('36', '305', 'CS Teaching Building', 'Meeting Room', 40,
7       'South Campus II'),
8      ('36', '105', 'CS Teaching Building', 'Lecture Room 1', 80,
9       'South Campus II'),
10     ('10', '100', 'Language Teaching Building', 'Teaching Room A', 30,
11      'South Campus 1'),
12     ('10', '102', 'Language Teaching Building', 'Teaching Room B', 30,
13      'South Campus 1'),
14     ('53', '904a', 'Comprehensive Experimental Building',
15      'Office 1', 10, 'South Campus 2'),
16     ('53', '904b', 'Comprehensive Experimental Building',
17      'Cluster Room', 3, 'South Campus 2'),
18     ('7', '200', 'Main Teaching Building', 'Auditorium', 120,
19      'South Campus 1'),
20     ('36', '106', 'CS Teaching Building', 'Lecture Room 2', 80,
21      'South Campus 2');
```

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

# Daten Ändern (1)



- Für einen Computer sind das natürlich zwei verschiedene Dinge.
- Dadurch, dass wir die selben Daten wieder und wieder eingeben müssen, werden solche Fehler wahrscheinlicher.
- Immer wenn jemand Daten eingeben muss gibt es eine gewisse Wahrscheinlichkeit, dass die Person einen Fehler macht.
- Je öfter wir Daten eingeben müssen, desto größer die Chance, Fehler zu machen.
- Die Verletzung der 2NF zwingt uns, die selben Informationen öfter einzugeben.

```
1  /** Insert data into the database. */
2
3  -- Insert several room records.
4  INSERT INTO building_room (building_number, room_number, building_name,
5                             room_name, capacity, address) VALUES
6      ('36', '305', 'CS Teaching Building', 'Meeting Room', 40,
7       'South Campus II'),
8      ('36', '105', 'CS Teaching Building', 'Lecture Room 1', 80,
9       'South Campus II'),
10     ('10', '100', 'Language Teaching Building', 'Teaching Room A', 30,
11      'South Campus 1'),
12     ('10', '102', 'Language Teaching Building', 'Teaching Room B', 30,
13      'South Campus 1'),
14     ('53', '904a', 'Comprehensive Experimental Building',
15      'Office 1', 10, 'South Campus 2'),
16     ('53', '904b', 'Comprehensive Experimental Building',
17      'Cluster Room', 3, 'South Campus 2'),
18     ('7', '200', 'Main Teaching Building', 'Auditorium', 120,
19      'South Campus 1'),
20     ('36', '106', 'CS Teaching Building', 'Lecture Room 2', 80,
21      'South Campus 2');
```

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

# Daten Ändern (1)



- Dadurch, dass wir die selben Daten wieder und wieder eingeben müssen, werden solche Fehler wahrscheinlicher.
- Immer wenn jemand Daten eingeben muss gibt es eine gewisse Wahrscheinlichkeit, dass die Person einen Fehler macht.
- Je öfter wir Daten eingeben müssen, desto größer die Chance, Fehler zu machen.
- Die Verletzung der 2NF zwingt uns, die selben Informationen öfter einzugeben.
- Dadurch erhöht sie die Chance, Fehler zu machen.

```
1  /** Insert data into the database. */
2
3  -- Insert several room records.
4  INSERT INTO building_room (building_number, room_number, building_name,
5                             room_name, capacity, address) VALUES
6      ('36', '305', 'CS Teaching Building', 'Meeting Room', 40,
7       'South Campus II'),
8      ('36', '105', 'CS Teaching Building', 'Lecture Room 1', 80,
9       'South Campus II'),
10     ('10', '100', 'Language Teaching Building', 'Teaching Room A', 30,
11      'South Campus 1'),
12     ('10', '102', 'Language Teaching Building', 'Teaching Room B', 30,
13      'South Campus 1'),
14     ('53', '904a', 'Comprehensive Experimental Building',
15      'Office 1', 10, 'South Campus 2'),
16     ('53', '904b', 'Comprehensive Experimental Building',
17      'Cluster Room', 3, 'South Campus 2'),
18     ('7', '200', 'Main Teaching Building', 'Auditorium', 120,
19      'South Campus 1'),
20     ('36', '106', 'CS Teaching Building', 'Lecture Room 2', 80,
21      'South Campus 2');
```

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

# Daten Ändern (1)



- Immer wenn jemand Daten eingeben muss gibt es eine gewisse Wahrscheinlichkeit, dass die Person einen Fehler macht.
- Je öfter wir Daten eingeben müssen, desto größer die Chance, Fehler zu machen.
- Die Verletzung der 2NF zwingt uns, die selben Informationen öfter einzugeben.
- Dadurch erhöht sie die Chance, Fehler zu machen.
- Die Verletzung der 2NF hat den Fehler nicht ausgelöst, aber ihn wahrscheinlicher gemacht.

```
1  /** Insert data into the database. */
2
3  -- Insert several room records.
4  INSERT INTO building_room (building_number, room_number, building_name,
5                             room_name, capacity, address) VALUES
6
7      ('36', '305', 'CS Teaching Building', 'Meeting Room', 40,
8       'South Campus II'),
9      ('36', '105', 'CS Teaching Building', 'Lecture Room 1', 80,
10      'South Campus II'),
11      ('10', '100', 'Language Teaching Building', 'Teaching Room A', 30,
12      'South Campus 1'),
13      ('10', '102', 'Language Teaching Building', 'Teaching Room B', 30,
14      'South Campus 1'),
15      ('53', '904a', 'Comprehensive Experimental Building',
16       'Office 1', 10, 'South Campus 2'),
17      ('53', '904b', 'Comprehensive Experimental Building',
18       'Cluster Room', 3, 'South Campus 2'),
19      ('7', '200', 'Main Teaching Building', 'Auditorium', 120,
20      'South Campus 1'),
21      ('36', '106', 'CS Teaching Building', 'Lecture Room 2', 80,
22      'South Campus 2');

```

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

```

# Daten Ändern (1)



- Je öfter wir Daten eingeben müssen, desto größer die Chance, Fehler zu machen.
- Die Verletzung der 2NF zwingt uns, die selben Informationen öfter einzugeben.
- Dadurch erhöht sie die Chance, Fehler zu machen.
- Die Verletzung der 2NF hat den Fehler nicht ausgelöst, aber ihn wahrscheinlicher gemacht.
- Wir können das Problem auf zwei Arten lösen.

```
1  /** Insert data into the database. */
2
3  -- Insert several room records.
4  INSERT INTO building_room (building_number, room_number, building_name,
5                             room_name, capacity, address) VALUES
6      ('36', '305', 'CS Teaching Building', 'Meeting Room', 40,
7       'South Campus II'),
8      ('36', '105', 'CS Teaching Building', 'Lecture Room 1', 80,
9       'South Campus II'),
10     ('10', '100', 'Language Teaching Building', 'Teaching Room A', 30,
11      'South Campus 1'),
12     ('10', '102', 'Language Teaching Building', 'Teaching Room B', 30,
13      'South Campus 1'),
14     ('53', '904a', 'Comprehensive Experimental Building',
15      'Office 1', 10, 'South Campus 2'),
16     ('53', '904b', 'Comprehensive Experimental Building',
17      'Cluster Room', 3, 'South Campus 2'),
18     ('7', '200', 'Main Teaching Building', 'Auditorium', 120,
19      'South Campus 1'),
20     ('36', '106', 'CS Teaching Building', 'Lecture Room 2', 80,
21      'South Campus 2');
```

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

# Daten Ändern (1)



- Die Verletzung der 2NF zwingt uns, die selben Informationen öfter einzugeben.
- Dadurch erhöht sie die Chance, Fehler zu machen.
- Die Verletzung der 2NF hat den Fehler nicht ausgelöst, aber ihn wahrscheinlicher gemacht.
- Wir können das Problem auf zwei Arten lösen.
- Wir können die Zeilen auswählen, für die `address = 'South Campus 2'` oder `address = 'South Campus II'` gilt.

```
1  /** Find all the rooms available in South Campus 2. */
2
3  -- Several rooms are missing, because the data is inconsistent:
4  -- Their address was 'South Campus II'.
5  SELECT building_number, room_number FROM building_room
6         WHERE address = 'South Campus 2';
7
8  -- Now we get these rooms.
9  SELECT building_number, room_number FROM building_room
10         WHERE address = 'South Campus 2' OR address = 'South Campus II';
```

```
1  $ psql "postgres://postgres:XXX@localhost/violation" -v ON_ERROR_STOP=1
2      ↪ -ebf select_a_2.sql
3  building_number | room_number
4  -----+-----
5  36              | 106
6  (1 row)
7
7  building_number | room_number
8  -----+-----
9  36              | 305
10 36              | 105
11 36              | 106
12 (3 rows)
13
14 # psql 16.11 succeeded with exit code 0.
```

# Daten Ändern (1)

- Dadurch erhöht sie die Chance, Fehler zu machen.
- Die Verletzung der 2NF hat den Fehler nicht ausgelöst, aber ihn wahrscheinlicher gemacht.
- Wir können das Problem auf zwei Arten lösen.
- Wir können die Zeilen auswählen, für die `address = 'South Campus 2'` oder `address = 'South Campus II'` gilt.
- Oder wir reparieren die falschen Daten mit einem `UPDATE`.

```
1  /** Find all the rooms available in South Campus 2. */
2
3  -- Several rooms are missing, because the data is inconsistent:
4  -- Their address was 'South Campus II'.
5  SELECT building_number, room_number FROM building_room
6         WHERE address = 'South Campus 2';
7
8  -- Now we get these rooms.
9  SELECT building_number, room_number FROM building_room
10         WHERE address = 'South Campus 2' OR address = 'South Campus II';
11
12 -- We can fix this problem with an UPDATE instruction.
13 UPDATE building_room SET address = 'South Campus 2'
14        WHERE address = 'South Campus II' RETURNING building_number;
15
16 -- This query now also works, because the addresses are now consistent.
17 SELECT building_number, room_number FROM building_room
18        WHERE address = 'South Campus 2';
```

```
1  $ psql "postgres://postgres:XXX@localhost/violation" -v ON_ERROR_STOP=1
   ↪ -ebf update_a.sql
2  building_number | room_number
3  -----+-----
4  36              | 106
5  (1 row)
6
7  building_number | room_number
8  -----+-----
9  36              | 305
10 36              | 105
11 36              | 106
12 (3 rows)
13
14 building_number
15 -----
16 36
17 36
18 (2 rows)
19
20 UPDATE 2
21 building_number | room_number
22 -----+-----
23 36              | 106
24 36              | 305
25 36              | 105
26 (3 rows)
27
28 # psql 16.11 succeeded with exit code 0.
```



# Daten Ändern (1)

- Die Verletzung der 2NF hat den Fehler nicht ausgelöst, aber ihn wahrscheinlicher gemacht.
- Wir können das Problem auf zwei Arten lösen.
- Wir können die Zeilen auswählen, für die `address = 'South Campus 2'` oder `address = 'South Campus II'` gilt.
- Oder wir reparieren die falschen Daten mit einem `UPDATE`.
- Wir würden die Gebäudeadresse auf `'South Campus 2'` setzen, wenn ein Gebäude in `'South Campus II'` liegt.

```
1  /** Find all the rooms available in South Campus 2. */
2
3  -- Several rooms are missing, because the data is inconsistent:
4  -- Their address was 'South Campus II'.
5  SELECT building_number, room_number FROM building_room
6         WHERE address = 'South Campus 2';
7
8  -- Now we get these rooms.
9  SELECT building_number, room_number FROM building_room
10         WHERE address = 'South Campus 2' OR address = 'South Campus II';
11
12 -- We can fix this problem with an UPDATE instruction.
13 UPDATE building_room SET address = 'South Campus 2'
14        WHERE address = 'South Campus II' RETURNING building_number;
15
16 -- This query now also works, because the addresses are now consistent.
17 SELECT building_number, room_number FROM building_room
18        WHERE address = 'South Campus 2';
```

```
1  $ psql "postgres://postgres:XXX@localhost/violation" -v ON_ERROR_STOP=1
2      ↪ -ebf update_a.sql
3  building_number | room_number
4  -----+-----
5  36              | 106
6  (1 row)
7
8  building_number | room_number
9  -----+-----
10 36              | 305
1136              | 105
1236              | 106
13(3 rows)
14
15 building_number
16 -----
1736
1836
19(2 rows)
20
21 UPDATE 2
22 building_number | room_number
23 -----+-----
2436              | 106
2536              | 305
2636              | 105
27(3 rows)
28
29 # psql 16.11 succeeded with exit code 0.
```



# Daten Ändern (1)

- Wir können das Problem auf zwei Arten lösen.
- Wir können die Zeilen auswählen, für die `address = 'South Campus 2'` oder `address = 'South Campus II'` gilt.
- Oder wir reparieren die falschen Daten mit einem `UPDATE`.
- Wir würden die Gebäudeadresse auf `'South Campus 2'` setzen, wenn ein Gebäude in `'South Campus II'` liegt.
- Natürlich schreiben wir auch wieder ein `RETURNING`-Statement<sup>52</sup>, um die veränderten Zeilen anzuzeigen.

```
1  /** Find all the rooms available in South Campus 2. */
2
3  -- Several rooms are missing, because the data is inconsistent:
4  -- Their address was 'South Campus II'.
5  SELECT building_number, room_number FROM building_room
6         WHERE address = 'South Campus 2';
7
8  -- Now we get these rooms.
9  SELECT building_number, room_number FROM building_room
10         WHERE address = 'South Campus 2' OR address = 'South Campus II';
11
12 -- We can fix this problem with an UPDATE instruction.
13 UPDATE building_room SET address = 'South Campus 2'
14        WHERE address = 'South Campus II' RETURNING building_number;
15
16 -- This query now also works, because the addresses are now consistent.
17 SELECT building_number, room_number FROM building_room
18        WHERE address = 'South Campus 2';
```

```
1  $ psql "postgres://postgres:XXX@localhost/violation" -v ON_ERROR_STOP=1
2  ↪ -ebf update_a.sql
3  building_number | room_number
4  -----+-----
5  36               | 106
6  (1 row)
7
8  building_number | room_number
9  -----+-----
10 36               | 305
1136               | 105
1236               | 106
13(3 rows)
14
15 building_number
16 -----
1736
1836
19(2 rows)
20
21 UPDATE 2
22 building_number | room_number
23 -----+-----
2436               | 106
2536               | 305
2636               | 105
27(3 rows)
28
29 # psql 16.11 succeeded with exit code 0.
```



# Daten Ändern (1)

- Wir können die Zeilen auswählen, für die `address = 'South Campus 2'` oder `address = 'South Campus II'` gilt.
- Oder wir reparieren die falschen Daten mit einem `UPDATE`.
- Wir würden die Gebäudeadresse auf `'South Campus 2'` setzen, wenn ein Gebäude in `'South Campus II'` liegt.
- Natürlich schreiben wir auch wieder ein `RETURNING`-Statement<sup>52</sup>, um die veränderten Zeilen anzuzeigen.
- Nun funktioniert auch die erste Anfrage richtig.

```
1  /** Find all the rooms available in South Campus 2. */
2
3  -- Several rooms are missing, because the data is inconsistent:
4  -- Their address was 'South Campus II'.
5  SELECT building_number, room_number FROM building_room
6         WHERE address = 'South Campus 2';
7
8  -- Now we get these rooms.
9  SELECT building_number, room_number FROM building_room
10         WHERE address = 'South Campus 2' OR address = 'South Campus II';
11
12 -- We can fix this problem with an UPDATE instruction.
13 UPDATE building_room SET address = 'South Campus 2'
14         WHERE address = 'South Campus II' RETURNING building_number;
15
16 -- This query now also works, because the addresses are now consistent.
17 SELECT building_number, room_number FROM building_room
18         WHERE address = 'South Campus 2';
```

```
1  $ psql "postgres://postgres:XXX@localhost/violation" -v ON_ERROR_STOP=1
   ↳ -ebf update_a.sql
2  building_number | room_number
3  -----+-----
4  36              | 106
5  (1 row)
6
7  building_number | room_number
8  -----+-----
9  36              | 305
10 36              | 105
11 36              | 106
12 (3 rows)
13
14 building_number
15 -----
16 36
17 36
18 (2 rows)
19
20 UPDATE 2
21 building_number | room_number
22 -----+-----
23 36              | 106
24 36              | 305
25 36              | 105
26 (3 rows)
27
28 # psql 16.11 succeeded with exit code 0.
```

# Daten Ändern (2)



- Ändern wir nun den Namen des Building 36 von „CS Teaching Building“ zu „Computer Science Building“.

```
1  /** Change the name of Building 36 to Computer Science Building. */
2
3  UPDATE building_room SET building_name = 'Computer Science Building'
4      WHERE address = 'South Campus 2' AND building_number = '36'
5      RETURNING building_number, building_name, room_number;
```

```
1  $ psql "postgres://postgres:XXX@localhost/violation" -v ON_ERROR_STOP=1
   ↪ -ebf update_b.sql
2  building_number |      building_name      | room_number
3  -----+-----+-----
4  36              | Computer Science Building | 105
5  36              | Computer Science Building | 106
6  36              | Computer Science Building | 305
7  (3 rows)
8
9  UPDATE 3
10 # psql 16.11 succeeded with exit code 0.
```

# Daten Ändern (2)



- Ändern wir nun den Namen des Building 36 von „CS Teaching Building“ zu „Computer Science Building“.
- Das kann mit der Anfrage rechts gemacht werden.

```
1  /** Change the name of Building 36 to Computer Science Building. */
2
3  UPDATE building_room SET building_name = 'Computer Science Building'
4     WHERE address = 'South Campus 2' AND building_number = '36'
5     RETURNING building_number, building_name, room_number;
```

```
1  $ psql "postgres://postgres:XXX@localhost/violation" -v ON_ERROR_STOP=1
   ↪ -ebf update_b.sql
2  building_number |      building_name      | room_number
3  -----+-----+-----
4  36              | Computer Science Building | 105
5  36              | Computer Science Building | 106
6  36              | Computer Science Building | 305
7  (3 rows)
8
9  UPDATE 3
10 # psql 16.11 succeeded with exit code 0.
```

# Daten Ändern (2)



- Ändern wir nun den Namen des Building 36 von „CS Teaching Building“ zu „Computer Science Building“.
- Das kann mit der Anfrage rechts gemacht werden.
- Wir benutzen ein `UPDATE`-Statement auf die Tabelle `building_room`.

```
1  /** Change the name of Building 36 to Computer Science Building. */
2
3  UPDATE building_room SET building_name = 'Computer Science Building'
4      WHERE address = 'South Campus 2' AND building_number = '36'
5      RETURNING building_number, building_name, room_number;
```

```
1  $ psql "postgres://postgres:XXX@localhost/violation" -v ON_ERROR_STOP=1
   ↪ -ebf update_b.sql
2  building_number | building_name | room_number
3  -----+-----+-----
4  36              | Computer Science Building | 105
5  36              | Computer Science Building | 106
6  36              | Computer Science Building | 305
7  (3 rows)
8
9  UPDATE 3
10 # psql 16.11 succeeded with exit code 0.
```

# Daten Ändern (2)



- Ändern wir nun den Namen des Building 36 von „CS Teaching Building“ zu „Computer Science Building“.
- Das kann mit der Anfrage rechts gemacht werden.
- Wir benutzen ein `UPDATE`-Statement auf die Tabelle `building_room`.
- Dieses Mal wollen wir alle Zeilen bearbeiten, bei denen `address = 'South Campus 2'` und `building_number = '36'` gilt.

```
1  /** Change the name of Building 36 to Computer Science Building. */
2
3  UPDATE building_room SET building_name = 'Computer Science Building'
4      WHERE address = 'South Campus 2' AND building_number = '36'
5      RETURNING building_number, building_name, room_number;
```

```
1  $ psql "postgres://postgres:XXX@localhost/violation" -v ON_ERROR_STOP=1
   ↪ -ebf update_b.sql
2  building_number | building_name | room_number
3  -----+-----+-----
4  36              | Computer Science Building | 105
5  36              | Computer Science Building | 106
6  36              | Computer Science Building | 305
7  (3 rows)
8
9  UPDATE 3
10 # psql 16.11 succeeded with exit code 0.
```

# Daten Ändern (2)



- Ändern wir nun den Namen des Building 36 von „CS Teaching Building“ zu „Computer Science Building“.
- Das kann mit der Anfrage rechts gemacht werden.
- Wir benutzen ein `UPDATE`-Statement auf die Tabelle `building_room`.
- Dieses Mal wollen wir alle Zeilen bearbeiten, bei denen `address = 'South Campus 2'` und `building_number = '36'` gilt.
- Für alle diese Zeilen setzen wir `building_name = 'Computer Science Building'`.

```
1  /** Change the name of Building 36 to Computer Science Building. */
2
3  UPDATE building_room SET building_name = 'Computer Science Building'
4      WHERE address = 'South Campus 2' AND building_number = '36'
5      RETURNING building_number, building_name, room_number;
```

```
1  $ psql "postgres://postgres:XXX@localhost/violation" -v ON_ERROR_STOP=1
   ↪ -ebf update_b.sql
2  building_number | building_name | room_number
3  -----+-----+-----
4  36              | Computer Science Building | 105
5  36              | Computer Science Building | 106
6  36              | Computer Science Building | 305
7  (3 rows)
8
9  UPDATE 3
10 # psql 16.11 succeeded with exit code 0.
```

# Daten Ändern (2)



- Das kann mit der Anfrage rechts gemacht werden.
- Wir benutzen ein `UPDATE`-Statement auf die Tabelle `building_room`.
- Dieses Mal wollen wir alle Zeilen bearbeiten, bei denen `address = 'South Campus 2'` und `building_number = '36'` gilt.
- Für alle diese Zeilen setzen wir `building_name = 'Computer Science Building'`.
- Wir benutzen wieder ein `RETURNING`-Statement<sup>52</sup>, um die geänderten Zeilen auszugeben.

```
1  /** Change the name of Building 36 to Computer Science Building. */
2
3  UPDATE building_room SET building_name = 'Computer Science Building'
4     WHERE address = 'South Campus 2' AND building_number = '36'
5     RETURNING building_number, building_name, room_number;
```

```
1  $ psql "postgres://postgres:XXX@localhost/violation" -v ON_ERROR_STOP=1
   ↪ -ebf update_b.sql
2
3  building_number | building_name          | room_number
4  -----+-----+-----
5  36              | Computer Science Building | 105
6  36              | Computer Science Building | 106
7  36              | Computer Science Building | 305
8  (3 rows)
9
10 UPDATE 3
10 # psql 16.11 succeeded with exit code 0.
```

## Daten Ändern (2)



- Wir benutzen ein `UPDATE`-Statement auf die Tabelle `building_room`.
- Dieses Mal wollen wir alle Zeilen bearbeiten, bei denen `address = 'South Campus 2'` und `building_number = '36'` gilt.
- Für alle diese Zeilen setzen wir `building_name = 'Computer Science Building'`.
- Wir benutzen wieder ein `RETURNING`-Statement<sup>52</sup>, um die geänderten Zeilen auszugeben.
- Wie Sie sehen, sind das drei Zeilen.

```
1  /** Change the name of Building 36 to Computer Science Building. */
2
3  UPDATE building_room SET building_name = 'Computer Science Building'
4      WHERE address = 'South Campus 2' AND building_number = '36'
5      RETURNING building_number, building_name, room_number;
```

```
1  $ psql "postgres://postgres:XXX@localhost/violation" -v ON_ERROR_STOP=1
   ↪ -ebf update_b.sql
2
3  building_number | building_name          | room_number
4  -----+-----+-----
5  36              | Computer Science Building | 105
6  36              | Computer Science Building | 106
7  36              | Computer Science Building | 305
8  (3 rows)
9
10 UPDATE 3
# psql 16.11 succeeded with exit code 0.
```

# Daten Ändern (2)



- Dieses Mal wollen wir alle Zeilen bearbeiten, bei denen  
`address = 'South Campus 2'` und  
`building_number = '36'` gilt.
- Für alle diese Zeilen setzen wir  
`building_name =`  
`'Computer Science Building'`.
- Wir benutzen wieder ein `RETURNING`-Statement<sup>52</sup>, um die geänderten Zeilen auszugeben.
- Wie Sie sehen, sind das drei Zeilen.
- Beachten Sie aber, dass wir nur **ein einziges** Datum geändert haben.

```
1  /** Change the name of Building 36 to Computer Science Building. */
2
3  UPDATE building_room SET building_name = 'Computer Science Building'
4     WHERE address = 'South Campus 2' AND building_number = '36'
5     RETURNING building_number, building_name, room_number;
```

```
1  $ psql "postgres://postgres:XXX@localhost/violation" -v ON_ERROR_STOP=1
   ↪ -ebf update_b.sql
2  building_number | building_name | room_number
3  -----+-----+-----
4  36              | Computer Science Building | 105
5  36              | Computer Science Building | 106
6  36              | Computer Science Building | 305
7  (3 rows)
8
9  UPDATE 3
10 # psql 16.11 succeeded with exit code 0.
```

# Daten Ändern (2)



- Für alle diese Zeilen setzen wir `building_name = 'Computer Science Building'`.
- Wir benutzen wieder ein `RETURNING`-Statement<sup>52</sup>, um die geänderten Zeilen auszugeben.
- Wie Sie sehen, sind das drei Zeilen.
- Beachten Sie aber, dass wir nur **ein** **einziges** Datum geändert haben.
- Aber wir mussten **drei** Zeilen verändern.

```
1  /** Change the name of Building 36 to Computer Science Building. */
2
3  UPDATE building_room SET building_name = 'Computer Science Building'
4     WHERE address = 'South Campus 2' AND building_number = '36'
5     RETURNING building_number, building_name, room_number;
```

```
1  $ psql "postgres://postgres:XXX@localhost/violation" -v ON_ERROR_STOP=1
   ↪ -ebf update_b.sql
2  building_number | building_name          | room_number
3  -----+-----+-----
4  36              | Computer Science Building | 105
5  36              | Computer Science Building | 106
6  36              | Computer Science Building | 305
7  (3 rows)
8
9  UPDATE 3
10 # psql 16.11 succeeded with exit code 0.
```

## Daten Ändern (2)

- Wir benutzen wieder ein `RETURNING`-Statement<sup>52</sup>, um die geänderten Zeilen auszugeben.
- Wie Sie sehen, sind das drei Zeilen.
- Beachten Sie aber, dass wir nur **ein einziges** Datum geändert haben.
- Aber wir mussten **drei** Zeilen verändern.

```
1  /** Change the name of Building 36 to Computer Science Building. */
2
3  UPDATE building_room SET building_name = 'Computer Science Building'
4      WHERE address = 'South Campus 2' AND building_number = '36'
5      RETURNING building_number, building_name, room_number;
```

```
1  $ psql "postgres://postgres:XXX@localhost/violation" -v ON_ERROR_STOP=1
   ↪ -ebf update_b.sql
2  building_number |          building_name          | room_number
3  -----+-----+-----
4  36              | Computer Science Building | 105
5  36              | Computer Science Building | 106
6  36              | Computer Science Building | 305
7  (3 rows)
8
9  UPDATE 3
10 # psql 16.11 succeeded with exit code 0.
```

### Definition: Update-Anomalie

Eine Situation, wo das Verändern von einer Information das Ändern von mehreren Zeilen erfordert nennt man *Update-Anomalie* (EN: *update anomaly*).

