



合肥大學
HEFEI UNIVERSITY



Datenbanken

12. Fabrik-Datenbank: Joins und Views

Thomas Weise (汤卫思)
tweise@hfuu.edu.cn

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

应用优化研究所
人工智能与大数据学院
合肥大学
中国安徽省合肥市

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. LEFT JOIN
3. INNER JOIN
4. Sichten / Views
5. Die Sicht Benutzen
6. Zusammenfassung



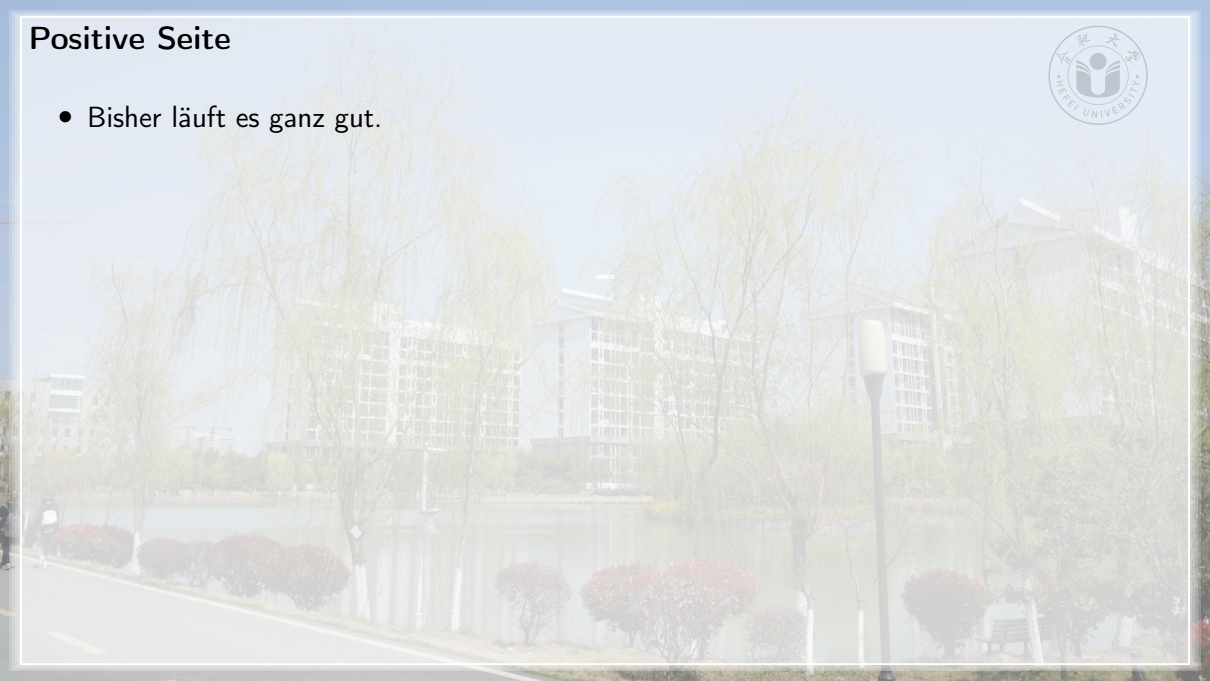


Einleitung



Positive Seite

- Bisher läuft es ganz gut.



Positive Seite



- Bisher läuft es ganz gut.
- Wir haben gelernt, wie wir die Daten, die in einem Managementsystem für eine Fabrik anfallen, in unterschiedliche Kategorien einteilen können.



- Bisher läuft es ganz gut.
- Wir haben gelernt, wie wir die Daten, die in einem Managementsystem für eine Fabrik anfallen, in unterschiedliche Kategorien einteilen können.
- Wir haben gelernt, wie wir mit dem DBMS PostgreSQL interagieren können und wie wir für jede dieser Datenkategorien eine Tabelle in einer Datenbank erstellen können.

Positive Seite



- Bisher läuft es ganz gut.
- Wir haben gelernt, wie wir die Daten, die in einem Managementsystem für eine Fabrik anfallen, in unterschiedliche Kategorien einteilen können.
- Wir haben gelernt, wie wir mit dem DBMS PostgreSQL interagieren können und wie wir für jede dieser Datenkategorien eine Tabelle in einer Datenbank erstellen können.
- Das ist schonmal sehr schön.

Positive Seite



- Bisher läuft es ganz gut.
- Wir haben gelernt, wie wir die Daten, die in einem Managementsystem für eine Fabrik anfallen, in unterschiedliche Kategorien einteilen können.
- Wir haben gelernt, wie wir mit dem DBMS PostgreSQL interagieren können und wie wir für jede dieser Datenkategorien eine Tabelle in einer Datenbank erstellen können.
- Das ist schonmal sehr schön.
- Wir haben auch gelernt, wie wir die Integrität unserer Daten schützen können.

Positive Seite



- Bisher läuft es ganz gut.
- Wir haben gelernt, wie wir die Daten, die in einem Managementsystem für eine Fabrik anfallen, in unterschiedliche Kategorien einteilen können.
- Wir haben gelernt, wie wir mit dem DBMS PostgreSQL interagieren können und wie wir für jede dieser Datenkategorien eine Tabelle in einer Datenbank erstellen können.
- Das ist schonmal sehr schön.
- Wir haben auch gelernt, wie wir die Integrität unserer Daten schützen können.
- Auf der einen Seite kennen wir Datentypen wie `DECIMAL` und `DATE`, die sicherstellen, dass unsere Daten bestimmte Kriterien einhalten.

Positive Seite



- Bisher läuft es ganz gut.
- Wir haben gelernt, wie wir die Daten, die in einem Managementsystem für eine Fabrik anfallen, in unterschiedliche Kategorien einteilen können.
- Wir haben gelernt, wie wir mit dem DBMS PostgreSQL interagieren können und wie wir für jede dieser Datenkategorien eine Tabelle in einer Datenbank erstellen können.
- Das ist schonmal sehr schön.
- Wir haben auch gelernt, wie wir die Integrität unserer Daten schützen können.
- Auf der einen Seite kennen wir Datentypen wie `DECIMAL` und `DATE`, die sicherstellen, dass unsere Daten bestimmte Kriterien einhalten.
- Auf der anderen Seite kennen wir Einschränkungen wie `NOT NULL`, `UNIQUE`, `CONSTRAINT` und `REFERENCES`.



- Bisher läuft es ganz gut.
- Wir haben gelernt, wie wir die Daten, die in einem Managementsystem für eine Fabrik anfallen, in unterschiedliche Kategorien einteilen können.
- Wir haben gelernt, wie wir mit dem DBMS PostgreSQL interagieren können und wie wir für jede dieser Datenkategorien eine Tabelle in einer Datenbank erstellen können.
- Das ist schonmal sehr schön.
- Wir haben auch gelernt, wie wir die Integrität unserer Daten schützen können.
- Auf der einen Seite kennen wir Datentypen wie `DECIMAL` und `DATE`, die sicherstellen, dass unsere Daten bestimmte Kriterien einhalten.
- Auf der anderen Seite kennen wir Einschränkungen wie `NOT NULL`, `UNIQUE`, `CONSTRAINT` und `REFERENCES`.
- Damit sind Datenbanken Spreadsheets, wie wir sie mit Microsoft Excel oder LibreOffice Calc erstellen können, weit überlegen.

Negative Seite

- Auf der anderen Seite ist das Arbeiten mit Datenbanken kompliziert.



Negative Seite



- Auf der anderen Seite ist das Arbeiten mit Datenbanken kompliziert.
- Die Interaktion mit dem PostgreSQL-Server über psql ist nicht einfach und andere DBMS sind auch nicht einfacher zu bedienen.



Negative Seite



- Auf der anderen Seite ist das Arbeiten mit Datenbanken kompliziert.
- Die Interaktion mit dem PostgreSQL-Server über psql ist nicht einfach und andere DBMS sind auch nicht einfacher zu bedienen.
- Ärgerlich ist, dass Daten, die mehrere Tabellen verbinden, schwer zu verstehen sind.



Negative Seite



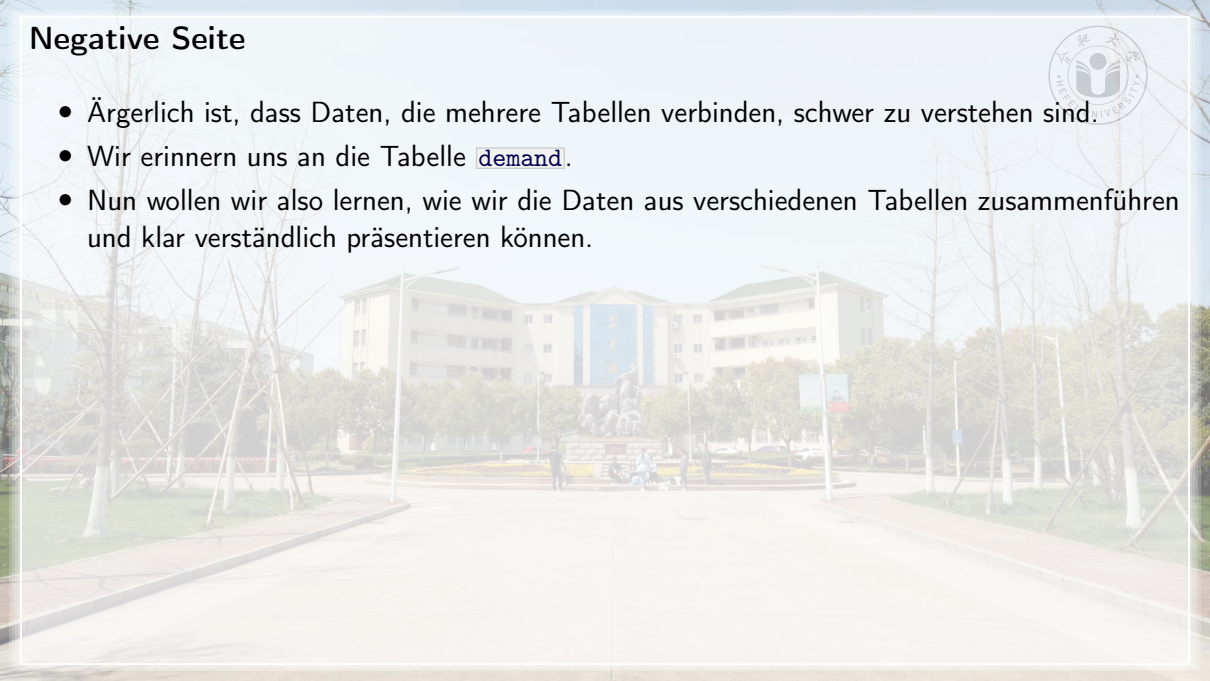
- Ärgerlich ist, dass Daten, die mehrere Tabellen verbinden, schwer zu verstehen sind.
- Wir erinnern uns an die Tabelle `demand`:

```
1 $ psql "postgres://boss:superboss123@localhost/factory" -v ON_ERROR_STOP
   ↳ =1 -ebf insert_into_table_demand.sql
2 id | customer | product | amount | ordered
3 -----+-----+-----+-----+-----
4 (0 rows)
5
6 INSERT 0 8
7 id | customer | product | amount | ordered
8 -----+-----+-----+-----+-----
9 1 | 1 | 7 | 12 | 2024-11-21
10 2 | 2 | 3 | 2 | 2024-12-09
11 3 | 3 | 2 | 7 | 2024-12-16
12 4 | 2 | 5 | 7 | 2024-12-30
13 5 | 1 | 5 | 3 | 2025-01-05
14 6 | 2 | 6 | 4 | 2025-01-12
15 7 | 3 | 11 | 10 | 2025-01-16
16 8 | 2 | 3 | 6 | 2025-02-05
17 (8 rows)
18
19 # psql 16.11 succeeded with exit code 0.
```

Negative Seite



- Ärgerlich ist, dass Daten, die mehrere Tabellen verbinden, schwer zu verstehen sind.
- Wir erinnern uns an die Tabelle `demand`.
- Nun wollen wir also lernen, wie wir die Daten aus verschiedenen Tabellen zusammenführen und klar verständlich präsentieren können.





LEFT JOIN



Bestellungen der Kunden



- Wir wollen nun eine Liste unserer Kunden.

Bestellungen der Kunden



- Wir wollen nun eine Liste unserer Kunden.
- Für jeden Kunden wollen wir die `id`-Werte aller ihrer Bestellungen.

Bestellungen der Kunden



- Wir wollen nun eine Liste unserer Kunden.
- Für jeden Kunden wollen wir die `id`-Werte aller ihrer Bestellungen.
- Mit anderen Worten, wir wollen eine SQL-Anfrage, die zwei Spalten produziert.

Bestellungen der Kunden



- Wir wollen nun eine Liste unserer Kunden.
- Für jeden Kunden wollen wir die `id`-Werte aller ihrer Bestellungen.
- Mit anderen Worten, wir wollen eine SQL-Anfrage, die zwei Spalten produziert.
- In der ersten Spalte – nennen wir sie `name` – wollen wir die Namen der Kunden.



