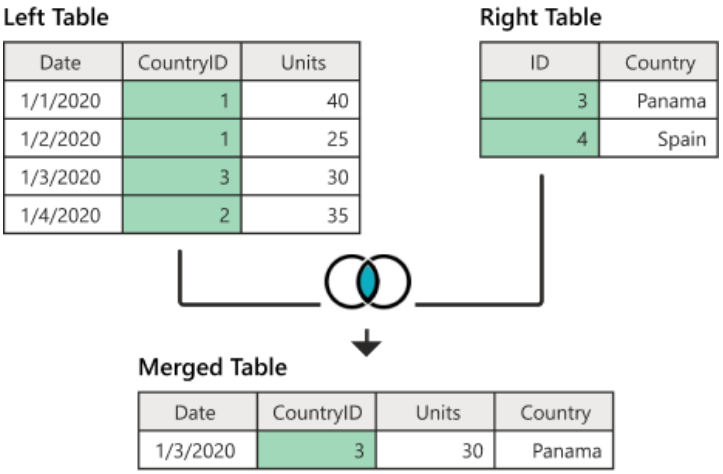
Der JOIN-Befehl

Theorie



David Tyndall

Januar 2023

Inhaltsverzeichnis

1 Informationen auf mehrere Tabellen verteilen

Bis jetzt waren alle notwendigen Informationen jeweils in *einer* Tabelle abgespeichert. Dies macht Sinn für Informationen, die eng zusammengehören. Zum Beispiel sind der Vor- und Nachname oder das Geschlecht Attribute einer bestimmten Person, also Informationen, die untrennbar verbunden sind mit dieser bestimmten Person. In einer Personen-Tabelle würden alle diese Attribute in einer Zeile gespeichert.

In vielen Problemstellungen ist man jedoch interessiert an Informationen, die nicht ganz so eng zusammengehören. Man speichert die Informationen dann nicht nur in einer, sondern in *mehreren* Tabellen ab.

Als Beispiel wollen wir eine Personengruppe bei einem Restaurantbesuch modellieren. Die Personen werden dabei weiterhin in einer Personen-Tabelle abgespeichert. Aber wo soll beispielsweise die Menükarte gespeichert werden? Offensichtlich ist es nicht sinnvoll, sie in die Personen-Tabelle zu integrieren. Vielmehr würde man die Menükarte in einer eigenen Tabelle mit allen zugehörigen Attributen (z. B. Name des Menüs, Preis, etc.) speichern.

ID	Vorname	Nachname
1	Jonas	Allenmann
2	Elias	Wirth
3	Jonas	Kopp
4	Severin	Meier

Tabelle 1: Personen

ID	Menu	Preis (CHF)
1	Spaghetti	21
2	Ravioli	19
3	Schnitzel mit Pommes	23
4	Pilzrisotto	18
5	Cordon bleu	23

Tabelle 2: Hauptgerichte der Speisekarte

2 Beziehungen zwischen Tabellen

Aufgabe 1: Wir wollen die Bestellungen aller Personen darstellen. Du kannst davon ausgehen, dass jede Person maximal einen Hauptgang bestellt.

a) Wie könnte man die Bestellungen auf möglichst einfache Weise darstellen?

b) Findest du eine Möglichkeit zur Darstellung ausschliesslich unter Verwendung von Tabellen? (Tabellen sind das einzige „Medium“, das in SQL-Datenbanken für die Datenspeicherung zur Verfügung steht.)

c) Wie viele Kombinationsmöglichkeiten gibt es, wenn jede Person genau einen Hauptgang bestellt?

2.1 Beziehung – was ist das?

Dieses Kapitel trägt den Titel „Beziehung *zwischen Tabellen*“ – was soll man darunter verstehen?