## Daten Ändern (2)



- Beachten Sie aber, dass wir nur **ein einziges** Datum geändert haben.
- Aber wir mussten **drei** Zeilen verändern.
- Diese Update-Anomalie wird direkt durch die Verletzung der 2NF ausgelöst.

```
1  /** Change the name of Building 36 to Computer Science Building. */
2
3  UPDATE building_room SET building_name = 'Computer Science Building'
4     WHERE address = 'South Campus 2' AND building_number = '36'
5     RETURNING building_number, building_name, room_number;
```

```
1  $ psql "postgres://postgres:XXX@localhost/violation" -v ON_ERROR_STOP=1
   ↪ -ebf update_b.sql
2  building_number | building_name          | room_number
3  -----+-----+-----
4  36              | Computer Science Building | 105
5  36              | Computer Science Building | 106
6  36              | Computer Science Building | 305
7  (3 rows)
8
9  UPDATE 3
10 # psql 16.11 succeeded with exit code 0.
```

## Daten Ändern (2)



- Beachten Sie aber, dass wir nur **ein einziges** Datum geändert haben.
- Aber wir mussten **drei** Zeilen verändern.
- Diese Update-Anomalie wird direkt durch die Verletzung der 2NF ausgelöst.
- Das selbe gilt für eine weitere Anomalie:

```
1  /** Change the name of Building 36 to Computer Science Building. */
2
3  UPDATE building_room SET building_name = 'Computer Science Building'
4     WHERE address = 'South Campus 2' AND building_number = '36'
5     RETURNING building_number, building_name, room_number;
```

```
1  $ psql "postgres://postgres:XXX@localhost/violation" -v ON_ERROR_STOP=1
   ↪ -ebf update_b.sql
2
3  building_number | building_name          | room_number
4  -----+-----+-----
5  36              | Computer Science Building | 105
6  36              | Computer Science Building | 106
7  36              | Computer Science Building | 305
8  (3 rows)
9
10 UPDATE 3
# psql 16.11 succeeded with exit code 0.
```

## Daten Ändern (2)

- Beachten Sie aber, dass wir nur **ein einziges** Datum geändert haben.
- Aber wir mussten **drei** Zeilen verändern.
- Diese Update-Anomalie wird direkt durch die Verletzung der 2NF ausgelöst.
- Das selbe gilt für eine weitere Anomalie:

```
1  /** Change the name of Building 36 to Computer Science Building. */
2
3  UPDATE building_room SET building_name = 'Computer Science Building'
4      WHERE address = 'South Campus 2' AND building_number = '36'
5      RETURNING building_number, building_name, room_number;

1  $ psql "postgres://postgres:XXX@localhost/violation" -v ON_ERROR_STOP=1
   ↪ -ebf update_b.sql
2  building_number |          building_name          | room_number
3  -----+-----+-----
4  36              | Computer Science Building | 105
5  36              | Computer Science Building | 106
6  36              | Computer Science Building | 305
7  (3 rows)
8
9  UPDATE 3
10 # psql 16.11 succeeded with exit code 0.
```

### Definition: Einfüge-Anomalie

Eine Situation wo das Einfügen von Daten in die Datenbank unmöglich wird, weil andere Daten noch nicht da sind, nennt man *Einfüge-Anomalie* (EN: *insertion anomaly*)<sup>59</sup>.

## Daten Ändern (2)



- Aber wir mussten **drei** Zeilen verändern.
- Diese Update-Anomalie wird direkt durch die Verletzung der 2NF ausgelöst.
- Das selbe gilt für eine weitere Anomalie:
- Z. B. ist es nicht möglich, die Information über einen Raum einzufügen, ohne Informationen über ein Gebäude einzufügen (und andersherum).

```
1  /** Change the name of Building 36 to Computer Science Building. */
2
3  UPDATE building_room SET building_name = 'Computer Science Building'
4     WHERE address = 'South Campus 2' AND building_number = '36'
5     RETURNING building_number, building_name, room_number;
```

```
1  $ psql "postgres://postgres:XXX@localhost/violation" -v ON_ERROR_STOP=1
   ↪ -ebf update_b.sql
2  building_number | building_name | room_number
3  -----+-----+-----
4  36              | Computer Science Building | 105
5  36              | Computer Science Building | 106
6  36              | Computer Science Building | 305
7  (3 rows)
8
9  UPDATE 3
10 # psql 16.11 succeeded with exit code 0.
```

## Daten Ändern (2)



- Diese Update-Anomalie wird direkt durch die Verletzung der 2NF ausgelöst.
- Das selbe gilt für eine weitere Anomalie:
- Z. B. ist es nicht möglich, die Information über einen Raum einzufügen, ohne Informationen über ein Gebäude einzufügen (und andersherum).
- In unserem Speziellfall hier ist das sowieso nicht möglich und nicht sinnvoll.

```
1  /** Change the name of Building 36 to Computer Science Building. */
2
3  UPDATE building_room SET building_name = 'Computer Science Building'
4     WHERE address = 'South Campus 2' AND building_number = '36'
5     RETURNING building_number, building_name, room_number;
```

```
1  $ psql "postgres://postgres:XXX@localhost/violation" -v ON_ERROR_STOP=1
   ↪ -ebf update_b.sql
2  building_number | building_name | room_number
3  -----+-----+-----
4  36              | Computer Science Building | 105
5  36              | Computer Science Building | 106
6  36              | Computer Science Building | 305
7  (3 rows)
8
9  UPDATE 3
10 # psql 16.11 succeeded with exit code 0.
```

# Daten Ändern (2)



- Das selbe gilt für eine weitere Anomalie:
- Z. B. ist es nicht möglich, die Information über einen Raum einzufügen, ohne Informationen über ein Gebäude einzufügen (und andersherum).
- In unserem Speziellfall hier ist das sowieso nicht möglich und nicht sinnvoll.
- Aber es kann viele andere Situationen geben, wo die Verletzung der 2NF dieses als Problem auslöst.

```
1  /** Change the name of Building 36 to Computer Science Building. */
2
3  UPDATE building_room SET building_name = 'Computer Science Building'
4     WHERE address = 'South Campus 2' AND building_number = '36'
5     RETURNING building_number, building_name, room_number;
```

```
1  $ psql "postgres://postgres:XXX@localhost/violation" -v ON_ERROR_STOP=1
   ↪ -ebf update_b.sql
2  building_number | building_name          | room_number
3  -----+-----+-----
4  36              | Computer Science Building | 105
5  36              | Computer Science Building | 106
6  36              | Computer Science Building | 305
7  (3 rows)
8
9  UPDATE 3
10 # psql 16.11 succeeded with exit code 0.
```

## Daten Ändern (2)



- Z. B. ist es nicht möglich, die Information über einen Raum einzufügen, ohne Informationen über ein Gebäude einzufügen (und andersherum).
- In unserem Speziellfall hier ist das sowieso nicht möglich und nicht sinnvoll.
- Aber es kann viele andere Situationen geben, wo die Verletzung der 2NF dieses als Problem auslöst.
- In [37] z. B., wird ein interessanter Fall aus dem Gebiet der Warenhaltung vorgestellt.

```
1  /** Change the name of Building 36 to Computer Science Building. */
2
3  UPDATE building_room SET building_name = 'Computer Science Building'
4      WHERE address = 'South Campus 2' AND building_number = '36'
5      RETURNING building_number, building_name, room_number;
```

```
1  $ psql "postgres://postgres:XXX@localhost/violation" -v ON_ERROR_STOP=1
   ↪ -ebf update_b.sql
2
3  building_number | building_name | room_number
4  -----+-----+-----
5  36              | Computer Science Building | 105
6  36              | Computer Science Building | 106
7  36              | Computer Science Building | 305
8  (3 rows)
9
10 UPDATE 3
10 # psql 16.11 succeeded with exit code 0.
```

## Daten Ändern (2)



- In unserem Speziellfall hier ist das sowieso nicht möglich und nicht sinnvoll.
- Aber es kann viele andere Situationen geben, wo die Verletzung der 2NF dieses als Problem auslöst.
- In [37] z. B., wird ein interessanter Fall aus dem Gebiet der Warenhaltung vorgestellt.
- Die Namen und Adressen von Lagerhäusern werden zusammen mit den Namen und Mengen der Produkte in ihnen *in einer Tabelle* gespeichert.

```
1  /** Change the name of Building 36 to Computer Science Building. */
2
3  UPDATE building_room SET building_name = 'Computer Science Building'
4     WHERE address = 'South Campus 2' AND building_number = '36'
5     RETURNING building_number, building_name, room_number;
```

```
1  $ psql "postgres://postgres:XXX@localhost/violation" -v ON_ERROR_STOP=1
   ↪ -ebf update_b.sql
2  building_number | building_name          | room_number
3  -----+-----+-----
4  36              | Computer Science Building | 105
5  36              | Computer Science Building | 106
6  36              | Computer Science Building | 305
7  (3 rows)
8
9  UPDATE 3
10 # psql 16.11 succeeded with exit code 0.
```

# Daten Ändern (2)



- In unserem Speziellfall hier ist das sowieso nicht möglich und nicht sinnvoll.
- Aber es kann viele andere Situationen geben, wo die Verletzung der 2NF dieses als Problem auslöst.
- In [37] z. B., wird ein interessanter Fall aus dem Gebiet der Warenhaltung vorgestellt.
- Die Namen und Adressen von Lagerhäusern werden zusammen mit den Namen und Mengen der Produkte in ihnen *in einer Tabelle* gespeichert.
- Dieses Design einer einzelnen Tabelle verletzt die 2NF.

```
1  /** Change the name of Building 36 to Computer Science Building. */
2
3  UPDATE building_room SET building_name = 'Computer Science Building'
4     WHERE address = 'South Campus 2' AND building_number = '36'
5     RETURNING building_number, building_name, room_number;
```

```
1  $ psql "postgres://postgres:XXX@localhost/violation" -v ON_ERROR_STOP=1
   ↪ -ebf update_b.sql
2  building_number |      building_name      | room_number
3  -----+-----+-----
4  36              | Computer Science Building | 105
5  36              | Computer Science Building | 106
6  36              | Computer Science Building | 305
7  (3 rows)
8
9  UPDATE 3
10 # psql 16.11 succeeded with exit code 0.
```

## Daten Ändern (2)



- In [37] z. B., wird ein interessanter Fall aus dem Gebiet der Warenhaltung vorgestellt.
- Die Namen und Adressen von Lagerhäusern werden zusammen mit den Namen und Mengen der Produkte in ihnen *in einer Tabelle* gespeichert.
- Dieses Design einer einzelnen Tabelle verletzt die 2NF.
- Die Daten waren genau in vier Spalten gespeichert (Name des Lagerhauses, Adresse des Lagerhauses, Name des Produkts, Menge des Produkts).

```
1  /** Change the name of Building 36 to Computer Science Building. */
2
3  UPDATE building_room SET building_name = 'Computer Science Building'
4     WHERE address = 'South Campus 2' AND building_number = '36'
5     RETURNING building_number, building_name, room_number;
```

```
1  $ psql "postgres://postgres:XXX@localhost/violation" -v ON_ERROR_STOP=1
   ↪ -ebf update_b.sql
2  building_number | building_name | room_number
3  -----+-----+-----
4  36              | Computer Science Building | 105
5  36              | Computer Science Building | 106
6  36              | Computer Science Building | 305
7  (3 rows)
8
9  UPDATE 3
10 # psql 16.11 succeeded with exit code 0.
```

# Daten Ändern (2)



- Die Namen und Adressen von Lagerhäusern werden zusammen mit den Namen und Mengen der Produkte in ihnen *in einer Tabelle* gespeichert.
- Dieses Design einer einzelnen Tabelle verletzt die 2NF.
- Die Daten waren genau in vier Spalten gespeichert (Name des Lagerhauses, Adresse des Lagerhauses, Name des Produkts, Menge des Produkts).
- Dadurch ist es nicht möglich, einen Datensatz für ein Lagerhaus zu erstellen, in dem noch keine Produkte gespeichert sind.

```
1  /** Change the name of Building 36 to Computer Science Building. */
2
3  UPDATE building_room SET building_name = 'Computer Science Building'
4      WHERE address = 'South Campus 2' AND building_number = '36'
5      RETURNING building_number, building_name, room_number;
```

```
1  $ psql "postgres://postgres:XXX@localhost/violation" -v ON_ERROR_STOP=1
   ↪ -ebf update_b.sql
2
3  building_number | building_name          | room_number
4  -----+-----+-----
5  36              | Computer Science Building | 105
6  36              | Computer Science Building | 106
7  36              | Computer Science Building | 305
8  (3 rows)
9
10 UPDATE 3
# psql 16.11 succeeded with exit code 0.
```

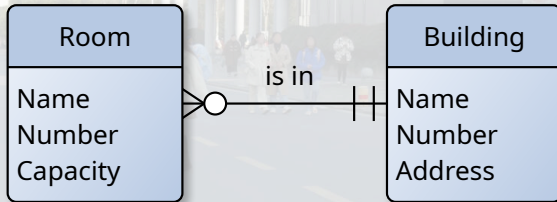


# Lösung: Wiederherstellung der 2. Normalform



# Lösung

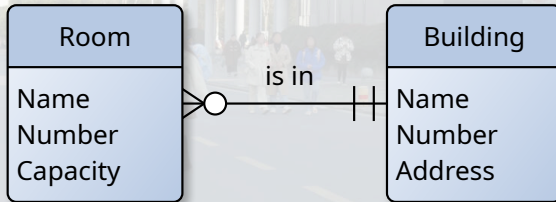
- Die Verletzung der 2NF kann – direkt oder indirekt – mehrere Probleme und Anomalien auslösen.



| public.building_room     |                     |        |
|--------------------------|---------------------|--------|
| 🔑 <b>building_number</b> | <b>varchar(4)</b>   | « pk » |
| 🔑 <b>room_number</b>     | <b>varchar(4)</b>   | « pk » |
| 🕒 <b>building_name</b>   | <b>varchar(100)</b> | « nn » |
| 🕒 <b>room_name</b>       | <b>varchar(100)</b> | « nn » |
| 🕒 <b>capacity</b>        | <b>integer</b>      | « nn » |
| 🕒 <b>address</b>         | <b>varchar(100)</b> | « nn » |
| 🏠 <b>br_pk</b>           | <b>constraint</b>   | « pk » |

# Lösung

- Die Verletzung der 2NF kann – direkt oder indirekt – mehrere Probleme und Anomalien auslösen.
- Normalisieren wir also die Datenbank von vornhin nach der 2NF.

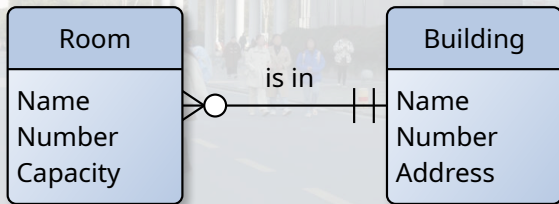


| public.building_room     |                     |        |
|--------------------------|---------------------|--------|
| 🔑 <b>building_number</b> | <b>varchar(4)</b>   | « pk » |
| 🔑 <b>room_number</b>     | <b>varchar(4)</b>   | « pk » |
| 🔵 <b>building_name</b>   | <b>varchar(100)</b> | « nn » |
| 🔵 <b>room_name</b>       | <b>varchar(100)</b> | « nn » |
| 🔵 <b>capacity</b>        | <b>integer</b>      | « nn » |
| 🔵 <b>address</b>         | <b>varchar(100)</b> | « nn » |
| 🔷 <b>br_pk</b>           | <b>constraint</b>   | « pk » |

# Lösung



- Die Verletzung der 2NF kann – direkt oder indirekt – mehrere Probleme und Anomalien auslösen.
- Normalisieren wir also die Datenbank von vorn nach der 2NF.
- In unserem ursprünglichen Design hatten wir eine einzige Tabelle: `building_room`.

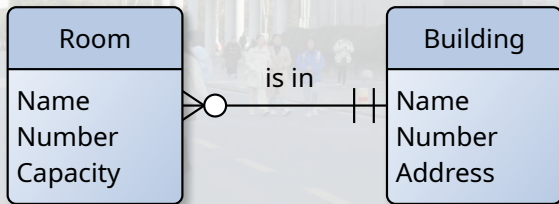


| public.building_room   |                     |        |
|------------------------|---------------------|--------|
| <b>building_number</b> | <b>varchar(4)</b>   | « pk » |
| <b>room_number</b>     | <b>varchar(4)</b>   | « pk » |
| <b>building_name</b>   | <b>varchar(100)</b> | « nn » |
| <b>room_name</b>       | <b>varchar(100)</b> | « nn » |
| <b>capacity</b>        | <b>integer</b>      | « nn » |
| <b>address</b>         | <b>varchar(100)</b> | « nn » |
| <b>br_pk</b>           | <b>constraint</b>   | « pk » |

# Lösung



- Die Verletzung der 2NF kann – direkt oder indirekt – mehrere Probleme und Anomalien auslösen.
- Normalisieren wir also die Datenbank von vorn nach der 2NF.
- In unserem ursprünglichen Design hatten wir eine einzige Tabelle: `building_room`.
- Diese Tabelle hatte einen zusammengesetzten Primärschlüssel bestehend aus den Spalten `building_number` und `room_number`.

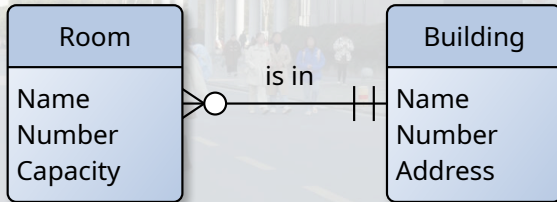


| public.building_room     |                     |        |
|--------------------------|---------------------|--------|
| 🔑 <b>building_number</b> | <b>varchar(4)</b>   | « pk » |
| 🔑 <b>room_number</b>     | <b>varchar(4)</b>   | « pk » |
| ⦿ <b>building_name</b>   | <b>varchar(100)</b> | « nn » |
| ⦿ <b>room_name</b>       | <b>varchar(100)</b> | « nn » |
| ⦿ <b>capacity</b>        | <b>integer</b>      | « nn » |
| ⦿ <b>address</b>         | <b>varchar(100)</b> | « nn » |
| 🔑 <b>br_pk</b>           | <b>constraint</b>   | « pk » |

# Lösung



- Die Verletzung der 2NF kann – direkt oder indirekt – mehrere Probleme und Anomalien auslösen.
- Normalisieren wir also die Datenbank von vorn nach der 2NF.
- In unserem ursprünglichen Design hatten wir eine einzige Tabelle: `building_room`.
- Diese Tabelle hatte einen zusammengesetzten Primärschlüssel bestehend aus den Spalten `building_number` und `room_number`.
- Die Spalten `building_name` und `address` hängen aber nur von `building_number` ab.

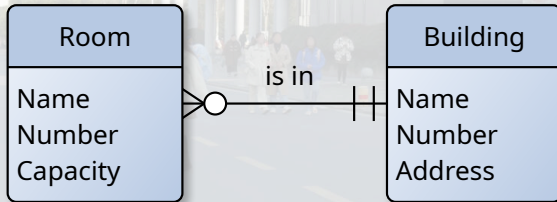


| public.building_room                                                                                       |                     |        |
|------------------------------------------------------------------------------------------------------------|---------------------|--------|
|  <b>building_number</b> | <b>varchar(4)</b>   | « pk » |
|  <b>room_number</b>     | <b>varchar(4)</b>   | « pk » |
|  <b>building_name</b>   | <b>varchar(100)</b> | « nn » |
|  <b>room_name</b>       | <b>varchar(100)</b> | « nn » |
|  <b>capacity</b>        | <b>integer</b>      | « nn » |
|  <b>address</b>         | <b>varchar(100)</b> | « nn » |
|  <b>br_pk</b>           | <b>constraint</b>   | « pk » |

# Lösung



- Normalisieren wir also die Datenbank von vorn nach der 2NF.
- In unserem ursprünglichen Design hatten wir eine einzige Tabelle: `building_room`.
- Diese Tabelle hatte einen zusammengesetzten Primärschlüssel bestehend aus den Spalten `building_number` und `room_number`.
- Die Spalten `building_name` und `address` hängen aber nur von `building_number` ab.
- Das führte dann zu den Anomalien, zu Redundanz und erhöhte die Wahrscheinlichkeit, Fehler zu machen.