Bestellungen der Kunden



- Wir wollen nun eine Liste unserer Kunden.
- Für jeden Kunden wollen wir die `id`-Werte aller ihrer Bestellungen.
- Mit anderen Worten, wir wollen eine SQL-Anfrage, die zwei Spalten produziert.
- In der ersten Spalte – nennen wir sie `name` – wollen wir die Namen der Kunden.
- In der zweiten Spalte – nennen wir sie `demand_id` – wollen wir die `id`-Werte der Bestellungen des Kunden.

Bestellungen der Kunden



- Wir wollen nun eine Liste unserer Kunden.
- Für jeden Kunden wollen wir die `id`-Werte aller ihrer Bestellungen.
- Mit anderen Worten, wir wollen eine SQL-Anfrage, die zwei Spalten produziert.
- In der ersten Spalte – nennen wir sie `name` – wollen wir die Namen der Kunden.
- In der zweiten Spalte – nennen wir sie `demand_id` – wollen wir die `id`-Werte der Bestellungen des Kunden.
- Wenn Kunden mehrere Bestellungen aufgegeben haben, dann werden sie mehrmals im Ergebnis auftauchen, jeweils mit einer Bestellung.

Bestellungen der Kunden



- Wir wollen nun eine Liste unserer Kunden.
- Für jeden Kunden wollen wir die `id`-Werte aller ihrer Bestellungen.
- Mit anderen Worten, wir wollen eine SQL-Anfrage, die zwei Spalten produziert.
- In der ersten Spalte – nennen wir sie `name` – wollen wir die Namen der Kunden.
- In der zweiten Spalte – nennen wir sie `demand_id` – wollen wir die `id`-Werte der Bestellungen des Kunden.
- Wenn Kunden mehrere Bestellungen aufgegeben haben, dann werden sie mehrmals im Ergebnis auftauchen, jeweils mit einer Bestellung.
- Wenn ein Kunde keine Bestellungen aufgegeben hat, dann soll er trotzdem auftauchen, aber nur einmal, mit einem `NULL`-Wert als `demand_id`.

Erstellen der Anfrage



- Im ersten Schritt brauchen wir also alle Kundennamen.

Erstellen der Anfrage



- Im ersten Schritt brauchen wir also alle Kundennamen.
- `SELECT name FROM customer;` macht das.

Erstellen der Anfrage



- Im ersten Schritt brauchen wir also alle Kundennamen.
- `SELECT name FROM customer;` macht das.
- Nun müssen wir die `id`-Werte der Tabelle `customer` mit der Spalte `customer` der Tabelle `demand` referenzieren.

Erstellen der Anfrage



- Im ersten Schritt brauchen wir also alle Kundennamen.
- `SELECT name FROM customer;` macht das.
- Nun müssen wir die `id`-Werte der Tabelle `customer` mit der Spalte `customer` der Tabelle `demand` referenzieren.
- Das geht mit einem so-geannten `LEFT JOIN`³⁶.

LEFT JOIN Syntax

```
1  -- Die Syntax eines LEFT JOIN. The Syntax of a LEFT JOIN.
2  -- Wir vereinigen Informationen aus den Tabellen Table_1 und Table_2.
3
4  -- Oft, aber nicht immer, hat Table_1 die Spalte 'a' als Primär-
5  -- schlüssel, der von Spalte 'b' in Table_2 als Fremdschlüssel
6  -- referenziert wird.
7  -- In diesem Beispiel geben wir die Werte von Spalte 'x' der Table_1
8  -- und 'y' von Table_2 zurück, nachdem wir beide Tabellen vereinigt
9  -- haben.
10 -- Für Zeilen in Table_1 ohne passende Zeilen in Table_2 werden
11 -- Ergebniszeilen generiert, bei denen alle Werte, die eigentlich von
12 -- Table_2 kommen sollten, mit NULL belegt.
```

```
13 SELECT Table_1.x, Table_2.y FROM Table_1
14 LEFT JOIN Table_2 ON (Table_1.a = Table_2.b);
```

```
15
16 -- Table_1 Table_2 Query Result
17 -- a | x b | y x | y
18 -- +-----+-----+-----+
19 -- 1 | Hello 1 | A Hello | A
20 -- 2 | World 2 | B World | B
21 -- 3 | from 2 | C World | C
22 -- 4 | HFUU 4 | D World | E
23 -- 2 | E from | <NULL>
24 -- HFUU | D
```



LEFT JOIN: Syntax

- Angenommen, Tabelle `Table_1` hat den Primärschlüssel `a`.



```
1  -- Die Syntax eines LEFT JOIN. The Syntax of a LEFT JOIN.
2  -- Wir vereinigen Informationen aus den Tabellen Table_1 und Table_2.
3
4  -- Oft, aber nicht immer, hat Table_1 die Spalte 'a' als Primär-
5  -- schlüssel, der von Spalte 'b' in Table_2 als Fremdschlüssel
6  -- referenziert wird.
7  -- In diesem Beispiel geben wir die Werte von Spalte 'x' der Table_1
8  -- und 'y' von Table_2 zurück, nachdem wir beide Tabellen vereinigt
9  -- haben.
10 -- Für Zeilen in Table_1 ohne passende Zeilen in Table_2 werden
11 -- Ergebniszeilen generiert, bei denen alle Werte, die eigentlich von
12 -- Table_2 kommen sollten, mit NULL belegt.
13 SELECT Table_1.x, Table_2.y FROM Table_1
14 LEFT JOIN Table_2 ON (Table_1.a = Table_2.b);
```

LEFT JOIN: Syntax



- Angenommen, Tabelle `Table_1` hat den Primärschlüssel `a` und Tabelle `Table_2` hat eine Spalte `b`, die als Fremdschlüssel auf `Table_1.a` verweist.

```
1  -- Die Syntax eines LEFT JOIN. The Syntax of a LEFT JOIN.
2  -- Wir vereinigen Informationen aus den Tabellen Table_1 und Table_2.
3
4  -- Oft, aber nicht immer, hat Table_1 die Spalte 'a' als Primär-
5  -- schlüssel, der von Spalte 'b' in Table_2 als Fremdschlüssel
6  -- referenziert wird.
7  -- In diesem Beispiel geben wir die Werte von Spalte 'x' der Table_1
8  -- und 'y' von Table_2 zurück, nachdem wir beide Tabellen vereinigt
9  -- haben.
10 -- Für Zeilen in Table_1 ohne passende Zeilen in Table_2 werden
11 -- Ergebniszeilen generiert, bei denen alle Werte, die eigentlich von
12 -- Table_2 kommen sollten, mit NULL belegt.
13 SELECT Table_1.x, Table_2.y FROM Table_1
14 LEFT JOIN Table_2 ON (Table_1.a = Table_2.b);
```

LEFT JOIN: Syntax



- Angenommen, Tabelle `Table_1` hat den Primärschlüssel `a` und Tabelle `Table_2` hat eine Spalte `b`, die als Fremdschlüssel auf `Table_1.a` verweist.
- `SELECT ... FROM Table_1 LEFT JOIN Table_2 ON (Table_1.a = Table_2.b)` geht dann Zeile für Zeile durch `Table_1` und sucht für jeden Wert von `Table_1.a` alle passenden Zeilen aus `Table_2` heraus, also alle Zeilen, für die `Table_1.a = Table_2.b` gilt.

```
1  -- Die Syntax eines LEFT JOIN. The Syntax of a LEFT JOIN.
2  -- Wir vereinigen Informationen aus den Tabellen Table_1 und Table_2.
3
4  -- Oft, aber nicht immer, hat Table_1 die Spalte 'a' als Primär-
5  -- schlüssel, der von Spalte 'b' in Table_2 als Fremdschlüssel
6  -- referenziert wird.
7  -- In diesem Beispiel geben wir die Werte von Spalte 'x' der Table_1
8  -- und 'y' von Table_2 zurück, nachdem wir beide Tabellen vereinigt
9  -- haben.
10 -- Für Zeilen in Table_1 ohne passende Zeilen in Table_2 werden
11 -- Ergebniszeilen generiert, bei denen alle Werte, die eigentlich von
12 -- Table_2 kommen sollten, mit NULL belegt.
13 SELECT Table_1.x, Table_2.y FROM Table_1
14      LEFT JOIN Table_2 ON (Table_1.a = Table_2.b);
```

LEFT JOIN: Syntax



- Angenommen, Tabelle `Table_1` hat den Primärschlüssel `a` und Tabelle `Table_2` hat eine Spalte `b`, die als Fremdschlüssel auf `Table_1.a` verweist.
- `SELECT ... FROM Table_1 LEFT JOIN Table_2 ON (Table_1.a = Table_2.b)` geht dann Zeile für Zeile durch `Table_1` und sucht für jeden Wert von `Table_1.a` alle passenden Zeilen aus `Table_2` heraus, also alle Zeilen, für die `Table_1.a = Table_2.b` gilt.
- Jede dieser Zeilen wird dann eine Zeile im Ergebnis.

```
1  -- Die Syntax eines LEFT JOIN. The Syntax of a LEFT JOIN.
2  -- Wir vereinigen Informationen aus den Tabellen Table_1 und Table_2.
3
4  -- Oft, aber nicht immer, hat Table_1 die Spalte 'a' als Primär-
5  -- schlüssel, der von Spalte 'b' in Table_2 als Fremdschlüssel
6  -- referenziert wird.
7  -- In diesem Beispiel geben wir die Werte von Spalte 'x' der Table_1
8  -- und 'y' von Table_2 zurück, nachdem wir beide Tabellen vereinigt
9  -- haben.
10 -- Für Zeilen in Table_1 ohne passende Zeilen in Table_2 werden
11 -- Ergebniszeilen generiert, bei denen alle Werte, die eigentlich von
12 -- Table_2 kommen sollten, mit NULL belegt.
13 SELECT Table_1.x, Table_2.y FROM Table_1
14      LEFT JOIN Table_2 ON (Table_1.a = Table_2.b);
```

LEFT JOIN: Syntax



- Angenommen, Tabelle `Table_1` hat den Primärschlüssel `a` und Tabelle `Table_2` hat eine Spalte `b`, die als Fremdschlüssel auf `Table_1.a` verweist.
- `SELECT ... FROM Table_1 LEFT JOIN Table_2 ON (Table_1.a = Table_2.b)` geht dann Zeile für Zeile durch `Table_1` und sucht für jeden Wert von `Table_1.a` alle passenden Zeilen aus `Table_2` heraus, also alle Zeilen, für die `Table_1.a = Table_2.b` gilt.
- Jede dieser Zeilen wird dann eine Zeile im Ergebnis.
- Gibt es keine solche Zeile, so trotzdem eine Ergebniszeile generiert, jedoch alle Werte, die vielleicht aus `Table_2` übernommen würden, werden als `NULL` angegeben.

LEFT JOIN: Syntax



- Angenommen, Tabelle `Table_1` hat den Primärschlüssel `a` und Tabelle `Table_2` hat eine Spalte `b`, die als Fremdschlüssel auf `Table_1.a` verweist.
- `SELECT ... FROM Table_1 LEFT JOIN Table_2 ON (Table_1.a = Table_2.b)` geht dann Zeile für Zeile durch `Table_1` und sucht für jeden Wert von `Table_1.a` alle passenden Zeilen aus `Table_2` heraus, also alle Zeilen, für die `Table_1.a = Table_2.b` gilt.
- Jede dieser Zeilen wird dann eine Zeile im Ergebnis.
- Gibt es keine solche Zeile, so trotzdem eine Ergebniszeile generiert, jedoch alle Werte, die vielleicht aus `Table_2` übernommen würden, werden als `NULL` angegeben.
- Beachten Sie: Um Klarheit zu schaffen, können wir den Tabellennamen als Prefix, gefolgt von einem Punkt, dem Spaltennamen voranstellen.

LEFT JOIN: Syntax



- Angenommen, Tabelle `Table_1` hat den Primärschlüssel `a` und Tabelle `Table_2` hat eine Spalte `b`, die als Fremdschlüssel auf `Table_1.a` verweist.
- `SELECT ... FROM Table_1 LEFT JOIN Table_2 ON (Table_1.a = Table_2.b)` geht dann Zeile für Zeile durch `Table_1` und sucht für jeden Wert von `Table_1.a` alle passenden Zeilen aus `Table_2` heraus, also alle Zeilen, für die `Table_1.a = Table_2.b` gilt.
- Jede dieser Zeilen wird dann eine Zeile im Ergebnis.
- Gibt es keine solche Zeile, so trotzdem eine Ergebniszeile generiert, jedoch alle Werte, die vielleicht aus `Table_2` übernommen würden, werden als `NULL` angegeben.
- Beachten Sie: Um Klarheit zu schaffen, können wir den Tabellennamen als Prefix, gefolgt von einem Punkt, dem Spaltennamen voranstellen.
- `Table_1.a` bedeutet also „Spalte `a` aus Tabelle `Table_1`“.

