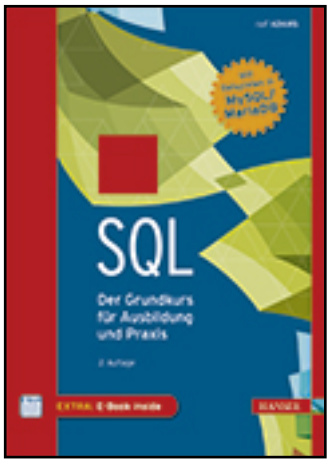
SQL

Mit Beispielen in MySQL/MariaDB

Bearbeitet von
Ralf Adams

2., aktualisierte Auflage 2016. Buch inkl. Online-Nutzung. 454 S. Softcover
ISBN 978 3 446 45074 5
Format (B x L): 17,9 x 24,4 cm
Gewicht: 945 g

HANSER



Leseprobe

Ralf Adams

SQL

Der Grundkurs für Ausbildung und Praxis. Mit Beispielen in MySQL/MariaDB

ISBN (Buch): 978-3-446-45074-5

ISBN (E-Book): 978-3-446-45079-0

Weitere Informationen oder Bestellungen unter

http://www.hanser-fachbuch.de/978-3-446-45074-5

sowie im Buchhandel.

Dieses Buch möchte ich allen Lehrerinnen und Lehrern der ehemaligen Aufbaurealschule in Eslohe, Sauerland, widmen.

Besonders denke ich dabei an meinen Klassenlehrer und jetzigen Schulleiter, Herrn Schmidt. Durch Ihr stetes Bemühen um jeden Einzelnen sind Sie mir menschlich und heute als Lehrer fachlich ein Vorbild.

Inhalt

Vorwort

Vorwort zur 1. Auflage

Und noch'n SQL-Buch. Es gibt so viele SQL-Bücher, dass man berechtigt die Frage stellen kann, warum man noch eines braucht. Ich kann die Frage nur indirekt beantworten. Als Lehrer für Anwendungsentwicklung an einem Berufskolleg habe ich über Jahre erlebt, dass die Auszubildenden sich sehr mit den üblichen Büchern abmühen.

Die fachliche Qualität dieser Bücher ist unbestritten. Aber die Sprache ist meist von *IT-Profi* zu *IT-Profi*, und genau damit sind Auszubildende und Berufsanfänger oft überfordert – zumindest wird der Einstieg erschwert.

Ich habe daher begonnen, leicht verständliche Skripte zu schreiben, aus denen sich dieses Buch speist. Dabei werden Befehle didaktisch reduziert und Beispiele möglichst lebensnah ausgesucht. Fachbegriffe werden nur verwendet, wenn sie IT-sprachlicher Umgang sind; akademische Begriffe werden vermieden, wobei ich ihre Berechtigung nicht in Abrede stellen möchte.

Primärziel ist ein möglichst umfangreicher Ersteinstieg (Grundkurs), der dann durch berufliche Praxis ausgebaut werden kann. Trotzdem vertiefe ich an vielen Stellen im Buch den Einblick in SQL oder den MySQL-Server (Vertiefendes) – zum einen, um zu zeigen, dass ich auch ein bisschen was drauf habe, zum anderen, um Neugierde und Jagdtrieb beim Leser[1] zu wecken.

Ein weiterer Grund für dieses Buch ist, dass es mir großen Spaß gemacht hat, es zu schreiben. Ich hoffe, dass es Ihnen genau soviel Spaß macht, es zu lesen und damit zu arbeiten. Falls Sie mich fachlich korrigieren oder ergänzen möchten, senden Sie mir doch bitte eine E-Mail an *sqlbuch@ralfadams.de*.

Der Titel des Buches ist SQL und nicht MySQL. Ich habe deshalb an vielen Stellen den Unterschied zwischen MySQL und dem SQL92-Standard aufgezeigt. Trotzdem wird es schwer sein, die Beispiele *einfach so* auf andere DBMS wie Oracle oder MS-SQL zu übertragen. Die im Internet unter *http://downloads.hanser.de* verfügbaren Quelltexte sind jedenfalls nur unter MySQL 5.5 getestet worden. Trotzdem ist dies kein MySQL-Buch. Auf jeden Fall werden Sie ein Verständnis für den Aufbau und die Funktionsweise der Befehle erwerben, sodass Sie leicht die verschiedenen SQL-Dialekte adaptieren können.

[1] Der besseren Lesbarkeit wegen verzichte ich auf weiblich/männlich-Konstruktionen. Bitte verstehen Sie dies nicht als stillschweigende Hinnahme des geringen Frauenanteils in den IT-Berufen.

Wie ist das Buch zu lesen?

- Teil I *Was man so wissen sollte*
  Als Grundkurs werden hier die zentralen Begriffe wie Datenbank und Datenbankmanagementsystem eingeführt. Anschließend wird unser Beispiel-DBMS MySQL/MariaDB vorgestellt. Falls Sie auf anderen Systemen arbeiten, können Sie diesen Abschnitt überspringen. Nun folgt eine Einführung in die Theorie. Zwar wird in den nachfolgenden Kapiteln immer wieder auf diese Theorie zurückgegriffen, aber Sie können dieses Kapitel auch erst einmal überspringen, wenn Sie nur einen Befehl erlernen möchten. Allen, die sich auf eine Prüfung vorbereiten müssen, empfehle ich aber dringend, dieses Kapitel zu lesen und zu lernen. Zum Schluss wird das im Buch verwendete Beispiel eines Online-Shops inhaltlich entwickelt und als ER-Modell dokumentiert.
- Teil II *Eine Datenbank aufbauen*
  Es werden einige Installationen von MySQL vorgestellt. Leider kann ich nicht alle attraktiven Zielplattformen beschreiben. Es geht hier um die Bereitstellung einer Arbeitsumgebung für einen Entwickler oder besser für den Leser dieses Buches. Nach der Installation werden die Datenbank und die Tabellen des Online-Shops angelegt. Zu den Tabellen werden Indizes eingeführt. Dabei werden immer wieder vertiefende Inhalte angeboten, die bei einem Grundkurs erstmal übersprungen werden können. Zum Schluss werden noch Daten in die Tabellen eingefügt oder aus anderen Datenquellen übernommen.
- Teil III *Eine Datenbank verändern*
  Die Datenbankeigenschaften, die Tabellenstrukturen und die Tabelleninhalte werden verändert oder gelöscht.
- Teil IV *Eine Datenbank auswerten*
  Der umfangreichste Teil des Buchs. Hier werden die Auswertungen mit `SELECT` in allen möglichen Varianten vorgestellt. Besonders die Verbindung mit `JOIN` und Unterabfragen werden an vielen Beispielen verständlich gemacht. Fast alle Inhalte gehören zum Grundkurs und sollte man daher z.B. für Prüfungen wissen.
- Teil V *Anweisungen kapseln*
  Transaktionen, Prozeduren, Funktionen, Trigger und Ereignisse gehören nicht zwangsläufig zum Grundkurs, werden hier aber wegen ihrer fachlichen Attraktivität und ihrer wachsenden Bedeutung vorgestellt.
- Teil VI *Anhänge*
  Eine Datenbank zu administrieren, ist – bis auf die Benutzerrechte – so DBMS-spezifisch, dass ich dies in den Anhang geschoben habe. Leider musste ich aus Platzmangel das Thema Cluster aussparen. Es folgt eine (My)SQL-Referenz. Anschließend werden Zeichensätze und Sortierungen, die für die Sprache *Deutsch* relevant sind, aufgelistet. Da ich den MySQL-Client verwende, stelle ich auch noch den Client etwas genauer vor. Zum Schluss folgen noch ausgewählte Quelltexte zum Selbststudium.

Die Lösungen zu den Aufgaben können unter *http://downloads.hanser.de* zwecks Vergleich heruntergeladen werden.

