SQL

Der Grundkurs für Ausbildung und Praxis

12 Differenzierte Auswertungen

Auswertungen mithilfe von Aggregatfunktionen erstellen.

- Grundkurs
  - Einfache Statistik mit `MIN()`, `MAX()`, `SUM()`, `COUNT()` und `AVG()`
  - Tabellen mit `GROUP BY` in Gruppen zerlegen
  - Aggregatfunktionen auf Gruppen anwenden
  - Gruppenergebnisse mit `HAVING` aussortieren
- Vertiefendes
  - Aggregatfunktionen
  - Summenbildung mit `WITH ROLLUP`
  - Gruppieren nach Ausdrücken
  - Gruppieren nach mehr als einer Spalte
  - Einfluss von Indizes auf Gruppierungen

Die Quelltexte dieses Kapitels stehen in der Datei `listing09.sql` (siehe Listing 28.9, Listing 28.49 und Listing 28.34).

12.1 Statistisches mit Aggregatfunktionen

Die Daten, die bisher aus einem `SELECT` kamen, sind immer Originaldaten gewesen. Soll heißen, es wurden nur Texte oder Zahlen angezeigt, die in den Tabellen so abgelegt waren.

Mithilfe von Aggregatfunktionen werden nun Auswertungen über die Daten erstellt. Eine Übersicht der verfügbaren Aggregatfunktionen finden Sie in Abschnitt 26.2.3. Der *ausdruck* in Abschnitt 26.2.3 ist meist ein Spaltenname oder eine mathematische Kombination aus Spaltennamen.

Die wichtigsten Aggregatfunktionen sind: `MIN()`, `MAX()`, `SUM()`, `COUNT()` und `AVG()`. Mit `MIN()` und `MAX()` lassen sich der minimale und maximale Wert einer Liste ermitteln. `SUM()` addiert die Werte einer Liste auf, und `COUNT()` zählt die Anzahl von Werten. Mit `AVG()` wird das arithmetische Mittel einer Werteliste berechnet.

Diese Funktionen lassen sich überall da einbauen, wo man mit Werten arbeitet. Beim `SELECT` beispielsweise in der Spaltenliste oder beim `WHERE` in den Vergleichen usw. Wir wollen mal die meisten Aggregatfunktionen an unseren Daten ausprobieren:

Was ist der durchschnittliche Preis unserer Artikel?

Der Artikelpreis steht in der Spalte `einzelpreis`. Da dieser mit der Option `NOT NULL` erstellt wurde, erwarten wir keine Schwierigkeiten.

```
SELECT AVG(einzelpreis) FROM artikel;
------------------+
| AVG(einzelpreis) |
+------------------+
|    12.5777777778 |
+------------------+
```

Wie viele Zeilen hat eine Tabelle?

Wir verwenden hier die Tabelle `kunde`, es könnte aber auf diese Art und Weise von jeder beliebigen Tabelle die Anzahl der Zeilen ermittelt werden.

```
SELECT COUNT(*) FROM kunde;
+----------+
| COUNT(*) |
+----------+
|        6 |
+----------+
```

Wie viele Kunden haben eine eigene Lieferadresse?

Da der Inhalt des Fremdschlüssels `liefer_adresse_id` den Wert `NULL` hat, wenn keine eigene Lieferadresse erfasst ist, kann über diese Spalte die Anzahl ermittelt werden.

```
SELECT COUNT(liefer_adresse_id) FROM kunde;
+--------------------------+
| COUNT(liefer_adresse_id) |
+--------------------------+
|                        1 |
+--------------------------+
```

Wie viele unterschiedliche Rechnungsadressen gibt es?

Die Tabelle `kunde` hat in der Spalte `rechnung_adresse_id` die Fremdschlüsselwerte zu den Rechnungsadressen abgelegt. Die Anzahl unterschiedlicher Werte liefert mir das gewünschte Ergebnis.

```
SELECT COUNT(DISTINCT(rechnung_adresse_id)) FROM kunde;
+-------------------------------------+
| COUNT(DISTINCT(rechnung_adresse_id)) |
+-------------------------------------+
|                                   4 |
+-------------------------------------+
```

Was ist unser teuerster und unser billigster Einzelpreis?