LEFT JOIN Syntax

-- Die Syntax eines LEFT JOIN. The Syntax of a LEFT JOIN.
-- Wir vereinigen Informationen aus den Tabellen Table_1 und Table_2.
-- Oft, aber nicht immer, hat Table_1 die Spalte 'a' als Primär-
-- schlüssel, der von Spalte 'b' in Table_2 als Fremdschlüssel
-- referenziert wird.
-- In diesem Beispiel geben wir die Werte von Spalte 'x' der Table_1
-- und 'y' von Table_2 zurück, nachdem wir beide Tabellen vereinigt
-- haben.
-- Für Zeilen in Table_1 ohne passende Zeilen in Table_2 werden
-- Ergebniszeilen generiert, bei denen alle Werte, die eigentlich von
-- Table_2 kommen sollten, mit NULL belegt.

```
SELECT Table_1.x, Table_2.y FROM Table_1  
LEFT JOIN Table_2 ON (Table_1.a = Table_2.b);
```

Table_1	Table_2	Query Result
a x	b y	x y
1 Hello	1 A	Hello A
2 World	2 B	World B
3 from	2 C	World C
4 HFUU	4 D	World E
	2 E	from <NULL>
		HFUU D



LEFT JOIN von customer mit demand



- Übertragen wir also diese Syntax auf unsere Situation.

```
1  /* Get the demands per customer. */  
2  
3  -- List all demand IDs for each customer.  
4  -- LEFT JOIN: 'Bobbo' also appears once, with a NULL demand.  
5  SELECT customer.name AS name, demand.id AS demand_id FROM customer  
6         LEFT JOIN demand ON (customer.id = demand.customer)  
7         ORDER BY name;
```

LEFT JOIN von customer mit demand



- Übertragen wir also diese Syntax auf unsere Situation.

```
$ psql "postgres://boss:superboss123@localhost/factory" -v ON_ERROR_STOP  
↪ =1 -ebf select_customer_demand_1.sql
```

```
name | demand_id  
-----+-----  
Bebba |          7  
Bebba |          3  
Bebbo |          8  
Bebbo |          4  
Bebbo |          2  
Bebbo |          6  
Bibbo |          1  
Bibbo |          5  
Bobbo |  
(9 rows)
```

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

Zählen der Bestellungen



- Jetzt wollen wir zählen, wie viele Bestellungen jeder Kunde bisher gemacht hat.

Zählen der Bestellungen



- Jetzt wollen wir zählen, wie viele Bestellungen jeder Kunde bisher gemacht hat.
- Wir kennen inzwischen alles, was wir dazu brauchen:

Zählen der Bestellungen



- Jetzt wollen wir zählen, wie viele Bestellungen jeder Kunde bisher gemacht hat.
- Wir kennen inzwischen alles, was wir dazu brauchen:
- Mit `LEFT JOIN` von der Tabelle `customer` auf die Tabelle `demand` bekommen wir die Bestellungen pro Kunde.

Zählen der Bestellungen



- Jetzt wollen wir zählen, wie viele Bestellungen jeder Kunde bisher gemacht hat.
- Wir kennen inzwischen alles, was wir dazu brauchen:
- Mit `LEFT JOIN` von der Tabelle `customer` auf die Tabelle `demand` bekommen wir die Bestellungen pro Kunde.
- Alles, was dann zu tun ist, ist die Daten nach Kundenname zu gruppieren (`GROUP BY name`).

Zählen der Bestellungen



- Jetzt wollen wir zählen, wie viele Bestellungen jeder Kunde bisher gemacht hat.
- Wir kennen inzwischen alles, was wir dazu brauchen:
- Mit `LEFT JOIN` von der Tabelle `customer` auf die Tabelle `demand` bekommen wir die Bestellungen pro Kunde.
- Alles, was dann zu tun ist, ist die Daten nach Kundenname zu gruppieren (`GROUP BY name`).
- Dann können wir die Zeilen pro `name`-Wert zählen, mit `COUNT(*)`.

LEFT JOIN: Bestellungen pro Kunde zählen



```
1  /* Get the number of demands per customer. */
2
3  -- Now we count the demands.
4  SELECT customer.name AS name, COUNT(*) AS demands FROM customer
5         LEFT JOIN demand ON (customer.id = demand.customer)
6         GROUP BY name ORDER BY name;
```

```
1  $ psql "postgres://boss:superboss123@localhost/factory" -v ON_ERROR_STOP
   ↪ =1 -ebf select_customer_demand_2.sql
2  name  | demands
3  -----+-----
4  Bebba  |      2
5  Bebbo  |      4
6  Bibbo  |      2
7  Bobbo  |      1
8  (4 rows)
9
10 # psql 16.11 succeeded with exit code 0.
```



INNER JOIN



Verkaufsdaten Zusammenführen



- Wir können nun Daten von verschiedenen Tabellen zusammenführen.

Verkaufsdaten Zusammenführen



- Wir können nun Daten von verschiedenen Tabellen zusammenführen.
- Erstellen wir eine Anfrage, die für jede Bestellung den Kundenname, Produktname, Preise, bestellte Anzahl und Bestelldatum heraussucht.

Verkaufsdaten Zusammenführen



- Wir können nun Daten von verschiedenen Tabellen zusammenführen.
- Erstellen wir eine Anfrage, die für jede Bestellung den Kundenname, Produktname, Preise, bestellte Anzahl und Bestelldatum herausucht.
- Das Ergebnis dieser Anfrage wäre dann eine klare und lesbare Übersicht unserer Geschäftstransaktionen.

Verkaufsdaten Zusammenführen



- Wir können nun Daten von verschiedenen Tabellen zusammenführen.
- Erstellen wir eine Anfrage, die für jede Bestellung den Kundenname, Produktname, Preise, bestellte Anzahl und Bestelldatum herausucht.
- Das Ergebnis dieser Anfrage wäre dann eine klare und lesbare Übersicht unserer Geschäftstransaktionen.
- Basierend auf unserem vorigen Beispiel können wir erwarten, dass wir nun zwei „Joins“ brauchen.

Verkaufsdaten Zusammenführen



- Wir können nun Daten von verschiedenen Tabellen zusammenführen.
- Erstellen wir eine Anfrage, die für jede Bestellung den Kundenname, Produktname, Preise, bestellte Anzahl und Bestelldatum heraussucht.
- Das Ergebnis dieser Anfrage wäre dann eine klare und lesbare Übersicht unserer Geschäftstransaktionen.
- Basierend auf unserem vorigen Beispiel können wir erwarten, dass wir nun zwei „Joins“ brauchen.
- Wir müssen ja Daten aus allen drei Tabellen – `demand`, `customer` und `product` – kombinieren.

Verkaufsdaten Zusammenführen



- Wir können nun Daten von verschiedenen Tabellen zusammenführen.
- Erstellen wir eine Anfrage, die für jede Bestellung den Kundenname, Produktname, Preise, bestellte Anzahl und Bestelldatum heraussucht.
- Das Ergebnis dieser Anfrage wäre dann eine klare und lesbare Übersicht unserer Geschäftstransaktionen.
- Basierend auf unserem vorigen Beispiel können wir erwarten, dass wir nun zwei „Joins“ brauchen.
- Wir müssen ja Daten aus allen drei Tabellen – `demand`, `customer` und `product` – kombinieren.
- Die Anfrage wird daher etwas komplizierter werden.

Grundidee

- Wir beginnen, die Anfrage zu konstruieren.



Grundidee



- Wir beginnen, die Anfrage zu konstruieren.
- Was wir wollen ist das: „Für jede Zeile in Tabelle `demand` brauchen wir die entsprechende Zeile in Tabelle `customer` und die entsprechende Zeile in Tabelle `product`.“

Grundidee



- Wir beginnen, die Anfrage zu konstruieren.
- Was wir wollen ist das: „Für jede Zeile in Tabelle `demand` brauchen wir die entsprechende Zeile in Tabelle `customer` und die entsprechende Zeile in Tabelle `product`.“
- Wir wissen, dass es immer eine entsprechende Zeile in der Tabelle `customer` geben muss, denn die `REFERENCES`-Einschränkung erzwingt das.

Grundidee



- Wir beginnen, die Anfrage zu konstruieren.
- Was wir wollen ist das: „Für jede Zeile in Tabelle `demand` brauchen wir die entsprechende Zeile in Tabelle `customer` und die entsprechende Zeile in Tabelle `product`.“
- Wir wissen, dass es immer eine entsprechende Zeile in der Tabelle `customer` geben muss, denn die `REFERENCES`-Einschränkung erzwingt das.
- Wir wissen, dass es immer eine entsprechende Zeile in der Tabelle `product` geben muss, denn die `REFERENCES`-Einschränkung erzwingt das.

Grundidee



- Wir beginnen, die Anfrage zu konstruieren.
- Was wir wollen ist das: „Für jede Zeile in Tabelle `demand` brauchen wir die entsprechende Zeile in Tabelle `customer` und die entsprechende Zeile in Tabelle `product`.“
- Wir wissen, dass es immer eine entsprechende Zeile in der Tabelle `customer` geben muss, denn die `REFERENCES`-Einschränkung erzwingt das.
- Wir wissen, dass es immer eine entsprechende Zeile in der Tabelle `product` geben muss, denn die `REFERENCES`-Einschränkung erzwingt das.
- Es gibt keine Zeile in `demand` ohne entsprechende Zeile in `customer` oder `product`.

Grundidee



- Wir beginnen, die Anfrage zu konstruieren.
- Was wir wollen ist das: „Für jede Zeile in Tabelle `demand` brauchen wir die entsprechende Zeile in Tabelle `customer` und die entsprechende Zeile in Tabelle `product`.“
- Wir wissen, dass es immer eine entsprechende Zeile in der Tabelle `customer` geben muss, denn die `REFERENCES`-Einschränkung erzwingt das.
- Wir wissen, dass es immer eine entsprechende Zeile in der Tabelle `product` geben muss, denn die `REFERENCES`-Einschränkung erzwingt das.
- Es gibt keine Zeile in `demand` ohne entsprechende Zeile in `customer` oder `product`.
- Wir könnten wieder `LEFT JOIN` verwenden, denn es könnte niemals `NULL` erzeugen.

- Wir beginnen, die Anfrage zu konstruieren.
- Was wir wollen ist das: „Für jede Zeile in Tabelle `demand` brauchen wir die entsprechende Zeile in Tabelle `customer` und die entsprechende Zeile in Tabelle `product`.“
- Wir wissen, dass es immer eine entsprechende Zeile in der Tabelle `customer` geben muss, denn die `REFERENCES`-Einschränkung erzwingt das.
- Wir wissen, dass es immer eine entsprechende Zeile in der Tabelle `product` geben muss, denn die `REFERENCES`-Einschränkung erzwingt das.
- Es gibt keine Zeile in `demand` ohne entsprechende Zeile in `customer` oder `product`.
- Wir könnten wieder `LEFT JOIN` verwenden, denn es könnte niemals `NULL` erzeugen.
- Lassen Sie uns aber stattdessen `INNER JOIN`³⁶ nutzen.

- Wir beginnen, die Anfrage zu konstruieren.
- Was wir wollen ist das: „Für jede Zeile in Tabelle `demand` brauchen wir die entsprechende Zeile in Tabelle `customer` und die entsprechende Zeile in Tabelle `product`.“
- Wir wissen, dass es immer eine entsprechende Zeile in der Tabelle `customer` geben muss, denn die `REFERENCES`-Einschränkung erzwingt das.
- Wir wissen, dass es immer eine entsprechende Zeile in der Tabelle `product` geben muss, denn die `REFERENCES`-Einschränkung erzwingt das.
- Es gibt keine Zeile in `demand` ohne entsprechende Zeile in `customer` oder `product`.
- Wir könnten wieder `LEFT JOIN` verwenden, denn es könnte niemals `NULL` erzeugen.
- Lassen Sie uns aber stattdessen `INNER JOIN`³⁶ nutzen.
- Anders als `LEFT JOIN` wird `INNER JOIN` nur Zeilen ausgeben, wenn alle Tabellen zueinanderpassende Zeilen haben.