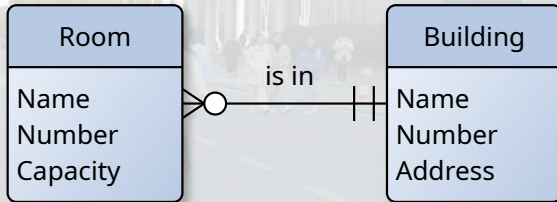


| public.building_room           |                           |        |
|--------------------------------|---------------------------|--------|
| 🔑 <code>building_number</code> | <code>varchar(4)</code>   | « pk » |
| 🔑 <code>room_number</code>     | <code>varchar(4)</code>   | « pk » |
| 🔵 <code>building_name</code>   | <code>varchar(100)</code> | « nn » |
| 🔵 <code>room_name</code>       | <code>varchar(100)</code> | « nn » |
| 🔵 <code>capacity</code>        | <code>integer</code>      | « nn » |
| 🔵 <code>address</code>         | <code>varchar(100)</code> | « nn » |
| 🔷 <code>br_pk</code>           | <code>constraint</code>   | « pk » |

# Lösung



- In unserem ursprünglichen Design hatten wir eine einzige Tabelle: `building_room`.
- Diese Tabelle hatte einen zusammengesetzten Primärschlüssel bestehend aus den Spalten `building_number` und `room_number`.
- Die Spalten `building_name` und `address` hängen aber nur von `building_number` ab.
- Das führte dann zu den Anomalien, zu Redundanz und erhöhte die Wahrscheinlichkeit, Fehler zu machen.
- Um Normalisierung unter der 2NF zu erreichen, müssen wir die Spalten `building_name` und `address` in eine eigene Tabelle tun.



| public.building_room                                                                                       |                     |        |
|------------------------------------------------------------------------------------------------------------|---------------------|--------|
|  <b>building_number</b> | <b>varchar(4)</b>   | « pk » |
|  <b>room_number</b>     | <b>varchar(4)</b>   | « pk » |
|  <b>building_name</b>   | <b>varchar(100)</b> | « nn » |
|  <b>room_name</b>       | <b>varchar(100)</b> | « nn » |
|  <b>capacity</b>        | <b>integer</b>      | « nn » |
|  <b>address</b>         | <b>varchar(100)</b> | « nn » |
|  <b>br_pk</b>           | <b>constraint</b>   | « pk » |

# Lösung



- Diese Tabelle hatte einen zusammengesetzten Primärschlüssel bestehend aus den Spalten `building_number` und `room_number`.
- Die Spalten `building_name` und `address` hängen aber nur von `building_number` ab.
- Das führte dann zu den Anomalien, zu Redundanz und erhöhte die Wahrscheinlichkeit, Fehler zu machen.
- Um Normalisierung unter der 2NF zu erreichen, müssen wir die Spalten `building_name` und `address` in eine eigene Tabelle tun.
- Wir nennen diese Tabelle `building`.



| public.building                                                                   |                              |                                  |
|-----------------------------------------------------------------------------------|------------------------------|----------------------------------|
|  | <code>building_number</code> | <code>varchar(4)</code> « pk »   |
|  | <code>building_name</code>   | <code>varchar(100)</code> « nn » |
|  | <code>address</code>         | <code>varchar(100)</code> « nn » |
|  | <code>building_pk</code>     | <code>constraint</code> « pk »   |

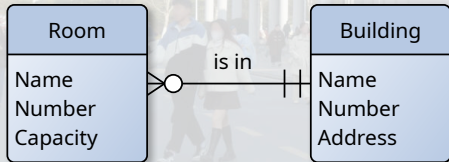
is\_in\_building

| public.room                                                                         |                              |                                   |
|-------------------------------------------------------------------------------------|------------------------------|-----------------------------------|
|  | <code>building_number</code> | <code>varchar(4)</code> « pk fk » |
|  | <code>room_number</code>     | <code>varchar(4)</code> « pk »    |
|  | <code>room_name</code>       | <code>varchar(100)</code> « nn »  |
|  | <code>capacity</code>        | <code>integer</code> « nn »       |
|  | <code>room_pk</code>         | <code>constraint</code> « pk »    |
|  | <code>building_fk</code>     | <code>constraint</code> « fk »    |

# Lösung



- Die Spalten `building_name` und `address` hängen aber nur von `building_number` ab.
- Das führte dann zu den Anomalien, zu Redundanz und erhöhte die Wahrscheinlichkeit, Fehler zu machen.
- Um Normalisierung unter der 2NF zu erreichen, müssen wir die Spalten `building_name` und `address` in eine eigene Tabelle tun.
- Wir nennen diese Tabelle `building`.
- Natürlich braucht diese einen Primärschlüssel.



| public.building              |                           |        |
|------------------------------|---------------------------|--------|
| <code>building_number</code> | <code>varchar(4)</code>   | « pk » |
| <code>building_name</code>   | <code>varchar(100)</code> | « nn » |
| <code>address</code>         | <code>varchar(100)</code> | « nn » |
| <code>building_pk</code>     | <code>constraint</code>   | « pk » |

is\_in\_building

| public.room                  |                           |           |
|------------------------------|---------------------------|-----------|
| <code>building_number</code> | <code>varchar(4)</code>   | « pk fk » |
| <code>room_number</code>     | <code>varchar(4)</code>   | « pk »    |
| <code>room_name</code>       | <code>varchar(100)</code> | « nn »    |
| <code>capacity</code>        | <code>integer</code>      | « nn »    |
| <code>room_pk</code>         | <code>constraint</code>   | « pk »    |
| <code>building_fk</code>     | <code>constraint</code>   | « fk »    |

# Lösung



- Das führte dann zu den Anomalien, zu Redundanz und erhöhte die Wahrscheinlichkeit, Fehler zu machen.
- Um Normalisierung unter der 2NF zu erreichen, müssen wir die Spalten `building_name` und `address` in eine eigene Tabelle tun.
- Wir nennen diese Tabelle `building`.
- Natürlich braucht diese einen Primärschlüssel.
- Dafür nehmen wir `building_number`.



| public.building |                              |                           |        |
|-----------------|------------------------------|---------------------------|--------|
|                 | <code>building_number</code> | <code>varchar(4)</code>   | « pk » |
|                 | <code>building_name</code>   | <code>varchar(100)</code> | « nn » |
|                 | <code>address</code>         | <code>varchar(100)</code> | « nn » |
|                 | <code>building_pk</code>     | <code>constraint</code>   | « pk » |

is\_in\_building

| public.room |                              |                           |           |
|-------------|------------------------------|---------------------------|-----------|
|             | <code>building_number</code> | <code>varchar(4)</code>   | « pk fk » |
|             | <code>room_number</code>     | <code>varchar(4)</code>   | « pk »    |
|             | <code>room_name</code>       | <code>varchar(100)</code> | « nn »    |
|             | <code>capacity</code>        | <code>integer</code>      | « nn »    |
|             | <code>room_pk</code>         | <code>constraint</code>   | « pk »    |
|             | <code>building_fk</code>     | <code>constraint</code>   | « fk »    |

# Lösung



- Um Normalisierung unter der 2NF zu erreichen, müssen wir die Spalten `building_name` und `address` in eine eigene Tabelle tun.
- Wir nennen diese Tabelle `building`.
- Natürlich braucht diese einen Primärschlüssel.
- Dafür nehmen wir `building_number`.
- Wir können `building_number` nehmen, denn in unserer Universität gibt es die Regel, dass keine zwei Gebäude die selbe Nummer haben dürfen.



| public.building |                              |                           |        |
|-----------------|------------------------------|---------------------------|--------|
|                 | <code>building_number</code> | <code>varchar(4)</code>   | « pk » |
|                 | <code>building_name</code>   | <code>varchar(100)</code> | « nn » |
|                 | <code>address</code>         | <code>varchar(100)</code> | « nn » |
|                 | <code>building_pk</code>     | <code>constraint</code>   | « pk » |

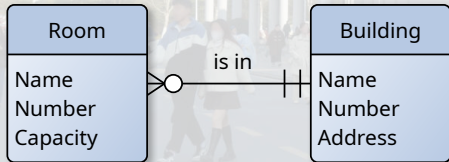
is\_in\_building

| public.room |                              |                           |           |
|-------------|------------------------------|---------------------------|-----------|
|             | <code>building_number</code> | <code>varchar(4)</code>   | « pk fk » |
|             | <code>room_number</code>     | <code>varchar(4)</code>   | « pk »    |
|             | <code>room_name</code>       | <code>varchar(100)</code> | « nn »    |
|             | <code>capacity</code>        | <code>integer</code>      | « nn »    |
|             | <code>room_pk</code>         | <code>constraint</code>   | « pk »    |
|             | <code>building_fk</code>     | <code>constraint</code>   | « fk »    |

# Lösung



- Wir nennen diese Tabelle `building`.
- Natürlich braucht diese einen Primärschlüssel.
- Dafür nehmen wir `building_number`.
- Wir können `building_number` nehmen, denn in unserer Universität gibt es die Regel, dass keine zwei Gebäude die selbe Nummer haben dürfen.
- Dann benennen wir die Tabelle `building_room` zu `room` um.



| public.building |                              |                           |        |
|-----------------|------------------------------|---------------------------|--------|
|                 | <code>building_number</code> | <code>varchar(4)</code>   | « pk » |
|                 | <code>building_name</code>   | <code>varchar(100)</code> | « nn » |
|                 | <code>address</code>         | <code>varchar(100)</code> | « nn » |
|                 | <code>building_pk</code>     | <code>constraint</code>   | « pk » |

is\_in\_building

| public.room |                              |                           |           |
|-------------|------------------------------|---------------------------|-----------|
|             | <code>building_number</code> | <code>varchar(4)</code>   | « pk fk » |
|             | <code>room_number</code>     | <code>varchar(4)</code>   | « pk »    |
|             | <code>room_name</code>       | <code>varchar(100)</code> | « nn »    |
|             | <code>capacity</code>        | <code>integer</code>      | « nn »    |
|             | <code>room_pk</code>         | <code>constraint</code>   | « pk »    |
|             | <code>building_fk</code>     | <code>constraint</code>   | « fk »    |

# Lösung



- Natürlich braucht diese einen Primärschlüssel.
- Dafür nehmen wir `building_number`.
- Wir können `building_number` nehmen, denn in unserer Universität gibt es die Regel, dass keine zwei Gebäude die selbe Nummer haben dürfen.
- Dann benennen wir die Tabelle `building_room` zu `room` um.
- Jede Zeile repräsentiert nun einen einzelnen Raum in einem Gebäude.



| public.building |                              |                           |        |
|-----------------|------------------------------|---------------------------|--------|
|                 | <code>building_number</code> | <code>varchar(4)</code>   | « pk » |
|                 | <code>building_name</code>   | <code>varchar(100)</code> | « nn » |
|                 | <code>address</code>         | <code>varchar(100)</code> | « nn » |
|                 | <code>building_pk</code>     | <code>constraint</code>   | « pk » |

is\_in\_building

| public.room |                              |                           |           |
|-------------|------------------------------|---------------------------|-----------|
|             | <code>building_number</code> | <code>varchar(4)</code>   | « pk fk » |
|             | <code>room_number</code>     | <code>varchar(4)</code>   | « pk »    |
|             | <code>room_name</code>       | <code>varchar(100)</code> | « nn »    |
|             | <code>capacity</code>        | <code>integer</code>      | « nn »    |
|             | <code>room_pk</code>         | <code>constraint</code>   | « pk »    |
|             | <code>building_fk</code>     | <code>constraint</code>   | « fk »    |

# Lösung



- Dafür nehmen wir `building_number`.
- Wir können `building_number` nehmen, denn in unserer Universität gibt es die Regel, dass keine zwei Gebäude die selbe Nummer haben dürfen.
- Dann benennen wir die Tabelle `building_room` zu `room` um.
- Jede Zeile repräsentiert nun einen einzelnen Raum in einem Gebäude.
- Diese Tabelle hat immer noch den zusammengesetzten Primärschlüssel aus den Spalten `building_number` und `room_number`.



| public.building |                              |                           |        |
|-----------------|------------------------------|---------------------------|--------|
|                 | <code>building_number</code> | <code>varchar(4)</code>   | « pk » |
|                 | <code>building_name</code>   | <code>varchar(100)</code> | « nn » |
|                 | <code>address</code>         | <code>varchar(100)</code> | « nn » |
|                 | <code>building_pk</code>     | <code>constraint</code>   | « pk » |

is\_in\_building

| public.room |                              |                           |           |
|-------------|------------------------------|---------------------------|-----------|
|             | <code>building_number</code> | <code>varchar(4)</code>   | « pk fk » |
|             | <code>room_number</code>     | <code>varchar(4)</code>   | « pk »    |
|             | <code>room_name</code>       | <code>varchar(100)</code> | « nn »    |
|             | <code>capacity</code>        | <code>integer</code>      | « nn »    |
|             | <code>room_pk</code>         | <code>constraint</code>   | « pk »    |
|             | <code>building_fk</code>     | <code>constraint</code>   | « fk »    |

# Lösung



- Wir können `building_number` nehmen, denn in unserer Universität gibt es die Regel, dass keine zwei Gebäude die selbe Nummer haben dürfen.
- Dann benennen wir die Tabelle `building_room` zu `room` um.
- Jede Zeile repräsentiert nun einen einzelnen Raum in einem Gebäude.
- Diese Tabelle hat immer noch den zusammengesetzten Primärschlüssel aus den Spalten `building_number` und `room_number`.
- Es gibt keinen Grund, das zu ändern.



| public.building |                              |                           |        |
|-----------------|------------------------------|---------------------------|--------|
|                 | <code>building_number</code> | <code>varchar(4)</code>   | « pk » |
|                 | <code>building_name</code>   | <code>varchar(100)</code> | « nn » |
|                 | <code>address</code>         | <code>varchar(100)</code> | « nn » |
|                 | <code>building_pk</code>     | <code>constraint</code>   | « pk » |

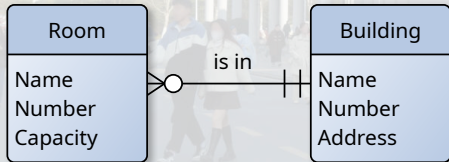
is\_in\_building

| public.room |                              |                           |           |
|-------------|------------------------------|---------------------------|-----------|
|             | <code>building_number</code> | <code>varchar(4)</code>   | « pk fk » |
|             | <code>room_number</code>     | <code>varchar(4)</code>   | « pk »    |
|             | <code>room_name</code>       | <code>varchar(100)</code> | « nn »    |
|             | <code>capacity</code>        | <code>integer</code>      | « nn »    |
|             | <code>room_pk</code>         | <code>constraint</code>   | « pk »    |
|             | <code>building_fk</code>     | <code>constraint</code>   | « fk »    |

# Lösung



- Dann benennen wir die Tabelle `building_room` zu `room` um.
- Jede Zeile repräsentiert nun einen einzelnen Raum in einem Gebäude.
- Diese Tabelle hat immer noch den zusammengesetzten Primärschlüssel aus den Spalten `building_number` und `room_number`.
- Es gibt keinen Grund, das zu ändern.
- Genau genommen ist das sogar besser, als einen Ersatzschlüssel zu verwenden.



| public.building |                              |                           |        |
|-----------------|------------------------------|---------------------------|--------|
|                 | <code>building_number</code> | <code>varchar(4)</code>   | « pk » |
|                 | <code>building_name</code>   | <code>varchar(100)</code> | « nn » |
|                 | <code>address</code>         | <code>varchar(100)</code> | « nn » |
|                 | <code>building_pk</code>     | <code>constraint</code>   | « pk » |

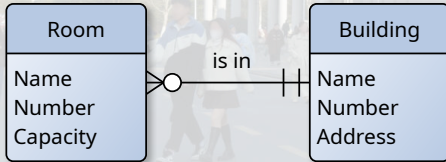
is\_in\_building

| public.room |                              |                           |           |
|-------------|------------------------------|---------------------------|-----------|
|             | <code>building_number</code> | <code>varchar(4)</code>   | « pk fk » |
|             | <code>room_number</code>     | <code>varchar(4)</code>   | « pk »    |
|             | <code>room_name</code>       | <code>varchar(100)</code> | « nn »    |
|             | <code>capacity</code>        | <code>integer</code>      | « nn »    |
|             | <code>room_pk</code>         | <code>constraint</code>   | « pk »    |
|             | <code>building_fk</code>     | <code>constraint</code>   | « fk »    |

# Lösung



- Jede Zeile repräsentiert nun einen einzelnen Raum in einem Gebäude.
- Diese Tabelle hat immer noch den zusammengesetzten Primärschlüssel aus den Spalten `building_number` und `room_number`.
- Es gibt keinen Grund, das zu ändern.
- Genau genommen ist das sogar besser, als einen Ersatzschlüssel zu verwenden.
- Es erzwingt, dass wir Räume nur speichern können *nachdem* wir die entsprechenden Gebäudedatensätze angelegt haben.



| public.building              |                           |        |
|------------------------------|---------------------------|--------|
| <code>building_number</code> | <code>varchar(4)</code>   | « pk » |
| <code>building_name</code>   | <code>varchar(100)</code> | « nn » |
| <code>address</code>         | <code>varchar(100)</code> | « nn » |
| <code>building_pk</code>     | <code>constraint</code>   | « pk » |

is\_in\_building

| public.room                  |                           |           |
|------------------------------|---------------------------|-----------|
| <code>building_number</code> | <code>varchar(4)</code>   | « pk fk » |
| <code>room_number</code>     | <code>varchar(4)</code>   | « pk »    |
| <code>room_name</code>       | <code>varchar(100)</code> | « nn »    |
| <code>capacity</code>        | <code>integer</code>      | « nn »    |
| <code>room_pk</code>         | <code>constraint</code>   | « pk »    |
| <code>building_fk</code>     | <code>constraint</code>   | « fk »    |

# Lösung



- Diese Tabelle hat immer noch den zusammengesetzten Primärschlüssel aus den Spalten `building_number` und `room_number`.
- Es gibt keinen Grund, das zu ändern.
- Genau genommen ist das sogar besser, als einen Ersatzschlüssel zu verwenden.
- Es erzwingt, dass wir Räume nur speichern können *nachdem* wir die entsprechenden Gebäudedatensätze angelegt haben.
- In der Tabelle gibt es noch die zwei Spalten `room_name` und `capacity`.



| public.building                                                                   |                              |                           |        |
|-----------------------------------------------------------------------------------|------------------------------|---------------------------|--------|
|  | <code>building_number</code> | <code>varchar(4)</code>   | « pk » |
|  | <code>building_name</code>   | <code>varchar(100)</code> | « nn » |
|  | <code>address</code>         | <code>varchar(100)</code> | « nn » |
|  | <code>building_pk</code>     | <code>constraint</code>   | « pk » |

is\_in\_building

| public.room                                                                         |                              |                           |           |
|-------------------------------------------------------------------------------------|------------------------------|---------------------------|-----------|
|  | <code>building_number</code> | <code>varchar(4)</code>   | « pk fk » |
|  | <code>room_number</code>     | <code>varchar(4)</code>   | « pk »    |
|  | <code>room_name</code>       | <code>varchar(100)</code> | « nn »    |
|  | <code>capacity</code>        | <code>integer</code>      | « nn »    |
|  | <code>room_pk</code>         | <code>constraint</code>   | « pk »    |
|  | <code>building_fk</code>     | <code>constraint</code>   | « fk »    |

# Lösung



- Es gibt keinen Grund, das zu ändern.
- Genau genommen ist das sogar besser, als einen Ersatzschlüssel zu verwenden.
- Es erzwingt, dass wir Räume nur speichern können *nachdem* wir die entsprechenden Gebäudedatensätze angelegt haben.
- In der Tabelle gibt es noch die zwei Spalten `room_name` und `capacity`.
- Diese beiden nicht-Schlüssel-Attribute sind funktional vom Primärschlüssel abhängig – und zwar vom *kompletten* Primärschlüssel.



| public.building |              |        |
|-----------------|--------------|--------|
| building_number | varchar(4)   | « pk » |
| building_name   | varchar(100) | « nn » |
| address         | varchar(100) | « nn » |
| building_pk     | constraint   | « pk » |

is\_in\_building

| public.room     |              |           |
|-----------------|--------------|-----------|
| building_number | varchar(4)   | « pk fk » |
| room_number     | varchar(4)   | « pk »    |
| room_name       | varchar(100) | « nn »    |
| capacity        | integer      | « nn »    |
| room_pk         | constraint   | « pk »    |
| building_fk     | constraint   | « fk »    |

# Lösung



- Genau genommen ist das sogar besser, als einen Ersatzschlüssel zu verwenden.
- Es erzwingt, dass wir Räume nur speichern können *nachdem* wir die entsprechenden Gebäudedatensätze angelegt haben.
- In der Tabelle gibt es noch die zwei Spalten `room_name` und `capacity`.
- Diese beiden nicht-Schlüssel-Attribute sind funktional vom Primärschlüssel abhängig – und zwar vom *kompletten* Primärschlüssel.
- Dieses logische Modell repräsentiert unser konzeptuelles Modell auch viel besser.



| public.building |                 |              |        |
|-----------------|-----------------|--------------|--------|
|                 | building_number | varchar(4)   | « pk » |
|                 | building_name   | varchar(100) | « nn » |
|                 | address         | varchar(100) | « nn » |
|                 | building_pk     | constraint   | « pk » |

is\_in\_building

| public.room |                 |              |           |
|-------------|-----------------|--------------|-----------|
|             | building_number | varchar(4)   | « pk fk » |
|             | room_number     | varchar(4)   | « pk »    |
|             | room_name       | varchar(100) | « nn »    |
|             | capacity        | integer      | « nn »    |
|             | room_pk         | constraint   | « pk »    |
|             | building_fk     | constraint   | « fk »    |

# Tabelle building



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

```
1 -- object: public.building | type: TABLE --
2 -- DROP TABLE IF EXISTS public.building CASCADE;
3 CREATE TABLE public.building (
4     building_number varchar(4) NOT NULL,
5     building_name varchar(100) NOT NULL,
6     address varchar(100) NOT NULL,
7     CONSTRAINT building_pk PRIMARY KEY (building_number)
8 );
9 -- ddl-end --
10 ALTER TABLE public.building OWNER TO postgres;
11 -- ddl-end --
```

```
1 $ psql "postgres://postgres:XXX@localhost/fixed" -v ON_ERROR_STOP=1 -bf
   ↳ 04_public_building_table_5080.sql
2 CREATE TABLE
3 ALTER TABLE
4 # psql 16.11 succeeded with exit code 0.
```

# Tabelle building



- Hier ist das auto-generierte Skript zum Erstellen der Tabelle `building`.
- Sie bekommt die Spalten `building_number`, `building_name` und `address`.

```
1  -- object: public.building | type: TABLE --
2  -- DROP TABLE IF EXISTS public.building CASCADE;
3  CREATE TABLE public.building (
4      building_number varchar(4) NOT NULL,
5      building_name varchar(100) NOT NULL,
6      address varchar(100) NOT NULL,
7      CONSTRAINT building_pk PRIMARY KEY (building_number)
8  );
9  -- ddl-end --
10 ALTER TABLE public.building OWNER TO postgres;
11 -- ddl-end --
```

```
1  $ psql "postgres://postgres:XXX@localhost/fixed" -v ON_ERROR_STOP=1 -ebf
   ↳ 04_public_building_table_5080.sql
2  CREATE TABLE
3  ALTER TABLE
4  # psql 16.11 succeeded with exit code 0.
```

# Tabelle building



- Hier ist das auto-generierte Skript zum Erstellen der Tabelle `building`.
- Sie bekommt die Spalten `building_number`, `building_name` und `address`.
- `building_number` ist der Primärschlüssel.

```
1  -- object: public.building | type: TABLE --
2  -- DROP TABLE IF EXISTS public.building CASCADE;
3  CREATE TABLE public.building (
4      building_number varchar(4) NOT NULL,
5      building_name varchar(100) NOT NULL,
6      address varchar(100) NOT NULL,
7      CONSTRAINT building_pk PRIMARY KEY (building_number)
8  );
9  -- ddl-end --
10 ALTER TABLE public.building OWNER TO postgres;
11 -- ddl-end --
```

```
1  $ psql "postgres://postgres:XXX@localhost/fixed" -v ON_ERROR_STOP=1 -ebf
   ↳ 04_public_building_table-5080.sql
2  CREATE TABLE
3  ALTER TABLE
4  # psql 16.11 succeeded with exit code 0.
```

# Tabelle building



- Hier ist das auto-generierte Skript zum Erstellen der Tabelle `building`.
- Sie bekommt die Spalten `building_number`, `building_name` und `address`.
- `building_number` ist der Primärschlüssel.
- Wir hätten noch eine `UNIQUE`-Einschränkung auf `building_name` machen können, um zu verhindern, dass zwei Gebäude den selben Namen bekommen können ... aber das lasse ich Ihnen als Übung.

```
1 -- object: public.building | type: TABLE --
2 -- DROP TABLE IF EXISTS public.building CASCADE;
3 CREATE TABLE public.building (
4     building_number varchar(4) NOT NULL,
5     building_name varchar(100) NOT NULL,
6     address varchar(100) NOT NULL,
7     CONSTRAINT building_pk PRIMARY KEY (building_number)
8 );
9 -- ddl-end --
10 ALTER TABLE public.building OWNER TO postgres;
11 -- ddl-end --
```

```
1 $ psql "postgres://postgres:XXX@localhost/fixed" -v ON_ERROR_STOP=1 -ebf
   ↳ 04_public_building_table_5080.sql
2 CREATE TABLE
3 ALTER TABLE
4 # psql 16.11 succeeded with exit code 0.
```

# Tabelle room



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

```
1  -- object: public.room | type: TABLE --
2  -- DROP TABLE IF EXISTS public.room CASCADE;
3  CREATE TABLE public.room (
4      building_number varchar(4) NOT NULL,
5      room_number varchar(4) NOT NULL,
6      room_name varchar(100) NOT NULL,
7      capacity integer NOT NULL,
8      CONSTRAINT room_pk PRIMARY KEY (building_number,room_number)
9  );
10 -- ddl-end --
11 ALTER TABLE public.room OWNER TO postgres;
12 -- ddl-end --
```

```
1  $ psql "postgres://postgres:XXX@localhost/fixed" -v ON_ERROR_STOP=1 -ebf
   ↳ 03_public_room_table_5071.sql
2  CREATE TABLE
3  ALTER TABLE
4  # psql 16.11 succeeded with exit code 0.
```

# Tabelle room



- Hier ist das auto-generierte Skript zum Erstellen der Tabelle `room`.
- Ihre ersten beiden Spalten, `building_number` und `room_number`, bilden den zusammengesetzten Primärschlüssel.

```
1  -- object: public.room | type: TABLE --
2  -- DROP TABLE IF EXISTS public.room CASCADE;
3  CREATE TABLE public.room (
4      building_number varchar(4) NOT NULL,
5      room_number varchar(4) NOT NULL,
6      room_name varchar(100) NOT NULL,
7      capacity integer NOT NULL,
8      CONSTRAINT room_pk PRIMARY KEY (building_number,room_number)
9  );
10 -- ddl-end --
11 ALTER TABLE public.room OWNER TO postgres;
12 -- ddl-end --
```

```
1  $ psql "postgres://postgres:XXX@localhost/fixed" -v ON_ERROR_STOP=1 -ebf
   ↳ 03_public_room_table_5071.sql
2  CREATE TABLE
3  ALTER TABLE
4  # psql 16.11 succeeded with exit code 0.
```

# Tabelle room



- Hier ist das auto-generierte Skript zum Erstellen der Tabelle `room`.
- Ihre ersten beiden Spalten, `building_number` und `room_number`, bilden den zusammengesetzten Primärschlüssel.
- Die anderen beiden Spalten (`room_name` und `capacity`) hängen funktional vom gesamten Schlüssel ab.

```
1  -- object: public.room | type: TABLE --
2  -- DROP TABLE IF EXISTS public.room CASCADE;
3  CREATE TABLE public.room (
4      building_number varchar(4) NOT NULL,
5      room_number varchar(4) NOT NULL,
6      room_name varchar(100) NOT NULL,
7      capacity integer NOT NULL,
8      CONSTRAINT room_pk PRIMARY KEY (building_number,room_number)
9  );
10 -- ddl-end --
11 ALTER TABLE public.room OWNER TO postgres;
12 -- ddl-end --
```

```
1  $ psql "postgres://postgres:XXX@localhost/fixed" -v ON_ERROR_STOP=1 -ebf
   ↳ 03_public_room_table_5071.sql
2  CREATE TABLE
3  ALTER TABLE
4  # psql 16.11 succeeded with exit code 0.
```

# Fremdschlüssel



- Hier ist das auto-generierte Skript zum Erstellen der Fremdschlüssel-Einschränkung, die die Zeilen in Tabelle `room` mit denen in Tabelle `building` verbindet.