Danksagung

Als Erstes möchte ich mich bei Frau Margarete Metzger vom Hanser Verlag bedanken. Sie hat dieses Buch erst möglich gemacht. Zum Ende hin wurde das Buchprojekt von Frau Brigitte Bauer-Schiewek betreut, der ich ebenfalls meinen Dank aussprechen möchte. Herr Jürgen Dubau hat sprachliche Ausrutscher und flapsige Formulierungen glatt gebügelt. Für die LaTeX-Probleme war Herr Stephan Korell zuständig. Seine schnellen und guten Lösungen haben die Bearbeitung sehr erleichtert. Das Layout wurde von Frau Irene Weilhart betreut; wenn Ihnen das Layout gefällt, ist das ihrem Sinn für Ästhetik zu verdanken.

Ich möchte meinen Kollegen Dr. Andreas Alef und Marco Bakera, mit denen ich das Vergnügen habe, an der Technischen Beruflichen Schule 1 in Bochum (*http://www.tbs1.de*) zu unterrichten, für ihre kritischen und aufmunternden Kommentare danken.

Besonders will ich meine Schülerinnen und Schüler erwähnen. Die hier vorgestellten Beispiele und Konzepte sind in großen Teilen durch ihre schonungslose Kritik an bestehenden Lehrmaterialien entstanden. Das penetrante *Kapiere ich nicht!* hat mich immer weiter angespornt, es noch verständlicher zu versuchen. Falls dieses Buch SQL gut vermittelt, ist das auch deren Verdienst.

Zum Schluss noch eine Abbitte an meine Familie. Sie hat in den Monaten der Entstehung dieses Buchs oft zurückstecken müssen. Dafür, dass ich von allen – insbesondere von meiner Frau Barbara – so intensiv unterstützt wurde, bin ich sehr dankbar. Das ist nicht selbstverständlich, und ich freue mich darauf, dies mit vielen Runden Kniffel und Rummikub auszugleichen.

Ralf Adams, Juli 2012

Bemerkungen zur überarbeiteten 2. Auflage

- Ich beziehe mich nicht mehr auf den SQL92-Standard, sondern auf den SQL3-Standard.
- Skripte und Beispiele sind auf folgenden Servern getestet worden: *MySQL Community Server 5.7.12* unter Ubuntu, *MariaDB 10.1.13* unter Windows 10 und *PostgreSQL 9.5* unter Windows 10.
- MariaDB hat MySQL in vielen Anwendungen abgelöst. Die Gründe dafür und die entsprechende Diskussion darüber möchte ich hier nicht wiedergeben. Da beide den gleichen SQL-Dialekt sprechen, konnte ich an den meisten Stellen im Buch die Gültigkeit auf MariaDB erweitern. Unterschiede konnte ich bei den verwendeten Skripten nicht feststellen.
- Ich habe an mehreren Stellen auch PostgreSQL-Beispiele eingebaut. Zum einen, weil diese näher am SQL3-Standard sind, und zum anderen, weil es manche Befehlsvarianten in MySQL und MariaDB nicht gibt (z.B. der rekursive `SELECT` in Abschnitt 11.5 auf Seite 201).
- Aus diesem Grunde habe ich auch für die meisten Skripte PostgreSQL-Varianten erstellt. Diese können Sie im entsprechenden Verzeichnis des Hanser Verlags herunterladen.
- Die SQL-Skripte der ersten Auflage haben an manchen Stellen Fehlermeldungen ausgegeben. Diese Fehlermeldungen waren gewollt – z.B. das Löschen einer Zeile mit einem

Fremdschlüsselconstraint – und im Quelltext durch Kommentare dokumentiert. Leider ist dies von vielen so interpretiert worden, als seien die Skripte fehlerhaft. Ich habe deshalb die entsprechenden Stellen in den Skripten auskommentiert.

- Bitte beachten Sie, dass die Pfadangaben in den Skripten mit `LOAD DATA INFILE` angepasst werden müssen, je nachdem, wo Sie die Daten entpacken.
- Ich habe angefangen, für die Aufgaben Musterlösungen bei YouTube (*http://www.youtube.com/channel/UCu4ZybNXw1y4Rs4Mgx-4HKw*) einzustellen. In diesen Videos kann ich einfach besser erklären, worauf es bei den Lösungen ankommt.
- Beim Test der Skripte unter MySQL 5.6.19 ist ein Fehler des Servers aufgetreten (siehe [Ada14]).

Bei Frau Sylvia Hasselbach vom Hanser Verlag möchte ich mich dafür bedanken, dass sie die 2. Auflage angestoßen und ermöglicht hat.

Abschließend möchte ich mich für die vielen guten Kontakte und positiven wie kritischen Rückmeldungen über *sqlbuch@ralfadams.de* bedanken. Es hat mich sehr gefreut, dass durch mein Buch vielen das Erlernen von SQL Spaß gemacht hat.

Ralf Adams, Juni 2016

11 Tabellen verbinden

Daten für Auswertungen verbinden.

- Grundkurs
  - Kartesisches Produkt, CROSS JOIN
  - INNER JOIN mit zwei und mehr Tabellen
  - LEFT und RIGHT OUTER JOIN
- Vertiefendes
  - Abkürzung mit USING
  - NATURAL JOIN
  - JOIN als Datenquelle
  - Verknüpfung über Nichtschlüsselspalten
  - EQUI JOIN
  - OUTER JOIN und referentielle Integrität
  - SELF JOIN
  - Common Table Expression (WITH)
  - Einfluss von Indizes auf JOIN

Spätestens jetzt sind wir im nichttrivialen Bereich von `SELECT` angekommen. Daten aus mehreren Tabellen zusammenzuführen, ist Alltagsgeschäft und wird von vielen DBMS-Tools unterstützt. Trotzdem muss man das Handwerk dahinter verstehen, denn jeder DB-Designer muss wissen, wie die Daten später wieder zusammengebastelt werden müssen. Ohne den Aufwand zu kennen, lässt sich kein seriöses ER-Modell erstellen[1].

Die Quelltexte dieses Kapitels stehen in der Datei `listing08.sql` (siehe Listing 28.8 auf Seite 445 und Listing 28.27 auf Seite 502).

[1] So, wie wir es in den ersten Kapiteln getan haben ;-).

11.1 Heiße Liebe: Primär-/Fremdschlüsselpaare

SQL3
MySQL/MariaDB

```
SELECT [DISTINCT]
  {*|spaltenliste|ausdruck}
  FROM tabellenname[, tabellenname]*
  [WHERE bedingung]
  [ORDER BY spaltenname [ASC|DESC] [,spaltenname [ASC|DESC]]*]
;
```

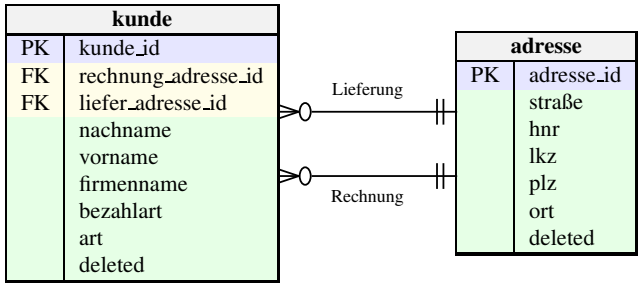


Bild 11.1 ER-Modell: `kunde` und `adresse`

Die bisher durchgeführten Auswertungen fanden immer auf *einer* Tabelle statt. Wenn wir uns die ER-Modelle für den Online-Shop anschauen, erkennen wir, dass oft inhaltlich zusammenhängende Informationen auf mehrere Tabellen verteilt sind.[2]