INNER JOIN Syntax

```
1  -- Die Syntax von einem INNER JOIN.
2  -- Wir vereinigen Informationen aus den Tabellen Table_1 und Table_2.
3
4  -- Oft, aber nicht immer, hat Table_1 die Spalte 'a' als Primär-
5  -- schlüssel, der von Spalte 'b' in Table_2 als Fremdschlüssel
6  -- referenziert wird.
7  -- In diesem Beispiel geben wir die Werte von Spalte 'x' der Table_1
8  -- und 'y' von Table_2 zurück, nachdem wir beide Tabellen vereinigt
9  -- haben.
10 -- Für Zeilen in Table_1 ohne passende Zeilen in Table_2, wird keine
11 -- Ergebniszeile generiert.
```

```
12 SELECT Table_1.x, Table_2.y FROM Table_1
13     INNER JOIN Table_2 ON (Table_1.a = Table_2.b);
```

```
14
15 -- Table_1 Table_2 Query Result
16 -- a | x b | y x | y
17 -- +-----+ +-----+
18 -- 1 | Hello 1 | A Hello | A
19 -- 2 | World 2 | B World | B
20 -- 3 | from 2 | C World | C
21 -- 4 | HFUU 4 | D World | E
22 -- 2 | E HFUU | D
```

INNER JOIN: Syntax



- Angenommen, Tabelle `Table_1` hat den Primärschlüssel `a`.

```
1  -- Die Syntax von einem INNER JOIN.
2  -- Wir vereinigen Informationen aus den Tabellen Table_1 und Table_2.
3
4  -- Oft, aber nicht immer, hat Table_1 die Spalte 'a' als Primär-
5  -- schlüssel, der von Spalte 'b' in Table_2 als Fremdschlüssel
6  -- referenziert wird.
7  -- In diesem Beispiel geben wir die Werte von Spalte 'x' der Table_1
8  -- und 'y' von Table_2 zurück, nachdem wir beide Tabellen vereinigt
9  -- haben.
10 -- Für Zeilen in Table_1 ohne passende Zeilen in Table_2, wird keine
11 -- Ergebniszeile generiert.
12 SELECT Table_1.x, Table_2.y FROM Table_1
13      INNER JOIN Table_2 ON (Table_1.a = Table_2.b);
```

INNER JOIN: Syntax



- Angenommen, Tabelle `Table_1` hat den Primärschlüssel `a` und Tabelle `Table_2` hat eine Spalte `b`, die als Fremdschlüssel auf `Table_1.a` verweist.

```
1  -- Die Syntax von einem INNER JOIN.
2  -- Wir vereinigen Informationen aus den Tabellen Table_1 und Table_2.
3
4  -- Oft, aber nicht immer, hat Table_1 die Spalte 'a' als Primär-
5  -- schlüssel, der von Spalte 'b' in Table_2 als Fremdschlüssel
6  -- referenziert wird.
7  -- In diesem Beispiel geben wir die Werte von Spalte 'x' der Table_1
8  -- und 'y' von Table_2 zurück, nachdem wir beide Tabellen vereinigt
9  -- haben.
10 -- Für Zeilen in Table_1 ohne passende Zeilen in Table_2, wird keine
11 -- Ergebniszeile generiert.
12 SELECT Table_1.x, Table_2.y FROM Table_1
13      INNER JOIN Table_2 ON (Table_1.a = Table_2.b);
```

INNER JOIN: Syntax



- Angenommen, Tabelle `Table_1` hat den Primärschlüssel `a` und Tabelle `Table_2` hat eine Spalte `b`, die als Fremdschlüssel auf `Table_1.a` verweist.
- `SELECT ... FROM Table_1 INNER JOIN Table_2 ON (Table_1.a = Table_2.b)` geht dann Zeile für Zeile durch `Table_1` und sucht für jeden Wert von `Table_1.a` alle passenden Zeilen aus `Table_2` heraus, also alle Zeilen, für die `Table_1.a = Table_2.b` gilt.

```
1  -- Die Syntax von einem INNER JOIN.
2  -- Wir vereinigen Informationen aus den Tabellen Table_1 und Table_2.
3
4  -- Oft, aber nicht immer, hat Table_1 die Spalte 'a' als Primär-
5  -- schlüssel, der von Spalte 'b' in Table_2 als Fremdschlüssel
6  -- referenziert wird.
7  -- In diesem Beispiel geben wir die Werte von Spalte 'x' der Table_1
8  -- und 'y' von Table_2 zurück, nachdem wir beide Tabellen vereinigt
9  -- haben.
10 -- Für Zeilen in Table_1 ohne passende Zeilen in Table_2, wird keine
11 -- Ergebniszeile generiert.
12 SELECT Table_1.x, Table_2.y FROM Table_1
13      INNER JOIN Table_2 ON (Table_1.a = Table_2.b);
```

INNER JOIN: Syntax



- Angenommen, Tabelle `Table_1` hat den Primärschlüssel `a` und Tabelle `Table_2` hat eine Spalte `b`, die als Fremdschlüssel auf `Table_1.a` verweist.
- `SELECT ... FROM Table_1 INNER JOIN Table_2 ON (Table_1.a = Table_2.b)` geht dann Zeile für Zeile durch `Table_1` und sucht für jeden Wert von `Table_1.a` alle passenden Zeilen aus `Table_2` heraus, also alle Zeilen, für die `Table_1.a = Table_2.b` gilt.
- Jede dieser Zeilen wird dann eine Zeile im Ergebnis.

```
1  -- Die Syntax von einem INNER JOIN.
2  -- Wir vereinigen Informationen aus den Tabellen Table_1 und Table_2.
3
4  -- Oft, aber nicht immer, hat Table_1 die Spalte 'a' als Primär-
5  -- schlüssel, der von Spalte 'b' in Table_2 als Fremdschlüssel
6  -- referenziert wird.
7  -- In diesem Beispiel geben wir die Werte von Spalte 'x' der Table_1
8  -- und 'y' von Table_2 zurück, nachdem wir beide Tabellen vereinigt
9  -- haben.
10 -- Für Zeilen in Table_1 ohne passende Zeilen in Table_2, wird keine
11 -- Ergebniszeile generiert.
12 SELECT Table_1.x, Table_2.y FROM Table_1
13      INNER JOIN Table_2 ON (Table_1.a = Table_2.b);
```

INNER JOIN: Syntax



- Angenommen, Tabelle `Table_1` hat den Primärschlüssel `a` und Tabelle `Table_2` hat eine Spalte `b`, die als Fremdschlüssel auf `Table_1.a` verweist.
- `SELECT ... FROM Table_1 INNER JOIN Table_2 ON (Table_1.a = Table_2.b)` geht dann Zeile für Zeile durch `Table_1` und sucht für jeden Wert von `Table_1.a` alle passenden Zeilen aus `Table_2` heraus, also alle Zeilen, für die `Table_1.a = Table_2.b` gilt.
- Jede dieser Zeilen wird dann eine Zeile im Ergebnis.
- Gibt es keine solche Zeile, **wird anders als bei `LEFT JOIN` keine Ergebniszeile generiert.**

INNER JOIN Syntax

```
1  -- Die Syntax von einem INNER JOIN.
2  -- Wir vereinigen Informationen aus den Tabellen Table_1 und Table_2.
3
4  -- Oft, aber nicht immer, hat Table_1 die Spalte 'a' als Primär-
5  -- schlüssel, der von Spalte 'b' in Table_2 als Fremdschlüssel
6  -- referenziert wird.
7  -- In diesem Beispiel geben wir die Werte von Spalte 'x' der Table_1
8  -- und 'y' von Table_2 zurück, nachdem wir beide Tabellen vereinigt
9  -- haben.
10 -- Für Zeilen in Table_1 ohne passende Zeilen in Table_2, wird keine
11 -- Ergebniszeile generiert.
```

```
12 SELECT Table_1.x, Table_2.y FROM Table_1
13      INNER JOIN Table_2 ON (Table_1.a = Table_2.b);
```

```
14
15 --      Table_1      Table_2      Query Result
16 --      a | x      b | y      x | y
17 --      -+-----+-----+-----+
18 --      1 | Hello      1 | A      Hello | A
19 --      2 | World      2 | B      World | B
20 --      3 | from      2 | C      World | C
21 --      4 | HFUU      4 | D      World | E
22 --      2 | E      HFUU | D
```

INNER JOIN: Mehrere INNER JOIN zusammen



- Um Daten aus mehreren Tabellen zu kombinieren, brauchen wir mehrere Joins.

```
-- Die Syntax von *zwei* kombinierten INNER JOINS.
-- Wir vereinigen Informationen der Tabellen Table_1, Table_2, Table_3.

-- Nur wenn Zeilen in allen drei Tabellen passen, werden Ergebniszeilen
-- generiert.
SELECT Table_1.x, Table_2.y, Table_3.z FROM Table_1
      INNER JOIN Table_2 ON (Table_1.a = Table_2.b)
      INNER JOIN Table_3 ON (Table_2.y = Table_3.c);
```

```
-- Table_1 Table_2 Table_3 Query Result
-- a | x      b | y      c | z      x | y | z
-- +-----+-----+-----+-----+
-- 1 | Hello    1 | A      A | 10     Hello | A | 10
-- 2 | World    2 | B      B | 20     World | B | 20
-- 3 | from     2 | C      B | 30     World | B | 30
-- 4 | HFUU     4 | D      C | 40     World | C | 40
--                2 | E      D | 50     HFUU  | D | 50
--                F | 60
```

Anfrage Zusammenbauen



- Wir fangen an mit `SELECT <something> FROM demand.`

Anfrage Zusammenbauen



- Wir fangen an mit `SELECT <something> FROM demand.`
- Dann schreiben wir `INNER JOIN customer ON (customer.id = demand.customer).`

Anfrage Zusammenbauen



- Wir fangen an mit `SELECT <something> FROM demand`.
- Dann schreiben wir `INNER JOIN customer ON (customer.id = demand.customer)`.
- Für jede Zeile in `demand` gibt uns das die passende Zeile in `customer`.

Anfrage Zusammenbauen



- Wir fangen an mit `SELECT <something> FROM demand`.
- Dann schreiben wir `INNER JOIN customer ON (customer.id = demand.customer)`.
- Für jede Zeile in `demand` gibt uns das die passende Zeile in `customer`.
- Nun fügen wir einfach das nächste `INNER JOIN` an, in dem wir direkt danach `INNER JOIN product ON (product.id = demand.product)` schreiben.

Anfrage Zusammenbauen



- Wir fangen an mit `SELECT <something> FROM demand`.
- Dann schreiben wir `INNER JOIN customer ON (customer.id = demand.customer)`.
- Für jede Zeile in `demand` gibt uns das die passende Zeile in `customer`.
- Nun fügen wir einfach das nächste `INNER JOIN` an, in dem wir direkt danach `INNER JOIN product ON (product.id = demand.product)` schreiben.
- Damit bekommen wir die passende Zeile aus der Tabelle `product`.

Anfrage Zusammenbauen



- Wir fangen an mit `SELECT <something> FROM demand.`
- Dann schreiben wir `INNER JOIN customer ON (customer.id = demand.customer).`
- Für jede Zeile in `demand` gibt uns das die passende Zeile in `customer`.
- Nun fügen wir einfach das nächste `INNER JOIN` an, in dem wir direkt danach `INNER JOIN product ON (product.id = demand.product)` schreiben.
- Damit bekommen wir die passende Zeile aus der Tabelle `product`.
- Da wir diesmal sowohl Kunden- als auch Produktnamen haben, selektieren wir `customer.name AS customer_name` und `product.name AS product_name`.

Anfrage Zusammenbauen



- Wir fangen an mit `SELECT <something> FROM demand`.
- Dann schreiben wir `INNER JOIN customer ON (customer.id = demand.customer)`.
- Für jede Zeile in `demand` gibt uns das die passende Zeile in `customer`.
- Nun fügen wir einfach das nächste `INNER JOIN` an, in dem wir direkt danach `INNER JOIN product ON (product.id = demand.product)` schreiben.
- Damit bekommen wir die passende Zeile aus der Tabelle `product`.
- Da wir diesmal sowohl Kunden- als auch Produktnamen haben, selektieren wir `customer.name AS customer_name` und `product.name AS product_name`.
- Wir selektieren ebenfalls `product.price AS price`, `demand.amount AS amount` und `demand.ordered as ordered`.

Anfrage Zusammenbauen



- Wir fangen an mit `SELECT <something> FROM demand`.
- Dann schreiben wir `INNER JOIN customer ON (customer.id = demand.customer)`.
- Für jede Zeile in `demand` gibt uns das die passende Zeile in `customer`.
- Nun fügen wir einfach das nächste `INNER JOIN` an, in dem wir direkt danach `INNER JOIN product ON (product.id = demand.product)` schreiben.
- Damit bekommen wir die passende Zeile aus der Tabelle `product`.
- Da wir diesmal sowohl Kunden- als auch Produktnamen haben, selektieren wir `customer.name AS customer_name` und `product.name AS product_name`.
- Wir selektieren ebenfalls `product.price AS price`, `demand.amount AS amount` und `demand.ordered as ordered`.
- Wir sortieren die Ausgabe mit `ORDER BY` lexikographisch nach `customer_name`, `ordered` und `product_name`, `price` und `amount`.