Die Tabelle `artikel` enthält in der Spalte `einzelpreis` unsere Preise.

```
SELECT MAX(einzelpreis), MIN(einzelpreis) FROM artikel;
+------------------+------------------+
| MAX(einzelpreis) | MIN(einzelpreis) |
+------------------+------------------+
|        56.260000 |         0.520000 |
+------------------+------------------+
```

Wie viele Einzelartikel sind bestellt worden?

Die Bestellmenge steht in der Tabelle `bestellung_position` in der Spalte `menge`.

```
SELECT SUM(menge) FROM bestellung_position;
+------------+
| SUM(menge) |
+------------+
|  96.000000 |
+------------+
```

Wie hoch ist das Bestellvolumen?

Hier werden zwei Tabellen zur Auswertung benötigt: `bestellung_position` und `artikel`. Bilden Sie zuerst den `INNER JOIN` wie in Abschnitt 11.2.1 beschrieben.

```
mysql> SELECT
    ->  bp.menge, a.einzelpreis
    ->  FROM
    ->   bestellung_position bp INNER JOIN artikel a USING(artikel_id);
+-----------+-------------+
| menge     | einzelpreis |
+-----------+-------------+
| 30.000000 |    0.520000 |
| 50.000000 |    3.420000 |
|  1.000000 |   20.100000 |
| 10.000000 |    0.520000 |
|  5.000000 |   15.100000 |
+-----------+-------------+
```

Der Wert einer Position lässt sich nun einfach dadurch ermitteln, dass die beiden Werte `menge` und `einzelpreis` miteinander multipliziert werden:

```
mysql> SELECT
    ->  bp.menge * a.einzelpreis 'Positionswert'
    ->  FROM
    ->   bestellung_position bp INNER JOIN artikel a USING(artikel_id);
+------------------+
| Positionswert    |
+------------------+
|  15.600000000000 |
| 171.000000000000 |
|  20.100000000000 |
|   5.200000000000 |
|  75.500000000000 |
+------------------+
```

Zum Schluss werden die Positionswerte summiert.

```
1  SELECT
2   SUM(bp.menge * a.einzelpreis)
3   FROM
4    bestellung_position bp INNER JOIN artikel a USING(artikel_id);
5  +------------------------------+
6  | SUM(bp.menge * a.einzelpreis) |
7  +------------------------------+
8  |             287.400000000000 |
9  +------------------------------+
```

Wir haben hier ein Beispiel dafür, dass als Parameter einer Aggregatfunktion nicht nur Spaltennamen, sondern auch Ausdrücke vorkommen können.

Aufgabe 12.1: Hier ein paar Übungsausgaben. Für die Übungsausgaben habe ich in `listing09.sql` noch einen Lagerbestand aufgebaut.

a) Ergänzen Sie das ER-Modell in Bild 3.2 um die Lagerverwaltung in `listing09.sql`.
b) Ermitteln Sie die durchschnittliche Bestellmenge.
c) Ermitteln Sie die Anzahl der Artikel mit der Währung `USD`.
d) Ermitteln Sie die Anzahl der Privatkunden.
e) Ermitteln Sie die Anzahl der Kunden, die noch keine Bestellung aufgegeben haben.
f) Was ist unsere kleinste und was unsere größte Bestellmenge?
g) Ermitteln Sie, wie viele *ml* Tinte noch auf Lager sind.
h) Ermitteln Sie den Wert des Lagers.

12.2 Tabelle in Gruppen zerlegen

Die Aggregatfunktionen liefern uns bis jetzt ihre Ergebnisse bezogen auf die ganze Tabelle, also *eine* Zahl. Es kommt aber viel häufiger vor, dass man die Auswertung pro Kunde, pro Postleitzahl, pro Bestellung, pro Artikel etc. vornehmen möchte, z.B. Anzahl der Bestellpositionen pro Artikel. Wir brauchen ein Werkzeug, was die Anwendung von Aggregatfunktionen auf Teiltabellen (Gruppen) ermöglicht.