So stehen beispielsweise die Adressdaten eines Kunden in einer anderen Tabelle als sein Name. Für ein Rechnungsschreiben werden beide Informationen wieder miteinander zu verknüpfen sein. Eine Variante des `SELECT`s erlaubt die Verwendung mehrerer Tabellen in einem `SELECT`. Wollen wir mal schauen, was dabei herauskommt:

```
mysql> SELECT
    ->  kunde_id, nachname, vorname, rechnung_adresse_id, adresse_id
    ->  FROM
    ->   kunde, adresse
    -> ;
+----------+-------------+---------+--------------------+------------+
| kunde_id | nachname    | vorname | rechnung_adresse_id | adresse_id |
+----------+-------------+---------+--------------------+------------+
|        1 | Gamdschie   | Samweis |                  1 |          1 |
|        2 | Beutlin     | Frodo   |                  2 |          1 |
|        3 | Beutlin     | Bilbo   |                  2 |          1 |
|        4 | Telcontar   | Elessar |                  3 |          1 |
|        5 | Earendilionn | Elrond  |                  4 |          1 |
|        6 | Eichenschild | Thorin  |               NULL |          1 |
|        1 | Gamdschie   | Samweis |                  1 |          2 |
```

[2] Dies ist gerade der entscheidende Unterschied zu objektorientierten Datenbanken oder NoSQL.

```
|         2 | Beutlin      | Frodo   |                   2 |          2 |
|         3 | Beutlin      | Bilbo   |                   2 |          2 |
|         4 | Telcontar    | Elessar |                   3 |          2 |
|         5 | Earendilionn | Elrond  |                   4 |          2 |
|         6 | Eichenschild | Thorin  |                NULL |          2 |
|         1 | Gamdschie    | Samweis |                   1 |          3 |
|         2 | Beutlin      | Frodo   |                   2 |          3 |
|         3 | Beutlin      | Bilbo   |                   2 |          3 |
|         4 | Telcontar    | Elessar |                   3 |          3 |
|         5 | Earendilionn | Elrond  |                   4 |          3 |
|         6 | Eichenschild | Thorin  |                NULL |          3 |
|         1 | Gamdschie    | Samweis |                   1 |          4 |
|         2 | Beutlin      | Frodo   |                   2 |          4 |
|         3 | Beutlin      | Bilbo   |                   2 |          4 |
|         4 | Telcontar    | Elessar |                   3 |          4 |
|         5 | Earendilionn | Elrond  |                   4 |          4 |
|         6 | Eichenschild | Thorin  |                NULL |          4 |
|         1 | Gamdschie    | Samweis |                   1 |          5 |
|         2 | Beutlin      | Frodo   |                   2 |          5 |
|         3 | Beutlin      | Bilbo   |                   2 |          5 |
|         4 | Telcontar    | Elessar |                   3 |          5 |
|         5 | Earendilionn | Elrond  |                   4 |          5 |
|         6 | Eichenschild | Thorin  |                NULL |          5 |
|         1 | Gamdschie    | Samweis |                   1 |         10 |
|         2 | Beutlin      | Frodo   |                   2 |         10 |
|         3 | Beutlin      | Bilbo   |                   2 |         10 |
|         4 | Telcontar    | Elessar |                   3 |         10 |
|         5 | Earendilionn | Elrond  |                   4 |         10 |
|         6 | Eichenschild | Thorin  |                NULL |         10 |
|         1 | Gamdschie    | Samweis |                   1 |         11 |
|         2 | Beutlin      | Frodo   |                   2 |         11 |
|         3 | Beutlin      | Bilbo   |                   2 |         11 |
|         4 | Telcontar    | Elessar |                   3 |         11 |
|         5 | Earendilionn | Elrond  |                   4 |         11 |
|         6 | Eichenschild | Thorin  |                NULL |         11 |
+-----------+--------------+---------+---------------------+------------+
42 rows in set (0.00 sec)
```

In Zeile 4 werden zwei Tabellen hinter dem `FROM` angegeben. Das ist neu; bisher stand hier immer nur *die eine* Tabelle.

Im Ergebnis werden mir 42 Zeilen einer *neuen* Tabelle angezeigt. Aber wie sind diese Zeilen gebildet worden? Es fällt auf, dass die `kunde_id` sich nach dem Schema $1,2,3,4,5,6$ wiederholt. Ebenso interessant ist, dass die `adresse_id` mit jeweils sechs gleichen Werten auftaucht.

Betrachtet man nur die Werte von `kunde_id` und `adresse_id`, so hat man Datenpaare, die wie folgt aufgebaut sind: Jede Adresse ist mit allen Kunden kombiniert worden. Adresse 1 mit Kunde 1 bis 6, Adresse 2 mit Kunde 1 bis 6 usw. Da es sechs Kunden und sieben Adressen gibt, erhalten wir 42 Kombinationen, das *Kartesische Produkt*.

Definition 38: Kartesisches Produkt
Seien A und B zwei endliche Mengen, $a \in A$ und $b \in B$, dann ist die Menge aller unterschiedlichen Paare (a,b) das *Kartesische Produkt* K der Mengen A und B.

Diese Definition lässt $A = B$ zu. Wenn n die Anzahl der Elemente in A ist und m die Anzahl der Elemente in B, dann hat K $n \times m$ viele Elemente.

Definition 39: CROSS JOIN
Das Kartesische Produkt zweier Mengen wird auch *CROSS JOIN* oder *Kreuzprodukt* genannt.

Toll, ein Kartesisches Produkt[3], ja und?

Aufgabe 11.1: Betrachten Sie genau die Zahlenpaare `rechnung_adresse_id` und `adresse_id`. Formulieren Sie eine `WHERE`-Klausel, die genau die Zeilen übrig lässt, die zu einem Kunden die richtige Adressnummer angeben.

In den Zeilen 9, 16, 17, 24 und 31 stehen jeweils die gleichen Werte. Ausformuliert bedeutet dies: Im Fremdschlüssel `rechnung_adresse_id` steht der gleiche Wert wie in `adresse_id`. Das sind genau die Elemente des Kartesischen Produkts, die eine gültige Verknüpfung zwischen den beiden Tabellen darstellen. Die `WHERE`-Klausel sollte somit nur diese Zeilen übrig lassen:

```
1  mysql> SELECT
2      ->  kunde_id, nachname, vorname, rechnung_adresse_id, adresse_id
3      ->  FROM
4      ->   kunde, adresse
5      ->  WHERE
6      ->   rechnung_adresse_id = adresse_id
7      -> ;
8  +----------+--------------+---------+---------------------+------------+
9  | kunde_id | nachname     | vorname | rechnung_adresse_id | adresse_id |
10 +----------+--------------+---------+---------------------+------------+
11 |        1 | Gamdschie    | Samweis |                   1 |          1 |
12 |        2 | Beutlin      | Frodo   |                   2 |          2 |
13 |        3 | Beutlin      | Bilbo   |                   2 |          2 |
14 |        4 | Telcontar    | Elessar |                   3 |          3 |
15 |        5 | Earendilionn | Elrond  |                   4 |          4 |
16 +----------+--------------+---------+---------------------+------------+
17 5 rows in set (0.00 sec)
```

Aufgabe 11.2: Was ist mit den Adressen mit den Primärschlüsselwerten 5, 10 und 11 passiert? Wie kann man das inhaltlich interpretieren? Und was ist mit dem Kunden *Thorin Eichenschild*?

Die Reduzierung des Kartesischen Produkts auf die Zeilen mit passenden Schlüsselpaaren ist schon ein `INNER JOIN`. Was uns nun noch fehlt, ist ein *richtiger* Befehl dazu.

[3] Wird von den Azubis gerne *Orgienjoin* genannt.

11.2 INNER JOIN zwischen zwei Tabellen

Definition 40: INNER JOIN
Der *INNER JOIN* zweier Tabellen ist die Teilmenge des kartesischen Produkts, für welche gilt, dass die Fremdschlüsselwerte zu den Primärschlüsselwerten passen.