Die Anfrage



```
1  /* Query the sales information. */
2
3  -- We combine the three tables customer, product, and demand.
4  -- Basically, for each row in 'demand', we find the corresponding rows
5  -- in the 'customer' and 'product' tables (via the INNER JOIN).
6  -- This gives us one long row with all the information for one 'demand'.
7  -- We now choose only some elements of this long row and rename them.
8  -- We extract the name of the customer and refer to it "customer_name".
9  -- We extract the name of the product and refer to it as "product_name".
10 -- We also print the "amount" from each customer demand and the price.
11 -- We also print when the purchase as made.
12 SELECT customer.name AS customer_name, product.name AS product_name,
13         product.price AS price,          demand.amount AS amount,
14         demand.ordered AS ordered
15 FROM demand INNER JOIN customer ON (customer.id = demand.customer)
16         INNER JOIN product ON (product.id = demand.product)
17 ORDER BY customer_name, ordered, product_name, price, amount;
```

Die Anfrage



```
$ psql "postgres://boss:superboss123@localhost/factory" -v ON_ERROR_STOP  
↪ =1 -ebf select_sale_data.sql
```

customer_name	product_name	price	amount	ordered
Bebba	Shoe, Size 37	152.99	7	2024-12-16
Bebba	Large Purse	150.00	10	2025-01-16
Bebbo	Shoe, Size 38	154.99	2	2024-12-09
Bebbo	Shoe, Size 40	158.99	7	2024-12-30
Bebbo	Shoe, Size 41	160.99	4	2025-01-12
Bebbo	Shoe, Size 38	154.99	6	2025-02-05
Bibbo	Shoe, Size 42	162.99	12	2024-11-21
Bibbo	Shoe, Size 40	158.99	3	2025-01-05

```
(8 rows)
```

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



Sichten / Views



Sichten / Views



- Nun haben wir sehr schön die Verkaufsdaten unserer Fabrik aufbereitet.

Sichten / Views



- Nun haben wir sehr schön die Verkaufsdaten unserer Fabrik aufbereitet.
- Wahrscheinlich brauchen wir diese Anfrage sehr häufig, wahrscheinlich jedes Mal leicht verändert.



Sichten / Views



- Nun haben wir sehr schön die Verkaufsdaten unserer Fabrik aufbereitet.
- Wahrscheinlich brauchen wir diese Anfrage sehr häufig, wahrscheinlich jedes Mal leicht verändert.
- Vielleicht wollen wir nur Bestellungen aus dem aktuellen Jahr.

Sichten / Views



- Nun haben wir sehr schön die Verkaufsdaten unserer Fabrik aufbereitet.
- Wahrscheinlich brauchen wir diese Anfrage sehr häufig, wahrscheinlich jedes Mal leicht verändert.
- Vielleicht wollen wir nur Bestellungen aus dem aktuellen Jahr. Dann würden wir eine WHERE-Klausel anfügen.

Sichten / Views



- Nun haben wir sehr schön die Verkaufsdaten unserer Fabrik aufbereitet.
- Wahrscheinlich brauchen wir diese Anfrage sehr häufig, wahrscheinlich jedes Mal leicht verändert.
- Vielleicht wollen wir nur Bestellungen aus dem aktuellen Jahr. Dann würden wir eine WHERE-Klausel anfügen.
- Vielleicht wollen wir die Geldmenge pro Kunde aufaddieren.

Sichten / Views



- Nun haben wir sehr schön die Verkaufsdaten unserer Fabrik aufbereitet.
- Wahrscheinlich brauchen wir diese Anfrage sehr häufig, wahrscheinlich jedes Mal leicht verändert.
- Vielleicht wollen wir nur Bestellungen aus dem aktuellen Jahr. Dann würden wir eine WHERE-Klausel anfügen.
- Vielleicht wollen wir die Geldmenge pro Kunde aufaddieren. Oder pro Produkt.

Sichten / Views



- Nun haben wir sehr schön die Verkaufsdaten unserer Fabrik aufbereitet.
- Wahrscheinlich brauchen wir diese Anfrage sehr häufig, wahrscheinlich jedes Mal leicht verändert.
- Vielleicht wollen wir nur Bestellungen aus dem aktuellen Jahr. Dann würden wir eine WHERE-Klausel anfügen.
- Vielleicht wollen wir die Geldmenge pro Kunde aufaddieren. Oder pro Produkt.
- Jedes Mal würden wir eine ähnliche Abfrage schreiben.

Sichten / Views



- Nun haben wir sehr schön die Verkaufsdaten unserer Fabrik aufbereitet.
- Wahrscheinlich brauchen wir diese Anfrage sehr häufig, wahrscheinlich jedes Mal leicht verändert.
- Vielleicht wollen wir nur Bestellungen aus dem aktuellen Jahr. Dann würden wir eine [WHERE](#)-Klausel anfügen.
- Vielleicht wollen wir die Geldmenge pro Kunde aufaddieren. Oder pro Produkt.
- Jedes Mal würden wir eine ähnliche Abfrage schreiben. Das ist nicht gut.



- Nun haben wir sehr schön die Verkaufsdaten unserer Fabrik aufbereitet.
- Wahrscheinlich brauchen wir diese Anfrage sehr häufig, wahrscheinlich jedes Mal leicht verändert.
- Vielleicht wollen wir nur Bestellungen aus dem aktuellen Jahr. Dann würden wir eine [WHERE](#)-Klausel anfügen.
- Vielleicht wollen wir die Geldmenge pro Kunde aufaddieren. Oder pro Produkt.
- Jedes Mal würden wir eine ähnliche Abfrage schreiben. Das ist nicht gut.
- Zum Glück gibt es eine Lösung: Wir können Anfragen als sogenannte *Views* (Sichten) speichern!¹⁸.

- Nun haben wir sehr schön die Verkaufsdaten unserer Fabrik aufbereitet.
- Wahrscheinlich brauchen wir diese Anfrage sehr häufig, wahrscheinlich jedes Mal leicht verändert.
- Vielleicht wollen wir nur Bestellungen aus dem aktuellen Jahr. Dann würden wir eine **WHERE**-Klausel anfügen.
- Vielleicht wollen wir die Geldmenge pro Kunde aufaddieren. Oder pro Produkt.
- Jedes Mal würden wir eine ähnliche Abfrage schreiben. Das ist nicht gut.
- Zum Glück gibt es eine Lösung: Wir können Anfragen als sogenannte *Views* (Sichten) speichern!¹⁸.
- Ein View funktioniert im Grunde wie eine Tabelle.

Problem mit unserer Anfrage

- Wir brauchen unsere Anfrage also nur als Sicht zu speichern.



Problem mit unserer Anfrage



- Wir brauchen unsere Anfrage also nur als Sicht zu speichern.
- Als wir das tun wollen, fällt uns ein Problem auf.



Problem mit unserer Anfrage



- Wir brauchen unsere Anfrage also nur als Sicht zu speichern.
- Als wir das tun wollen, fällt uns ein Problem auf: Kundennamen sind nicht notwendigerweise eindeutig (nicht UNIQUE).

Problem mit unserer Anfrage



- Wir brauchen unsere Anfrage also nur als Sicht zu speichern.
- Als wir das tun wollen, fällt uns ein Problem auf: Kundennamen sind nicht notwendigerweise eindeutig (nicht **UNIQUE**).
- Wenn wir unsere Anfrage später nutzen wollen, um Verkäufe pro Kunde aufzuaddieren, dann würden alle Verkäufe nach Kundennamen gruppiert und dann summiert.

Problem mit unserer Anfrage



- Wir brauchen unsere Anfrage also nur als Sicht zu speichern.
- Als wir das tun wollen, fällt uns ein Problem auf: Kundennamen sind nicht notwendigerweise eindeutig (nicht **UNIQUE**).
- Wenn wir unsere Anfrage später nutzen wollen, um Verkäufe pro Kunde aufzuaddieren, dann würden alle Verkäufe nach Kundennamen gruppiert und dann summiert. Wenn zwei Kunden den selben Namen haben, werden ihre Bestellungen vermengt.

Problem mit unserer Anfrage



- Wir brauchen unsere Anfrage also nur als Sicht zu speichern.
- Als wir das tun wollen, fällt uns ein Problem auf: Kundennamen sind nicht notwendigerweise eindeutig (nicht **UNIQUE**).
- Wenn wir unsere Anfrage später nutzen wollen, um Verkäufe pro Kunde aufzuaddieren, dann würden alle Verkäufe nach Kundennamen gruppiert und dann summiert. Wenn zwei Kunden den selben Namen haben, werden ihre Bestellungen vermengt.
- Telefonnummern sind aber eindeutig in unserer Datenbank.

Problem mit unserer Anfrage



- Wir brauchen unsere Anfrage also nur als Sicht zu speichern.
- Als wir das tun wollen, fällt uns ein Problem auf: Kundennamen sind nicht notwendigerweise eindeutig (nicht `UNIQUE`).
- Wenn wir unsere Anfrage später nutzen wollen, um Verkäufe pro Kunde aufzuaddieren, dann würden alle Verkäufe nach Kundennamen gruppiert und dann summiert. Wenn zwei Kunden den selben Namen haben, werden ihre Bestellungen vermengt.
- Telefonnummern sind aber eindeutig in unserer Datenbank.
- Als `customer_name`, werden wir also `customer.name || ', ' || customer.phone` benutzen.

Problem mit unserer Anfrage



- Wir brauchen unsere Anfrage also nur als Sicht zu speichern.
- Als wir das tun wollen, fällt uns ein Problem auf: Kundennamen sind nicht notwendigerweise eindeutig (nicht `UNIQUE`).
- Wenn wir unsere Anfrage später nutzen wollen, um Verkäufe pro Kunde aufzuaddieren, dann würden alle Verkäufe nach Kundennamen gruppiert und dann summiert. Wenn zwei Kunden den selben Namen haben, werden ihre Bestellungen vermengt.
- Telefonnummern sind aber eindeutig in unserer Datenbank.
- Als `customer_name`, werden wir also `customer.name ||', '|| customer.phone` benutzen.
- Der doppelten Strich `||` steht für das aneinanderreihen von Zeichenketten.

Problem mit unserer Anfrage



- Wir brauchen unsere Anfrage also nur als Sicht zu speichern.
- Als wir das tun wollen, fällt uns ein Problem auf: Kundennamen sind nicht notwendigerweise eindeutig (nicht `UNIQUE`).
- Wenn wir unsere Anfrage später nutzen wollen, um Verkäufe pro Kunde aufzuaddieren, dann würden alle Verkäufe nach Kundennamen gruppiert und dann summiert. Wenn zwei Kunden den selben Namen haben, werden ihre Bestellungen vermengt.
- Telefonnummern sind aber eindeutig in unserer Datenbank.
- Als `customer_name`, werden wir also `customer.name ||', '|| customer.phone` benutzen.
- Der doppelten Strich `||` steht für das aneinanderreihen von Zeichenketten.
`'Hello ' || 'Word!'` ergibt `'Hello World!'`.

Problem mit unserer Anfrage



- Wir brauchen unsere Anfrage also nur als Sicht zu speichern.
- Als wir das tun wollen, fällt uns ein Problem auf: Kundennamen sind nicht notwendigerweise eindeutig (nicht `UNIQUE`).
- Wenn wir unsere Anfrage später nutzen wollen, um Verkäufe pro Kunde aufzuaddieren, dann würden alle Verkäufe nach Kundennamen gruppiert und dann summiert. Wenn zwei Kunden den selben Namen haben, werden ihre Bestellungen vermengt.
- Telefonnummern sind aber eindeutig in unserer Datenbank.
- Als `customer_name`, werden wir also `customer.name ||', '|| customer.phone` benutzen.
- Der doppelten Strich `||` steht für das aneinanderreihen von Zeichenketten.
`'Hello ' || 'Word!'` ergibt `'Hello World!'`.
- Problem gelöst: Da Telefonnummern eindeutig sind, sind Name + Telefonnummer auch eindeutig.

Anfrage als Sicht speichern und benutzen



- Wir können unsere Anfrage nun als Sicht speichern.

Anfrage als Sicht speichern und benutzen



- Wir können unsere Anfrage nun als Sicht speichern.
- Das geht sehr einfach: Wir schreiben `CREATE VIEW sale AS ...`, wobei `...` durch unsere Anfrage zu ersetzen ist.

Anfrage als Sicht speichern und benutzen



- Wir können unsere Anfrage nun als Sicht speichern.
- Das geht sehr einfach: Wir schreiben `CREATE VIEW sale AS ...`, wobei `...` durch unsere Anfrage zu ersetzen ist.
- Damit wird unsere Anfrage unter dem Namen `sale` gespeichert.

Anfrage als Sicht speichern und benutzen



- Wir können unsere Anfrage nun als Sicht speichern.
- Das geht sehr einfach: Wir schreiben `CREATE VIEW sale AS ...`, wobei `...` durch unsere Anfrage zu ersetzen ist.
- Damit wird unsere Anfrage unter dem Namen `sale` gespeichert.
- Sie kann dann wie eine Tabelle verwendet werden (nur dass man keine Daten oder ändern anfügen kann).

Anfrage als Sicht speichern und benutzen



- Wir können unsere Anfrage nun als Sicht speichern.
- Das geht sehr einfach: Wir schreiben `CREATE VIEW sale AS ...`, wobei `...` durch unsere Anfrage zu ersetzen ist.
- Damit wird unsere Anfrage unter dem Namen `sale` gespeichert.
- Sie kann dann wie eine Tabelle verwendet werden (nur dass man keine Daten oder ändern anfügen kann).
- Wir können schreiben `SELECT * FROM sale;`.