SQL:2023, **MySQL/MariaDB**, **PostgreSQL**, **T-SQL**

```
SELECT [DISTINCT]
  {*|spaltenliste|ausdruck}
  FROM
    from_ausdruck
  [WHERE bedingung]
  [GROUP BY spaltenliste|ausdruck]
  [ORDER BY spaltenname [ASC|DESC] [,spaltenname [ASC|DESC]]*]
  [LIMIT [offset,] anzahl]
  [INTO OUTFILE 'dateiname' exportoptionen]
;
```

Wie viele Positionen pro Bestellungen gibt es?

Da es irgendwas mit *Anzahl* zu tun hat, ist `COUNT()` unser Mann.

```
mysql> SELECT COUNT(*)
    ->  FROM bestellung_position
    -> ;
+----------+
| COUNT(*) |
+----------+
|        5 |
+----------+
```

Jetzt wissen wir die Anzahl der Positionen überhaupt. Wir wollen aber wissen, wie viele es pro Bestellung sind. Wir müssen also ein `GROUP BY` einfügen. Aber was steht in *spaltenliste* hinter dem `GROUP BY`? Die Spalte, die bestimmt, zu welcher Gruppe die Zeile gehört. In unserem Fall der Fremdschlüssel auf die Tabelle `bestellung`.

```
1  mysql> SELECT
2      ->  bestellung_id Bestellnummer, COUNT(*) ‘Anzahl der Positionen‘
3      ->  FROM
4      ->   bestellung_position
5      ->  GROUP BY
6      ->   bestellung_id
7      -> ;
8  +---------------+-----------------------+
9  | Bestellnummer | Anzahl der Positionen |
10 +---------------+-----------------------+
11 |             1 |                     3 |
12 |             2 |                     2 |
13 +---------------+-----------------------+
```

Noch mal: Schauen wir uns die erste Version an, so wird die Aggregatfunktion `COUNT(*)` auf die gesamte Tabelle angewendet. In der zweiten Version weisen wir an, dass die Tabelle in Gruppen zerlegt wird. Als Unterscheidungsmerkmal wird in Zeile 6 die Spalte `bestellung_id` angegeben. Das bedeutet, dass alle Zeilen mit dem gleichen Spaltenwert zur gleichen Gruppe gehören. Daher gibt es jetzt zwei Gruppen: für jede `bestellung_id` eine.

Ist eine Tabelle in Gruppen zerlegt worden, so werden die Aggregatfunktionen immer pro Gruppe ausgeführt. In unserem Fall bedeutet das, dass pro Bestellung `COUNT(*)` die Zeilen in `bestellung_position` zählt.

Hinweis: In Zeile 2 wird ein Alias mit Leerzeichen verwendet. Deshalb muss um den Alias eine Stringbegrenzung erfolgen. Die *normalen* Hochkomma ’ oder Gänsefüßchen " können in MySQL/MariaDB nicht verwendet werden, da an dieser Stelle auch Stringkonstanten stehen könnten. Als neues Zeichen wird auch ein Hochkomma ‘ verwendet, aber das auf der Taste links neben dem Backspace.

Wir wollen wissen, wie oft ein Artikel bestellt wurde

Dabei soll der Artikelname mit ausgegeben werden. Zuerst wenden wir bzgl. der beiden Tabellen `bestellung_position` und `artikel` das in Abschnitt 11.2.1 beschriebene Verfahren an, um den Artikelname zu erfahren.

```
mysql> SELECT a.bezeichnung, bp.menge
    ->  FROM
    ->   bestellung_position bp INNER JOIN artikel a USING(artikel_id)
    -> ;
+---------------+-----------+
| bezeichnung   | menge     |
+---------------+-----------+
| Silberzwiebel | 30.000000 |
| Tulpenzwiebel | 50.000000 |
| Spaten        |  1.000000 |
| Silberzwiebel | 10.000000 |
| Schaufel      |  5.000000 |
+---------------+-----------+
```