Die Formulierung über das Kartesische Produkt mit der passenden `WHERE`-Klausel ist für die Programmierung ein wenig sperrig. Stellen Sie sich vor, dass die Ergebnismenge weitere Bedingungen erfüllen muss oder keine echten Tabellen verwendet werden, sondern Unterabfragen[4]. Deshalb gibt es eine eigene Syntax für den INNER JOIN:

SQL3
MySQL/MariaDB

```
SELECT [DISTINCT]
  {*|spaltenliste|ausdruck}
  FROM
    tabellenname_fk INNER JOIN tabellenname_pk
       ON tabellenname_fk.fk = tabellenname_pk.pk
  [WHERE bedingung]
  [ORDER BY spaltenname [ASC|DESC] [,spaltenname [ASC|DESC]]*]
;
```

Der obige Befehl sähe umgebaut so aus:

```
1 SELECT
2  kunde_id, nachname, vorname, rechnung_adresse_id, adresse_id
3  FROM
4   kunde INNER JOIN adresse ON rechnung_adresse_id = adresse_id;
```

Aufgabe 11.3: Bauen Sie den letzten Befehl so um, dass folgende Ausgabe erzeugt wird:

```
+--------------+---------+-----------------+-----+--------------+
| nachname     | vorname | strasse         | hnr | ort          |
+--------------+---------+-----------------+-----+--------------+
| Gamdschie    | Samweis | Beutelhaldenweg | 5   | Hobbingen    |
| Beutlin      | Frodo   | Beutelhaldenweg | 1   | Hobbingen    |
| Beutlin      | Bilbo   | Beutelhaldenweg | 1   | Hobbingen    |
| Telcontar    | Elessar | Auf der Feste   | 1   | Minas Tirith |
| Earendilionn | Elrond  | Letztes Haus    | 4   | Bruchtal     |
+--------------+---------+-----------------+-----+--------------+
```

[4] Siehe Kapitel 13 auf Seite 219

11.2.1 Bauanleitung für einen INNER JOIN

Die Programmierung eines `INNER JOIN` fällt meinen Schülerinnen und Schülern oft schwer. Deshalb will ich hier ein wenig ausführlicher beschreiben, wie man einen `INNER JOIN` zusammenbauen kann. Die passende Aufgabe: Wir wollen zu einer Bankverbindung die Bankleitzahl und den Banknamen wissen.

1. **Ermitteln der beteiligten Tabellen:** In unserem Fall sind es die Tabellen `bankverbindung` und `bank`. Schreiben Sie sich diese Tabellen auf: eine auf die linke Seite vom Blatt und eine auf die rechte Seite.

2. **Ermitteln der Primär- und Fremdschlüssel:** Schreiben Sie unter den Tabellennamen jeweils den Primärschlüssel und alle vorkommenden Fremdschlüssel.

   bankverbindung
   kunde_id
   bankverbindung_id
   bank_id

   bank
   bank_id

3. **Fremdschlüssel festlegen:** In einer der Tabellen muss ein Fremdschlüssel vorkommen, der auf die andere Tabelle zeigt. Wenn Sie beim Design alles richtig gemacht haben und die Namenskonvention beachten, finden Sie diesen sehr schnell. Es kommen zwei Fremdschlüssel infrage: `kunde_id` und `bank_id`. Da eine der Tabellen `bank` heißt und wir der Namenskonvention gefolgt sind, muss es `bank_id` sein. Markieren Sie den Fremdschlüssel z.B. mit einem Textmarker.

   bankverbindung
   kunde_id
   bankverbindung_id
   bank_id

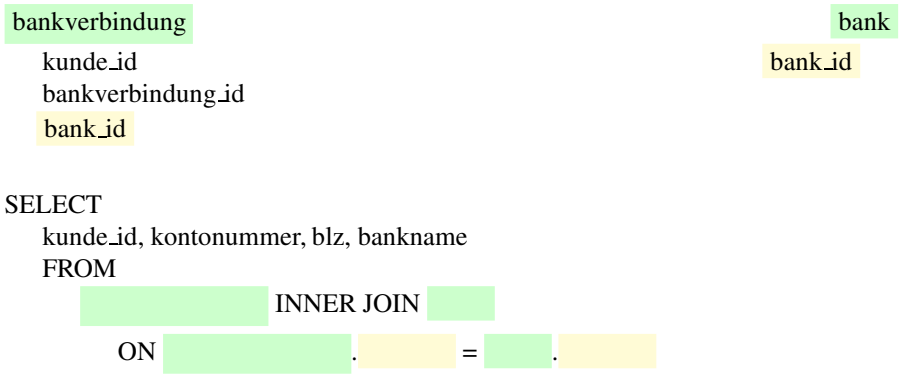
   bank
   bank_id

4. **Primärschlüssel festlegen:** Jetzt markieren Sie in der gleichen Farbe in der anderen Tabelle den dazugehörigen Primärschlüssel. Hier ist es die Spalte `bank_id`

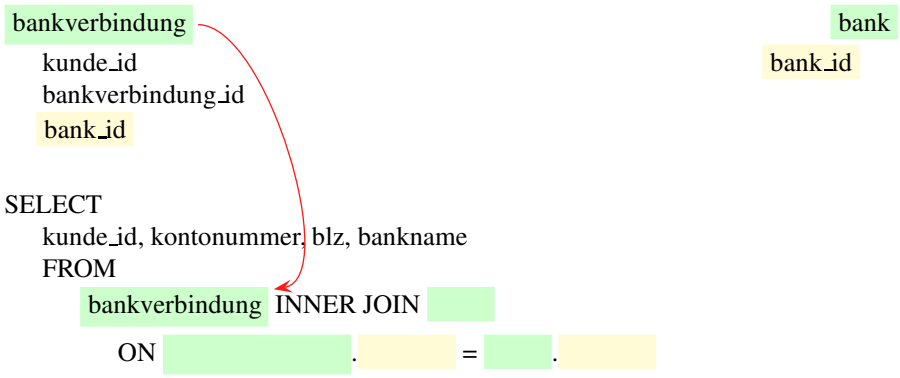
   bankverbindung
   kunde_id
   bankverbindung_id
   bank_id

   bank
   bank_id

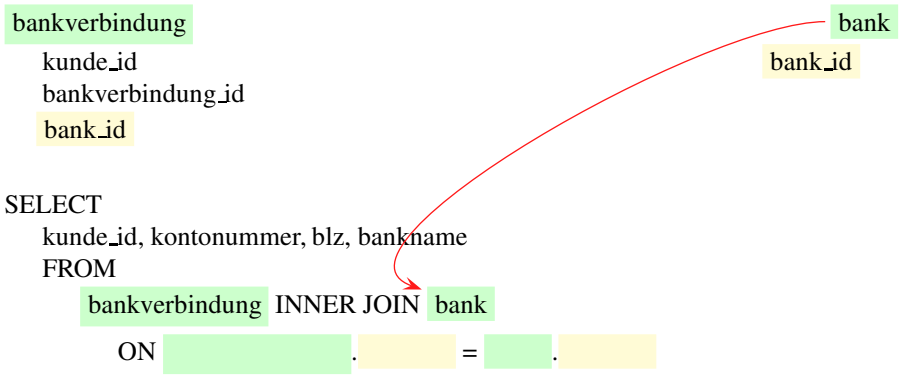
5. **Schablone benutzen:** Jetzt schreiben Sie auf das Blatt die Schablone für den `INNER JOIN`. Das Ganze sollte jetzt so aussehen:



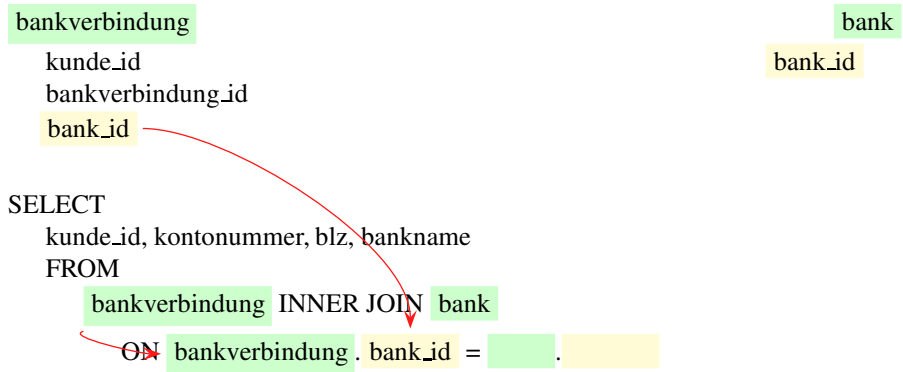
6. **Fremdschlüsseltabelle eintragen:** Fügen Sie links vom `INNER JOIN` den Tabellennamen für die Tabelle mit dem markierten Fremdschlüssel ein.