Die Anfrage



```
1  /* Create a view showing the sales information. */
2
3  -- We combine the three tables customer, product, and demand.
4  -- Basically, for each row in 'demand', we find the corresponding rows
5  -- in the 'customer' and 'product' tables (via the INNER JOIN).
6  -- This gives us one long row with all the information for one 'demand'.
7  -- We now choose only some elements of this long row and rename them.
8  -- We extract the name of the customer combined with their phone number
9  -- and refer to it "customer_name".
10 -- We extract the name of the product and refer to it as "product_name".
11 -- We also print the "amount" from each customer demand and the price.
12 -- We also print when the purchase as made.
13 CREATE VIEW sale AS
14     SELECT customer.name || ', ' || customer.phone AS customer_name,
15            product.name AS product_name, product.price AS price,
16            demand.amount AS amount, demand.ordered AS ordered
17 FROM demand INNER JOIN customer ON (customer.id = demand.customer)
18            INNER JOIN product ON (product.id = demand.product)
19 ORDER BY customer_name, ordered, product_name, price, amount;
20
21 -- We can use the view as if it was a table!
22 SELECT * FROM sale;
```

Die Anfrage



```
$ psql "postgres://boss:superboss123@localhost/factory" -v ON_ERROR_STOP  
↪ =1 -ebf create_view_sale.sql
```

```
CREATE VIEW
```

customer_name	product_name	price	amount	ordered
Bebba , 333333333333	Shoe , Size 37	152.99	7	2024-12-16
Bebba , 333333333333	Large Purse	150.00	10	2025-01-16
Bebbo , 555555555555	Shoe , Size 38	154.99	2	2024-12-09
Bebbo , 555555555555	Shoe , Size 40	158.99	7	2024-12-30
Bebbo , 555555555555	Shoe , Size 41	160.99	4	2025-01-12
Bebbo , 555555555555	Shoe , Size 38	154.99	6	2025-02-05
Bibbo , 999999999999	Shoe , Size 42	162.99	12	2024-11-21
Bibbo , 999999999999	Shoe , Size 40	158.99	3	2025-01-05

```
(8 rows)
```

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



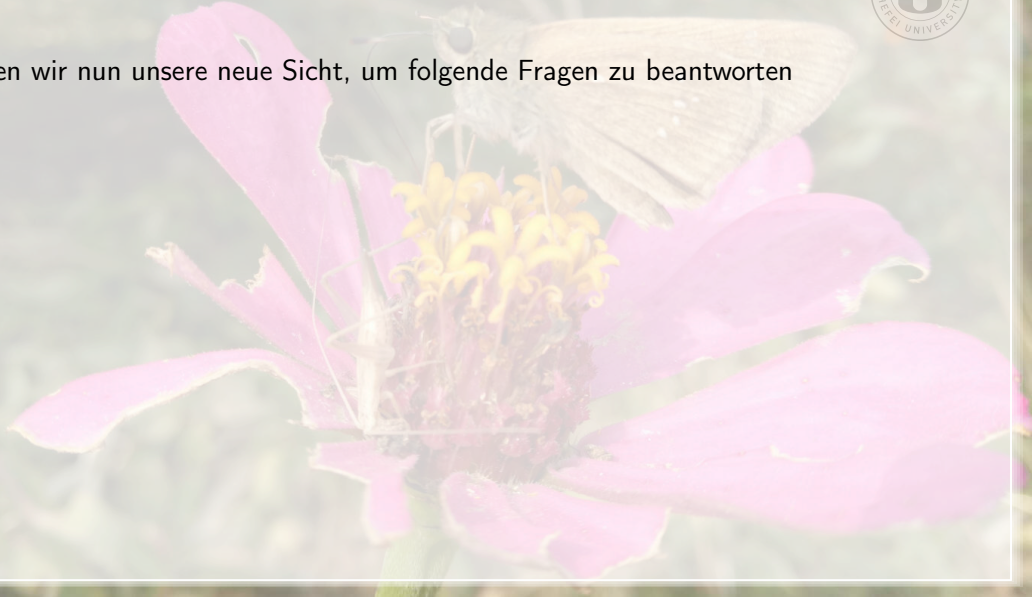
Die Sicht Benutzen



Benutzen unserer neuen Sicht



- Benutzen wir nun unsere neue Sicht, um folgende Fragen zu beantworten



Benutzen unserer neuen Sicht



- Benutzen wir nun unsere neue Sicht, um folgende Fragen zu beantworten:
 - Welcher Kunde hat den meisten Umsatz gemacht?

Benutzen unserer neuen Sicht



- Benutzen wir nun unsere neue Sicht, um folgende Fragen zu beantworten:
 - Welcher Kunde hat den meisten Umsatz gemacht?
 - Welches Produkt hat den meisten Umsatz gemacht?

Benutzen unserer neuen Sicht



- Benutzen wir nun unsere neue Sicht, um folgende Fragen zu beantworten:
 - Welcher Kunde hat den meisten Umsatz gemacht?
 - Welches Produkt hat den meisten Umsatz gemacht?
- Wir können den Gesamtpreis jeder Bestellung einfach ausrechnen.

Benutzen unserer neuen Sicht



- Benutzen wir nun unsere neue Sicht, um folgende Fragen zu beantworten:
 - Welcher Kunde hat den meisten Umsatz gemacht?
 - Welches Produkt hat den meisten Umsatz gemacht?
- Wir können den Gesamtpreis jeder Bestellung einfach ausrechnen. Wir kennen ja die Anzahl der bestellten Einheiten (`amount`) und den Preis pro Einheit (`price`).

Benutzen unserer neuen Sicht



- Benutzen wir nun unsere neue Sicht, um folgende Fragen zu beantworten:
 - Welcher Kunde hat den meisten Umsatz gemacht?
 - Welches Produkt hat den meisten Umsatz gemacht?
- Wir können den Gesamtpreis jeder Bestellung einfach ausrechnen. Wir kennen ja die Anzahl der bestellten Einheiten (`amount`) und den Preis pro Einheit (`price`).
- Pro Zeile müssen wir also einfach `amount * price` berechnen.

Benutzen unserer neuen Sicht



- Benutzen wir nun unsere neue Sicht, um folgende Fragen zu beantworten:
 - Welcher Kunde hat den meisten Umsatz gemacht?
 - Welches Produkt hat den meisten Umsatz gemacht?
- Wir können den Gesamtpreis jeder Bestellung einfach ausrechnen. Wir kennen ja die Anzahl der bestellten Einheiten (`amount`) und den Preis pro Einheit (`price`).
- Pro Zeile müssen wir also einfach `amount * price` berechnen.
- Da wir die Summe dieser Werte pro Kunde bzw. pro Produkt wollen, müssen wir die Zeilen jeweils nach `customer_name` oder `product_name` gruppieren.

Benutzen unserer neuen Sicht



- Benutzen wir nun unsere neue Sicht, um folgende Fragen zu beantworten:
 - Welcher Kunde hat den meisten Umsatz gemacht?
 - Welches Produkt hat den meisten Umsatz gemacht?
- Wir können den Gesamtpreis jeder Bestellung einfach ausrechnen. Wir kennen ja die Anzahl der bestellten Einheiten (`amount`) und den Preis pro Einheit (`price`).
- Pro Zeile müssen wir also einfach `amount * price` berechnen.
- Da wir die Summe dieser Werte pro Kunde bzw. pro Produkt wollen, müssen wir die Zeilen jeweils nach `customer_name` oder `product_name` gruppieren.
- Pro Gruppe berechnen wir dann `SUM(amount * price)`¹.

Benutzen unserer neuen Sicht



- Benutzen wir nun unsere neue Sicht, um folgende Fragen zu beantworten:
 - Welcher Kunde hat den meisten Umsatz gemacht?
 - Welches Produkt hat den meisten Umsatz gemacht?
- Wir können den Gesamtpreis jeder Bestellung einfach ausrechnen. Wir kennen ja die Anzahl der bestellten Einheiten (`amount`) und den Preis pro Einheit (`price`).
- Pro Zeile müssen wir also einfach `amount * price` berechnen.
- Da wir die Summe dieser Werte pro Kunde bzw. pro Produkt wollen, müssen wir die Zeilen jeweils nach `customer_name` oder `product_name` gruppieren.
- Pro Gruppe berechnen wir dann `SUM(amount * price)`¹.
- Schließlich sortieren wir die Werte absteigend, weil wir ja die Bestseller am Anfang haben wollen.

Verkaufserlös nach Kunde

```
1  /* Get the total sales per customer. */
2  SELECT customer_name, SUM(amount * price) AS customer_sale FROM sale
3  GROUP BY customer_name ORDER BY customer_sale DESC;
```

```
1  $ psql "postgres://boss:superboss123@localhost/factory" -v ON_ERROR_STOP
   ↪ =1 -ebf select_from_view_sale_1a.sql
2  customer_name      | customer_sale
3  -----+-----
4  Bebbo , 55555555555 |      2996.81
5  Bebba , 33333333333 |      2570.93
6  Bibbo , 99999999999 |      2432.85
7  (3 rows)
8
9  # psql 16.11 succeeded with exit code 0.
```

Verkaufserlös nach Produkt

```
1  /* Get the total sales per product. */
2  SELECT product_name, SUM(amount) AS total_amount,
3         SUM(amount * price) AS product_sale FROM sale
4         GROUP BY product_name ORDER BY product_sale DESC;
```

```
1  $ psql "postgres://boss:superboss123@localhost/factory" -v ON_ERROR_STOP
   ↪ =1 -ebf select_from_view_sale_1b.sql
```

```
2  product_name | total_amount | product_sale
3  -----+-----+-----
4  Shoe, Size 42 |          12 |      1955.88
5  Shoe, Size 40 |          10 |      1589.90
6  Large Purse   |          10 |      1500.00
7  Shoe, Size 38 |           8 |      1239.92
8  Shoe, Size 37 |           7 |      1070.93
9  Shoe, Size 41 |           4 |       643.96
10 (6 rows)
```

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

Weitere Schritte

- Wie konnten die Sicht tatsächlich wie eine Tabelle verwenden.



Weitere Schritte



- Wie konnten die Sicht tatsächlich wie eine Tabelle verwenden.
- Man kann es sich so vorstellen: Wenn wir ein **SELECT** über sie ausführen, dann wird intern zuerst die Anfrage unserer Sicht ausgeführt und unser **SELECT** dann darüber angewendet.



Weitere Schritte



- Wie konnten die Sicht tatsächlich wie eine Tabelle verwenden.
- Man kann es sich so vorstellen: Wenn wir ein `SELECT` über sie ausführen, dann wird intern zuerst die Anfrage unserer Sicht ausgeführt und unser `SELECT` dann darüber angewendet.
- Nehmen wir nun an, dass unsere Datenbank schon mehrere Jahre existiert.



Weitere Schritte



- Wie konnten die Sicht tatsächlich wie eine Tabelle verwenden.
- Man kann es sich so vorstellen: Wenn wir ein `SELECT` über sie ausführen, dann wird intern zuerst die Anfrage unserer Sicht ausgeführt und unser `SELECT` dann darüber angewendet.
- Nehmen wir nun an, dass unsere Datenbank schon mehrere Jahre existiert.
- Wir wollen dann vielleicht nicht alle Verkäufe, die jemals stattfanden, aufsummieren.



Weitere Schritte



- Wie konnten die Sicht tatsächlich wie eine Tabelle verwenden.
- Man kann es sich so vorstellen: Wenn wir ein `SELECT` über sie ausführen, dann wird intern zuerst die Anfrage unserer Sicht ausgeführt und unser `SELECT` dann darüber angewendet.
- Nehmen wir nun an, dass unsere Datenbank schon mehrere Jahre existiert.
- Wir wollen dann vielleicht nicht alle Verkäufe, die jemals stattfanden, aufsummieren.
- Stattdessen wollen wir vielleicht nur die Verkäufe im Jahr 2025 beachten.

Weitere Schritte



- Wie konnten die Sicht tatsächlich wie eine Tabelle verwenden.
- Man kann es sich so vorstellen: Wenn wir ein `SELECT` über sie ausführen, dann wird intern zuerst die Anfrage unserer Sicht ausgeführt und unser `SELECT` dann darüber angewendet.
- Nehmen wir nun an, dass unsere Datenbank schon mehrere Jahre existiert.
- Wir wollen dann vielleicht nicht alle Verkäufe, die jemals stattfanden, aufsummieren.
- Stattdessen wollen wir vielleicht nur die Verkäufe im Jahr 2025 beachten.
- Alles, was wir dafür ändern müssen, ist eine `WHERE`-Klausel an unsere Anfragen anhängen, mit der Bedingung `(ordered >= '2025-01-01')` and `(ordered < '2026-01-01')`.