Jetzt heißt es herauszufinden, nach welcher Spalte die Gruppen gebildet werden sollen. Da hilft ein Tipp: Wenn die Aufgabenstellung Formulierungen wie *pro Artikel* oder *für jeden Kunden* enthält, dann sind dies in der Regel die Gruppierungsmerkmale. In unserem Beispiel wird demnach nach der Spalte `artikel_id` gruppiert. Die Aggregatfunktion, um die Anzahl der Artikel zu ermitteln, ist hier `SUM()` und nicht `COUNT()`, da pro Zeile mehr als ein Artikel verkauft wird.

```
mysql> SELECT
    ->  a.bezeichnung ‘Artikelname‘, SUM(bp.menge) ‘Anzahl bestellter Artikel‘
    ->  FROM
    ->   bestellung_position bp INNER JOIN artikel a USING(artikel_id)
    ->  GROUP BY
    ->   artikel_id
    -> ;
+---------------+--------------------------+
| Artikelname   | Anzahl bestellter Artikel |
+---------------+--------------------------+
| Silberzwiebel |                40.000000 |
| Tulpenzwiebel |                50.000000 |
| Schaufel      |                 5.000000 |
| Spaten        |                 1.000000 |
+---------------+--------------------------+
```

Jetzt ist es aber so, dass hier die Artikel, die in keiner Bestellung vorkommen, gar nicht aufgeführt werden. `Ein RIGHT OUTER JOIN` könnte helfen:

```
mysql> SELECT
    ->  a.bezeichnung ‘Artikelname‘, SUM(bp.menge) ‘Anzahl bestellter Artikel‘
    ->  FROM
    ->   bestellung_position bp RIGHT JOIN artikel a USING(artikel_id)
    ->  GROUP BY
    ->   artikel_id;
+---------------+--------------------------+
| Artikelname   | Anzahl bestellter Artikel |
+---------------+--------------------------+
| Papier (100)  |                     NULL |
| Tinte (gold)  |                     NULL |
| Tinte (rot)   |                     NULL |
| Tinte (blau)  |                     NULL |
| Feder         |                     NULL |
| Silberzwiebel |                40.000000 |
| Tulpenzwiebel |                50.000000 |
| Schaufel      |                 5.000000 |
| Spaten        |                 1.000000 |
+---------------+--------------------------+
```

Wie Sie die Ausgabe NULL in eine schöne Ausgabe verwandeln können, erfahren Sie in Kapitel 15. Ach was, ich kann mich nicht beherrschen:

```
mysql> SELECT
    ->  a.bezeichnung 'Artikelname',
    ->  CASE
    ->   WHEN SUM(bp.menge) IS NULL THEN 0
    ->   ELSE SUM(bp.menge)
    ->  END AS 'Anzahl bestellter Artikel'
    ->  FROM
    ->   bestellung_position bp RIGHT JOIN artikel a USING(artikel_id)
    ->  GROUP BY
    ->   artikel_id
    -> ;
+--------------+--------------------------+
| Artikelname  | Anzahl bestellter Artikel |
+--------------+--------------------------+
| Papier (100) |                        0 |
| Tinte (gold) |                        0 |
| Tinte (rot)  |                        0 |
| Tinte (blau) |                        0 |
| Feder        |                        0 |
| Silberzwiebel |                40.000000 |
| Tulpenzwiebel |                50.000000 |
| Schaufel     |                 5.000000 |
| Spaten       |                 1.000000 |
+--------------+--------------------------+
```

Mit dem CASE können Sie eine Fallunterscheidung auf Werte vornehmen und die Ausgabe danach anpassen. Den Rest erzähle ich Ihnen wirklich erst später ;-).