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

```
1 $ psql "postgres://postgres:XXX@localhost/fixed" -v ON_ERROR_STOP=1 -ebf
  ↳ 05_public_room_building_fk_constraint_5099.sql
2 ALTER TABLE
3 # psql 16.11 succeeded with exit code 0.
```

# Fremdschlüssel



- Hier ist das auto-generierte Skript zum Erstellen der Fremdschlüssel-Einschränkung, die die Zeilen in Tabelle `room` mit denen in Tabelle `building` verbindet.
- Damit wird `building_number` erst zum Fremdschlüssel.

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

1  $ psql "postgres://postgres:XXX@localhost/fixed" -v ON_ERROR_STOP=1 -ebf
   ↳ 05_public_room_building_fk_constraint_5099.sql
2  ALTER TABLE
3  # psql 16.11 succeeded with exit code 0.
```

# Daten Einfügen



- Jetzt können wir genau die gleichen Daten wie vorhin einfügen.

```
1  /** Insert data into the database. */
2
3  -- Insert several room records.
4  INSERT INTO building (building_number, building_name, address) VALUES
5      ('36', 'CS Teaching Building', 'South Campus II'),
6      ('10', 'Language Teaching Building', 'South Campus 1'),
7      ('53', 'Comprehensive Experimental Building', 'South Campus 2'),
8      ('7', 'Main Teaching Building', 'South Campus 1');
9
10 -- Insert several room records.
11 INSERT INTO room (building_number, room_number, room_name,
12                  capacity) VALUES
13     ('36', '305', 'Meeting Room', 40),
14     ('36', '105', 'Lecture Room 1', 80),
15     ('10', '100', 'Teaching Room A', 30),
16     ('10', '102', 'Teaching Room B', 30),
17     ('53', '904a', 'Office 1', 10),
18     ('53', '904b', 'Cluster Room', 3),
19     ('7', '200', 'Auditorium', 120),
20     ('36', '106', 'Lecture Room 2', 80);
```

```
1  $ psql "postgres://postgres:XXX@localhost/fixed" -v ON_ERROR_STOP=1 -ebf
   ↪ insert.sql
2  INSERT 0 4
3  INSERT 0 8
4  # psql 16.11 succeeded with exit code 0.
```

# Daten Einfügen



- Jetzt können wir genau die gleichen Daten wie vorhin einfügen.
- Wir brauchen dafür jetzt zwei **INSERT INTO**-Statements.

```
1  /** Insert data into the database. */
2
3  -- Insert several room records.
4  INSERT INTO building (building_number, building_name, address) VALUES
5      ('36', 'CS Teaching Building', 'South Campus II'),
6      ('10', 'Language Teaching Building', 'South Campus 1'),
7      ('53', 'Comprehensive Experimental Building', 'South Campus 2'),
8      ('7', 'Main Teaching Building', 'South Campus 1');
9
10 -- Insert several room records.
11 INSERT INTO room (building_number, room_number, room_name,
12                  capacity) VALUES
13     ('36', '305', 'Meeting Room', 40),
14     ('36', '105', 'Lecture Room 1', 80),
15     ('10', '100', 'Teaching Room A', 30),
16     ('10', '102', 'Teaching Room B', 30),
17     ('53', '904a', 'Office 1', 10),
18     ('53', '904b', 'Cluster Room', 3),
19     ('7', '200', 'Auditorium', 120),
20     ('36', '106', 'Lecture Room 2', 80);
```

```
1  $ psql "postgres://postgres:XXX@localhost/fixed" -v ON_ERROR_STOP=1 -ebf
   ↪ insert.sql
2  INSERT 0 4
3  INSERT 0 8
4  # psql 16.11 succeeded with exit code 0.
```

# Daten Einfügen



- Jetzt können wir genau die gleichen Daten wie vorhin einfügen.
- Wir brauchen dafür jetzt zwei `INSERT INTO`-Statements.
- Zuerst speichern wir alle Gebäudedaten in Tabelle `building`.

```
1  /** Insert data into the database. */
2
3  -- Insert several room records.
4  INSERT INTO building (building_number, building_name, address) VALUES
5      ('36', 'CS Teaching Building', 'South Campus II'),
6      ('10', 'Language Teaching Building', 'South Campus 1'),
7      ('53', 'Comprehensive Experimental Building', 'South Campus 2'),
8      ('7', 'Main Teaching Building', 'South Campus 1');
9
10 -- Insert several room records.
11 INSERT INTO room (building_number, room_number, room_name,
12                  capacity) VALUES
13     ('36', '305', 'Meeting Room', 40),
14     ('36', '105', 'Lecture Room 1', 80),
15     ('10', '100', 'Teaching Room A', 30),
16     ('10', '102', 'Teaching Room B', 30),
17     ('53', '904a', 'Office 1', 10),
18     ('53', '904b', 'Cluster Room', 3),
19     ('7', '200', 'Auditorium', 120),
20     ('36', '106', 'Lecture Room 2', 80);
```

```
1  $ psql "postgres://postgres:XXX@localhost/fixed" -v ON_ERROR_STOP=1 -ebf
   ↪ insert.sql
2  INSERT 0 4
3  INSERT 0 8
4  # psql 16.11 succeeded with exit code 0.
```

# Daten Einfügen



- Jetzt können wir genau die gleichen Daten wie vorhin einfügen.
- Wir brauchen dafür jetzt zwei `INSERT INTO`-Statements.
- Zuerst speichern wir alle Gebäudedaten in Tabelle `building`.
- Dann speichern wir die Raumdaten in Tabelle `room`.

```
1  /** Insert data into the database. */
2
3  -- Insert several room records.
4  INSERT INTO building (building_number, building_name, address) VALUES
5      ('36', 'CS Teaching Building', 'South Campus II'),
6      ('10', 'Language Teaching Building', 'South Campus 1'),
7      ('53', 'Comprehensive Experimental Building', 'South Campus 2'),
8      ('7', 'Main Teaching Building', 'South Campus 1');
9
10 -- Insert several room records.
11 INSERT INTO room (building_number, room_number, room_name,
12                  capacity) VALUES
13     ('36', '305', 'Meeting Room', 40),
14     ('36', '105', 'Lecture Room 1', 80),
15     ('10', '100', 'Teaching Room A', 30),
16     ('10', '102', 'Teaching Room B', 30),
17     ('53', '904a', 'Office 1', 10),
18     ('53', '904b', 'Cluster Room', 3),
19     ('7', '200', 'Auditorium', 120),
20     ('36', '106', 'Lecture Room 2', 80);
```

```
1  $ psql "postgres://postgres:XXX@localhost/fixed" -v ON_ERROR_STOP=1 -ebf
   ↪ insert.sql
2  INSERT 0 4
3  INSERT 0 8
4  # psql 16.11 succeeded with exit code 0.
```

# Daten Einfügen



- Jetzt können wir genau die gleichen Daten wie vorhin einfügen.
- Wir brauchen dafür jetzt zwei `INSERT INTO`-Statements.
- Zuerst speichern wir alle Gebäudedaten in Tabelle `building`.
- Dann speichern wir die Raumdaten in Tabelle `room`.
- Wir sehen sofort, dass es jetzt viel weniger Redundanz gibt.

```
1  /** Insert data into the database. */
2
3  -- Insert several room records.
4  INSERT INTO building (building_number, building_name, address) VALUES
5      ('36', 'CS Teaching Building', 'South Campus II'),
6      ('10', 'Language Teaching Building', 'South Campus 1'),
7      ('53', 'Comprehensive Experimental Building', 'South Campus 2'),
8      ('7', 'Main Teaching Building', 'South Campus 1');
9
10 -- Insert several room records.
11 INSERT INTO room (building_number, room_number, room_name,
12                  capacity) VALUES
13     ('36', '305', 'Meeting Room', 40),
14     ('36', '105', 'Lecture Room 1', 80),
15     ('10', '100', 'Teaching Room A', 30),
16     ('10', '102', 'Teaching Room B', 30),
17     ('53', '904a', 'Office 1', 10),
18     ('53', '904b', 'Cluster Room', 3),
19     ('7', '200', 'Auditorium', 120),
20     ('36', '106', 'Lecture Room 2', 80);
```

```
1  $ psql "postgres://postgres:XXX@localhost/fixed" -v ON_ERROR_STOP=1 -ebf
   ↪ insert.sql
2  INSERT 0 4
3  INSERT 0 8
4  # psql 16.11 succeeded with exit code 0.
```

# Daten Einfügen



- Wir brauchen dafür jetzt zwei `INSERT INTO`-Statements.
- Zuerst speichern wir alle Gebäudedaten in Tabelle `building`.
- Dann speichern wir die Raumdaten in Tabelle `room`.
- Wir sehen sofort, dass es jetzt viel weniger Redundanz gibt.
- Die Daten über jedes Gebäude werden genau einmal gespeichert.

```
1  /** Insert data into the database. */
2
3  -- Insert several room records.
4  INSERT INTO building (building_number, building_name, address) VALUES
5      ('36', 'CS Teaching Building', 'South Campus II'),
6      ('10', 'Language Teaching Building', 'South Campus 1'),
7      ('53', 'Comprehensive Experimental Building', 'South Campus 2'),
8      ('7', 'Main Teaching Building', 'South Campus 1');
9
10 -- Insert several room records.
11 INSERT INTO room (building_number, room_number, room_name,
12                  capacity) VALUES
13     ('36', '305', 'Meeting Room', 40),
14     ('36', '105', 'Lecture Room 1', 80),
15     ('10', '100', 'Teaching Room A', 30),
16     ('10', '102', 'Teaching Room B', 30),
17     ('53', '904a', 'Office 1', 10),
18     ('53', '904b', 'Cluster Room', 3),
19     ('7', '200', 'Auditorium', 120),
20     ('36', '106', 'Lecture Room 2', 80);
```

```
1  $ psql "postgres://postgres:XXX@localhost/fixed" -v ON_ERROR_STOP=1 -ebf
   ↪ insert.sql
2  INSERT 0 4
3  INSERT 0 8
4  # psql 16.11 succeeded with exit code 0.
```

# Daten Einfügen



- Zuerst speichern wir alle Gebäudedaten in Tabelle `building`.
- Dann speichern wir die Raumdaten in Tabelle `room`.
- Wir sehen sofort, dass es jetzt viel weniger Redundanz gibt.
- Die Daten über jedes Gebäude werden genau einmal gespeichert.
- Vorher haben wir sie für jeden Raum speichern müssen.

```
1  /** Insert data into the database. */
2
3  -- Insert several room records.
4  INSERT INTO building (building_number, building_name, address) VALUES
5      ('36', 'CS Teaching Building', 'South Campus II'),
6      ('10', 'Language Teaching Building', 'South Campus 1'),
7      ('53', 'Comprehensive Experimental Building', 'South Campus 2'),
8      ('7', 'Main Teaching Building', 'South Campus 1');
9
10 -- Insert several room records.
11 INSERT INTO room (building_number, room_number, room_name,
12                  capacity) VALUES
13     ('36', '305', 'Meeting Room', 40),
14     ('36', '105', 'Lecture Room 1', 80),
15     ('10', '100', 'Teaching Room A', 30),
16     ('10', '102', 'Teaching Room B', 30),
17     ('53', '904a', 'Office 1', 10),
18     ('53', '904b', 'Cluster Room', 3),
19     ('7', '200', 'Auditorium', 120),
20     ('36', '106', 'Lecture Room 2', 80);
```

```
1  $ psql "postgres://postgres:XXX@localhost/fixed" -v ON_ERROR_STOP=1 -ebf
   ↪ insert.sql
2  INSERT 0 4
3  INSERT 0 8
4  # psql 16.11 succeeded with exit code 0.
```

# Daten Einfügen



- Dann speichern wir die Raumdaten in Tabelle `room`.
- Wir sehen sofort, dass es jetzt viel weniger Redundanz gibt.
- Die Daten über jedes Gebäude werden genau einmal gespeichert.
- Vorher haben wir sie für jeden Raum speichern müssen.
- Weil wir die Daten weniger oft schreiben müssen, wird auch die Wahrscheinlichkeit, Fehler zu machen, kleiner.

```
1  /** Insert data into the database. */
2
3  -- Insert several room records.
4  INSERT INTO building (building_number, building_name, address) VALUES
5      ('36', 'CS Teaching Building', 'South Campus II'),
6      ('10', 'Language Teaching Building', 'South Campus 1'),
7      ('53', 'Comprehensive Experimental Building', 'South Campus 2'),
8      ('7', 'Main Teaching Building', 'South Campus 1');
9
10 -- Insert several room records.
11 INSERT INTO room (building_number, room_number, room_name,
12                  capacity) VALUES
13     ('36', '305', 'Meeting Room', 40),
14     ('36', '105', 'Lecture Room 1', 80),
15     ('10', '100', 'Teaching Room A', 30),
16     ('10', '102', 'Teaching Room B', 30),
17     ('53', '904a', 'Office 1', 10),
18     ('53', '904b', 'Cluster Room', 3),
19     ('7', '200', 'Auditorium', 120),
20     ('36', '106', 'Lecture Room 2', 80);
```

```
1  $ psql "postgres://postgres:XXX@localhost/fixed" -v ON_ERROR_STOP=1 -ebf
   ↪ insert.sql
2  INSERT 0 4
3  INSERT 0 8
4  # psql 16.11 succeeded with exit code 0.
```

# Daten Einfügen



- Wir sehen sofort, dass es jetzt viel weniger Redundanz gibt.
- Die Daten über jedes Gebäude werden genau einmal gespeichert.
- Vorher haben wir sie für jeden Raum speichern müssen.
- Weil wir die Daten weniger oft schreiben müssen, wird auch die Wahrscheinlichkeit, Fehler zu machen, kleiner.
- Und wenn wir Fehler machen, dann sind sie leichter zu sehen, weil wir weniger Datensätze anschauen müssen.

```
1  /** Insert data into the database. */
2
3  -- Insert several room records.
4  INSERT INTO building (building_number, building_name, address) VALUES
5      ('36', 'CS Teaching Building', 'South Campus II'),
6      ('10', 'Language Teaching Building', 'South Campus 1'),
7      ('53', 'Comprehensive Experimental Building', 'South Campus 2'),
8      ('7', 'Main Teaching Building', 'South Campus 1');
9
10 -- Insert several room records.
11 INSERT INTO room (building_number, room_number, room_name,
12                  capacity) VALUES
13     ('36', '305', 'Meeting Room', 40),
14     ('36', '105', 'Lecture Room 1', 80),
15     ('10', '100', 'Teaching Room A', 30),
16     ('10', '102', 'Teaching Room B', 30),
17     ('53', '904a', 'Office 1', 10),
18     ('53', '904b', 'Cluster Room', 3),
19     ('7', '200', 'Auditorium', 120),
20     ('36', '106', 'Lecture Room 2', 80);
```

```
1  $ psql "postgres://postgres:XXX@localhost/fixed" -v ON_ERROR_STOP=1 -ebf
   ↳ insert.sql
2  INSERT 0 4
3  INSERT 0 8
4  # psql 16.11 succeeded with exit code 0.
```

# Daten Löschen

- Reproduzieren wir jetzt das erste „Problem“ von vorhin.

```
1  /** Delete room 904a and 904b, which deletes the entire building. */
2
3  -- Get the list of all buildings.
4  SELECT building_number, building_name FROM building;
5
6  -- Delete rooms 904a and 904b in building 53, which deletes building 53.
7  DELETE FROM room WHERE building_number = '53'
8      AND room_number LIKE '904%'
9      RETURNING building_number, room_number;
10
11 -- Get the list of all buildings again: Building 53 disappeared.
12 SELECT building_number, building_name FROM building;
```

```
1  $ psql "postgres://postgres:XXX@localhost/fixed" -v ON_ERROR_STOP=1 -ebf
   ↪ delete.sql
2
3  building_number |          building_name
4  -----+-----
5  36              | CS Teaching Building
6  10              | Language Teaching Building
7  53              | Comprehensive Experimental Building
8  7              | Main Teaching Building
9  (4 rows)
10
11 building_number | room_number
12 -----+-----
13  53              | 904a
14  53              | 904b
15  (2 rows)
16
17 DELETE 2
18
19 building_number |          building_name
20 -----+-----
21  36              | CS Teaching Building
22  10              | Language Teaching Building
23  53              | Comprehensive Experimental Building
24  7              | Main Teaching Building
25  (4 rows)
26
27 # psql 16.11 succeeded with exit code 0.
```

# Daten Löschen

- Reproduzieren wir jetzt das erste „Problem“ von vorhin.
- Damals hatten wir uns zuerst eine Liste von Gebäuden generiert.

```
1  /** Delete room 904a and 904b, which deletes the entire building. */
2
3  -- Get the list of all buildings.
4  SELECT building_number, building_name FROM building;
5
6  -- Delete rooms 904a and 904b in building 53, which deletes building 53.
7  DELETE FROM room WHERE building_number = '53'
8      AND room_number LIKE '904%'
9      RETURNING building_number, room_number;
10
11 -- Get the list of all buildings again: Building 53 disappeared.
12 SELECT building_number, building_name FROM building;
```

```
1  $ psql "postgres://postgres:XXX@localhost/fixed" -v ON_ERROR_STOP=1 -ebf
   ↳ delete.sql
2
3  building_number |          building_name
4  -----+-----
5  36              | CS Teaching Building
6  10              | Language Teaching Building
7  53              | Comprehensive Experimental Building
8  7              | Main Teaching Building
9  (4 rows)
10
11 building_number | room_number
12 -----+-----
13 53              | 904a
14 53              | 904b
15 (2 rows)
16
17 DELETE 2
18
19 building_number |          building_name
20 -----+-----
21 36              | CS Teaching Building
22 10              | Language Teaching Building
23 53              | Comprehensive Experimental Building
24 7              | Main Teaching Building
25 (4 rows)
26
27 # psql 16.11 succeeded with exit code 0.
```

# Daten Löschen

- Reproduzieren wir jetzt das erste „Problem“ von vorhin.
- Damals hatten wir uns zuerst eine Liste von Gebäuden generiert.
- Wir hatten gesehen, dass es vier Gebäude gab.

```
1  /** Delete room 904a and 904b, which deletes the entire building. */
2
3  -- Get the list of all buildings.
4  SELECT building_number, building_name FROM building;
5
6  -- Delete rooms 904a and 904b in building 53, which deletes building 53.
7  DELETE FROM room WHERE building_number = '53'
8      AND room_number LIKE '904%'
9      RETURNING building_number, room_number;
10
11 -- Get the list of all buildings again: Building 53 disappeared.
12 SELECT building_number, building_name FROM building;
```

```
1  $ psql "postgres://postgres:XXX@localhost/fixed" -v ON_ERROR_STOP=1 -ebf
   ↪ delete.sql
2
3  building_number | building_name
4  -----+-----
5  36              | CS Teaching Building
6  10              | Language Teaching Building
7  53              | Comprehensive Experimental Building
8  7              | Main Teaching Building
9  (4 rows)
10
11 building_number | room_number
12 -----+-----
13  53              | 904a
14  53              | 904b
15  (2 rows)
16
17 DELETE 2
18
19 building_number | building_name
20 -----+-----
21 36              | CS Teaching Building
22 10              | Language Teaching Building
23 53              | Comprehensive Experimental Building
24 7              | Main Teaching Building
25  (4 rows)
26
27 # psql 16.11 succeeded with exit code 0.
```

# Daten Löschen

- Reproduzieren wir jetzt das erste „Problem“ von vorhin.
- Damals hatten wir uns zuerst eine Liste von Gebäuden generiert.
- Wir hatten gesehen, dass es vier Gebäude gab.
- Dann hatten wir die beiden Räume 904a und 904b gelöscht.

```
1  /** Delete room 904a and 904b, which deletes the entire building. */
2
3  -- Get the list of all buildings.
4  SELECT building_number, building_name FROM building;
5
6  -- Delete rooms 904a and 904b in building 53, which deletes building 53.
7  DELETE FROM room WHERE building_number = '53'
8      AND room_number LIKE '904%'
9      RETURNING building_number, room_number;
10
11 -- Get the list of all buildings again: Building 53 disappeared.
12 SELECT building_number, building_name FROM building;
```

```
1  $ psql "postgres://postgres:XXX@localhost/fixed" -v ON_ERROR_STOP=1 -ebf
   ↪ delete.sql
2
3  building_number | building_name
4  -----
5  36              | CS Teaching Building
6  10              | Language Teaching Building
7  53              | Comprehensive Experimental Building
8  7              | Main Teaching Building
9  (4 rows)
10
11 building_number | room_number
12 -----
13 53              | 904a
14 53              | 904b
15 (2 rows)
16
17 DELETE 2
18
19 building_number | building_name
20 -----
21 36              | CS Teaching Building
22 10              | Language Teaching Building
23 53              | Comprehensive Experimental Building
24 7              | Main Teaching Building
25 (4 rows)
26
27 # psql 16.11 succeeded with exit code 0.
```

# Daten Löschen

- Reproduzieren wir jetzt das erste „Problem“ von vorhin.
- Damals hatten wir uns zuerst eine Liste von Gebäuden generiert.
- Wir hatten gesehen, dass es vier Gebäude gab.
- Dann hatten wir die beiden Räume 904a und 904b gelöscht.
- Dadurch war auch Gebäude 53 mit verschwunden, denn dessen Existenz war an die beiden Raum-Datensätze gebunden.