Gemeint ist vielmehr die Beziehung zwischen *Zeilen* zweier Tabellen, respektive Elementen zweier Mengen.[1] In der Informatik und in der Mathematik nennt man diese Beziehung eine *Zuordnung*. In Aufgabe 1 hast du schon ein Beispiel dafür gesehen: Personen bestellen Menüs. Das sind konkrete Zuordnungen von Menüs auf Personen. In einer graphischen Darstellung zeichnet man so eine Zuordnung oft mit einer Linie oder einem Pfeil (vgl. Abb. 1).

Die Menge der Zuordnungen bezeichnet man als *Relation*. In unserem Restaurantbeispiel ist die Relation also die *Menge aller Bestellungen*. In Abbildung 1 ist die Relation die *Menge aller Pfeile*. Eine Relation ist immer eine Teilmenge (oder identisch mit) der Menge aller grundsätzlich möglichen Zuordnungen.

Wir wollen die Relationen nun kategorisieren. Wir betrachten hierzu die sogenannten *Multiplizitäten* der Relationen, d. h. wir schauen, wie oft ein einzelnes Element (also eine Zeile) zugeordnet werden kann.

2.2 N:1-Beziehung

In Aufgabe 1 haben wir eine Relation zwischen den Tabellen betrachtet, welche folgende Eigenschaften hat:

Sicht Person: Eine Person wählt maximal 1 Menü. Dabei darf das gleiche Menü mehrmals gewählt werden.

Sicht Menü: Ein Menü kann von verschiedenen Personen gewählt werden.

Man nennt diese Zuordnung eine *N:1-Beziehung*. Anders ausgedrückt: „*N* (mehrere) Personen können *1* (einziges) Menü wählen“. In einem Graphen kann man die N:1-Beziehung wie folgt darstellen:

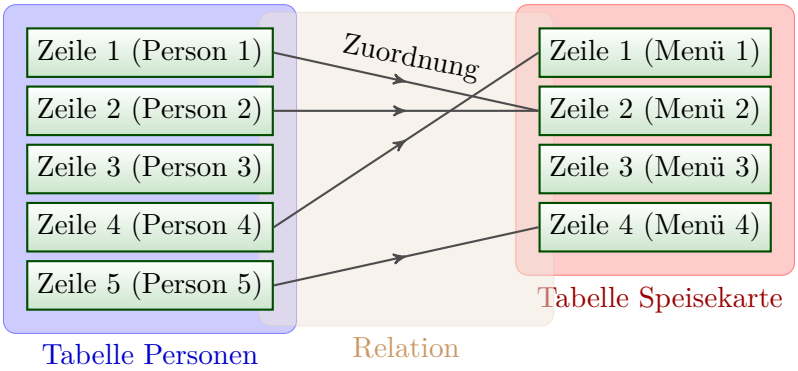


Abb. 1: N:1-Beziehung

[1] Eine Tabelle kann als *Menge* von Zeilen aufgefasst werden. Dann sind die einzelnen Tabellenzeilen jeweils die *Elemente* dieser Menge.

Einige Bemerkungen zum Graph:

- Von keiner Person darf mehr als ein Pfeil ausgehen. Dies rechtfertigt die *1* im Namen der N:1-Beziehung. Geht kein Pfeil von einer Person aus, so ist das in Ordnung (vgl. Person 3).
- Ein Menü darf von beliebig vielen Pfeilen erreicht werden (vgl. Menü 2). Dies rechtfertigt das *N* im Namen der N:1-Beziehung. Es ist auch zulässig, wenn das Menü von keinem Pfeil erreicht wird (vgl. Menü 3).
- Fasst man die Tabellen als *Mengen* auf, ist die Menge der Personen im Graph blau hinterlegt, und die Menge der Menüs rot. In Mengen ausgedrückt bedeutet die N:1-Beziehung:
  - Jedem Element der Personenmenge (also einer Person) darf maximal ein Element aus der Menümenge (also ein Menü) zugeordnet werden.
  - Einem Element der Menümenge dürfen beliebig viele Elemente aus der Personenmenge zugeordnet werden.