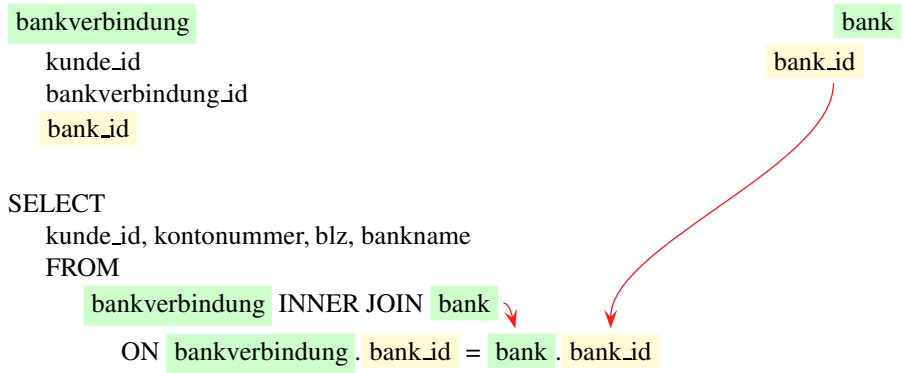
7. **Primärschlüsseltabelle eintragen:** Fügen Sie rechts vom `INNER JOIN` den Tabellennamen für die Tabelle mit dem markierten Primärschlüssel ein.



8. **Fremdschlüssel eintragen:** Fügen Sie zwischen dem `ON` und dem Gleichheitszeichen den Namen der Fremdschlüsseltabelle, einen Punkt und den Namen des Fremdschlüssels ein.



9. **Primärschlüssel eintragen:** Fügen Sie nach dem = den Namen der Primärschlüsseltabelle, einen Punkt und den Namen des Primärschlüssels ein.



10. **Fertig!**

```
mysql> SELECT
    ->  kunde_id, kontonummer, blz, bankname
    ->  FROM
    ->   bankverbindung INNER JOIN bank
    ->    ON bankverbindung.bank_id = bank.bank_id
    -> ;
+----------+-------------+----------+---------------+
| kunde_id | kontonummer | blz      | bankname      |
+----------+-------------+----------+---------------+
|        1 | 1111111111  | 10010010 | Postbank      |
|        1 | 1111111112  | 10010010 | Postbank      |
|        2 | 2222222221  | 10060198 | Pax-Bank      |
|        3 | 3333333331  | 10060198 | Pax-Bank      |
|        4 | 4444444441  | 12070000 | Deutsche Bank |
|        5 | 5555555551  | 12070000 | Deutsche Bank |
+----------+-------------+----------+---------------+
```

Ein zweites Beispiel: Zu einem Kunden werden die Kontonummern ausgegeben.

1. **Ermitteln der beteiligten Tabellen**: Es sind `kunde` und `bankverbindung`.
2. **Ermitteln der Primär- und Fremdschlüssel:** In der Tabelle `bankverbindung` gibt es den zusammengesetzten Primärschlüssel mit `kunde_id` und `bankverbindung_nr` und den Fremdschlüssel `bank_id`. In der Tabelle `kunde` gibt es den Primärschlüssel

kunde_id und die beiden Fremdschlüssel rechnung_adresse_id und liefer_adresse_id.

3. **Fremdschlüssel festlegen:** Da wir die Namenskonvention beachtet haben, brauchen wir nur nach einem Fremdschlüssel suchen, der so wie die andere Tabelle heißt. Dies ist in unserem Fall kunde_id.
4. **Primärschlüssel festlegen:** Jetzt wird die Spalte kunde_id in der Tabelle kunde markiert.
5. **Schablone benutzen:** Jetzt schreiben Sie auf das Blatt die Schablone für den INNER JOIN.
6. **Fremdschlüsseltabelle eintragen:** Fügen Sie links vom INNER JOIN den Tabellennamen für die Tabelle mit dem markierten Fremdschlüssel ein.
7. **Primärschlüsseltabelle eintragen:** Fügen Sie an die passende Stelle den Tabellennamen für die Tabelle mit dem markierten Primärschlüssel ein.
8. **Fremdschlüssel eintragen:** Fügen Sie an die passende Stelle den Spaltennamen des markierten Fremdschlüssels ein.
9. **Primärschlüssel eintragen:** Fügen Sie an die passende Stelle den Spaltennamen des markierten Primärschlüssels ein.
10. **Fertig!**

```
1  mysql> SELECT
2      ->  nachname, vorname, kontonummer
3      ->  FROM
4      ->   bankverbindung b INNER JOIN kunde k ON b.kunde_id = k.kunde_id;
5  +--------------+---------+-------------+
6  | nachname     | vorname | kontonummer |
7  +--------------+---------+-------------+
8  | Gamdschie    | Samweis | 1111111111  |
9  | Gamdschie    | Samweis | 1111111112  |
10 | Beutlin      | Frodo   | 2222222221  |
11 | Beutlin      | Bilbo   | 3333333331  |
12 | Telcontar    | Elessar | 4444444441  |
13 | Earendilionn | Elrond  | 5555555551  |
14 +--------------+---------+-------------+
```

Haben Sie gesehen, dass in Zeile 4 die Tabellennamen durch Aliase ersetzt wurden?

Aufgabe 11.4: Geben Sie zu allen Kunden den Namen und die Rechnungsadresse aus.

Aufgabe 11.5: Geben Sie zu allen Kunden den Namen und die Lieferadresse aus. Interpretieren Sie das Ergebnis.

Aufgabe 11.6: Geben Sie zu jedem Kunden den Namen und die Bestellungen nach Kundennamen und Bestelldatum sortiert aus. Die letzte Bestellung des Kunden soll zuerst erscheinen.

Aufgabe 11.7: Geben Sie zu jeder Bestellung die Kundennummer, das Bestelldatum und die Positionen aus. Die Sortierung soll nach Kundennummer, Bestellnummer und Position erfolgen.

Aufgabe 11.8: Geben Sie zu jeder Position den Artikelnamen aus. Es soll nach Artikelname sortiert werden.

11.2.2 Abkürzende Schreibweisen

SQL3
MySQL/MariaDB

```
SELECT [DISTINCT]
  {*|spaltenliste|ausdruck}
  FROM
    tabellenname_fk {INNER|NATURAL} JOIN tabellenname_pk
      [{ON tabellenname_fk.fk = tabellenname_pk.pk|USING(spaltenname)}]
  [WHERE bedingung]
  [ORDER BY spaltenname [ASC|DESC] [,spaltenname [ASC|DESC]]*]
;
```

Falls der Name des Fremdschlüssels genau der gleiche ist wie der des Primärschlüssels, kann man die `ON`-Klausel durch eine kürzere Variante mit `USING` ersetzen:

```
1 SELECT
2  kunde_id, kontonummer, blz, bankname
3  FROM
4   bankverbindung INNER JOIN bank USING (bank_id)
5 ;
```

Bei den Fremdschlüsseln, die anders als die Primärschlüssel heißen, wie beispielsweise `rechnung_adresse_id`, ist eine solche Abkürzung nicht möglich.

Aufgabe 11.9: Bauen Sie Ihre Lösungen zu den Aufgaben um, wenn ein `USING` möglich ist.

Sind die Fremdschlüssel-/Primärschlüsselspalten die einzigen Spalten, die in beiden Tabellen gleich sind und die Verknüpfung definieren, spricht man von einem `NATURAL JOIN`.