```
1  /** Delete room 904a and 904b, which deletes the entire building. */
2
3  -- Get the list of all buildings.
4  SELECT building_number, building_name FROM building;
5
6  -- Delete rooms 904a and 904b in building 53, which deletes building 53.
7  DELETE FROM room WHERE building_number = '53'
8      AND room_number LIKE '904%'
9      RETURNING building_number, room_number;
10
11 -- Get the list of all buildings again: Building 53 disappeared.
12 SELECT building_number, building_name FROM building;
```

```
1  $ psql "postgres://postgres:XXX@localhost/fixed" -v ON_ERROR_STOP=1 -ebf
   ↳ delete.sql
2
3  building_number | building_name
4  -----+-----
5  36              | CS Teaching Building
6  10              | Language Teaching Building
7  53              | Comprehensive Experimental Building
8  7              | Main Teaching Building
9  (4 rows)
10
11 building_number | room_number
12 -----+-----
13 53              | 904a
14 53              | 904b
15 (2 rows)
16
17 DELETE 2
18
19 building_number | building_name
20 -----+-----
21 36              | CS Teaching Building
22 10              | Language Teaching Building
23 53              | Comprehensive Experimental Building
24 7              | Main Teaching Building
25 (4 rows)
26
27 # psql 16.11 succeeded with exit code 0.
```

# Daten Löschen

- Damals hatten wir uns zuerst eine Liste von Gebäuden generiert.
- Wir hatten gesehen, dass es vier Gebäude gab.
- Dann hatten wir die beiden Räume 904a und 904b gelöscht.
- Dadurch war auch Gebäude 53 mit verschwunden, denn dessen Existenz war an die beiden Raum-Datensätze gebunden.
- Wir fangen also wieder damit an, eine Liste aller Gebäude zu erstellen.

```
1  /** Delete room 904a and 904b, which deletes the entire building. */
2
3  -- Get the list of all buildings.
4  SELECT building_number, building_name FROM building;
5
6  -- Delete rooms 904a and 904b in building 53, which deletes building 53.
7  DELETE FROM room WHERE building_number = '53'
8      AND room_number LIKE '904%'
9      RETURNING building_number, room_number;
10
11 -- Get the list of all buildings again: Building 53 disappeared.
12 SELECT building_number, building_name FROM building;
```

```
1  $ psql "postgres://postgres:XXX@localhost/fixed" -v ON_ERROR_STOP=1 -ebf
   ↪ delete.sql
2
3  building_number | building_name
4  -----+-----
5  36              | CS Teaching Building
6  10              | Language Teaching Building
7  53              | Comprehensive Experimental Building
8  7              | Main Teaching Building
9  (4 rows)
10
11 building_number | room_number
12 -----+-----
13  53              | 904a
14  53              | 904b
15  (2 rows)
16
17 DELETE 2
18 building_number | building_name
19 -----+-----
20  36              | CS Teaching Building
21  10              | Language Teaching Building
22  53              | Comprehensive Experimental Building
23  7              | Main Teaching Building
24  (4 rows)
25
26 # psql 16.11 succeeded with exit code 0.
```

# Daten Löschen

- Wir hatten gesehen, dass es vier Gebäude gab.
- Dann hatten wir die beiden Räume 904a und 904b gelöscht.
- Dadurch war auch Gebäude 53 mit verschwunden, denn dessen Existenz war an die beiden Raum-Datensätze gebunden.
- Wir fangen also wieder damit an, eine Liste aller Gebäude zu erstellen.
- Das geht nun über

```
SELECT building_number,  
building_name FROM building;
```

```
1  /** Delete room 904a and 904b, which deletes the entire building. */  
2  
3  -- Get the list of all buildings.  
4  SELECT building_number, building_name FROM building;  
5  
6  -- Delete rooms 904a and 904b in building 53, which deletes building 53.  
7  DELETE FROM room WHERE building_number = '53'  
8      AND room_number LIKE '904%'  
9      RETURNING building_number, room_number;  
10  
11 -- Get the list of all buildings again: Building 53 disappeared.  
12 SELECT building_number, building_name FROM building;
```

```
1  $ psql "postgres://postgres:XXX@localhost/fixed" -v ON_ERROR_STOP=1 -ebf  
   ↪ delete.sql  
2  
3  building_number | building_name  
4  -----  
5  36              | CS Teaching Building  
6  10              | Language Teaching Building  
7  53              | Comprehensive Experimental Building  
8  7              | Main Teaching Building  
9  (4 rows)  
10  
11 building_number | room_number  
12 -----  
13 53              | 904a  
14 53              | 904b  
15 (2 rows)  
16  
17 DELETE 2  
18  
19 building_number | building_name  
20 -----  
21 36              | CS Teaching Building  
22 10              | Language Teaching Building  
23 53              | Comprehensive Experimental Building  
24 7              | Main Teaching Building  
25 (4 rows)  
  
# psql 16.11 succeeded with exit code 0.
```

# Daten Löschen

- Dann hatten wir die beiden Räume 904a und 904b gelöscht.
- Dadurch war auch Gebäude 53 mit verschwunden, denn dessen Existenz war an die beiden Raum-Datensätze gebunden.
- Wir fangen also wieder damit an, eine Liste aller Gebäude zu erstellen.
- Das geht nun über  
`SELECT building_number, building_name FROM building;`
- Im Vergleich zu vorhin brauchen wir das `DISTINCT` nicht mehr.

```
1  /** Delete room 904a and 904b, which deletes the entire building. */
2
3  -- Get the list of all buildings.
4  SELECT building_number, building_name FROM building;
5
6  -- Delete rooms 904a and 904b in building 53, which deletes building 53.
7  DELETE FROM room WHERE building_number = '53'
8      AND room_number LIKE '904%'
9      RETURNING building_number, room_number;
10
11 -- Get the list of all buildings again: Building 53 disappeared.
12 SELECT building_number, building_name FROM building;
```

```
1  $ psql "postgres://postgres:XXX@localhost/fixed" -v ON_ERROR_STOP=1 -ebf
   ↪ delete.sql
2
3  building_number | building_name
4  -----+-----
5  36              | CS Teaching Building
6  10              | Language Teaching Building
7  53              | Comprehensive Experimental Building
8  7              | Main Teaching Building
9  (4 rows)
10
11 building_number | room_number
12 -----+-----
13  53              | 904a
14  53              | 904b
15  (2 rows)
16
17 DELETE 2
18 building_number | building_name
19 -----+-----
20  36              | CS Teaching Building
21  10              | Language Teaching Building
22  53              | Comprehensive Experimental Building
23  7              | Main Teaching Building
24  (4 rows)
25
# psql 16.11 succeeded with exit code 0.
```

# Daten Löschen

- Dadurch war auch Gebäude 53 mit verschwunden, denn dessen Existenz war an die beiden Raum-Datensätze gebunden.
- Wir fangen also wieder damit an, eine Liste aller Gebäude zu erstellen.
- Das geht nun über  
`SELECT building_number, building_name FROM building;`
- Im Vergleich zu vorhin brauchen wir das `DISTINCT` nicht mehr.
- Damals, in Tabelle `building_room`, tauchte jede Gebäudenummer und jeder Gebäudename einmal für jeden Raum auf.

```
1  /** Delete room 904a and 904b, which deletes the entire building. */
2
3  -- Get the list of all buildings.
4  SELECT building_number, building_name FROM building;
5
6  -- Delete rooms 904a and 904b in building 53, which deletes building 53.
7  DELETE FROM room WHERE building_number = '53'
8      AND room_number LIKE '904%'
9      RETURNING building_number, room_number;
10
11 -- Get the list of all buildings again: Building 53 disappeared.
12 SELECT building_number, building_name FROM building;
```

```
1  $ psql "postgres://postgres:XXX@localhost/fixed" -v ON_ERROR_STOP=1 -ebf
   ↪ delete.sql
2
3  building_number | building_name
4  -----
5  36              | CS Teaching Building
6  10              | Language Teaching Building
7  53              | Comprehensive Experimental Building
8  7              | Main Teaching Building
9  (4 rows)
10
11 building_number | room_number
12 -----+-----
13 53              | 904a
14 53              | 904b
15 (2 rows)
16
17 DELETE 2
18 building_number | building_name
19 -----+-----
20 36              | CS Teaching Building
21 10              | Language Teaching Building
22 53              | Comprehensive Experimental Building
23 7              | Main Teaching Building
24 (4 rows)
25
26 # psql 16.11 succeeded with exit code 0.
```

# Daten Löschen

- Wir fangen also wieder damit an, eine Liste aller Gebäude zu erstellen.
- Das geht nun über  
`SELECT building_number, building_name FROM building;`
- Im Vergleich zu vorher brauchen wir das `DISTINCT` nicht mehr.
- Damals, in Tabelle `building_room`, tauchte jede Gebäudenummer und jeder Gebäudename einmal für jeden Raum auf.
- In Tabelle `building` kommen Sie aber immer nur einmal vor.

```
1  /** Delete room 904a and 904b, which deletes the entire building. */
2
3  -- Get the list of all buildings.
4  SELECT building_number, building_name FROM building;
5
6  -- Delete rooms 904a and 904b in building 53, which deletes building 53.
7  DELETE FROM room WHERE building_number = '53'
8      AND room_number LIKE '904%'
9      RETURNING building_number, room_number;
10
11 -- Get the list of all buildings again: Building 53 disappeared.
12 SELECT building_number, building_name FROM building;
```

```
1  $ psql "postgres://postgres:XXX@localhost/fixed" -v ON_ERROR_STOP=1 -ebf
   ↪ delete.sql
2
3  building_number | building_name
4  -----
5  36              | CS Teaching Building
6  10              | Language Teaching Building
7  53              | Comprehensive Experimental Building
8  7              | Main Teaching Building
9  (4 rows)
10
11 building_number | room_number
12 -----
13 53              | 904a
14 53              | 904b
15 (2 rows)
16
17 DELETE 2
18 building_number | building_name
19 -----
20 36              | CS Teaching Building
21 10              | Language Teaching Building
22 53              | Comprehensive Experimental Building
23 7              | Main Teaching Building
24 (4 rows)
25
26 # psql 16.11 succeeded with exit code 0.
```

# Daten Löschen

- Das geht nun über  
`SELECT building_number,  
building_name FROM building;`
- Im Vergleich zu vorhin brauchen wir das `DISTINCT` nicht mehr.
- Damals, in Tabelle `building_room`, tauchte jede Gebäudenummer und jeder Gebäudename einmal für jeden Raum auf.
- In Tabelle `building` kommen Sie aber immer nur einmal vor.
- Daher ist das Weglassen doppelter Zeilen, das `DISTINCT` macht, nicht mehr nötig.

```
1  /** Delete room 904a and 904b, which deletes the entire building. */
2
3  -- Get the list of all buildings.
4  SELECT building_number, building_name FROM building;
5
6  -- Delete rooms 904a and 904b in building 53, which deletes building 53.
7  DELETE FROM room WHERE building_number = '53'
8      AND room_number LIKE '904%'
9      RETURNING building_number, room_number;
10
11 -- Get the list of all buildings again: Building 53 disappeared.
12 SELECT building_number, building_name FROM building;
```

```
1  $ psql "postgres://postgres:XXX@localhost/fixed" -v ON_ERROR_STOP=1 -ebf
   ↳ delete.sql
2
3  building_number | building_name
4  -----+-----
5  36              | CS Teaching Building
6  10              | Language Teaching Building
7  53              | Comprehensive Experimental Building
8  7              | Main Teaching Building
9  (4 rows)
10
11 building_number | room_number
12 -----+-----
13  53              | 904a
14  53              | 904b
15  (2 rows)
16
17 DELETE 2
18 building_number | building_name
19 -----+-----
20 36              | CS Teaching Building
21 10              | Language Teaching Building
22 53              | Comprehensive Experimental Building
23 7              | Main Teaching Building
24 (4 rows)
25
26 # psql 16.11 succeeded with exit code 0.
```

# Daten Löschen

- Im Vergleich zu vorher brauchen wir das **DISTINCT** nicht mehr.
- Damals, in Tabelle **building\_room**, tauchte jede Gebäudenummer und jeder Gebäudename einmal für jeden Raum auf.
- In Tabelle **building** kommen Sie aber immer nur einmal vor.
- Daher ist das Weglassen doppelter Zeilen, das **DISTINCT** macht, nicht mehr nötig.
- Wir bekommen wieder eine Liste von vier Gebäuden.

```
1  /** Delete room 904a and 904b, which deletes the entire building. */
2
3  -- Get the list of all buildings.
4  SELECT building_number, building_name FROM building;
5
6  -- Delete rooms 904a and 904b in building 53, which deletes building 53.
7  DELETE FROM room WHERE building_number = '53'
8      AND room_number LIKE '904%'
9      RETURNING building_number, room_number;
10
11 -- Get the list of all buildings again: Building 53 disappeared.
12 SELECT building_number, building_name FROM building;
```

```
1  $ psql "postgres://postgres:XXX@localhost/fixed" -v ON_ERROR_STOP=1 -ebf
   ↪ delete.sql
```

| building_number | building_name                       |
|-----------------|-------------------------------------|
| 36              | CS Teaching Building                |
| 10              | Language Teaching Building          |
| 53              | Comprehensive Experimental Building |
| 7               | Main Teaching Building              |

(4 rows)

| building_number | room_number |
|-----------------|-------------|
| 53              | 904a        |
| 53              | 904b        |

(2 rows)

```
DELETE 2
building_number | building_name
-----+-----
36              | CS Teaching Building
10              | Language Teaching Building
53              | Comprehensive Experimental Building
7              | Main Teaching Building
(4 rows)
```

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

# Daten Löschen

- Damals, in Tabelle `building_room`, tauchte jede Gebäudenummer und jeder Gebäudename einmal für jeden Raum auf.
- In Tabelle `building` kommen Sie aber immer nur einmal vor.
- Daher ist das Weglassen doppelter Zeilen, das `DISTINCT` macht, nicht mehr nötig.
- Wir bekommen wieder eine Liste von vier Gebäuden.
- Wir löschen nun die beiden Räume 904a und 904b von Gebäude 53 mit einer `DELETE`-Anfrage.

```
1  /** Delete room 904a and 904b, which deletes the entire building. */
2
3  -- Get the list of all buildings.
4  SELECT building_number, building_name FROM building;
5
6  -- Delete rooms 904a and 904b in building 53, which deletes building 53.
7  DELETE FROM room WHERE building_number = '53'
8      AND room_number LIKE '904%'
9      RETURNING building_number, room_number;
10
11 -- Get the list of all buildings again: Building 53 disappeared.
12 SELECT building_number, building_name FROM building;
```

```
1  $ psql "postgres://postgres:XXX@localhost/fixed" -v ON_ERROR_STOP=1 -ebf
   ↪ delete.sql
2
3  building_number | building_name
4  -----+-----
5  36              | CS Teaching Building
6  10              | Language Teaching Building
7  53              | Comprehensive Experimental Building
8  7              | Main Teaching Building
9  (4 rows)
10
11 building_number | room_number
12 -----+-----
13  53              | 904a
14  53              | 904b
15  (2 rows)
16
17 DELETE 2
18 building_number | building_name
19 -----+-----
20  36              | CS Teaching Building
21  10              | Language Teaching Building
22  53              | Comprehensive Experimental Building
23  7              | Main Teaching Building
24  (4 rows)
25
26 # psql 16.11 succeeded with exit code 0.
```

# Daten Löschen

- In Tabelle `building` kommen Sie aber immer nur einmal vor.
- Daher ist das Weglassen doppelter Zeilen, das `DISTINCT` macht, nicht mehr nötig.
- Wir bekommen wieder eine Liste von vier Gebäuden.
- Wir löschen nun die beiden Räume 904a und 904b von Gebäude 53 mit einer `DELETE`-Anfrage.
- Das `RETURNING`-Statement gibt uns die beiden gelöschten Räume nochmal zurück.

```
1  /** Delete room 904a and 904b, which deletes the entire building. */
2
3  -- Get the list of all buildings.
4  SELECT building_number, building_name FROM building;
5
6  -- Delete rooms 904a and 904b in building 53, which deletes building 53.
7  DELETE FROM room WHERE building_number = '53'
8      AND room_number LIKE '904%'
9      RETURNING building_number, room_number;
10
11 -- Get the list of all buildings again: Building 53 disappeared.
12 SELECT building_number, building_name FROM building;
```

```
1  $ psql "postgres://postgres:XXX@localhost/fixed" -v ON_ERROR_STOP=1 -ebf
   ↳ delete.sql
2
3  building_number | building_name
4  -----+-----
5  36              | CS Teaching Building
6  10              | Language Teaching Building
7  53              | Comprehensive Experimental Building
8  7              | Main Teaching Building
9  (4 rows)
10
11 building_number | room_number
12 -----+-----
13  53              | 904a
14  53              | 904b
15  (2 rows)
16
17 DELETE 2
18 building_number | building_name
19 -----+-----
20  36              | CS Teaching Building
21  10              | Language Teaching Building
22  53              | Comprehensive Experimental Building
23  7              | Main Teaching Building
24  (4 rows)
25
26 # psql 16.11 succeeded with exit code 0.
```

# Daten Löschen

- Daher ist das Weglassen doppelter Zeilen, das **DISTINCT** macht, nicht mehr nötig.
- Wir bekommen wieder eine Liste von vier Gebäuden.
- Wir löschen nun die beiden Räume 904a und 904b von Gebäude 53 mit einer **DELETE**-Anfrage.
- Das **RETURNING**-Statement gibt uns die beiden gelöschten Räume nochmal zurück.
- Damals hat das Löschen der Räume auch zum Verschwinden des Gebäudes 53 geführt.

```
1  /** Delete room 904a and 904b, which deletes the entire building. */
2
3  -- Get the list of all buildings.
4  SELECT building_number, building_name FROM building;
5
6  -- Delete rooms 904a and 904b in building 53, which deletes building 53.
7  DELETE FROM room WHERE building_number = '53'
8      AND room_number LIKE '904%'
9      RETURNING building_number, room_number;
10
11 -- Get the list of all buildings again: Building 53 disappeared.
12 SELECT building_number, building_name FROM building;
```

```
1  $ psql "postgres://postgres:XXX@localhost/fixed" -v ON_ERROR_STOP=1 -ebf
   ↪ delete.sql
2
3  building_number | building_name
4  -----
5  36              | CS Teaching Building
6  10              | Language Teaching Building
7  53              | Comprehensive Experimental Building
8  7              | Main Teaching Building
9  (4 rows)
10
11 building_number | room_number
12 -----
13 53              | 904a
14 53              | 904b
15 (2 rows)
16
17 DELETE 2
18
19 building_number | building_name
20 -----
21 36              | CS Teaching Building
22 10              | Language Teaching Building
23 53              | Comprehensive Experimental Building
24 7              | Main Teaching Building
25 (4 rows)
26
27 # psql 16.11 succeeded with exit code 0.
```

# Daten Löschen

- Wir bekommen wieder eine Liste von vier Gebäuden.
- Wir löschen nun die beiden Räume 904a und 904b von Gebäude 53 mit einer **DELETE**-Anfrage.
- Das **RETURNING**-Statement gibt uns die beiden gelöschten Räume nochmal zurück.
- Damals hat das Löschen der Räume auch zum Verschwinden des Gebäudes 53 geführt.
- Das war das Ergebnis der Verletzung der 2NF.

```
1  /** Delete room 904a and 904b, which deletes the entire building. */
2
3  -- Get the list of all buildings.
4  SELECT building_number, building_name FROM building;
5
6  -- Delete rooms 904a and 904b in building 53, which deletes building 53.
7  DELETE FROM room WHERE building_number = '53'
8      AND room_number LIKE '904%'
9      RETURNING building_number, room_number;
10
11 -- Get the list of all buildings again: Building 53 disappeared.
12 SELECT building_number, building_name FROM building;
```

```
1  $ psql "postgres://postgres:XXX@localhost/fixed" -v ON_ERROR_STOP=1 -ebf
   ↪ delete.sql
2
3  building_number | building_name
4  -----
5  36              | CS Teaching Building
6  10              | Language Teaching Building
7  53              | Comprehensive Experimental Building
8  7              | Main Teaching Building
9  (4 rows)
10
11 building_number | room_number
12 -----
13 53              | 904a
14 53              | 904b
15 (2 rows)
16
17 DELETE 2
18 building_number | building_name
19 -----
20 36              | CS Teaching Building
21 10              | Language Teaching Building
22 53              | Comprehensive Experimental Building
23 7              | Main Teaching Building
24 (4 rows)
25
26 # psql 16.11 succeeded with exit code 0.
```

# Daten Löschen

- Wir löschen nun die beiden Räume 904a und 904b von Gebäude 53 mit einer **DELETE**-Anfrage.
- Das **RETURNING**-Statement gibt uns die beiden gelöschten Räume nochmal zurück.
- Damals hat das Löschen der Räume auch zum Verschwinden des Gebäudes 53 geführt.
- Das war das Ergebnis der Verletzung der 2NF.
- Deshalb gab es damals nach dem Löschen der Räume nur noch drei Gebäude.

```
1  /** Delete room 904a and 904b, which deletes the entire building. */
2
3  -- Get the list of all buildings.
4  SELECT building_number, building_name FROM building;
5
6  -- Delete rooms 904a and 904b in building 53, which deletes building 53.
7  DELETE FROM room WHERE building_number = '53'
8      AND room_number LIKE '904%'
9      RETURNING building_number, room_number;
10
11 -- Get the list of all buildings again: Building 53 disappeared.
12 SELECT building_number, building_name FROM building;
```

```
1  $ psql "postgres://postgres:XXX@localhost/fixed" -v ON_ERROR_STOP=1 -ebf
   ↳ delete.sql
2
3  building_number | building_name
4  -----+-----
5  36              | CS Teaching Building
6  10              | Language Teaching Building
7  53              | Comprehensive Experimental Building
8  7              | Main Teaching Building
9  (4 rows)
10
11 building_number | room_number
12 -----+-----
13  53              | 904a
14  53              | 904b
15  (2 rows)
16
17 DELETE 2
18 building_number | building_name
19 -----+-----
20  36              | CS Teaching Building
21  10              | Language Teaching Building
22  53              | Comprehensive Experimental Building
23  7              | Main Teaching Building
24  (4 rows)
25
26 # psql 16.11 succeeded with exit code 0.
```

# Daten Löschen

- Das **RETURNING**-Statement gibt uns die beiden gelöschten Räume nochmal zurück.
- Damals hat das Löschen der Räume auch zum Verschwinden des Gebäudes 53 geführt.
- Das war das Ergebnis der Verletzung der 2NF.
- Deshalb gab es damals nach dem Löschen der Räume nur noch drei Gebäude.
- Das ist jetzt nicht mehr so.

```
1  /** Delete room 904a and 904b, which deletes the entire building. */
2
3  -- Get the list of all buildings.
4  SELECT building_number, building_name FROM building;
5
6  -- Delete rooms 904a and 904b in building 53, which deletes building 53.
7  DELETE FROM room WHERE building_number = '53'
8      AND room_number LIKE '904%'
9      RETURNING building_number, room_number;
10
11 -- Get the list of all buildings again: Building 53 disappeared.
12 SELECT building_number, building_name FROM building;
```

```
1  $ psql "postgres://postgres:XXX@localhost/fixed" -v ON_ERROR_STOP=1 -ebf
   ↪ delete.sql
2
3  building_number | building_name
4  -----+-----
5  36              | CS Teaching Building
6  10              | Language Teaching Building
7  53              | Comprehensive Experimental Building
8  7              | Main Teaching Building
9  (4 rows)
10
11 building_number | room_number
12 -----+-----
13  53              | 904a
14  53              | 904b
15  (2 rows)
16
17 DELETE 2
18 building_number | building_name
19 -----+-----
20  36              | CS Teaching Building
21  10              | Language Teaching Building
22  53              | Comprehensive Experimental Building
23  7              | Main Teaching Building
24  (4 rows)
25
26 # psql 16.11 succeeded with exit code 0.
```

# Daten Löschen

- Damals hat das Löschen der Räume auch zum Verschwinden des Gebäudes 53 geführt.
- Das war das Ergebnis der Verletzung der 2NF.
- Deshalb gab es damals nach dem Löschen der Räume nur noch drei Gebäude.
- Das ist jetzt nicht mehr so.
- Alle vier Gebäude sind noch da.

```
1  /** Delete room 904a and 904b, which deletes the entire building. */
2
3  -- Get the list of all buildings.
4  SELECT building_number, building_name FROM building;
5
6  -- Delete rooms 904a and 904b in building 53, which deletes building 53.
7  DELETE FROM room WHERE building_number = '53'
8      AND room_number LIKE '904%'
9      RETURNING building_number, room_number;
10
11 -- Get the list of all buildings again: Building 53 disappeared.
12 SELECT building_number, building_name FROM building;
```

```
1  $ psql "postgres://postgres:XXX@localhost/fixed" -v ON_ERROR_STOP=1 -ebf
   ↪ delete.sql
2
3  building_number | building_name
4  -----
5  36              | CS Teaching Building
6  10              | Language Teaching Building
7  53              | Comprehensive Experimental Building
8  7              | Main Teaching Building
9  (4 rows)
10
11 building_number | room_number
12 -----+-----
13  53              | 904a
14  53              | 904b
15  (2 rows)
16
17 DELETE 2
18
19 building_number | building_name
20 -----+-----
21  36              | CS Teaching Building
22  10              | Language Teaching Building
23  53              | Comprehensive Experimental Building
24  7              | Main Teaching Building
25  (4 rows)
26
27 # psql 16.11 succeeded with exit code 0.
```

# Daten Ändern (1)



- Jetzt reproduzieren wir das „2. Problem“ von vorhin.