Hinweis: Im SQL:2023-Standard muss das Gruppierungsmerkmal in der Spaltenliste hinter dem SELECT auftauchen (siehe Zeile 10). PostgreSQL und T-SQL setzen diese Forderung um.

```
oshop=# SELECT
oshop-#  a.bezeichnung Artikelname,
oshop-#  CASE
oshop-#   WHEN SUM(bp.menge) IS NULL THEN 0
oshop-#   ELSE SUM(bp.menge)
oshop-#  END AS Anzahl_bestellter_Artikel
oshop-#  FROM
oshop-#   bestellung_position bp RIGHT JOIN artikel a USING(artikel_id)
oshop-#  GROUP BY
oshop-#   a.bezeichnung
oshop-# ;
```

Ein schönes Feature ist die Summenbildung bei Aggregatfunktionen mit WITH ROLLUP. Dabei wird am Ende der Auswertung eine zusätzliche Zeile eingefügt, welche die summierten Auswertungen enthält. Die Gruppierungsspalte (hier artikel_id) bekommt den Wert NULL zugewiesen.

In unserem Beispiel wird in Zeile 10 die Option angehängt. Dadurch werden in der letzten Zeile des Ergebnisses (Zeile 24) die Summe aller Werte der Spalte Anzahl bestellter Artikel ausgegeben.

```
mysql> SELECT
    ->  artikel_id,
    ->  CASE
    ->   WHEN SUM(bp.menge) IS NULL THEN 0
    ->   ELSE SUM(bp.menge)
    ->  END AS ‘Anzahl bestellter Artikel‘
    ->  FROM
    ->   bestellung_position bp RIGHT JOIN artikel a USING(artikel_id)
    ->  GROUP BY
    ->   artikel_id WITH ROLLUP
    -> ;
+------------+--------------------------+
| artikel_id | Anzahl bestellter Artikel |
+------------+--------------------------+
|       3001 |                        0 |
|       3005 |                        0 |
|       3006 |                        0 |
|       3007 |                        0 |
|       3010 |                        0 |
|       7856 |                40.000000 |
|       7863 |                50.000000 |
|       9010 |                 5.000000 |
|       9015 |                 1.000000 |
|       NULL |                96.000000 |    ← Hier steht die Summe!
+------------+--------------------------+
```

Werden mehr als ein Gruppierungselement benannt, werden Zwischenzeilen mit Zwischensummen ausgegeben.

12.3 Gruppenergebnisse filtern

SQL:2023, **MySQL/MariaDB**, **PostgreSQL**, **T-SQL**

```
SELECT [DISTINCT]
  {*|spaltenliste|ausdruck}
  FROM
    from_ausdruck
  [WHERE bedingung]
  [[GROUP BY spaltenliste|ausdruck]
    [HAVING bedingung]]
  [ORDER BY spaltenname [ASC|DESC] [,spaltenname [ASC|DESC]]*]
  [LIMIT [offset ,] anzahl]
  [INTO OUTFILE 'dateiname' exportoptionen]
;
```

Wir wollen nur solche Artikel sehen, die mehr als zehn Mal verkauft wurden. Intuitiv wollen wir dazu die WHERE-Klausel verwenden, müssen aber feststellen, dass das nicht geht.

```
mysql> SELECT
    ->  a.bezeichnung ‘Artikelname‘, SUM(bp.menge) ‘Anzahl bestellter Artikel‘
    ->  FROM
    ->   bestellung_position bp INNER JOIN artikel a USING(artikel_id)
    ->  WHERE
```

```
6       ->   SUM(bp.menge) > 10
7       ->  GROUP BY
8       ->   artikel_id
9       -> ;
10  ERROR 1111 (HY000): Invalid use of group function
```

Die Definition 35 sagt aus, dass die Operation auf die Zeilen eingeschränkt wird, für welche die Bedingung der WHERE-Klausel TRUE ergibt. Das Problem ist aber, dass wir gar nicht die Zeilen beschränken wollen, die in die Berechnungen einfließen, sondern die Ergebnisse, die aus den Berechnungen herausfließen. Dazu wird ein HAVING gebraucht.