Definition 41: NATURAL JOIN
Werden zwei Tabellen über die Spalten verknüpft, die den gleichen Namen und Inhalt haben, spricht man von einem *NATURAL JOIN*.

```
1 SELECT
2  kunde_id, kontonummer, blz, bankname
3  FROM
4   bankverbindung NATURAL JOIN bank
5 ;
```

Hinweis: Bitte beachten Sie, dass bei unseren Tabellen in einem `NATURAL JOIN` auch die Spalte `deleted` in die Verknüpfung mit einfließt.

11.2.3 Als Datenquelle für temporäre Tabellen

Wir haben oben die Bankverbindung zu einem Kunden ermittelt. Annahme: Wir wollen jetzt viele Auswertungen der Kunden inklusive der Bankverbindung machen, dann müssen jedes Mal im entsprechenden `SELECT` die Informationen mit `INNER JOIN` verknüpft werden.

Obwohl `INNER JOIN`-Anweisungen durch passend gewählte Indizes sehr beschleunigt werden, entsteht trotzdem eine Rechnerlast auf dem Server, die jedes Mal das gleiche – oder fast das gleiche – Ergebnis liefert[5].

Da es plausibel ist anzunehmen, dass sich die Kundenstammdaten (siehe Definition 34) für einen gewissen Auswertungszeitraum nicht ändern, könnte man statt dessen das Ergebnis des `INNER JOIN` in eine (temporäre) Tabelle ablegen und mit dieser weiterarbeiten.

Als Beispiel wollen wir die Kundendaten mit Rechnungsanschrift und allen Bestellungen ausgeben und danach die Kundendaten mit Rechnungsanschrift mit allen Bankverbindungen.

In beiden Fällen werden die Kundendaten mit Rechnungsanschrift benötigt. Das können wir schon:

```
1  SELECT
2   k.kunde_id, k.nachname, k.vorname, a.strasse, a.hnr, a.plz, a.ort
3   FROM
4    kunde k INNER JOIN adresse a
5     ON k.rechnung_adresse_id = a.adresse_id
6  ;
```

Sie bemerken bitte, dass hier abkürzende Alias (Zeile 4) verwendet werden. Das ist gerade bei Verknüpfungen sehr beliebt, da hier oft zwischen den Tabellen unterschieden werden muss. Diese Abfrage wird jetzt in eine Variante des `CREATE TABLE` eingebaut.

MySQL/MariaDB

```
CREATE [TEMPORARY] TABLE [IF NOT EXISTS] tabellenname
  SELECT auswahl
;
```

```
1  mysql> CREATE TEMPORARY TABLE tmp_kadresse
2      ->  SELECT
3      ->   k.kunde_id, k.nachname, k.vorname, a.strasse, a.hnr, a.plz, a.ort
4      ->   FROM
5      ->    kunde k INNER JOIN adresse a
6      ->     ON k.rechnung_adresse_id = a.adresse_id;
7
8  Query OK, 5 rows affected (0.09 sec)
9  Records: 5  Duplicates: 0  Warnings: 0
10
11 mysql> SELECT kunde_id, nachname, ort FROM tmp_kadresse;
12 +----------+-------------+-------------+
13 | kunde_id | nachname    | ort         |
```

[5] Wobei davon auszugehen ist, dass exakt gleiche Ergebnisse durch eine Cache-Strategie beschleunigt zur Verfügung gestellt werden können.

```
+----------+--------------+--------------+
|        1 | Gamdschie    | Hobbingen    |
|        2 | Beutlin      | Hobbingen    |
|        3 | Beutlin      | Hobbingen    |
|        4 | Telcontar    | Minas Tirith |
|        5 | Earendilionn | Bruchtal     |
+----------+--------------+--------------+
```

Jetzt zur ersten Auswertung: Alle Kunden mit Adressdaten und ihren Bestellungen:

```
mysql> SELECT
    ->  t.kunde_id, t.nachname, t.ort, b.bestellung_id, DATE(b.datum)
    ->  FROM
    ->   bestellung b INNER JOIN tmp_kadresse t USING (kunde_id);

+----------+-----------+-----------+--------------+--------------+
| kunde_id | nachname  | ort       | bestellung_id | DATE(b.datum) |
+----------+-----------+-----------+--------------+--------------+
|        1 | Gamdschie | Hobbingen |             1 | 2012-03-24    |
|        2 | Beutlin   | Hobbingen |             2 | 2012-03-23    |
+----------+-----------+-----------+--------------+--------------+
```

Aufgabe 11.10: Erzeugen Sie mit der Spaltenliste `t.*, b.bestellung_id, datum` eine neue temporäre Tabelle und verknüpfen Sie diese mit den Positionen der Bestellungen. Achten Sie auf eine sinnvolle Sortierung. ■

Jetzt zur zweiten Auswertung: Alle Kunden mit Adressdaten und allen Bankverbindungen. Das wollen wir jetzt besonders schön machen. Zuerst eine temporäre Tabelle für die Bankleitzahl und den Banknamen, und dann werden diese beiden temporären Tabellen wieder verknüpft. Und weil wir es jetzt können, wird am Ende noch eine temporäre Tabelle gebaut.

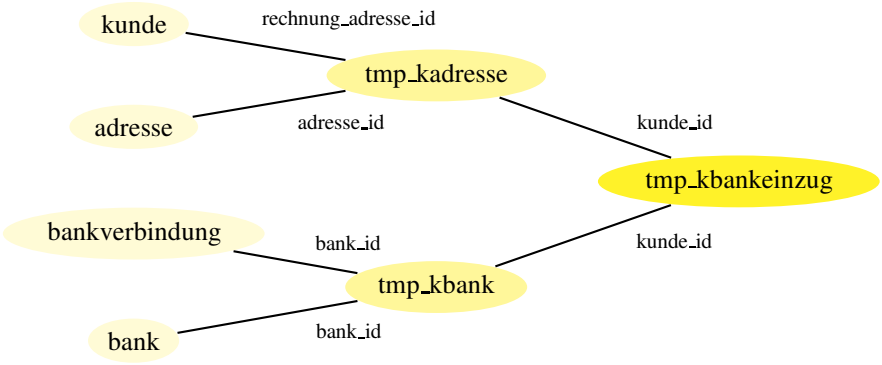
```
mysql> CREATE TEMPORARY TABLE tmp_kbank
    ->  SELECT
    ->   bv.kunde_id, bv.bankverbindung_nr, bv.kontonummer,
    ->   bv.iban, ba.blz, ba.bankname
    ->   FROM
    ->    bankverbindung bv INNER JOIN bank ba USING (bank_id);

mysql> CREATE TEMPORARY TABLE tmp_kbankeinzug
    ->  SELECT
    ->   ka.*, kb.bankverbindung_nr, kb.kontonummer,
    ->   kb.iban, kb.blz, kb.bankname
    ->   FROM
    ->    tmp_kadresse ka INNER JOIN tmp_kbank kb USING(kunde_id);

mysql> SELECT kunde_id, nachname, ort, bankname FROM tmp_kbankeinzug;

+----------+--------------+--------------+---------------+
| kunde_id | nachname     | ort          | bankname      |
+----------+--------------+--------------+---------------+
|        1 | Gamdschie    | Hobbingen    | Postbank      |
|        1 | Gamdschie    | Hobbingen    | Postbank      |
|        2 | Beutlin      | Hobbingen    | Pax-Bank      |
|        3 | Beutlin      | Hobbingen    | Pax-Bank      |
|        4 | Telcontar    | Minas Tirith | Deutsche Bank |
|        5 | Earendilionn | Bruchtal     | Deutsche Bank |
+----------+--------------+--------------+---------------+
```

Das Zusammenspiel der tatsächlichen und temporären Tabellen sei hier in einer Baumstruktur dargestellt. Die Blätter[6] sind die Ursprungstabellen. Zwei davon werden in jeweils eine temporäre Tabelle mit `INNER JOIN` zusammengefasst. Die beiden neu erzeugten temporären Tabellen werden wiederum in eine temporäre Tabelle vereinigt.