Verkaufserlös nach Kunde in 2025



```
1  /* Get the total sales per customer in 2025. */
2  SELECT customer_name, SUM(amount * price) AS customer_sale FROM sale
3      WHERE (ordered >= '2025-01-01') AND (ordered < '2026-01-01')
4      GROUP BY customer_name ORDER BY customer_sale DESC;
```

```
1  $ psql "postgres://boss:superboss123@localhost/factory" -v ON_ERROR_STOP
   ↪ =1 -ebf select_from_view_sale_2a.sql
2      customer_name      | customer_sale
3  -----+-----
4      Bebbio, 555555555555 |          1573.90
5      Bebbio, 333333333333 |          1500.00
6      Bibbo, 999999999999 |           476.97
7  (3 rows)
8
9  # psql 16.11 succeeded with exit code 0.
```

Verkaufserlös nach Produkt in 2025

```
1  /* Get the total sales per product in 2025. */
2  SELECT product_name, SUM(amount) AS total_amount,
3         SUM(amount * price) AS product_sale FROM sale
4         WHERE (ordered >= '2025-01-01') AND (ordered < '2026-01-01')
5         GROUP BY product_name ORDER BY product_sale DESC;
```

```
1  $ psql "postgres://boss:superboss123@localhost/factory" -v ON_ERROR_STOP
   ↪ =1 -ebf select_from_view_sale_2b.sql
2  product_name | total_amount | product_sale
3  -----+-----+-----
4  Large Purse  |           10 |       1500.00
5  Shoe, Size 38 |            6 |        929.94
6  Shoe, Size 41 |            4 |        643.96
7  Shoe, Size 40 |            3 |        476.97
8  (4 rows)
9
10 # psql 16.11 succeeded with exit code 0.
```



Zusammenfassung



Zusammenfassung



- Wir sind nun ein ganzes Stück stärker geworden.

Zusammenfassung



- Wir sind nun ein ganzes Stück stärker geworden.
- Mit **JOIN**s sind wir nun in der Lage, Daten aus verschiedenen Tabellen zusammenzufügen.

Zusammenfassung



- Wir sind nun ein ganzes Stück stärker geworden.
- Mit **JOIN**s sind wir nun in der Lage, Daten aus verschiedenen Tabellen zusammenzufügen.
- Das ist eines der wichtigsten Features, das Datenbanken uns bieten.

Zusammenfassung



- Wir sind nun ein ganzes Stück stärker geworden.
- Mit **JOINs** sind wir nun in der Lage, Daten aus verschiedenen Tabellen zusammenzufügen.
- Das ist eines der wichtigsten Features, das Datenbanken uns bieten.
- Wir haben auch gelernt, wie wir Sichten (Views) erstellen können, also gespeicherte Anfragen.

Zusammenfassung



- Wir sind nun ein ganzes Stück stärker geworden.
- Mit **JOIN**s sind wir nun in der Lage, Daten aus verschiedenen Tabellen zusammenzufügen.
- Das ist eines der wichtigsten Features, das Datenbanken uns bieten.
- Wir haben auch gelernt, wie wir Sichten (Views) erstellen können, also gespeicherte Anfragen.
- Damit ist es uns nicht nur möglich, bestimmte, oft benötigte Anfragen zu speichern um unnötiges, mehrmaliges eingeben zu verhindern.

Zusammenfassung



- Wir sind nun ein ganzes Stück stärker geworden.
- Mit **JOINs** sind wir nun in der Lage, Daten aus verschiedenen Tabellen zusammenzufügen.
- Das ist eines der wichtigsten Features, das Datenbanken uns bieten.
- Wir haben auch gelernt, wie wir Sichten (Views) erstellen können, also gespeicherte Anfragen.
- Damit ist es uns nicht nur möglich, bestimmte, oft benötigte Anfragen zu speichern um unnötiges, mehrmaliges eingeben zu verhindern.
- Sichten sind auch ein zentrales Werkzeug, um verschiedenen Werkzeugen und Benutzern ... nun ... verschiedene Sichten auf die Daten zu bieten.

Zusammenfassung



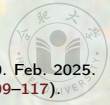
- Wir sind nun ein ganzes Stück stärker geworden.
- Mit **JOINs** sind wir nun in der Lage, Daten aus verschiedenen Tabellen zusammenzufügen.
- Das ist eines der wichtigsten Features, das Datenbanken uns bieten.
- Wir haben auch gelernt, wie wir Sichten (Views) erstellen können, also gespeicherte Anfragen.
- Damit ist es uns nicht nur möglich, bestimmte, oft benötigte Anfragen zu speichern um unnötiges, mehrmaliges eingeben zu verhindern.
- Sichten sind auch ein zentrales Werkzeug, um verschiedenen Werkzeugen und Benutzern ... nun ... verschiedene Sichten auf die Daten zu bieten.
- Wir können sie nutzen, um bestimmten Nutzern oder Applikationen nur bestimmte Teile unserer gespeicherten Daten zugänglich zu machen.



谢谢你们！
Thank you!
Vielen Dank!



References I



- [1] "Aggregate Functions". In: *PostgreSQL Documentation*. 17.4. The PostgreSQL Global Development Group (PGDG), 20. Feb. 2025. Kap. 9.21. URL: <https://www.postgresql.org/docs/17/functions-aggregate.html> (besucht am 2025-02-27) (siehe S. 109–117).
- [2] Adam Aspin und Karine Aspin. *Query Answers with MariaDB – Volume I: Introduction to SQL Queries*. Tetras Publishing, Okt. 2018. ISBN: 978-1-9996172-4-0. See also³ (siehe S. 137, 146).
- [3] Adam Aspin und Karine Aspin. *Query Answers with MariaDB – Volume II: In-Depth Querying*. Tetras Publishing, Okt. 2018. ISBN: 978-1-9996172-5-7. See also² (siehe S. 137, 146).
- [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. 146, 148).
- [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. 146).
- [6] Ben Beitler. *Hands-On Microsoft Access 2019*. Birmingham, England, UK: Packt Publishing Ltd, März 2020. ISBN: 978-1-83898-747-3 (siehe S. 146).
- [7] Tim Berners-Lee. *Re: Qualifiers on Hypertext links...* Geneva, Switzerland: World Wide Web project, European Organization for Nuclear Research (CERN) und Newsgroups: alt.hypertext, 6. Aug. 1991. URL: <https://www.w3.org/People/Berners-Lee/1991/08/art-6484.txt> (besucht am 2025-02-05) (siehe S. 148).
- [8] Alex Berson. *Client/Server Architecture*. 2. Aufl. Computer Communications Series. New York, NY, USA: McGraw-Hill, 29. März 1996. ISBN: 978-0-07-005664-0 (siehe S. 145).
- [9] Bernard Obeng Boateng. *Data Modeling with Microsoft Excel*. Birmingham, England, UK: Packt Publishing Ltd, Nov. 2023. ISBN: 978-1-80324-028-2 (siehe S. 146).
- [10] Silvia Botros und Jeremy Tinley. *High Performance MySQL*. 4. Aufl. Sebastopol, CA, USA: O'Reilly Media, Inc., Nov. 2021. ISBN: 978-1-4920-8051-0 (siehe S. 147).
- [11] Ed Bott. *Windows 11 Inside Out*. Hoboken, NJ, USA: Microsoft Press, Pearson Education, Inc., Feb. 2023. ISBN: 978-0-13-769132-6 (siehe S. 146).

References II



- [12] Ron Brash und Ganesh Naik. *Bash Cookbook*. Birmingham, England, UK: Packt Publishing Ltd, Juli 2018. ISBN: 978-1-78862-936-2 (siehe S. 145).
- [13] Jason Cannon. *High Availability for the LAMP Stack*. Shelter Island, NY, USA: Manning Publications, Juni 2022 (siehe S. 145, 147).
- [14] Donald D. Chamberlin. "50 Years of Queries". *Communications of the ACM (CACM)* 67(8):110–121, Aug. 2024. New York, NY, USA: Association for Computing Machinery (ACM). ISSN: 0001-0782. doi:10.1145/3649887. URL: <https://cacm.acm.org/research/50-years-of-queries> (besucht am 2025-01-09) (siehe S. 147).
- [15] Christmas, FL, USA: Simon Sez IT. *Microsoft Access 2021 – Beginner to Advanced*. Birmingham, England, UK: Packt Publishing Ltd, Aug. 2023. ISBN: 978-1-83546-911-8 (siehe S. 146).
- [16] 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. 148).
- [17] 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. 147).
- [18] "CREATE VIEW". 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-createview.html> (besucht am 2025-03-03) (siehe S. 81–90).
- [19] *Database Language SQL*. Techn. Ber. ANSI X3.135-1986. Washington, D.C., USA: American National Standards Institute (ANSI), 1986 (siehe S. 147).
- [20] 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. 147).
- [21] *Database Language SQL*. International Standard ISO 9075-1987. Geneva, Switzerland: International Organization for Standardization (ISO), 1987 (siehe S. 147).
- [22] Paul Deitel, Harvey Deitel und Abbey Deitel. *Internet & World Wide WebWJ: How to Program*. 5. Aufl. Hoboken, NJ, USA: Pearson Education, Inc., Nov. 2011. ISBN: 978-0-13-299045-5 (siehe S. 148).

References III



- [23] Pooyan Doozandeh und Frank E. Ritter. "Some Tips for Academic Writing and Using Microsoft Word". *XRDS: Crossroads, The ACM Magazine for Students* 26(1):10–11, Herbst 2019. New York, NY, USA: Association for Computing Machinery (ACM). ISSN: 1528-4972. doi:10.1145/3351470 (siehe S. 147).
- [24] 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. 146, 147).
- [25] Steve Fanning, Vasudev Narayanan, „flywire“, Olivier Hallot, Jean Hollis Weber, Jenna Sargent, Pulkit Krishna, Dan Lewis, Peter Schofield, Jochen Schiffers, Robert Großkopf, Jost Lange, Martin Fox, Hazel Russman, Steve Schwettman, Alain Romedenne, Andrew Pitonyak, Jean-Pierre Ledure, Drew Jensen und Randolph Gam. *Base Guide 7.3. Revision 1. Based on LibreOffice 7.3 Community*. Berlin, Germany: The Document Foundation, Aug. 2022. URL: <https://books.libreoffice.org/en/BG73/BG73-BaseGuide.pdf> (besucht am 2025-01-13) (siehe S. 146).
- [26] 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. 147).
- [27] Jonas Gamalielsson und Björn Lundell. "Long-Term Sustainability of Open Source Software Communities beyond a Fork: A Case Study of LibreOffice". In: *8th IFIP WG 2.13 International Conference on Open Source Systems: Long-Term Sustainability OSS'2012*. 10.–13. Sep. 2012, Hammamet, Tunisia. Hrsg. von Imed Hammouda, Björn Lundell, Tommi Mikkonen und Walt Scacchi. Bd. 378. IFIP Advances in Information and Communication Technology (IFIPAICT). Berlin/Heidelberg, Germany: Springer-Verlag GmbH Germany, 2012, S. 29–47. ISSN: 1868-4238. ISBN: 978-3-642-33441-2. doi:10.1007/978-3-642-33442-9_3 (siehe S. 146).
- [28] Dawn Griffiths. *Excel Cookbook – Receipts for Mastering Microsoft Excel*. Sebastopol, CA, USA: O'Reilly Media, Inc., Mai 2024. ISBN: 978-1-0981-4332-9 (siehe S. 146).
- [29] 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. 147).
- [30] 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. 147).

References IV



- [31] Michael Hausenblas. *Learning Modern Linux*. Sebastopol, CA, USA: O'Reilly Media, Inc., Apr. 2022. ISBN: 978-1-0981-0894-6 (siehe S. 146).
- [32] 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. 145, 148).
- [33] Manuel Hoffmann, Frank Nagle und Yanuo Zhou. *The Value of Open Source Software*. Working Paper 24-038. Boston, MA, USA: Harvard Business School, 1. Jan. 2024. URL: https://www.hbs.edu/ris/Publication%20Files/24-038_51f8444f-502c-4139-8bf2-56eb4b65c58a.pdf (besucht am 2025-06-04) (siehe S. 147).
- [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. 147).
- [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. 147).
- [36] "Joined Tables". In: *PostgreSQL Documentation*. 17.4. The PostgreSQL Global Development Group (PGDG), 20. Feb. 2025. Kap. 7.2.1.1. URL: <https://www.postgresql.org/docs/17/queries-table-expressions.html#QUERIES-JOIN> (besucht am 2025-03-01) (siehe S. 26–29, 54–61).
- [37] Jay LaCroix. *Mastering Ubuntu Server*. 4. Aufl. Birmingham, England, UK: Packt Publishing Ltd, Sep. 2022. ISBN: 978-1-80323-424-3 (siehe S. 147).
- [38] Joan Lambert und Curtis Frye. *Microsoft Office Step by Step (Office 2021 and Microsoft 365)*. Hoboken, NJ, USA: Microsoft Press, Pearson Education, Inc., Juni 2022. ISBN: 978-0-13-754493-6 (siehe S. 146).
- [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. 147).