Definition 48: HAVING
Durch ein *HAVING* werden die Ergebnisse eines GROUP BY verworfen, für welche die Bedingung nicht TRUE ergibt.

```
1   mysql> SELECT
2       ->  a.bezeichnung ‘Artikelname‘, SUM(bp.menge) ‘Anzahl bestellter Artikel‘
3       ->  FROM
4       ->   bestellung_position bp INNER JOIN artikel a USING(artikel_id)
5       ->  GROUP BY
6       ->   artikel_id
7       ->  HAVING
8       ->   SUM(bp.menge) > 10
9       -> ;
10  +---------------+--------------------------+
11  | Artikelname   | Anzahl bestellter Artikel |
12  +---------------+--------------------------+
13  | Silberzwiebel |                 40.000000 |
14  | Tulpenzwiebel |                 50.000000 |
15  +---------------+--------------------------+
```

In Zeile 7 wird der HAVING verwendet. Der Bau der Bedingung kann die gleichen Vergleichsoperatoren (siehe Abschnitt 26.3.1) und Verknüpfungsoperatoren (siehe Abschnitt 26.3.2) verwenden wie das WHERE.

Um den Unterschied zwischen WHERE und HAVING zu verdeutlichen, wird die obige Auswertung um solche Zeilen eingeschränkt, deren Artikelname mit Silber beginnen.

```
1   mysql> SELECT
2       ->  a.bezeichnung ‘Artikelname‘, SUM(bp.menge) ‘Anzahl bestellter Artikel‘
3       ->  FROM
4       ->   bestellung_position bp INNER JOIN artikel a USING(artikel_id)
5       ->  WHERE
6       ->   a.bezeichnung LIKE ’Silber%’
7       ->  GROUP BY
8       ->   artikel_id
9       ->  HAVING
10      ->   SUM(bp.menge) > 10
11      -> ;
12  +---------------+--------------------------+
13  | Artikelname   | Anzahl bestellter Artikel |
14  +---------------+--------------------------+
15  | Silberzwiebel |                 40.000000 |
16  +---------------+--------------------------+
```

Die WHERE-Klausel lässt nur Artikel zu, die das Präfix Silber haben. Deshalb steht das WHERE auch vor dem GROUP BY. Erst nach diesem Filter werden die restlichen Daten grup-

piert und ausgewertet. Die Berechnungsergebnisse selbst werden dann noch mit dem `HAVING` weiter gefiltert.

12.4 Noch Fragen?

12.4.1 Kann ich nach Ausdrücken gruppieren?

In der syntaktischen Beschreibung von `GROUP BY` in Abschnitt 12.3 steht *spaltenliste*, aber es können auch Ausdrücke verwendet werden.

Beispiel: Nach [Wik23b] steht die erste Ziffer der Bankleitzahl für das Clearinggebiet. Wir wollen wissen, wie viele Bankleitzahlen pro Clearinggebiet in der Tabelle `bank` sind.

```
1  mysql> SELECT
2      ->  SUBSTRING(blz, 1, 1) ‘Clearinggebiet‘, COUNT(*) ‘Anzahl‘
3      ->  FROM
4      ->   bank
5      ->  GROUP BY
6      ->   ‘Clearinggebiet‘
7      ->  ORDER BY ‘Anzahl‘ DESC
8      -> ;
9  +----------------+--------+
10 | Clearinggebiet | Anzahl |
11 +----------------+--------+
12 | 7              |   1249 |
13 | 5              |   1193 |
14 | 6              |   1100 |
15 | 2              |    989 |
16 | 4              |    556 |
17 | 3              |    527 |
18 | 8              |    287 |
19 | 1              |    205 |
20 +----------------+--------+
```

Um das Clearinggebiet zu ermitteln, verwende ich in Zeile 2 die Funktion `SUBSTRING()`. Diese schneidet mir aus einer Zeichenkette einen Abschnitt heraus. Der erste Buchstabe der Zeichenkette hat den Index 1. Als letzter Parameter der Funktion wird die Länge des Abschnitts festgelegt, hier ebenfalls 1.

Dadurch, dass ich den Alias `Clearinggebiet` vergeben habe, kann ich überall dort, wo der Ausdruck `SUBSTRING(blz, 1, 1)` gebraucht wird, den Alias verwenden. So auch in Zeile 6 beim `GROUP BY`.