Jede der Tabellen kann nun unabhängig voneinander verwendet werden, da die Datensätze in den temporären Tabellen keine Verweise auf die ursprünglichen Datensätze sind, sondern Kopien. Diese *Unabhängigkeit* ist sehr wichtig, wenn man mit vielen konkurrierenden Zugriffen auf die Tabellen `kunde` etc. rechnet.

Bei der Betrachtung von Transaktionen (siehe Kapitel 18 auf Seite 289) werden wir noch sehen, dass Tabellen für Operationen gesperrt werden. Durch die temporären Tabellen kann aber auf der Tabelle `kunde` gearbeitet werden, während Auswertungen auf `tmp_kadresse` stattfinden.

Hinweis: Ich habe oben erwähnt, dass es sich um Stammdaten handelt und somit während der Auswertungen eine geringe Änderungswahrscheinlichkeit besteht. Bei Bewegungsdaten ist das Verfahren genau abzuwägen. Vielleicht lassen sich ja Änderungen auf eine gewisse Uhrzeit festmachen, und die Auswertungen passieren außerhalb dieses Zeitfensters.

11.2.4 Ist Ihnen was aufgefallen?

Beim Bau der letzten temporären Tabelle gab es keinen Primärschlüssel!

```
mysql> DESCRIBE tmp_kadresse;
+----------+------------------+------+-----+---------+-------+
| Field    | Type             | Null | Key | Default | Extra |
+----------+------------------+------+-----+---------+-------+
| kunde_id | int(10) unsigned | NO   |     | 0       |       |
| nachname | varchar(255)     | NO   |     |         |       |
| vorname  | varchar(255)     | NO   |     |         |       |
| strasse  | varchar(255)     | NO   |     |         |       |
| hnr      | varchar(255)     | NO   |     |         |       |
```

[6] die Knoten ganz links

```
| plz      | char(5)          | NO   |     |         |       |
| ort      | varchar(255)     | NO   |     |         |       |
+----------+------------------+------+-----+---------+-------+

mysql> DESCRIBE tmp_kbank;
+-------------------+------------------+------+-----+---------+-------+
| Field             | Type             | Null | Key | Default | Extra |
+-------------------+------------------+------+-----+---------+-------+
| kunde_id          | int(10) unsigned | NO   |     | 0       |       |
| bankverbindung_nr | int(10) unsigned | NO   |     | 0       |       |
| kontonummer       | char(25)         | NO   |     |         |       |
| iban              | char(34)         | NO   |     |         |       |
| blz               | char(12)         | NO   |     |         |       |
| bankname          | varchar(255)     | NO   |     |         |       |
+-------------------+------------------+------+-----+---------+-------+
```

Und tatsächlich: Für einen `INNER JOIN` ist es nicht nötig, Primär-/Fremdschlüsselpaare zu bilden. Dem Befehl ist es völlig egal, was bei einem `INNER JOIN` in der `ON`- oder `USING`-Klausel steht. Es wird nur die Verträglichkeit der Datentypen untersucht.

Natürlich erfolgt die überwiegende Anzahl der Verknüpfungen auf Primär-/Fremdschlüsselpaaren. Aber hier haben wir ein Beispiel dafür, dass auch die Verknüpfung über Nichtschlüsselspalten[7] sinnvoll sein kann.

Es geht sogar noch weiter. Bei der `ON`-Klausel ist das Gleichheitszeichen nicht zwingend vorgeschrieben:

```
mysql> SELECT
    ->  k.kunde_id, k.vorname, a.strasse, a.hnr, a.plz, a.ort
    ->  FROM
    ->   kunde k INNER JOIN adresse a
    ->    ON k.rechnung_adresse_id <= a.adresse_id;
+----------+---------+-----------------+-----+-------+--------------+
| kunde_id | vorname | strasse         | hnr | plz   | ort          |
+----------+---------+-----------------+-----+-------+--------------+
|        1 | Samweis | Beutelhaldenweg | 5   | 67676 | Hobbingen    |
|        1 | Samweis | Beutelhaldenweg | 1   | 67676 | Hobbingen    |
|        2 | Frodo   | Beutelhaldenweg | 1   | 67676 | Hobbingen    |
|        3 | Bilbo   | Beutelhaldenweg | 1   | 67676 | Hobbingen    |
|        1 | Samweis | Auf der Feste   | 1   | 54786 | Minas Tirith |
|        2 | Frodo   | Auf der Feste   | 1   | 54786 | Minas Tirith |
|        3 | Bilbo   | Auf der Feste   | 1   | 54786 | Minas Tirith |
|        4 | Elessar | Auf der Feste   | 1   | 54786 | Minas Tirith |
|        1 | Samweis | Letztes Haus    | 4   | 87567 | Bruchtal     |
|        2 | Frodo   | Letztes Haus    | 4   | 87567 | Bruchtal     |
|        3 | Bilbo   | Letztes Haus    | 4   | 87567 | Bruchtal     |
[...]
|        2 | Frodo   | Baradur         | 1   | 62519 | Lugburz      |
|        3 | Bilbo   | Baradur         | 1   | 62519 | Lugburz      |
|        4 | Elessar | Baradur         | 1   | 62519 | Lugburz      |
|        5 | Elrond  | Baradur         | 1   | 62519 | Lugburz      |
|        1 | Samweis | Hochstrasse     | 4a  | 44879 | Bochum       |
|        2 | Frodo   | Hochstrasse     | 4a  | 44879 | Bochum       |
|        3 | Bilbo   | Hochstrasse     | 4a  | 44879 | Bochum       |
|        4 | Elessar | Hochstrasse     | 4a  | 44879 | Bochum       |
|        5 | Elrond  | Hochstrasse     | 4a  | 44879 | Bochum       |
```

[7] Immerhin sind es Fremdschlüssel.

```
|        1 | Samweis | Industriegebiet | 8   | 44878 | Bochum       |
|        2 | Frodo   | Industriegebiet | 8   | 44878 | Bochum       |
|        3 | Bilbo   | Industriegebiet | 8   | 44878 | Bochum       |
|        4 | Elessar | Industriegebiet | 8   | 44878 | Bochum       |
|        5 | Elrond  | Industriegebiet | 8   | 44878 | Bochum       |
+----------+---------+-----------------+-----+-------+--------------+
28 rows in set (0.00 sec)
```

In Zeile 5 ist anstelle des = ein <= verwendet worden. Ich kann mir zwar keine vernünftige Anwendung dafür ausdenken, will aber nicht bestreiten, dass es sie irgendwo gibt.

Wird aber die Verknüpfung über die Gleichheit hergestellt, hat das Ganze einen schönen Namen:

Definition 42: EQUI JOIN
Wird bei INNER JOIN auf die Gleichheit von Fremdschlüsselwert und Primärschlüsselwert getestet, spricht man von einem *EQUI JOIN*.

Sie haben sicher bei der Definition 40 auf Seite 183 bemerkt, dass nur allgemein von *passen* gesprochen wird. Was immer dieses *passen* auch bedeutet. Der Test auf Gleichheit ist aber so gängig, dass der Begriff `INNER JOIN` synonym für `EQUI JOIN` verwendet wird.

Hinweis: Lassen Sie sich nicht verwirren. Erstmal nach Primär-/ Fremdschlüsselpaaren suchen und diese mit = verknüpfen. In den absolut meisten Fällen sind Sie auf der sicheren Seite. Erst, wenn das so überhaupt nicht klappen sollte, denken Sie über Alternativen nach.

11.3 INNER JOIN über mehr als zwei Tabellen

Eine Möglichkeit, mehr als zwei Tabellen zu verknüpfen, haben Sie oben auf Seite 189 kennengelernt. Die Verwendung von temporären Tabellen bietet sich aber nur bei Stammdaten an oder wenn die Änderungen unerheblich für das Gesamtergebnis sind.

Trotzdem können wir aus der Episode mit den temporären Tabellen eine wichtige Schlussfolgerung ziehen: Das Ergebnis eines `INNER JOIN`s ist wieder eine Tabelle. Wie kann ich damit die Beschränkung umgehen, dass der `INNER JOIN` nur zwei Tabellen verknüpfen kann?