References V



- [40] *LibreOffice – The Document Foundation*. Berlin, Germany: The Document Foundation, 2024. URL: <https://www.libreoffice.org> (besucht am 2024-12-12) (siehe S. 146).
- [41] 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. 145).
- [42] Mark Lutz. *Learning Python*. 6. Aufl. Sebastopol, CA, USA: O'Reilly Media, Inc., März 2025. ISBN: 978-1-0981-7130-8 (siehe S. 147).
- [43] *MariaDB Server Documentation*. Milpitas, CA, USA: MariaDB, 2025. URL: <https://mariadb.com/kb/en/documentation> (besucht am 2025-04-24) (siehe S. 146).
- [44] Ron McFadyen und Cindy Miller. *Relational Databases and Microsoft Access*. 3. Aufl. Palatine, IL, USA: Harper College, 2014–2019. URL: <https://harpercollege.pressbooks.pub/relationaldatabases> (besucht am 2025-04-11) (siehe S. 146).
- [45] 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. 147).
- [46] *Microsoft Word*. Redmond, WA, USA: Microsoft Corporation, 2024. URL: <https://www.microsoft.com/en-us/microsoft-365/word> (besucht am 2024-12-12) (siehe S. 147).
- [47] Cameron Newham und Bill Rosenblatt. *Learning the Bash Shell – Unix Shell Programming: Covers Bash 3.0*. 3. Aufl. Sebastopol, CA, USA: O'Reilly Media, Inc., 2005. ISBN: 978-0-596-00965-6 (siehe S. 145).
- [48] Regina O. Obe und Leo S. Hsu. *PostgreSQL: Up and Running*. 3. Aufl. Sebastopol, CA, USA: O'Reilly Media, Inc., Okt. 2017. ISBN: 978-1-4919-6336-4 (siehe S. 147).
- [49] Robert Orfali, Dan Harkey und Jeri Edwards. *Client/Server Survival Guide*. 3. Aufl. Chichester, West Sussex, England, UK: John Wiley and Sons Ltd., 25. Jan. 1999. ISBN: 978-0-471-31615-2 (siehe S. 145).

References VI



- [50] Yasset Pérez-Riverol, Laurent Gatto, Rui Wang, Timo Sachsenberg, Julian Uszkoreit, Felipe da Veiga Leprevost, Christian Fufezan, Tobias Ternent, Stephen J. Eglén, Daniel S. Katz, Tom J. Pollard, Alexander Kononov, Robert M. Flight, Kai Blin und Juan Antonio Vizcaíno. "Ten Simple Rules for Taking Advantage of Git and GitHub". *PLOS Computational Biology* 12(7), 14. Juli 2016. San Francisco, CA, USA: Public Library of Science (PLOS). ISSN: **1553-7358**. doi:[10.1371/JOURNAL.PCBI.1004947](https://doi.org/10.1371/JOURNAL.PCBI.1004947) (siehe S. **145**).
- [51] *PostgreSQL Documentation*. 17.4. The PostgreSQL Global Development Group (PGDG), Feb. 2025. URL: <https://www.postgresql.org/docs/17/index.html> (besucht am 2025-02-25).
- [52] *PostgreSQL Essentials: Leveling Up Your Data Work*. Sebastopol, CA, USA: O'Reilly Media, Inc., März 2024 (siehe S. **147**).
- [53] Abhishek Ratan, Eric Chou, Pradeeban Kathiravelu und Dr. M.O. Faruque Sarker. *Python Network Programming*. Birmingham, England, UK: Packt Publishing Ltd, Jan. 2019. ISBN: **978-1-78883-546-6** (siehe S. **145**).
- [54] Federico Razzoli. *Mastering MariaDB*. Birmingham, England, UK: Packt Publishing Ltd, Sep. 2014. ISBN: **978-1-78398-154-0** (siehe S. **146**).
- [55] Mike Reichardt, Michael Gundall und Hans D. Schotten. "Benchmarking the Operation Times of NoSQL and MySQL Databases for Python Clients". In: *47th Annual Conference of the IEEE Industrial Electronics Society (IECON'2021)*. 13.–15. Okt. 2021, Toronto, ON, Canada. Piscataway, NJ, USA: Institute of Electrical and Electronics Engineers (IEEE), 2021, S. 1–8. ISSN: **2577-1647**. ISBN: **978-1-6654-3554-3**. doi:[10.1109/IECON48115.2021.9589382](https://doi.org/10.1109/IECON48115.2021.9589382) (siehe S. **147**).
- [56] Mark Richards und Neal Ford. *Fundamentals of Software Architecture: An Engineering Approach*. Sebastopol, CA, USA: O'Reilly Media, Inc., Jan. 2020. ISBN: **978-1-4920-4345-4** (siehe S. **145**).
- [57] Winfried Seimert. *LibreOffice 7.3 – Praxiswissen für Ein- und Umsteiger*. Blaufelden, Schwäbisch Hall, Baden-Württemberg, Germany: mitp Verlags GmbH & Co. KG, Apr. 2022. ISBN: **978-3-7475-0504-5** (siehe S. **146**).
- [58] 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. **146**).
- [59] Anna Skoulikari. *Learning Git*. Sebastopol, CA, USA: O'Reilly Media, Inc., Mai 2023. ISBN: **978-1-0981-3391-7** (siehe S. **145**).

References VII



- [60] 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. 147).
- [61] "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. 147).
- [62] 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. 143, 147).
- [63] 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 of⁶² (siehe S. 147).
- [64] Allen Taylor. *Introducing SQL and Relational Databases*. New York, NY, USA: Apress Media, LLC, Sep. 2018. ISBN: 978-1-4842-3841-7 (siehe S. 147).
- [65] 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. 147).
- [66] 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. 146).
- [67] Mariot Tsitoara. *Beginning Git and GitHub: Version Control, Project Management and Teamwork for the New Developer*. New York, NY, USA: Apress Media, LLC, März 2024. ISBN: 979-8-8688-0215-7 (siehe S. 145, 148).
- [68] Laurie A. Ulrich und Ken Cook. *Access For Dummies*. Hoboken, NJ, USA: For Dummies (Wiley), Dez. 2021. ISBN: 978-1-119-82908-9 (siehe S. 146).
- [69] Sander van Vugt. *Linux Fundamentals*. 2. Aufl. Hoboken, NJ, USA: Pearson IT Certification, Juni 2022. ISBN: 978-0-13-792931-3 (siehe S. 146).

References VIII



- [70] 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. 145–147).
- [71] 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. 147).
- [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. 147).
- [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. 147).
- [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. 145).
- [75] Pavlo V. Zahorodko und Pavlo V. Merzlykin. "An Approach for Processing and Document Flow Automation for Microsoft Word and LibreOffice Writer File Formats". In: *4th Workshop for Young Scientists in Computer Science & Software Engineering (CS&SE@SW'2021)*. 18. Dez. 2021, Virtual Event and Kryvyi Rih, Ukraine. Hrsg. von Arnold E. Kiv, Serhiy O. Semerikov, Vladimir N. Soloviev und Andrii M. Striuk. Bd. 3077 der Reihe CEUR Workshop Proceedings (CEUR-WS.org). Aachen, Nordrhein-Westfalen, Germany: CEUR-WS Team, Rheinisch-Westfälische Technische Hochschule (RWTH) Aachen, 2022, S. 66–82. ISSN: 1613-0073. URL: <https://ceur-ws.org/Vol-3077/paper12.pdf> (besucht am 2025-10-04) (siehe S. 146, 147).
- [76] Giorgio Zarrelli. *Mastering Bash*. Birmingham, England, UK: Packt Publishing Ltd, Juni 2017. ISBN: 978-1-78439-687-9 (siehe S. 145).

Glossary (in English) I



Bash is a the shell used under Ubuntu Linux, i.e., the program that „runs“ in the terminal and interprets your commands, allowing you to start and interact with other programs^{12,47,76}. Learn more at <https://www.gnu.org/software/bash>.

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

client-server architecture is a system design where a central server receives requests from one or multiple clients^{8,41,49,53,56}. These requests and responses are usually sent over network connections. A typical example for such a system is the World Wide Web (WWW), where web servers host websites and make them available to web browsers, the clients. Another typical example is the structure of database (DB) software, where a central server, the DBMS, offers access to the DB to the different clients. Here, the client can be some terminal software shipping with the DBMS, such as psql, or the different applications that access the DBs.

DB A *database* is an organized collection of structured information or data, typically stored electronically in a computer system. Databases are discussed in our book *Databases*⁷⁰.

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 DB⁷⁴.

Git is a distributed Version Control Systems (VCS) which allows multiple users to work on the same code while preserving the history of the code changes^{59,67}. Learn more at <https://git-scm.com>.

GitHub is a website where software projects can be hosted and managed via the Git VCS^{50,67}. Learn more at <https://github.com>.

IT information technology

LAMP Stack A system setup for web applications: Linux, Apache (a web server), MySQL, and the server-side scripting language PHP^{13,32}.

Glossary (in English) II



- LibreOffice is an open source office suite^{27,40,57} which is a good and free alternative to Microsoft Office. It offers software such as LibreOffice Writer, LibreOffice Calc, and LibreOffice Base. See⁷⁰ for more information and installation instructions.
- LibreOffice Base is a DBMS that can work on stand-alone files but also connect to other popular relational databases^{25,57}. It is part of LibreOffice^{27,40,57} and has functionality that is comparable to Microsoft Access^{6,15,68}.
- LibreOffice Calc is a spreadsheet software that allows you to arrange and perform calculations with data in a tabular grid. It is a free and open source spreadsheet software^{40,57}, i.e., an alternative to Microsoft Excel. It is part of LibreOffice^{27,40,57}.
- LibreOffice Writer is a free and open source text writing program⁷⁵ and part of LibreOffice^{27,40,57}. It is a good alternative to Microsoft Word.
- Linux is the leading open source operating system, i.e., a free alternative for Microsoft Windows^{4,31,58,66,69}. We recommend using it for this course, for software development, and for research. Learn more at <https://www.linux.org>. Its variant Ubuntu is particularly easy to use and install.
- MariaDB An open source relational database management system that has forked off from MySQL^{2,3,5,24,43,54}. See <https://mariadb.org> for more information.
- Microsoft Access is a DBMS that can work on DBs stored in single, stand-alone files but also connect to other popular relational databases^{6,15,44,68}. It is part of Microsoft Office. A free and open source alternative to this commercial software is LibreOffice Base.
- Microsoft Excel is a spreadsheet program that allows users to store, organize, manipulate, and calculate data in tabular structures^{9,28,38}. It is part of Microsoft Office. A free alternative to this commercial software is LibreOffice Calc^{40,57}.
- Microsoft Office is a commercial suite of office software, including Microsoft Excel, Microsoft Word, and Microsoft Access³⁸. LibreOffice is a free and open source alternative.
- Microsoft Windows is a commercial proprietary operating system¹¹. 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>.

Glossary (in English) III



Microsoft Word is one of the leading text writing programs^{23,46,75} and part of Microsoft Office. A free alternative to this commercial software is the LibreOffice Writer.

MySQL An open source relational database management system^{10,24,55,65,73}. MySQL is famous for its use in the LAMP Stack. See <https://www.mysql.com> for more information.

OSS Open source software, i.e., software that can freely be used, whose source code is made available in the internet, and which is usually developed cooperatively over the internet as well³³. Typical examples are Python, Linux, Git, and PostgreSQL.

PostgreSQL An open source object-relational DBMS^{26,48,52,65}. See <https://postgresql.org> for more information.

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

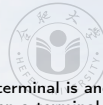
Python The Python programming language^{34,39,42,71}, i.e., what you will learn about in our book⁷¹. Learn more at <https://python.org>.

relational database A relational DB is a database that organizes data into rows (tuples, records) and columns (attributes), which collectively form tables (relations) where the data points are related to each other^{17,29,30,60,64,70,72}.

server In a client-server architecture, the server is a process that fulfills the requests of the clients. It usually waits for incoming communication carrying the requests from the clients. For each request, it takes the necessary actions, performs the required computations, and then sends a response with the result of the request. Typical examples for servers are web servers¹³ in the internet as well as DBMSes. It is also common to refer to the computer running the server processes as server as well, i.e., to call it the „server computer“³⁷.

SQL The *Structured Query Language* is basically a programming language for querying and manipulating relational databases^{14,19–21,35,45,61–64}. It is understood by many DBMSes. You find the Structured Query Language (SQL) commands supported by PostgreSQL in the reference⁶¹.

Glossary (in English) IV



- terminal** A terminal is a text-based window where you can enter commands and execute them^{4,16}. 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 `Win` + `R`, dann Schreiben von `cmd`, dann Drück auf `↵`. Under Ubuntu Linux, `Ctrl` + `Alt` + `T` opens a terminal, which then runs a Bash shell inside.
- Ubuntu** is a variant of the open source operating system Linux^{16,32}. 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>.
- VCS** A *Version Control System* is a software which allows you to manage and preserve the historical development of your program code⁶⁷. A distributed VCS allows multiple users to work on the same code and upload their changes to the server, which then preserves the change history. The most popular distributed VCS is Git.
- WWW** World Wide Web^{7,22}