Aufgabe 12.2: Wie viele Banken gibt es pro Bankengruppe (4. Ziffer der Bankleitzahl)?

12.4.2 Kann ich nach mehr als einer Spalte gruppieren?

In der syntaktischen Beschreibung von `GROUP BY` in Kapitel 12.3 steht hinter dem `GROUP BY` das Wort *spaltenliste*. Wir können Gruppen also auch über mehrere Elemente definieren. Bisher waren es einfache Spalten wie die `artikel_id`.

Beipiel: Wir wollen die Anzahl der Bestellungen pro Jahr und Monat wissen[1]:

```
1  mysql> SELECT
2      ->  YEAR(datum) 'Jahr', MONTH(datum) 'Monat', COUNT(*) 'Anzahl'
3      ->  FROM bestellung
4      ->  GROUP BY
5      ->   YEAR(datum), MONTH(datum)
6      ->  ORDER BY
7      ->   YEAR(datum) DESC, MONTH(datum) DESC
8      -> ;
9  +------+-------+--------+
10 | Jahr | Monat | Anzahl |
11 +------+-------+--------+
12 | 2012 |     4 |      1 |
13 | 2012 |     3 |      2 |
14 | 2011 |     1 |      3 |
15 +------+-------+--------+
```

Der entscheidende Teil ist die Zeile 5. Die Gruppierung wird zuerst nach dem Jahr vorgenommen, anschließend nach dem Monat. Die Funktion `YEAR()` liefert mir aus einer Datumsangabe die vierstellige Darstellung des Jahrs als Zahl; `MONTH()` liefert die ein- oder zweistellige Darstellung des Monats als Zahl. Beide Funktionen verwende ich in Zeile 7, um die Ausgabe nach dem Datum sinnvoll zu ordnen. Eine Übersicht mit vielen guten Beispiel finden Sie unter [MyS21a].

Aufgabe 12.3: Wie viele Banken gibt es pro Clearinggebiet und Bankengruppe (4. Stelle der Bankleitzahl)?

12.4.3 Wie kann ich `GROUP BY` beschleunigen?

Bei der Ausführung eines `GROUP BY` muss für jeden Datensatz entschieden werden, zu welcher Gruppe er gehört. Dazu müssen die Spalten und Ausdrücke ausgewertet werden. Wie bei der Sortierung (siehe Abschnitt 10.3) können Indizes diesen Vorgang erheblich beschleunigen.

Umgekehrt sind `GROUP BY`-Operationen, die keine Indexunterstützung haben, recht teuer. Besonders, wenn die Gruppierung über Ausdrücke erfolgt, muss der Ausdruck ausgewertet werden, was oft mit erheblicher Rechenleistung verbunden ist, da die Tabelle sequenziell durchlaufen werden muss. Schauen wir uns beispielsweise diesen Befehl an:

```
1  mysql> EXPLAIN
2      ->  SELECT
3      ->   SUBSTRING(blz, 1, 1) 'Clearinggebiet', COUNT(*) 'Anzahl'
4      ->   FROM
```

[1] Ich habe dazu in `listing09.sql` die Bestellungen um Daten erweitert.

```
5       ->    bank
6       ->   GROUP BY
7       ->    'Clearinggebiet'
8       ->   ORDER BY
9       ->  'Anzahl' DESC
10      -> \G
11 *************************** 1. row ***************************
12            id: 1
13   select_type: SIMPLE
14         table: bank
15    partitions: NULL
16          type: index
17 possible_keys: idx_bank_blzbankname
18           key: idx_bank_blzbankname
19       key_len: 1070
20           ref: NULL
21          rows: 6066
22      filtered: 100.00
23         Extra: Using index; Using temporary; Using filesort
```