Betrachten wir dazu den Klassiker für die Verknüpfung von mehr als zwei Tabellen: das Auflösen einer *n:m*-Verknüpfung (siehe Abschnitt 2.2.4 auf Seite 29). Wir haben eine *n:m*-Verknüpfung zwischen den Tabellen `artikel` und `warengruppe`.

Unser erster Versuch besteht darin, wie oben beschrieben vorzugehen, indem wir zwei *1:n*-Verknüpfungen erstellen:

1. **Ermitteln der beteiligten Tabellen**: `artikel_nm_warengruppe` und `artikel`.

2. **Ermitteln der Primär- und Fremdschlüssel:** In der Tabelle `artikel_nm_warengruppe` gibt es keine *echten* Primärschlüssel, aber die beiden Fremdschlüssel `warengruppe_id` und `artikel_id`. In der Tabelle `artikel` gibt es nur den Primärschlüssel `artikel_id`.
3. **Fremdschlüssel festlegen:** Laut Namenskonvention muss `artikel_id` in der Tabelle `artikel_nm_warengruppe` der gesuchte Fremdschlüssel sein.
4. **Primärschlüssel festlegen:** Wir markieren `artikel_id` in `artikel`.
5. **Schablone benutzen:** Jetzt schreiben Sie auf das Blatt die Schablone für den `INNER JOIN`.
6. **Fremdschlüsseltabelle eintragen:** Fügen Sie links vom `INNER JOIN` den Tabellennamen für die Tabelle mit dem markierten Fremdschlüssel ein.
7. **Primärschlüsseltabelle eintragen:** Fügen Sie an die passende Stelle den Tabellennamen für die Tabelle mit dem markierten Primärschlüssel ein.
8. **Fremdschlüssel eintragen:** Fügen Sie an die passende Stelle den Spaltennamen des markierten Fremdschlüssels ein.
9. **Primärschlüssel eintragen:** Fügen Sie an die passende Stelle den Spaltennamen des markierten Primärschlüssels ein.
10. **Fertig!**

```
mysql> SELECT
    ->  a.bezeichnung, nm.warengruppe_id
    ->  FROM
    ->   artikel_nm_warengruppe nm NATURAL JOIN artikel a;
+---------------+----------------+
| bezeichnung   | warengruppe_id |
+---------------+----------------+
| Feder         |              1 |
| Papier (100)  |              1 |
| Schaufel      |              3 |
| Schaufel      |              4 |
| Silberzwiebel |              2 |
| Silberzwiebel |              3 |
| Spaten        |              3 |
| Spaten        |              4 |
| Tinte (blau)  |              1 |
| Tinte (gold)  |              1 |
| Tinte (rot)   |              1 |
| Tulpenzwiebel |              2 |
| Tulpenzwiebel |              3 |
+---------------+----------------+
```

Das ganze Prozedere mit den Tabellen `artikel_nm_warengruppe` und `warengruppe` führt zu einer zweiten Verknüpfung:

```
mysql> SELECT
    ->  w.bezeichnung, nm.artikel_id
    ->  FROM
    ->   artikel_nm_warengruppe nm NATURAL JOIN warengruppe w;
+--------------+------------+
| bezeichnung  | artikel_id |
+--------------+------------+
| Bürobedarf   |       3001 |
| Bürobedarf   |       3005 |
| Bürobedarf   |       3006 |
```

```
| Bürobedarf   |       3007 |
| Bürobedarf   |       3010 |
| Gartenbedarf |       7856 |
| Gartenbedarf |       7863 |
| Gartenbedarf |       9010 |
| Gartenbedarf |       9015 |
| Pflanzen     |       7856 |
| Pflanzen     |       7863 |
| Werkzeug     |       9010 |
| Werkzeug     |       9015 |
+--------------+------------+
```

Und jetzt kommt's: Wir nehmen einfach den letzten `SELECT`, klammern die Verknüpfung und verwenden nicht mehr `NATURAL`, sondern `INNER JOIN`:

```
SELECT
 w.bezeichnung, nm.artikel_id
 FROM
  (artikel_nm_warengruppe nm INNER JOIN warengruppe w USING(warengruppe_id))
;
```

Der Inhalt der Klammer ist eine (!) Tabelle, und diese könnte der linke Teil einer neuen Verknüpfung sein:

```
SELECT
 w.bezeichnung, a.bezeichnung
 FROM
  (artikel_nm_warengruppe nm INNER JOIN warengruppe w USING (warengruppe_id))
                              INNER JOIN artikel a USING(artikel_id)
;
```

Und eine weitere gute Nachricht ist, dass man die Klammern nicht braucht; sie dienten nur dem besseren Verständnis:

```
SELECT
 w.bezeichnung Warengruppe, a.bezeichnung Artikel
 FROM
  artikel_nm_warengruppe nm INNER JOIN warengruppe w USING (warengruppe_id)
                            INNER JOIN artikel a USING(artikel_id)
;
```

Dieser `SELECT` liefert das gewünschte Ergebnis:

```
+--------------+---------------+
| Warengruppe  | Artikel       |
+--------------+---------------+
| Bürobedarf   | Papier (100)  |
| Bürobedarf   | Tinte (gold)  |
| Bürobedarf   | Tinte (rot)   |
| Bürobedarf   | Tinte (blau)  |
| Bürobedarf   | Feder         |
| Gartenbedarf | Silberwiebel  |
| Gartenbedarf | Tulpenzwiebel |
| Gartenbedarf | Schaufel      |
| Gartenbedarf | Spaten        |
| Pflanzen     | Silberwiebel  |
| Pflanzen     | Tulpenzwiebel |
| Werkzeug     | Schaufel      |
| Werkzeug     | Spaten        |
+--------------+---------------+
```

Die Spezifikation des SELECT lässt sich somit erweitern:

SQL3
MySQL/MariaDB

```
SELECT [DISTINCT]
  {*|spaltenliste|ausdruck}
  FROM
    tabname [{INNER|NATURAL} JOIN tabname
      [{ON tabname.spaltenname = tabname.spaltenname|USING(spaltenname)}]]*
  [WHERE bedingung]
  [ORDER BY spaltenname [ASC|DESC] [,spaltenname [ASC|DESC]]*]
;
```

Der Abschnitt rechts vom ersten *tabname* kann beliebig oft wiederholt werden, wobei beliebig auch bedeuten kann, dass er gar nicht vorkommt. Es wäre dann ein normaler SELECT.

Aufgabe 11.11: Warum konnte beim letzten SELECT kein NATURAL JOIN verwendet werden?

Aufgabe 11.12: Erweitern Sie diesen SELECT um die Positionen, in denen der Artikel vorkommt.

Aufgabe 11.13: Erweitern Sie das Ergebnis der letzten Aufgabe um die Daten der Bestellung.

Aufgabe 11.14: Erweitern Sie das Ergebnis der letzten Aufgabe um die Daten des Kunden.

Aufgabe 11.15: Erweitern Sie das Ergebnis der letzten Aufgabe um die Rechnungsadresse des Kunden.

11.4 Es muss nicht immer heiße Liebe sein: OUTER JOIN

Wir wollen die Kunden mit ihren ggf. vorhandenen Bestellungen wissen:

```
mysql> SELECT
    ->  k.kunde_id, k.nachname, k.vorname, b.datum
    ->  FROM
    ->   bestellung b INNER JOIN kunde k USING (kunde_id)
    ->  ORDER BY
    ->   k.nachname, k.vorname;
+----------+-----------+---------+---------------------+
| kunde_id | nachname  | vorname | datum               |
+----------+-----------+---------+---------------------+
|        2 | Beutlin   | Frodo   | 2012-03-23 16:11:00 |
|        1 | Gamdschie | Samweis | 2012-03-24 17:41:00 |
+----------+-----------+---------+---------------------+
```

Stichwortverzeichnis