```
1  /** Find all the rooms available in South Campus 2. */
2
3  -- Several rooms are missing, because the data is inconsistent:
4  -- Their address was 'South Campus II'.
5  SELECT room.building_number, room_number FROM room
6         INNER JOIN building ON
7             room.building_number = building.building_number
8         WHERE address = 'South Campus 2';

1  $ psql "postgres://postgres:XXX@localhost/fixed" -v ON_ERROR_STOP=1 -ebf
   ↪ select_a_1.sql
2  building_number | room_number
3  -----+-----
4  (0 rows)
5
6  # psql 16.11 succeeded with exit code 0.
```

# Daten Ändern (1)



- Jetzt reproduzieren wir das „2. Problem“ von vorhin.
- Wir wollen eine Liste aller Räume in South Campus 2.

```
1  /** Find all the rooms available in South Campus 2. */
2
3  -- Several rooms are missing, because the data is inconsistent:
4  -- Their address was 'South Campus II'.
5  SELECT room.building_number, room_number FROM room
6         INNER JOIN building ON
7             room.building_number = building.building_number
8         WHERE address = 'South Campus 2';
```

```
1  $ psql "postgres://postgres:XXX@localhost/fixed" -v ON_ERROR_STOP=1 -ebf
   ↳ select_a_1.sql
2  building_number | room_number
3  -----+-----
4  (0 rows)
5
6  # psql 16.11 succeeded with exit code 0.
```

# Daten Ändern (1)



- Jetzt reproduzieren wir das „2. Problem“ von vorhin.
- Wir wollen eine Liste aller Räume in South Campus 2.
- Wenn wir uns die notwendigen Schritte anschauen, um diese Informationen zu bekommen, dann sehen wir den Nachteil der 2NF: Wir haben nun eine kompliziertere Anfrage.

```
1  /** Find all the rooms available in South Campus 2. */
2
3  -- Several rooms are missing, because the data is inconsistent:
4  -- Their address was 'South Campus II'.
5  SELECT room.building_number, room_number FROM room
6         INNER JOIN building ON
7             room.building_number = building.building_number
8         WHERE address = 'South Campus 2';

1  $ psql "postgres://postgres:XXX@localhost/fixed" -v ON_ERROR_STOP=1 -ebf
   ↪ select_a_1.sql
2  building_number | room_number
3  -----+-----
4  (0 rows)
5
6  # psql 16.11 succeeded with exit code 0.
```

# Daten Ändern (1)



- Jetzt reproduzieren wir das „2. Problem“ von vorhin.
- Wir wollen eine Liste aller Räume in South Campus 2.
- Wenn wir uns die notwendigen Schritte anschauen, um diese Informationen zu bekommen, dann sehen wir den Nachteil der 2NF: Wir haben nun eine kompliziertere Anfrage.
- Vorhin war die Gebäudeadresse zusammen mit den Raumnummern in einer Zeile gespeichert.

```
1  /** Find all the rooms available in South Campus 2. */
2
3  -- Several rooms are missing, because the data is inconsistent:
4  -- Their address was 'South Campus II'.
5  SELECT room.building_number, room_number FROM room
6         INNER JOIN building ON
7             room.building_number = building.building_number
8         WHERE address = 'South Campus 2';
```

```
1  $ psql "postgres://postgres:XXX@localhost/fixed" -v ON_ERROR_STOP=1 -ebf
   ↪ select_a_1.sql
2  building_number | room_number
3  -----+-----
4  (0 rows)
5
6  # psql 16.11 succeeded with exit code 0.
```

# Daten Ändern (1)



- Jetzt reproduzieren wir das „2. Problem“ von vorhin.
- Wir wollen eine Liste aller Räume in South Campus 2.
- Wenn wir uns die notwendigen Schritte anschauen, um diese Informationen zu bekommen, dann sehen wir den Nachteil der 2NF: Wir haben nun eine kompliziertere Anfrage.
- Vorhin war die Gebäudeadresse zusammen mit den Raumnummern in einer Zeile gespeichert.
- Das ist nicht mehr so.

```
1  /** Find all the rooms available in South Campus 2. */
2
3  -- Several rooms are missing, because the data is inconsistent:
4  -- Their address was 'South Campus II'.
5  SELECT room.building_number, room_number FROM room
6         INNER JOIN building ON
7             room.building_number = building.building_number
8         WHERE address = 'South Campus 2';
```

```
1  $ psql "postgres://postgres:XXX@localhost/fixed" -v ON_ERROR_STOP=1 -ebf
   ↪ select_a_1.sql
2  building_number | room_number
3  -----+-----
4  (0 rows)
5
6  # psql 16.11 succeeded with exit code 0.
```

# Daten Ändern (1)



- Wir wollen eine Liste aller Räume in South Campus 2.
- Wenn wir uns die notwendigen Schritte anschauen, um diese Informationen zu bekommen, dann sehen wir den Nachteil der 2NF: Wir haben nun eine kompliziertere Anfrage.
- Vorhin war die Gebäudeadresse zusammen mit den Raumnummern in einer Zeile gespeichert.
- Das ist nicht mehr so.
- Die Rauminformation ist in Tabelle room gespeichert.

```
1  /** Find all the rooms available in South Campus 2. */
2
3  -- Several rooms are missing, because the data is inconsistent:
4  -- Their address was 'South Campus II'.
5  SELECT room.building_number, room_number FROM room
6         INNER JOIN building ON
7         room.building_number = building.building_number
8         WHERE address = 'South Campus 2';
```

```
1  $ psql "postgres://postgres:XXX@localhost/fixed" -v ON_ERROR_STOP=1 -ebf
   ↪ select_a_1.sql
2  building_number | room_number
3  -----+-----
4  (0 rows)
5
6  # psql 16.11 succeeded with exit code 0.
```

# Daten Ändern (1)



- Wenn wir uns die notwendigen Schritte anschauen, um diese Informationen zu bekommen, dann sehen wir den Nachteil der 2NF: Wir haben nun eine kompliziertere Anfrage.
- Vorhin war die Gebäudeadresse zusammen mit den Raumnummern in einer Zeile gespeichert.
- Das ist nicht mehr so.
- Die Rauminformation ist in Tabelle `room` gespeichert.
- Die Gebäudeadresse ist in Tabelle `building` gespeichert.

```
1  /** Find all the rooms available in South Campus 2. */
2
3  -- Several rooms are missing, because the data is inconsistent:
4  -- Their address was 'South Campus II'.
5  SELECT room.building_number, room_number FROM room
6         INNER JOIN building ON
7             room.building_number = building.building_number
8         WHERE address = 'South Campus 2';
```

```
1  $ psql "postgres://postgres:XXX@localhost/fixed" -v ON_ERROR_STOP=1 -ebf
   ↪ select_a_1.sql
2  building_number | room_number
3  -----+-----
4  (0 rows)
5
6  # psql 16.11 succeeded with exit code 0.
```

# Daten Ändern (1)



- Vorhin war die Gebäudeadresse zusammen mit den Raumnummern in einer Zeile gespeichert.
- Das ist nicht mehr so.
- Die Rauminformation ist in Tabelle `room` gespeichert.
- Die Gebäudeadresse ist in Tabelle `building` gespeichert.
- Wenn wir wissen wollen, ob ein Raum in South Campus 2 ist, dann müssen wir die Daten dieser beiden Tabellen kombinieren.

```
1  /** Find all the rooms available in South Campus 2. */
2
3  -- Several rooms are missing, because the data is inconsistent:
4  -- Their address was 'South Campus II'.
5  SELECT room.building_number, room_number FROM room
6         INNER JOIN building ON
7             room.building_number = building.building_number
8         WHERE address = 'South Campus 2';
```

```
1  $ psql "postgres://postgres:XXX@localhost/fixed" -v ON_ERROR_STOP=1 -ebf
2  ↪ select_a_1.sql
3  building_number | room_number
4  -----+-----
5  (0 rows)
6
7  # psql 16.11 succeeded with exit code 0.
```

# Daten Ändern (1)



- Das ist nicht mehr so.
- Die Rauminformation ist in Tabelle `room` gespeichert.
- Die Gebäudeadresse ist in Tabelle `building` gespeichert.
- Wenn wir wissen wollen, ob ein Raum in South Campus 2 ist, dann müssen wir die Daten dieser beiden Tabellen kombinieren.
- Ein `INNER JOIN` macht das.

```
1  /** Find all the rooms available in South Campus 2. */
2
3  -- Several rooms are missing, because the data is inconsistent:
4  -- Their address was 'South Campus II'.
5  SELECT room.building_number, room_number FROM room
6         INNER JOIN building ON
7         room.building_number = building.building_number
8         WHERE address = 'South Campus 2';

1  $ psql "postgres://postgres:XXX@localhost/fixed" -v ON_ERROR_STOP=1 -ef
   ↪ select_a_1.sql
2  building_number | room_number
3  -----+-----
4  (0 rows)
5
6  # psql 16.11 succeeded with exit code 0.
```

# Daten Ändern (1)



- Die Rauminformation ist in Tabelle `room` gespeichert.
- Die Gebäudeadresse ist in Tabelle `building` gespeichert.
- Wenn wir wissen wollen, ob ein Raum in South Campus 2 ist, dann müssen wir die Daten dieser beiden Tabellen kombinieren.
- Ein `INNER JOIN` macht das.
- Der Primärschlüssel von Tabelle `building` ist `building_number`.

```
1  /** Find all the rooms available in South Campus 2. */
2
3  -- Several rooms are missing, because the data is inconsistent:
4  -- Their address was 'South Campus II'.
5  SELECT room.building_number, room_number FROM room
6         INNER JOIN building ON
7             room.building_number = building.building_number
8         WHERE address = 'South Campus 2';
```

```
1  $ psql "postgres://postgres:XXX@localhost/fixed" -v ON_ERROR_STOP=1 -ebf
   ↪ select_a_1.sql
2  building_number | room_number
3  -----+-----
4  (0 rows)
5
6  # psql 16.11 succeeded with exit code 0.
```

# Daten Ändern (1)



- Die Gebäudeadresse ist in Tabelle `building` gespeichert.
- Wenn wir wissen wollen, ob ein Raum in South Campus 2 ist, dann müssen wir die Daten dieser beiden Tabellen kombinieren.
- Ein `INNER JOIN` macht das.
- Der Primärschlüssel von Tabelle `building` ist `building_number`.
- Die `building_number` ist Fremdschlüssel in Tabelle `room` (und auch Teil von deren Primärschlüssel).

```
1  /** Find all the rooms available in South Campus 2. */
2
3  -- Several rooms are missing, because the data is inconsistent:
4  -- Their address was 'South Campus II'.
5  SELECT room.building_number, room_number FROM room
6         INNER JOIN building ON
7         room.building_number = building.building_number
8         WHERE address = 'South Campus 2';
```

```
1  $ psql "postgres://postgres:XXX@localhost/fixed" -v ON_ERROR_STOP=1 -ebf
2      ↪ select_a_1.sql
3      building_number | room_number
4      -----+-----
5      (0 rows)
6
7  # psql 16.11 succeeded with exit code 0.
```

# Daten Ändern (1)



- Wenn wir wissen wollen, ob ein Raum in South Campus 2 ist, dann müssen wir die Daten dieser beiden Tabellen kombinieren.
- Ein `INNER JOIN` macht das.
- Der Primärschlüssel von Tabelle `building` ist `building_number`.
- Die `building_number` ist Fremdschlüssel in Tabelle `room` (und auch Teil von deren Primärschlüssel).
- Die `JOIN`-Bedingung `room.building_number = building.building_number` erlaubt uns also, die richtige Zeile in Tabelle `building` für jede Zeile in Tabelle `room` zu finden.

```
1  /** Find all the rooms available in South Campus 2. */
2
3  -- Several rooms are missing, because the data is inconsistent:
4  -- Their address was 'South Campus II'.
5  SELECT room.building_number, room_number FROM room
6         INNER JOIN building ON
7         room.building_number = building.building_number
8         WHERE address = 'South Campus 2';

1  $ psql "postgres://postgres:XXX@localhost/fixed" -v ON_ERROR_STOP=1 -ebf
   ↪ select_a_1.sql
   building_number | room_number
   -----+-----
2
3  (0 rows)
4
5
6  # psql 16.11 succeeded with exit code 0.
```

# Daten Ändern (1)



- Ein `INNER JOIN` macht das.
- Der Primärschlüssel von Tabelle `building` ist `building_number`.
- Die `building_number` ist Fremdschlüssel in Tabelle `room` (und auch Teil von deren Primärschlüssel).
- Die `JOIN`-Bedingung `room.building_number = building.building_number` erlaubt uns also, die richtige Zeile in Tabelle `building` für jede Zeile in Tabelle `room` zu finden.
- Wir führen diese etwas komplexere Anfrage aus.

```
1  /** Find all the rooms available in South Campus 2. */
2
3  -- Several rooms are missing, because the data is inconsistent:
4  -- Their address was 'South Campus II'.
5  SELECT room.building_number, room_number FROM room
6         INNER JOIN building ON
7         room.building_number = building.building_number
8         WHERE address = 'South Campus 2';
```

```
1  $ psql "postgres://postgres:XXX@localhost/fixed" -v ON_ERROR_STOP=1 -ebf
2  ↪ select_a_1.sql
3  building_number | room_number
4  -----+-----
5  (0 rows)
6
7  # psql 16.11 succeeded with exit code 0.
```

# Daten Ändern (1)



- Der Primärschlüssel von Tabelle `building` ist `building_number`.
- Die `building_number` ist Fremdschlüssel in Tabelle `room` (und auch Teil von deren Primärschlüssel).
- Die `JOIN`-Bedingung `room.building_number = building.building_number` erlaubt uns also, die richtige Zeile in Tabelle `building` für jede Zeile in Tabelle `room` zu finden.
- Wir führen diese etwas komplexere Anfrage aus.
- Überraschender Weise bekommen wir gar kein Ergebnis.

```
1  /** Find all the rooms available in South Campus 2. */
2
3  -- Several rooms are missing, because the data is inconsistent:
4  -- Their address was 'South Campus II'.
5  SELECT room.building_number, room_number FROM room
6         INNER JOIN building ON
7             room.building_number = building.building_number
8         WHERE address = 'South Campus 2';

1  $ psql "postgres://postgres:XXX@localhost/fixed" -v ON_ERROR_STOP=1 -ebf
   ↪ select_a_1.sql
2  building_number | room_number
3  -----+-----
4  (0 rows)
5
6  # psql 16.11 succeeded with exit code 0.
```

# Daten Ändern (1)



- Die `building_number` ist Fremdschlüssel in Tabelle `room` (und auch Teil von deren Primärschlüssel).
- Die `JOIN`-Bedingung `room.building_number = building.building_number` erlaubt uns also, die richtige Zeile in Tabelle `building` für jede Zeile in Tabelle `room` zu finden.
- Wir führen diese etwas komplexere Anfrage aus.
- Überraschender Weise bekommen wir gar kein Ergebnis.
- Es gibt keine Räume in South Campus 2??

```
1  /** Find all the rooms available in South Campus 2. */
2
3  -- Several rooms are missing, because the data is inconsistent:
4  -- Their address was 'South Campus II'.
5  SELECT room.building_number, room_number FROM room
6         INNER JOIN building ON
7             room.building_number = building.building_number
8         WHERE address = 'South Campus 2';

1  $ psql "postgres://postgres:XXX@localhost/fixed" -v ON_ERROR_STOP=1 -ebf
   ↪ select_a_1.sql
2  building_number | room_number
3  -----+-----
4  (0 rows)
5
6  # psql 16.11 succeeded with exit code 0.
```

# Daten Ändern (1)



- Die **JOIN**-Bedingung

`room.building_number = building.building_number` erlaubt uns also, die richtige Zeile in Tabelle `building` für jede Zeile in Tabelle `room` zu finden.

- Wir führen diese etwas komplexere Anfrage aus.
- Überraschender Weise bekommen wir gar kein Ergebnis.
- Es gibt keine Räume in South Campus 2??
- Gut, wir haben die beiden Räume von Gebäude 53 gelöscht, welche ja in South Campus 2 waren.

```
1  /** Find all the rooms available in South Campus 2. */
2
3  -- Several rooms are missing, because the data is inconsistent:
4  -- Their address was 'South Campus II'.
5  SELECT room.building_number, room_number FROM room
6         INNER JOIN building ON
7             room.building_number = building.building_number
8         WHERE address = 'South Campus 2';

1  $ psql "postgres://postgres:XXX@localhost/fixed" -v ON_ERROR_STOP=1 -ef
   ↪ select_a_1.sql
2  building_number | room_number
3  -----+-----
4  (0 rows)
5
6  # psql 16.11 succeeded with exit code 0.
```

# Daten Ändern (1)



- Wir führen diese etwas komplexere Anfrage aus.
- Überraschender Weise bekommen wir gar kein Ergebnis.
- Es gibt keine Räume in South Campus 2??
- Gut, wir haben die beiden Räume von Gebäude 53 gelöscht, welche ja in South Campus 2 waren.
- Also die sind weg.

```
1  /** Find all the rooms available in South Campus 2. */
2
3  -- Several rooms are missing, because the data is inconsistent:
4  -- Their address was 'South Campus II'.
5  SELECT room.building_number, room_number FROM room
6         INNER JOIN building ON
7             room.building_number = building.building_number
8         WHERE address = 'South Campus 2';
```

```
1  $ psql "postgres://postgres:XXX@localhost/fixed" -v ON_ERROR_STOP=1 -bf
2  ↪ select_a_1.sql
3  building_number | room_number
4  -----+-----
5  (0 rows)
6
7  # psql 16.11 succeeded with exit code 0.
```

# Daten Ändern (1)



- Überraschender Weise bekommen wir gar kein Ergebnis.
- Es gibt keine Räume in South Campus 2??
- Gut, wir haben die beiden Räume von Gebäude 53 gelöscht, welche ja in South Campus 2 waren.
- Also die sind weg.
- Aber es gibt dort immer noch Gebäude 36 mit seinen drei Räumen.

```
1  /** Find all the rooms available in South Campus 2. */
2
3  -- Several rooms are missing, because the data is inconsistent:
4  -- Their address was 'South Campus II'.
5  SELECT room.building_number, room_number FROM room
6         INNER JOIN building ON
7             room.building_number = building.building_number
8         WHERE address = 'South Campus 2';
```

```
1  $ psql "postgres://postgres:XXX@localhost/fixed" -v ON_ERROR_STOP=1 -ef
2  ↪ select_a_1.sql
3  building_number | room_number
4  -----+-----
5  (0 rows)
6
7  # psql 16.11 succeeded with exit code 0.
```

# Daten Ändern (1)



- Es gibt keine Räume in South Campus 2??
- Gut, wir haben die beiden Räume von Gebäude 53 gelöscht, welche ja in South Campus 2 waren.
- Also die sind weg.
- Aber es gibt dort immer noch Gebäude 36 mit seinen drei Räumen.
- Bei genauerer Betrachtung sehen wir, dass wir wieder die Adressen falsch geschrieben haben.

```
1  /** Insert data into the database. */
2
3  -- Insert several room records.
4  INSERT INTO building (building_number, building_name, address) VALUES
5      ('36', 'CS Teaching Building', 'South Campus II'),
6      ('10', 'Language Teaching Building', 'South Campus 1'),
7      ('53', 'Comprehensive Experimental Building', 'South Campus 2'),
8      ('7', 'Main Teaching Building', 'South Campus 1');
9
10 -- Insert several room records.
11 INSERT INTO room (building_number, room_number, room_name,
12                  capacity) VALUES
13      ('36', '305', 'Meeting Room', 40),
14      ('36', '105', 'Lecture Room 1', 80),
15      ('10', '100', 'Teaching Room A', 30),
16      ('10', '102', 'Teaching Room B', 30),
17      ('53', '904a', 'Office 1', 10),
18      ('53', '904b', 'Cluster Room', 3),
19      ('7', '200', 'Auditorium', 120),
20      ('36', '106', 'Lecture Room 2', 80);
```

```
1  $ psql "postgres://postgres:XXX@localhost/fixed" -v ON_ERROR_STOP=1 -ebf
   ↪ insert.sql
2  INSERT 0 4
3  INSERT 0 8
4  # psql 16.11 succeeded with exit code 0.
```

# Daten Ändern (1)



- Gut, wir haben die beiden Räume von Gebäude 53 gelöscht, welche ja in South Campus 2 waren.
- Also die sind weg.
- Aber es gibt dort immer noch Gebäude 36 mit seinen drei Räumen.
- Bei genauerer Betrachtung sehen wir, dass wir wieder die Adressen falsch geschrieben haben.
- Gebäude 36 ist in *South Campus II* anstatt von *South Campus 2*.

```
1  /** Insert data into the database. */
2
3  -- Insert several room records.
4  INSERT INTO building (building_number, building_name, address) VALUES
5      ('36', 'CS Teaching Building', 'South Campus II'),
6      ('10', 'Language Teaching Building', 'South Campus 1'),
7      ('53', 'Comprehensive Experimental Building', 'South Campus 2'),
8      ('7', 'Main Teaching Building', 'South Campus 1');
9
10 -- Insert several room records.
11 INSERT INTO room (building_number, room_number, room_name,
12                  capacity) VALUES
13     ('36', '305', 'Meeting Room', 40),
14     ('36', '105', 'Lecture Room 1', 80),
15     ('10', '100', 'Teaching Room A', 30),
16     ('10', '102', 'Teaching Room B', 30),
17     ('53', '904a', 'Office 1', 10),
18     ('53', '904b', 'Cluster Room', 3),
19     ('7', '200', 'Auditorium', 120),
20     ('36', '106', 'Lecture Room 2', 80);
```

```
1  $ psql "postgres://postgres:XXX@localhost/fixed" -v ON_ERROR_STOP=1 -ebf
   ↳ insert.sql
2  INSERT 0 4
3  INSERT 0 8
4  # psql 16.11 succeeded with exit code 0.
```

# Daten Ändern (1)



- Also die sind weg.
- Aber es gibt dort immer noch Gebäude 36 mit seinen drei Räumen.
- Bei genauerer Betrachtung sehen wir, dass wir wieder die Adressen falsch geschrieben haben.
- Gebäude 36 ist in *South Campus II* anstatt von *South Campus 2*.
- Weil wir den Gebäudedatensatz nur einmal gespeichert haben, betrifft das alle Räume darin.

```
1  /** Insert data into the database. */
2
3  -- Insert several room records.
4  INSERT INTO building (building_number, building_name, address) VALUES
5      ('36', 'CS Teaching Building', 'South Campus II'),
6      ('10', 'Language Teaching Building', 'South Campus 1'),
7      ('53', 'Comprehensive Experimental Building', 'South Campus 2'),
8      ('7', 'Main Teaching Building', 'South Campus 1');
9
10 -- Insert several room records.
11 INSERT INTO room (building_number, room_number, room_name,
12                  capacity) VALUES
13     ('36', '305', 'Meeting Room', 40),
14     ('36', '105', 'Lecture Room 1', 80),
15     ('10', '100', 'Teaching Room A', 30),
16     ('10', '102', 'Teaching Room B', 30),
17     ('53', '904a', 'Office 1', 10),
18     ('53', '904b', 'Cluster Room', 3),
19     ('7', '200', 'Auditorium', 120),
20     ('36', '106', 'Lecture Room 2', 80);
```

```
1  $ psql "postgres://postgres:XXX@localhost/fixed" -v ON_ERROR_STOP=1 -ebf
   ↳ insert.sql
2  INSERT 0 4
3  INSERT 0 8
4  # psql 16.11 succeeded with exit code 0.
```

# Daten Ändern (1)



- Aber es gibt dort immer noch Gebäude 36 mit seinen drei Räumen.
- Bei genauerer Betrachtung sehen wir, dass wir wieder die Adressen falsch geschrieben haben.
- Gebäude 36 ist in *South Campus II* anstatt von *South Campus 2*.
- Weil wir den Gebäudedatensatz nur einmal gespeichert haben, betrifft das alle Räume darin.
- Dieses Problem kann leicht gelöst werden, genau wie vorhin.