Unter der Überschrift `Extra` in Zeile 23 wird angegeben, wie der Befehl ausgeführt wird. Da ich auf die Bankleitzahl einen Index gesetzt hatte, wird dieser genutzt. Nun verwende ich ein Gruppierungsmerkmal, welches zugegeben sinnlos ist, aber keinen Index nutzen kann:

```
1  mysql> EXPLAIN
2      ->  SELECT
3      ->   CONCAT(lkz, SUBSTRING(blz, 6, 3)) 'Sinnlos', COUNT(*) 'Anzahl'
4      ->   FROM
5      ->    bank
6      ->   GROUP BY
7      ->    'Sinnlos'
8      ->   ORDER BY
9      ->    'Sinnlos'
10     -> \G
11 *************************** 1. row ***************************
12            id: 1
13   select_type: SIMPLE
14         table: bank
15    partitions: NULL
16          type: ALL
17 possible_keys: NULL
18           key: NULL
19       key_len: NULL
20           ref: NULL
21          rows: 6066
22      filtered: 100.00
23         Extra: Using temporary; Using filesort
```

Hier teilt mir MySQL mit, dass es eine temporäre Tabelle aufbauen muss und deren Inhalt mit Filesort sortiert. Keine Rede mehr davon, dass irgendwelche Indizes verwendet werden könnten.

Werden Gruppierungen nicht über Indizes unterstützt und Sie erwarten einen häufigen Zugriff auf nicht kleine Mengen, so sollten – falls möglich – entsprechende Indizes eingerichtet werden. Die Argumente für oder gegen das Anlegen von Indizes sind in Tabelle 6.1 zusammengefasst.

Falls die Gruppierungen nicht permanent gebraucht werden und zum Zeitpunkt der Auswertung nicht oder nur mit wenigen Änderungen der Daten zu rechnen ist, bietet sich das Anlegen einer (temporären) Tabelle an.

12.4.4 Parallele Bearbeitung – unterschiedliche Ergebnisse?

Bei nicht transaktionsfähigen Engines wie MyISAM wird die Anzahl der Zeilen in den Infos über die Tabelle vom System mitgepflegt. Es müssen also nicht alle Zeilen gezählt werden, um die Anzahl zu ermitteln. Bei transaktionsfähigen Engines wie InnoDB können in offenen Transaktionen unterschiedlich viele Zeilen der Tabelle parallel vorhanden sein (Näheres siehe Kapitel 19).

Die Anzahl der Zeilen muss daher aktiv ermittelt werden und kann in jeder Sitzung unterschiedlich sein, weil jede Sitzung Zeilen hinzugefügt oder gelöscht haben könnte, die in anderen Sitzungen noch nicht sichtbar sind.

12.5 Haben Sie keine Aufgaben für mich?

Aufgabe 12.4: Klar, hab' ich:

a) Ermitteln Sie pro Bankleitzahl die Anzahl der Banknamen. Sortieren Sie das Ergebnis nach Bankleitzahl.
b) Ermitteln Sie pro Adresse die Anzahl der Verwendungen als Rechnungsanschrift. Die nicht verwendeten Adressen sollen auch angezeigt werden. Diese haben die Anzahl 0. Sortieren Sie nach der Anzahl absteigend.
c) Geben Sie pro Warengruppe die Anzahl der Artikel aus. Es sollen auch die Warengruppen angezeigt werden, denen keine Artikel angehören.
d) Geben Sie pro Lieferant die Anzahl der gelieferten Artikel aus. Lieferanten, die keine Artikel liefern, sollen ebenfalls angezeigt werden.
e) Geben Sie den Lagerbestand pro Warengruppe aus. Sortieren Sie die Ausgabe nach Lagerbestand absteigend.
f) Geben Sie die Kundennamen mit mehr als einer Bestellung aus.
g) Geben Sie für jeden Kundennamen die durchschnittliche Anzahl der Bestellungen aus.
h) Geben Sie pro Kunde die durchschnittliche Menge an bestellten Artikeln aus. Die Ausgabe soll nach dem Durchschnitt aufsteigend sortiert werden.
i) Geben Sie den Wert der teuersten Einzelposition aus.
j) Geben Sie den durchschnittlichen Wert einer Einzelposition aus.
k) Ermitteln Sie, welche Artikel bestellt sind, deren Mindestmenge im Lager unterschritten ist.