```
1  /** Find all the rooms available in South Campus 2. */
2
3  -- Several rooms are missing, because the data is inconsistent:
4  -- Their address was 'South Campus II'.
5  SELECT room.building_number, room_number FROM room
6         INNER JOIN building ON
7             room.building_number = building.building_number
8         WHERE address = 'South Campus 2';
9
10 -- Now we get these rooms.
11 SELECT room.building_number, room_number FROM room
12         INNER JOIN building ON
13             room.building_number = building.building_number
14         WHERE address = 'South Campus 2' OR address = 'South Campus II';
```

```
1  $ psql "postgres://postgres:XXX@localhost/fixed" -v ON_ERROR_STOP=1 -ebf
2  ↵ select_a_2.sql
3  building_number | room_number
4  -----+-----
5  (0 rows)
6
7  building_number | room_number
8  -----+-----
9  36               | 305
10 36               | 105
11 36               | 106
12 (3 rows)
13
14 # psql 16.11 succeeded with exit code 0.
```

# Daten Ändern (1)



- Bei genauerer Betrachtung sehen wir, dass wir wieder die Adressen falsch geschrieben haben.
- Gebäude 36 ist in *South Campus II* anstatt von *South Campus 2*.
- Weil wir den Gebäudedatensatz nur einmal gespeichert haben, betrifft das alle Räume darin.
- Dieses Problem kann leicht gelöst werden, genau wie vorhin.
- Wir können unser Auswahlkriterium so ändern, dass auch South Campus II mit darin ist.

```
1  /** Find all the rooms available in South Campus 2. */
2
3  -- Several rooms are missing, because the data is inconsistent:
4  -- Their address was 'South Campus II'.
5  SELECT room.building_number, room_number FROM room
6         INNER JOIN building ON
7             room.building_number = building.building_number
8         WHERE address = 'South Campus 2';
9
10 -- Now we get these rooms.
11 SELECT room.building_number, room_number FROM room
12         INNER JOIN building ON
13             room.building_number = building.building_number
14         WHERE address = 'South Campus 2' OR address = 'South Campus II';
```

```
1  $ psql "postgres://postgres:XXX@localhost/fixed" -v ON_ERROR_STOP=1 -ebf
2  ↵ select_a_2.sql
3  building_number | room_number
4  -----+-----
5  (0 rows)
6
7  building_number | room_number
8  -----+-----
9  36               | 305
10 36               | 105
11 36               | 106
12 (3 rows)
13
14 # psql 16.11 succeeded with exit code 0.
```

# Daten Ändern (1)



- Gebäude 36 ist in *South Campus II* anstatt von *South Campus 2*.
- Weil wir den Gebäudedatensatz nur einmal gespeichert haben, betrifft das alle Räume darin.
- Dieses Problem kann leicht gelöst werden, genau wie vorhin.
- Wir können unser Auswahlkriterium so ändern, dass auch South Campus II mit darin ist.
- Aber das ist nicht zufriedenstellend.

```
1  /** Find all the rooms available in South Campus 2. */
2
3  -- Several rooms are missing, because the data is inconsistent:
4  -- Their address was 'South Campus II'.
5  SELECT room.building_number, room_number FROM room
6         INNER JOIN building ON
7             room.building_number = building.building_number
8         WHERE address = 'South Campus 2';
9
10 -- Now we get these rooms.
11 SELECT room.building_number, room_number FROM room
12         INNER JOIN building ON
13             room.building_number = building.building_number
14         WHERE address = 'South Campus 2' OR address = 'South Campus II';
```

```
1  $ psql "postgres://postgres:XXX@localhost/fixed" -v ON_ERROR_STOP=1 -ebf
2  ↵ select_a_2.sql
3  building_number | room_number
4  -----+-----
5  (0 rows)
6
7  building_number | room_number
8  -----+-----
9  36               | 305
10 36               | 105
11 36               | 106
12 (3 rows)
13
14 # psql 16.11 succeeded with exit code 0.
```

# Daten Ändern (1)

- Weil wir den Gebäudedatensatz nur einmal gespeichert haben, betrifft das alle Räume darin.
- Dieses Problem kann leicht gelöst werden, genau wie vorhin.
- Wir können unser Auswahlkriterium so ändern, dass auch South Campus II mit darin ist.
- Aber das ist nicht zufriedenstellend.
- Wir machen ein Update, in dem wir `SET address = 'South Campus 2'` auf alle Zeilen in Tabelle `building` anwenden, für die `address = 'South Campus II'` gilt.

```
1  /** Find all the rooms available in South Campus 2. */
2
3  -- Several rooms are missing, because the data is inconsistent:
4  -- Their address was 'South Campus II'.
5  SELECT room.building_number, room_number FROM room
6         INNER JOIN building ON
7             room.building_number = building.building_number
8         WHERE address = 'South Campus 2';
9
10 -- Now we get these rooms.
11 SELECT room.building_number, room_number FROM room
12         INNER JOIN building ON
13             room.building_number = building.building_number
14         WHERE address = 'South Campus 2' OR address = 'South Campus II';
15
16 -- We can fix this problem with an UPDATE instruction.
17 UPDATE building SET address = 'South Campus 2'
18         WHERE address = 'South Campus II' RETURNING building_number;
19
20 -- This query now also works, because the addresses are now consistent.
21 SELECT room.building_number, room_number FROM room
22         INNER JOIN building ON
23             room.building_number = building.building_number
24         WHERE address = 'South Campus 2';
```

```
1  $ psql "postgres://postgres:XXX@localhost/fixed" -v ON_ERROR_STOP=1 -ef
2  ↪ update_a.sql
3  building_number | room_number
4  -----+-----
5  (0 rows)
6
7  building_number | room_number
8  -----+-----
9  36               | 305
10 36               | 105
11 36               | 106
12 (3 rows)
13
14 building_number
15 -----
16 36
17 (1 row)
18
19 UPDATE 1
20 building_number | room_number
21 -----+-----
22 36               | 105
23 36               | 106
24 36               | 305
25 (3 rows)
26
27 # psql 16.11 succeeded with exit code 0.
```



# Daten Ändern (1)

- Dieses Problem kann leicht gelöst werden, genau wie vorher.
- Wir können unser Auswahlkriterium so ändern, dass auch South Campus II mit darin ist.
- Aber das ist nicht zufriedenstellend.
- Wir machen ein Update, in dem wir `SET address = 'South Campus 2'` auf alle Zeilen in Tabelle `building` anwenden, für die `address = 'South Campus II'` gilt.
- Wir liefern uns auch die Gebäudenummer der betroffenen Gebäude.

```
1  /** Find all the rooms available in South Campus 2. */
2
3  -- Several rooms are missing, because the data is inconsistent:
4  -- Their address was 'South Campus II'.
5  SELECT room.building_number, room_number FROM room
6         INNER JOIN building ON
7             room.building_number = building.building_number
8         WHERE address = 'South Campus 2';
9
10 -- Now we get these rooms.
11 SELECT room.building_number, room_number FROM room
12         INNER JOIN building ON
13             room.building_number = building.building_number
14         WHERE address = 'South Campus 2' OR address = 'South Campus II';
15
16 -- We can fix this problem with an UPDATE instruction.
17 UPDATE building SET address = 'South Campus 2'
18         WHERE address = 'South Campus II' RETURNING building_number;
19
20 -- This query now also works, because the addresses are now consistent.
21 SELECT room.building_number, room_number FROM room
22         INNER JOIN building ON
23             room.building_number = building.building_number
24         WHERE address = 'South Campus 2';
```

```
1  $ psql "postgres://postgres:XXX@localhost/fixed" -v ON_ERROR_STOP=1 -ebf
2  ↵ update_a.sql
3  building_number | room_number
4  -----+-----
5  (0 rows)
6
7  building_number | room_number
8  -----+-----
9  36               | 305
10 36               | 105
11 36               | 106
12 (3 rows)
13
14 building_number
15 -----
16 36
17 (1 row)
18
19 UPDATE 1
20 building_number | room_number
21 -----+-----
22 36               | 105
23 36               | 106
24 36               | 305
25 (3 rows)
26
27 # psql 16.11 succeeded with exit code 0.
```



# Daten Ändern (1)

- Wir können unser Auswahlkriterium so ändern, dass auch South Campus II mit darin ist.
- Aber das ist nicht zufriedenstellend.
- Wir machen ein Update, in dem wir `SET address = 'South Campus 2'` auf alle Zeilen in Tabelle `building` anwenden, für die `address = 'South Campus II'` gilt.
- Wir liefern uns auch die Gebäudenummer der betroffenen Gebäude.
- Wir sehen, dass nur ein Datensatz geändert wurde.

```
1  /** Find all the rooms available in South Campus 2. */
2
3  -- Several rooms are missing, because the data is inconsistent:
4  -- Their address was 'South Campus II'.
5  SELECT room.building_number, room_number FROM room
6         INNER JOIN building ON
7         room.building_number = building.building_number
8         WHERE address = 'South Campus 2';
9
10 -- Now we get these rooms.
11 SELECT room.building_number, room_number FROM room
12         INNER JOIN building ON
13         room.building_number = building.building_number
14         WHERE address = 'South Campus 2' OR address = 'South Campus II';
15
16 -- We can fix this problem with an UPDATE instruction.
17 UPDATE building SET address = 'South Campus 2'
18        WHERE address = 'South Campus II' RETURNING building_number;
19
20 -- This query now also works, because the addresses are now consistent.
21 SELECT room.building_number, room_number FROM room
22         INNER JOIN building ON
23         room.building_number = building.building_number
24         WHERE address = 'South Campus 2';

```

```
1  $ psql "postgres://postgres:XXX@localhost/fixed" -v ON_ERROR_STOP=1 -ebf
2  ↪ update_a.sql
3  building_number | room_number
4  -----+-----
5  (0 rows)
6
7  building_number | room_number
8  -----+-----
9  36               | 305
10 36               | 105
11 36               | 106
12 (3 rows)
13
14 building_number
15 -----
16 36
17 (1 row)
18
19 UPDATE 1
20 building_number | room_number
21 -----+-----
22 36               | 105
23 36               | 106
24 36               | 305
25 (3 rows)
26
27 # psql 16.11 succeeded with exit code 0.

```



# Daten Ändern (1)

- Aber das ist nicht zufriedenstellend.
- Wir machen ein Update, in dem wir `SET address = 'South Campus 2'` auf alle Zeilen in Tabelle `building` anwenden, für die `address = 'South Campus II'` gilt.
- Wir liefern uns auch die Gebäudenummer der betroffenen Gebäude.
- Wir sehen, dass nur ein Datensatz geändert wurde.
- Vorhin mussten wir zwei Datensätze ändern.

```
1  /** Find all the rooms available in South Campus 2. */
2
3  -- Several rooms are missing, because the data is inconsistent:
4  -- Their address was 'South Campus II'.
5  SELECT room.building_number, room_number FROM room
6         INNER JOIN building ON
7             room.building_number = building.building_number
8         WHERE address = 'South Campus 2';
9
10 -- Now we get these rooms.
11 SELECT room.building_number, room_number FROM room
12         INNER JOIN building ON
13             room.building_number = building.building_number
14         WHERE address = 'South Campus 2' OR address = 'South Campus II';
15
16 -- We can fix this problem with an UPDATE instruction.
17 UPDATE building SET address = 'South Campus 2'
18        WHERE address = 'South Campus II' RETURNING building_number;
19
20 -- This query now also works, because the addresses are now consistent.
21 SELECT room.building_number, room_number FROM room
22         INNER JOIN building ON
23             room.building_number = building.building_number
24        WHERE address = 'South Campus 2';
```

```
1  $ psql "postgres://postgres:XXX@localhost/fixe" -v ON_ERROR_STOP=1 -ef
2  ↪ update_a.sql
3  building_number | room_number
4  -----+-----
5  (0 rows)
6
7  building_number | room_number
8  -----+-----
9  36               | 305
10 36               | 105
11 36               | 106
12 (3 rows)
13
14 building_number
15 -----
16 36
17 (1 row)
18
19 UPDATE 1
20 building_number | room_number
21 -----+-----
22 36               | 105
23 36               | 106
24 36               | 305
25 (3 rows)
26
27 # psql 16.11 succeeded with exit code 0.
```



# Daten Ändern (1)

- Wir machen ein Update, in dem wir `SET address = 'South Campus 2'` auf alle Zeilen in Tabelle `building` anwenden, für die `address = 'South Campus II'` gilt.
- Wir liefern uns auch die Gebäudenummer der betroffenen Gebäude.
- Wir sehen, dass nur ein Datensatz geändert wurde.
- Vorhin mussten wir zwei Datensätze ändern.
- Das war ein Beispiel einer Update-Anomalie.

```
1  /** Find all the rooms available in South Campus 2. */
2
3  -- Several rooms are missing, because the data is inconsistent:
4  -- Their address was 'South Campus II'.
5  SELECT room.building_number, room_number FROM room
6         INNER JOIN building ON
7             room.building_number = building.building_number
8         WHERE address = 'South Campus 2';
9
10 -- Now we get these rooms.
11 SELECT room.building_number, room_number FROM room
12        INNER JOIN building ON
13            room.building_number = building.building_number
14        WHERE address = 'South Campus 2' OR address = 'South Campus II';
15
16 -- We can fix this problem with an UPDATE instruction.
17 UPDATE building SET address = 'South Campus 2'
18        WHERE address = 'South Campus II' RETURNING building_number;
19
20 -- This query now also works, because the addresses are now consistent.
21 SELECT room.building_number, room_number FROM room
22        INNER JOIN building ON
23            room.building_number = building.building_number
24        WHERE address = 'South Campus 2';
```

```
1  $ psql "postgres://postgres:XXX@localhost/fixe" -v ON_ERROR_STOP=1 -ef
2  ↪ update_a.sql
3  building_number | room_number
4  -----+-----
5  (0 rows)
6
7  building_number | room_number
8  -----+-----
9  36               | 305
10 36               | 105
11 36               | 106
12 (3 rows)
13
14 building_number
15 -----
16 36
17 (1 row)
18
19 UPDATE 1
20 building_number | room_number
21 -----+-----
22 36               | 105
23 36               | 106
24 36               | 305
25 (3 rows)
26
27 # psql 16.11 succeeded with exit code 0.
```



# Daten Ändern (1)

- Wir liefern uns auch die Gebäudenummer der betroffenen Gebäude.
- Wir sehen, dass nur ein Datensatz geändert wurde.
- Vorhin mussten wir zwei Datensätze ändern.
- Das war ein Beispiel einer Update-Anomalie.
- Und die ist jetzt weg.

```
1  /** Find all the rooms available in South Campus 2. */
2
3  -- Several rooms are missing, because the data is inconsistent:
4  -- Their address was 'South Campus II'.
5  SELECT room.building_number, room_number FROM room
6         INNER JOIN building ON
7         room.building_number = building.building_number
8         WHERE address = 'South Campus 2';
9
10 -- Now we get these rooms.
11 SELECT room.building_number, room_number FROM room
12         INNER JOIN building ON
13         room.building_number = building.building_number
14         WHERE address = 'South Campus 2' OR address = 'South Campus II';
15
16 -- We can fix this problem with an UPDATE instruction.
17 UPDATE building SET address = 'South Campus 2'
18         WHERE address = 'South Campus II' RETURNING building_number;
19
20 -- This query now also works, because the addresses are now consistent.
21 SELECT room.building_number, room_number FROM room
22         INNER JOIN building ON
23         room.building_number = building.building_number
24         WHERE address = 'South Campus 2';
```

```
1  $ psql "postgres://postgres:XXX@localhost/fixed" -v ON_ERROR_STOP=1 -ebf
2  ↪ update_a.sql
3  building_number | room_number
4  -----+-----
5  (0 rows)
6
7  building_number | room_number
8  -----+-----
9  36               | 305
10 36               | 105
11 36               | 106
12 (3 rows)
13
14 building_number
15 -----
16 36
17 (1 row)
18
19 UPDATE 1
20 building_number | room_number
21 -----+-----
22 36               | 105
23 36               | 106
24 36               | 305
25 (3 rows)
26
27 # psql 16.11 succeeded with exit code 0.
```



# Daten Ändern (1)

- Wir sehen, dass nur ein Datensatz geändert wurde.
- Vorhin mussten wir zwei Datensätze ändern.
- Das war ein Beispiel einer Update-Anomalie.
- Und die ist jetzt weg.
- Nach dem Update funktioniert die Anfrage wie erwartet.

```
1  /** Find all the rooms available in South Campus 2. */
2
3  -- Several rooms are missing, because the data is inconsistent:
4  -- Their address was 'South Campus II'.
5  SELECT room.building_number, room_number FROM room
6         INNER JOIN building ON
7         room.building_number = building.building_number
8         WHERE address = 'South Campus 2';
9
10 -- Now we get these rooms.
11 SELECT room.building_number, room_number FROM room
12         INNER JOIN building ON
13         room.building_number = building.building_number
14         WHERE address = 'South Campus 2' OR address = 'South Campus II';
15
16 -- We can fix this problem with an UPDATE instruction.
17 UPDATE building SET address = 'South Campus 2'
18        WHERE address = 'South Campus II' RETURNING building_number;
19
20 -- This query now also works, because the addresses are now consistent.
21 SELECT room.building_number, room_number FROM room
22         INNER JOIN building ON
23         room.building_number = building.building_number
24         WHERE address = 'South Campus 2';
```

```
1  $ psql "postgres://postgres:XXX@localhost/fixe" -v ON_ERROR_STOP=1 -ebl
2  ↵ update_a.sql
3  building_number | room_number
4  -----+-----
5  (0 rows)
6
7  building_number | room_number
8  -----+-----
9  36               | 305
10 36               | 105
11 36               | 106
12 (3 rows)
13
14 building_number
15 -----
16 36
17 (1 row)
18
19 UPDATE 1
20 building_number | room_number
21 -----+-----
22 36               | 105
23 36               | 106
24 36               | 305
25 (3 rows)
26
27 # psql 16.11 succeeded with exit code 0.
```



# Daten Ändern (1)

- Vorhin mussten wir zwei Datensätze ändern.
- Das war ein Beispiel einer Update-Anomalie.
- Und die ist jetzt weg.
- Nach dem Update funktioniert die Anfrage wie erwartet.
- Randnotiz: Der Fakt, dass das Problem mit der Adresse wieder auftauchen konnte, könnte so interpretiert werden, dass wie die Gebäudeadressen vielleicht in eine zusätzliche Tabelle `building_address` tun könnten.

```
1  /** Find all the rooms available in South Campus 2. */
2
3  -- Several rooms are missing, because the data is inconsistent:
4  -- Their address was 'South Campus II'.
5  SELECT room.building_number, room_number FROM room
6         INNER JOIN building ON
7         room.building_number = building.building_number
8         WHERE address = 'South Campus 2';
9
10 -- Now we get these rooms.
11 SELECT room.building_number, room_number FROM room
12        INNER JOIN building ON
13        room.building_number = building.building_number
14        WHERE address = 'South Campus 2' OR address = 'South Campus II';
15
16 -- We can fix this problem with an UPDATE instruction.
17 UPDATE building SET address = 'South Campus 2'
18        WHERE address = 'South Campus II' RETURNING building_number;
19
20 -- This query now also works, because the addresses are now consistent.
21 SELECT room.building_number, room_number FROM room
22        INNER JOIN building ON
23        room.building_number = building.building_number
24        WHERE address = 'South Campus 2';
```

```
1  $ psql "postgres://postgres:XXX@localhost/fixed" -v ON_ERROR_STOP=1 -ef
2  ↵ update_a.sql
3  building_number | room_number
4  -----+-----
5  (0 rows)
6
7  building_number | room_number
8  -----+-----
9  36               | 305
10 36               | 105
11 36               | 106
12 (3 rows)
13
14 building_number
15 -----
16 36
17 (1 row)
18
19 UPDATE 1
20 building_number | room_number
21 -----+-----
22 36               | 105
23 36               | 106
24 36               | 305
25 (3 rows)
26
27 # psql 16.11 succeeded with exit code 0.
```



# Daten Ändern (1)

- Das war ein Beispiel einer Update-Anomalie.
- Und die ist jetzt weg.
- Nach dem Update funktioniert die Anfrage wie erwartet.
- Randnotiz: Der Fakt, dass das Problem mit der Adresse wieder auftauchen konnte, könnte so interpretiert werden, dass wie die Gebäudeadressen vielleicht in eine zusätzliche Tabelle `building_address` tun könnten.
- Das würde das Problem komplett verhindern.

```
1  /** Find all the rooms available in South Campus 2. */
2
3  -- Several rooms are missing, because the data is inconsistent:
4  -- Their address was 'South Campus II'.
5  SELECT room.building_number, room_number FROM room
6         INNER JOIN building ON
7         room.building_number = building.building_number
8         WHERE address = 'South Campus 2';
9
10 -- Now we get these rooms.
11 SELECT room.building_number, room_number FROM room
12        INNER JOIN building ON
13        room.building_number = building.building_number
14        WHERE address = 'South Campus 2' OR address = 'South Campus II';
15
16 -- We can fix this problem with an UPDATE instruction.
17 UPDATE building SET address = 'South Campus 2'
18        WHERE address = 'South Campus II' RETURNING building_number;
19
20 -- This query now also works, because the addresses are now consistent.
21 SELECT room.building_number, room_number FROM room
22        INNER JOIN building ON
23        room.building_number = building.building_number
24        WHERE address = 'South Campus 2';
```

```
1  $ psql "postgres://postgres:XXX@localhost/fixed" -v ON_ERROR_STOP=1 -ebf
2  ↵ update_a.sql
3  building_number | room_number
4  -----+-----
5  (0 rows)
6
7  building_number | room_number
8  -----+-----
9  36               | 305
10 36               | 105
11 36               | 106
12 (3 rows)
13
14 building_number
15 -----
16 36
17 (1 row)
18
19 UPDATE 1
20 building_number | room_number
21 -----+-----
22 36               | 105
23 36               | 106
24 36               | 305
25 (3 rows)
26
27 # psql 16.11 succeeded with exit code 0.
```



# Daten Ändern (2)



- Das „3. Problem“ von vorhin war ein noch klareres Beispiel der Update-Anomalie.

```
1  /** Change the name of Building 36 to Computer Science Building. */
2
3  UPDATE building SET building_name = 'Computer Science Building'
4      WHERE address = 'South Campus 2' AND building_number = '36'
5      RETURNING building_number, building_name;
```

```
1  $ psql "postgres://postgres:XXX@localhost/fixed" -v ON_ERROR_STOP=1 -ebf
   ↪ update_b.sql
2  building_number |          building_name
3  -----+-----
4  36              | Computer Science Building
5  (1 row)
6
7  UPDATE 1
8  # psql 16.11 succeeded with exit code 0.
```

# Daten Ändern (2)



- Das „3. Problem“ von vorhin war ein noch klareres Beispiel der Update-Anomalie.
- Als wir den Name des Gebäudes 36 von *CS Teaching Building* auf *Computer Science Building* ändern wollten, mussten wir drei Zeilen anfassen.

```
1  /** Change the name of Building 36 to Computer Science Building. */
2
3  UPDATE building SET building_name = 'Computer Science Building'
4      WHERE address = 'South Campus 2' AND building_number = '36'
5      RETURNING building_number, building_name;
```

```
1  $ psql "postgres://postgres:XXX@localhost/fixed" -v ON_ERROR_STOP=1 -ebf
   ↪ update_b.sql
2  building_number |          building_name
3  -----+-----
4  36              | Computer Science Building
5  (1 row)
6
7  UPDATE 1
8  # psql 16.11 succeeded with exit code 0.
```

# Daten Ändern (2)



- Das „3. Problem“ von vorhin war ein noch klareres Beispiel der Update-Anomalie.
- Als wir den Name des Gebäudes 36 von *CS Teaching Building* auf *Computer Science Building* ändern wollten, mussten wir drei Zeilen anfassen.
- Wir wollten eine Information ändern, mussten aber drei Änderungen durchführen.

```
1  /** Change the name of Building 36 to Computer Science Building. */
2
3  UPDATE building SET building_name = 'Computer Science Building'
4      WHERE address = 'South Campus 2' AND building_number = '36'
5      RETURNING building_number, building_name;
```

```
1  $ psql "postgres://postgres:XXX@localhost/fixed" -v ON_ERROR_STOP=1 -ef
2      ↪ update_b.sql
3      building_number |          building_name
4      -----+-----
5      36              | Computer Science Building
6      (1 row)
7
8  UPDATE 1
9  # psql 16.11 succeeded with exit code 0.
```

# Daten Ändern (2)



- Das „3. Problem“ von vorhin war ein noch klareres Beispiel der Update-Anomalie.
- Als wir den Name des Gebäudes 36 von *CS Teaching Building* auf *Computer Science Building* ändern wollten, mussten wir drei Zeilen anfassen.
- Wir wollten eine Information ändern, mussten aber drei Änderungen durchführen.
- Jetzt, wo wir die 2NF einhalten, müssen wir nur noch genau eine Zeile in Tabelle `building` ändern.

```
1  /** Change the name of Building 36 to Computer Science Building. */
2
3  UPDATE building SET building_name = 'Computer Science Building'
4      WHERE address = 'South Campus 2' AND building_number = '36'
5      RETURNING building_number, building_name;
```

```
1  $ psql "postgres://postgres:XXX@localhost/fixed" -v ON_ERROR_STOP=1 -ebf
2      ↪ update_b.sql
3
4  building_number |      building_name
5  -----+-----
6  36              | Computer Science Building
7  (1 row)
8
9  UPDATE 1
10 # psql 16.11 succeeded with exit code 0.
```

# Daten Ändern (2)



- Das „3. Problem“ von vorhin war ein noch klareres Beispiel der Update-Anomalie.
- Als wir den Name des Gebäudes 36 von *CS Teaching Building* auf *Computer Science Building* ändern wollten, mussten wir drei Zeilen anfassen.
- Wir wollten eine Information ändern, mussten aber drei Änderungen durchführen.
- Jetzt, wo wir die 2NF einhalten, müssen wir nur noch genau eine Zeile in Tabelle `building` ändern.
- Die Anomalie ist weg.

```
1  /** Change the name of Building 36 to Computer Science Building. */
2
3  UPDATE building SET building_name = 'Computer Science Building'
4      WHERE address = 'South Campus 2' AND building_number = '36'
5      RETURNING building_number, building_name;
```

```
1  $ psql "postgres://postgres:XXX@localhost/fixed" -v ON_ERROR_STOP=1 -ebf
2      ↪ update_b.sql
3      building_number |          building_name
4      -----+-----
5      36              | Computer Science Building
6      (1 row)
7
8  UPDATE 1
9  # psql 16.11 succeeded with exit code 0.
```



# Zusammenfassung



## Zusammenfassung: Generell



- Die 2NF ist eine Regel, die uns dabei hilft, Redundanz zu verhindern.

## Zusammenfassung: Generell



- Die 2NF ist eine Regel, die uns dabei hilft, Redundanz zu verhindern.
- Sie besagt, dass es keine funktionale Abhängigkeit von nicht-Schlüssel-Attributen auf Teile von Schlüsseln geben soll.

## Zusammenfassung: Generell



- Die 2NF ist eine Regel, die uns dabei hilft, Redundanz zu verhindern.
- Sie besagt, dass es keine funktionale Abhängigkeit von nicht-Schlüssel-Attributen auf Teile von Schlüsseln geben soll.
- Als Ergebnis verringert sie die Wahrscheinlichkeit, Fehler bei der Dateneingabe zu machen, weil das Volumen der einzugebenden Daten kleiner wird.

## Zusammenfassung: Generell



- Die 2NF ist eine Regel, die uns dabei hilft, Redundanz zu verhindern.
- Sie besagt, dass es keine funktionale Abhängigkeit von nicht-Schlüssel-Attributen auf Teile von Schlüsseln geben soll.
- Als Ergebnis verringert sie die Wahrscheinlichkeit, Fehler bei der Dateneingabe zu machen, weil das Volumen der einzugebenden Daten kleiner wird.
- Die 2NF verringert auch Probleme, die durch Lösch- und Update-Anomalien auftreten können.

## Zusammenfassung: Generell



- Die 2NF ist eine Regel, die uns dabei hilft, Redundanz zu verhindern.
- Sie besagt, dass es keine funktionale Abhängigkeit von nicht-Schlüssel-Attributen auf Teile von Schlüsseln geben soll.
- Als Ergebnis verringert sie die Wahrscheinlichkeit, Fehler bei der Dateneingabe zu machen, weil das Volumen der einzugebenden Daten kleiner wird.
- Die 2NF verringert auch Probleme, die durch Lösch- und Update-Anomalien auftreten können.
- Sie macht allerdings unsere Anfragen etwas komplizierter.

## Zusammenfassung: Generell



- Die 2NF ist eine Regel, die uns dabei hilft, Redundanz zu verhindern.
- Sie besagt, dass es keine funktionale Abhängigkeit von nicht-Schlüssel-Attributen auf Teile von Schlüsseln geben soll.
- Als Ergebnis verringert sie die Wahrscheinlichkeit, Fehler bei der Dateneingabe zu machen, weil das Volumen der einzugebenden Daten kleiner wird.
- Die 2NF verringert auch Probleme, die durch Lösch- und Update-Anomalien auftreten können.
- Sie macht allerdings unsere Anfragen etwas komplizierter.
- Die 2NF zu implementieren bedeutet oft, Daten auf mehrere Tabellen zu verteilen.

# Zusammenfassung: Generell



- Die 2NF ist eine Regel, die uns dabei hilft, Redundanz zu verhindern.
- Sie besagt, dass es keine funktionale Abhängigkeit von nicht-Schlüssel-Attributen auf Teile von Schlüsseln geben soll.
- Als Ergebnis verringert sie die Wahrscheinlichkeit, Fehler bei der Dateneingabe zu machen, weil das Volumen der einzugebenden Daten kleiner wird.
- Die 2NF verringert auch Probleme, die durch Lösch- und Update-Anomalien auftreten können.
- Sie macht allerdings unsere Anfragen etwas komplizierter.
- Die 2NF zu implementieren bedeutet oft, Daten auf mehrere Tabellen zu verteilen.
- Wenn wir die Daten dann wieder zusammenbauen wollen, dann brauchen wir oft INNER JOINs.

# Zusammenfassung: Schlüssel



- Die Definition der 2NF ist etwas kompliziert.

# Zusammenfassung: Schlüssel



- Die Definition der 2NF ist etwas kompliziert.
- Sie gilt für zusammengesetzte Schlüssel und zwar für *alle* zusammengesetzten Schlüssel, nicht nur für Primärschlüssel.

# Zusammenfassung: Schlüssel



- Die Definition der 2NF ist etwas kompliziert.
- Sie gilt für zusammengesetzte Schlüssel und zwar für *alle* zusammengesetzten Schlüssel, nicht nur für Primärschlüssel.
- Es ist auch klar, warum das so ist:

# Zusammenfassung: Schlüssel



- Die Definition der 2NF ist etwas kompliziert.
- Sie gilt für zusammengesetzte Schlüssel und zwar für *alle* zusammengesetzten Schlüssel, nicht nur für Primärschlüssel.
- Es ist auch klar, warum das so ist:
- Wenn sie nur für Primärschlüssel gelten würde, dann hätten wir in unserem Beispiel oben die Tabelle building\_room einfach so ändern können, dass sie einen Ersatzprimärschlüssel verwendet.

# Zusammenfassung: Schlüssel



- Die Definition der 2NF ist etwas kompliziert.
- Sie gilt für zusammengesetzte Schlüssel und zwar für *alle* zusammengesetzten Schlüssel, nicht nur für Primärschlüssel.
- Es ist auch klar, warum das so ist:
- Wenn sie nur für Primärschlüssel gelten würde, dann hätten wir in unserem Beispiel oben die Tabelle building\_room einfach so ändern können, dass sie einen Ersatzprimärschlüssel verwendet.
- Da dieser nicht zusammengesetzt wäre, wäre die 2NF automatisch erfüllt.

# Zusammenfassung: Schlüssel



- Die Definition der 2NF ist etwas kompliziert.
- Sie gilt für zusammengesetzte Schlüssel und zwar für *alle* zusammengesetzten Schlüssel, nicht nur für Primärschlüssel.
- Es ist auch klar, warum das so ist:
- Wenn sie nur für Primärschlüssel gelten würde, dann hätten wir in unserem Beispiel oben die Tabelle building\_room einfach so ändern können, dass sie einen Ersatzprimärschlüssel verwendet.
- Da dieser nicht zusammengesetzt wäre, wäre die 2NF automatisch erfüllt.
- Das wäre aber sinnlos gewesen, da es keines der beschriebenen Probleme gelöst hätte.

# Zusammenfassung: Schlüssel



- Die Definition der 2NF ist etwas kompliziert.
- Sie gilt für zusammengesetzte Schlüssel und zwar für *alle* zusammengesetzten Schlüssel, nicht nur für Primärschlüssel.
- Es ist auch klar, warum das so ist:
- Wenn sie nur für Primärschlüssel gelten würde, dann hätten wir in unserem Beispiel oben die Tabelle building\_room einfach so ändern können, dass sie einen Ersatzprimärschlüssel verwendet.
- Da dieser nicht zusammengesetzt wäre, wäre die 2NF automatisch erfüllt.
- Das wäre aber sinnlos gewesen, da es keines der beschriebenen Probleme gelöst hätte.
- Deshalb ist die Definition so, wie sie ist.



谢谢你们！  
Thank you!  
Vielen Dank!



# References I



- [1] Adam Aspin und Karine Aspin. *Query Answers with MariaDB – Volume I: Introduction to SQL Queries*. Tetras Publishing, Okt. 2018. ISBN: 978-1-9996172-4-0. See also<sup>2</sup> (siehe S. 255, 264).
- [2] Adam Aspin und Karine Aspin. *Query Answers with MariaDB – Volume II: In-Depth Querying*. Tetras Publishing, Okt. 2018. ISBN: 978-1-9996172-5-7. See also<sup>1</sup> (siehe S. 255, 264).
- [3] 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. 263).
- [4] 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. 264, 265).
- [5] 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. 264).
- [6] 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. 265).
- [7] 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. 263).
- [8] 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. 264).
- [9] Ed Bott. *Windows 11 Inside Out*. Hoboken, NJ, USA: Microsoft Press, Pearson Education, Inc., Feb. 2023. ISBN: 978-0-13-769132-6 (siehe S. 264).
- [10] Ron Brash und Ganesh Naik. *Bash Cookbook*. Birmingham, England, UK: Packt Publishing Ltd, Juli 2018. ISBN: 978-1-78862-936-2 (siehe S. 263).

# References II



- [11] 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. 263).
- [12] Jason Cannon. *High Availability for the LAMP Stack*. Shelter Island, NY, USA: Manning Publications, Juni 2022 (siehe S. 264, 265).
- [13] 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. 265).
- [14] 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](http://bit.csc.lsu.edu/%7Echen/pdf/Chen_Pioneers.pdf) (besucht am 2025-03-06) (siehe S. 263).
- [15] Peter Pin-Shan Chen. "The Entity-Relationship Model – Toward a Unified View of Data". *ACM Transactions on Database Systems (TODS)* 1(1):9–36, März 1976. New York, NY, USA: Association for Computing Machinery (ACM). ISSN: 0362-5915. doi:10.1145/320434.320440 (siehe S. 256, 263).
- [16] Peter Pin-Shan Chen. "The Entity-Relationship Model: Toward a Unified View of Data". In: *1st International Conference on Very Large Data Bases (VLDB'1975)*. 22.–24. Sep. 1975, Framingham, MA, USA. Hrsg. von Douglas S. Kerr. New York, NY, USA: Association for Computing Machinery (ACM), 1975, S. 173. ISBN: 978-1-4503-3920-9. doi:10.1145/1282480.1282492. See<sup>15</sup> for a more comprehensive introduction. (Siehe S. 263).
- [17] 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. 265).
- [18] Edgar Frank „Ted“ Codd. "Normalized Data Base Structure: A Brief Tutorial". In: *ACM SIGFIDET Workshop on Data Description, Access, and Control*. 11.–12. Nov. 1971, San Diego, CA, USA. Hrsg. von Edgar Frank „Ted“ Codd und Albert L. Dean Jr. New York, NY, USA: Association for Computing Machinery (ACM), 1971, S. 1–17. ISBN: 978-1-4503-7300-5. doi:10.1145/1734714.1734716. See also<sup>19</sup> (siehe S. 5–13, 263).

# References III



- [19] Edgar Frank „Ted“ Codd. *Normalized Data Base Structure: A Brief Tutorial*. IBM Research Report RJ935. San Jose, CA, USA: IBM Research Laboratory, 1971. URL: [https://www.fsmwarden.com/Codd/Normalized%20data%20base%20structure\\_%20a%20brief%20tutorial\(1971,%20nov\).pdf](https://www.fsmwarden.com/Codd/Normalized%20data%20base%20structure_%20a%20brief%20tutorial(1971,%20nov).pdf) (besucht am 2025-05-04) (siehe S. 256).
- [20] 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. 263, 265).
- [21] Edgar Frank „Ted“ Codd. *Further Normalization of the Data Base Relational Model*. IBM Research Report RJ909. San Jose, CA, USA: IBM Research Laboratory, 31. Aug. 1971. URL: <https://forum.thethirdmanifesto.com/wp-content/uploads/asgarosforum/987737/00-efc-further-normalization.pdf> (besucht am 2025-05-04). Reprinted in and presented at <sup>54</sup> (siehe S. 5–13, 263).
- [22] *Database Language SQL*. Techn. Ber. ANSI X3.135-1986. Washington, D.C., USA: American National Standards Institute (ANSI), 1986 (siehe S. 265).
- [23] Christopher J. Date. *An Introduction to Database Systems*. 8. Aufl. Hoboken, NJ, USA: Pearson Education, Inc., Juli 2003. ISBN: 978-0-321-19784-9 (siehe S. 5–13, 263, 264).
- [24] 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. 265).
- [25] *Database Language SQL*. International Standard ISO 9075-1987. Geneva, Switzerland: International Organization for Standardization (ISO), 1987 (siehe S. 265).
- [26] Paul Deitel, Harvey Deitel und Abbey Deitel. *Internet & World Wide WebW[?]: How to Program*. 5. Aufl. Hoboken, NJ, USA: Pearson Education, Inc., Nov. 2011. ISBN: 978-0-13-299045-5 (siehe S. 265).
- [27] 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. 264).

# References IV



- [28] Ramez Elmasri und Shamkant Navathe. *Fundamentals of Database Systems*. 7. Aufl. Hoboken, NJ, USA: Pearson Education, Inc., Juni 2015. ISBN: **978-0-13-397077-7** (siehe S. **5–13, 263, 264**).
- [29] 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. **264**).
- [30] 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. **265**).
- [31] 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. **265**).
- [32] Michael Hausenblas. *Learning Modern Linux*. Sebastopol, CA, USA: O'Reilly Media, Inc., Apr. 2022. ISBN: **978-1-0981-0894-6** (siehe S. **264**).
- [33] 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. **264, 265**).
- [34] 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. **264**).
- [35] *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. **265**).
- [36] 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. **263**).

# References V



- [37] William (Bill) Kent. "A Simple Guide to Five Normal Forms in Relational Database Theory". *Communications of the ACM (CACM)* 26(2):120–125, Sep. 1982–Feb. 1983. New York, NY, USA: Association for Computing Machinery (ACM). ISSN: 0001-0782. doi:10.1145/358024.358054. URL: <https://www.cs.dartmouth.edu/~cs61/Resources/Papers/CACM%20Kent%20Five%20Normal%20Forms.pdf> (besucht am 2025-05-03) (siehe S. 5–24, 135–141, 263, 264).
- [38] Jay LaCroix. *Mastering Ubuntu Server*. 4. Aufl. Birmingham, England, UK: Packt Publishing Ltd, Sep. 2022. ISBN: 978-1-80323-424-3 (siehe S. 265).
- [39] 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. 264).
- [40] 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. 263).
- [41] Mark Lutz. *Learning Python*. 6. Aufl. Sebastopol, CA, USA: O'Reilly Media, Inc., März 2025. ISBN: 978-1-0981-7130-8 (siehe S. 264).
- [42] *MariaDB Server Documentation*. Milpitas, CA, USA: MariaDB, 2025. URL: <https://mariadb.com/kb/en/documentation> (besucht am 2025-04-24) (siehe S. 264).
- [43] 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. 265).
- [44] 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. 263).
- [45] 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. 264).
- [46] 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. 263).

# References VI



- [47] *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).
- [48] *PostgreSQL Essentials: Leveling Up Your Data Work*. Sebastopol, CA, USA: O'Reilly Media, Inc., März 2024 (siehe S. 264).
- [49] 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. 263).
- [50] Federico Razzoli. *Mastering MariaDB*. Birmingham, England, UK: Packt Publishing Ltd, Sep. 2014. ISBN: 978-1-78398-154-0 (siehe S. 264).
- [51] 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. 264).
- [52] "Returning Data from Modified Rows". In: *PostgreSQL Documentation*. 17.4. The PostgreSQL Global Development Group (PGDG), 20. Feb. 2025. Kap. 6.4. URL: <https://www.postgresql.org/docs/17/dml-returning.html> (besucht am 2025-04-21) (siehe S. 77–90, 100–131).
- [53] 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. 263).
- [54] Randall Rustin, Hrsg. *Data Base Systems: Courant Computer Science Symposium 6*. 24.–25. Mai 1971, New York, NY, USA. Englewood Cliffs, NJ, USA: Prentice-Hall, 1972. ISBN: 978-0-13-196741-0 (siehe S. 257).
- [55] Heinz Schweppe und Manuel Scholz. "Schema Definition with SQL / DDL (II)". In: *Einführung in die Datenbanksysteme. Datenbanken für die Bioinformatik*. Berlin, Germany: Freie Universität Berlin, Apr.–Okt. 2005. Kap. 4. URL: <https://www.inf.fu-berlin.de/lehre/SS05/19517-V/FolienEtc/dbs05-06-DDLSQL-2-2.pdf> (besucht am 2025-05-15) (siehe S. 30–37, 266).
- [56] 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](https://dbs.academy.lv/lection/dbs_LS04EN_erd.pdf) (besucht am 2025-03-29) (siehe S. 263).

# References VII



- [57] Yuriy Shamshin. *Databases*. Riga, Latvia: ISMA University of Applied Sciences, Mai 2024. URL: <https://dbs.academy.lv> (besucht am 2025-01-11).
- [58] Yuriy Shamshin. "Normalization". In: *Databases*. Riga, Latvia: ISMA University of Applied Sciences, Mai 2024. Kap. 07a. URL: [https://dbs.academy.lv/lection/dbs\\_LS07ENa\\_normalization.pdf](https://dbs.academy.lv/lection/dbs_LS07ENa_normalization.pdf) (besucht am 2025-05-03) (siehe S. 264).
- [59] Yuriy Shamshin. "RDM Normalization. Data Anomalies. Functional Dependency. Normal Forms.". In: *Databases*. Riga, Latvia: ISMA University of Applied Sciences, Mai 2024. Kap. 07. URL: [https://dbs.academy.lv/lection/dbs\\_LS07EN\\_normalization.pdf](https://dbs.academy.lv/lection/dbs_LS07EN_normalization.pdf) (besucht am 2025-05-03) (siehe S. 20–24, 77–99, 134, 264).
- [60] 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. 264).
- [61] John Miles Smith und Philip Yen-Tang Chang. "Optimizing the Performance of a Relational Algebra Database Interface". *Communications of the ACM (CACM)* 18(10):568–579, Okt. 1975. New York, NY, USA: Association for Computing Machinery (ACM). ISSN: 0001-0782. doi:10.1145/361020.361025 (siehe S. 265).
- [62] "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. 265).
- [63] 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. 261, 265).
- [64] 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 <sup>63</sup> (siehe S. 265).
- [65] Allen Taylor. *Introducing SQL and Relational Databases*. New York, NY, USA: Apress Media, LLC, Sep. 2018. ISBN: 978-1-4842-3841-7 (siehe S. 265).

# References VIII



- [66] 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. **264**).
- [67] 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. **264**).
- [68] Sander van Vugt. *Linux Fundamentals*. 2. Aufl. Hoboken, NJ, USA: Pearson IT Certification, Juni 2022. ISBN: **978-0-13-792931-3** (siehe S. **264**).
- [69] 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. **263**, **265**).
- [70] 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. **264**).
- [71] 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. **263**).
- [72] *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. **265**).
- [73] 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. **264**).
- [74] 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. **263**).
- [75] Giorgio Zarrelli. *Mastering Bash*. Birmingham, England, UK: Packt Publishing Ltd, Juni 2017. ISBN: **978-1-78439-687-9** (siehe S. **263**).

# Glossary (in English) I



1NF The first normal form (NF) in relational databases (DBs)<sup>20,23,28,37</sup>.

2NF The second normal form (NF) in relational DBs<sup>18,21,23,28,37</sup>.

3NF The third normal form (NF) in relational DBs<sup>18,21,23,28,37</sup>.

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<sup>10,44,75</sup>. 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<sup>7,40,46,49,53</sup>. These requests and responses are usually sent over network connections. A typical example for such a system is the World Wide Web (WWW), where web servers host websites and make them available to web browsers, the clients. Another typical example is the structure of DB software, where a central server, the DBMS, offers access to the DB to the different clients. Here, the client can be some terminal software shipping with the DBMS, such as psql, or the different applications that access the DBs.

DB A *database* is an organized collection of structured information or data, typically stored electronically in a computer system. Databases are discussed in our book *Databases*<sup>69</sup>.

DBMS A *database management system* is the software layer located between the user or application and the DB. The DBMS allows the user/application to create, read, write, update, delete, and otherwise manipulate the data in the DB<sup>74</sup>.

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

ERD Entity relationship diagrams show the relationships between objects, e.g., between the tables in a DB and how they reference each other<sup>3,11,14–16,36,56,71</sup>.

# Glossary (in English) II



**FD** A *functional dependency* exists between two groups of attributes  $Y$  and  $X$  if the values of  $X$  determine the values of  $Y$ . This is written as  $X \rightarrow Y$ .

**IT** information technology

**LAMP Stack** A system setup for web applications: Linux, Apache (a web server), MySQL, and the server-side scripting language PHP<sup>12,33</sup>.

**Linux** is the leading open source operating system, i.e., a free alternative for Microsoft Windows<sup>4,32,60,67,68</sup>. 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<sup>1,2,5,27,42,50</sup>. See <https://mariadb.org> for more information.

**Microsoft Windows** is a commercial proprietary operating system<sup>9</sup>. 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<sup>8,27,51,66,73</sup>. MySQL is famous for its use in the LAMP Stack. See <https://www.mysql.com> for more information.

**NF** The *normal forms* define guidelines for the design of relational DBs with the goal to avoid redundancy and to prevent inconsistencies and anomalies<sup>23,28,37,58,59</sup>. There are several normal forms, first normal form (1NF), second normal form (2NF), third normal form (3NF), and so on, each more restrictive than the other.

**PostgreSQL** An open source object-relational DBMS<sup>29,45,48,66</sup>. See <https://postgresql.org> for more information.

**psql** is the client program used to access the PostgreSQL DBMS server.

**Python** The Python programming language<sup>34,39,41,70</sup>, i.e., what you will learn about in our book<sup>70</sup>. Learn more at <https://python.org>.

# Glossary (in English) III



**relational database** A relational DB is a database that organizes data into rows (tuples, records) and columns (attributes), which collectively form tables (relations) where the data points are related to each other<sup>20,30,31,61,65,69,72</sup>.

**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 servers<sup>12</sup> 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“<sup>38</sup>.

**SQL** The *Structured Query Language* is basically a programming language for querying and manipulating relational databases<sup>13,22,24,25,35,43,62–65</sup>. It is understood by many DBMSes. You find the Structured Query Language (SQL) commands supported by PostgreSQL in the reference<sup>62</sup>.

**terminal** A terminal is a text-based window where you can enter commands and execute them<sup>4,17</sup>. 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 Drück auf  + , dann Schreiben von `cmd`, dann Drück auf . Under Ubuntu Linux,  +  +  opens a terminal, which then runs a Bash shell inside.

**Ubuntu** is a variant of the open source operating system Linux<sup>17,33</sup>. 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<sup>6,26</sup>

$i..j$  with  $i, j \in \mathbb{Z}$  and  $i \leq j$  is the set that contains all integer numbers in the inclusive range from  $i$  to  $j$ . For example,  $5..9$  is equivalent to  $\{5, 6, 7, 8, 9\}$

# Glossary (in English) IV



$\mathbb{R}$  the set of the real numbers.

$\Sigma(R)$  the relation schema of relation  $R$ <sup>55</sup>.

$\mathbb{Z}$  the set of the integers numbers including positive and negative numbers and 0, i.e.,  $\dots, -3, -2, -1, 0, 1, 2, 3, \dots$ , and so on. It holds that  $\mathbb{Z} \subset \mathbb{R}$ .