2.3 Weitere Multiplizitäten

Aufgabe 2: Findest du andere Möglichkeiten für Multiplizitäten von Relationen? Zeichne passende Graphen und gib ihnen sinnvolle Namen.
Zusatzaufgabe: Finde Alltagsbeispiele für die verschiedenen Multiplizitäten.

Wenn man nur auf die Anzahl der Elemente achtet, mit der jede Zeile verbunden wird, gibt es je nach Zählweise 3-4 Typen von Multiplizitäten:

- *1:1-Beziehung*: Jeder Zeile aus der ersten Tabelle wird maximal eine Zeile aus der anderen Tabelle zugeordnet. Dabei darf aus der zweiten Tabelle jede Zeile maximal einmal zugeordnet worden sein.
- *1:N-Beziehung*: Umkehrung der N:1-Beziehung. Oft werden diese beiden Multiplizitäten als gleichwertig betrachtet, da sie durch Vertauschen der Tabellen identisch gehandhabt werden können.
- *M:N-Beziehung*: Jeder Zeile aus der ersten Tabelle werden beliebig viele Zeilen aus der zweiten Tabelle zugeordnet. Umgekehrt darf jede Zeile aus der zweiten Tabelle beliebig vielen Zeilen aus der ersten Tabelle zugeordnet worden sein. Auch *keine* Zuordnung ist in beiden Tabellen erlaubt.

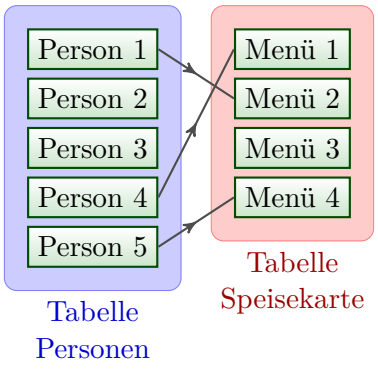


Abb. 2: 1:1-Beziehung

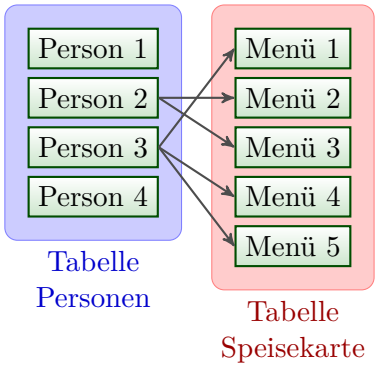


Abb. 3: 1:N-Beziehung

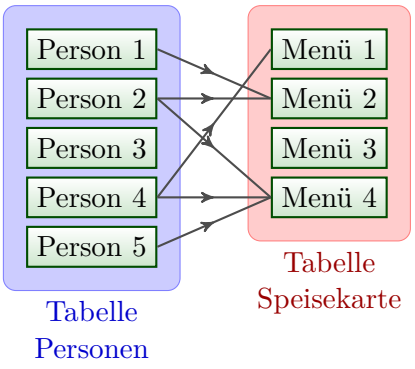


Abb. 4: M:N-Beziehung

Man kann die 1:1-, 1:N- und N:1-Beziehungen als Spezialfall der M:N-Beziehung auffassen. Da die ersten 3 Multiplizitäten in Tabellen schlanker umgesetzt werden können, macht man die Unterscheidung jedoch bewusst. Zudem ist die 1:1-Beziehung ein Spezialfall der 1:N- oder N:1-Beziehung.

Aufgabe 3: Welche Multiplizität haben die folgenden Relationen? Die Auswahl der Speisen sei jeweils auf die Speisekarte/verfügbaren Teller beschränkt.

a) Welche Hauptgerichte mögen die Besuchenden des Restaurants?

b) Was ist das Lieblingsmenü der Gäste?

c) Auf einem Buffet sind alle Teller schon angerichtet. Die Gäste wählen je einen Teller.

d) Welche Menüs erscheinen den einzelnen Personen zu teuer?

e) Welches Menü haben die Konsumierenden beim letzten Besuch bestellt?

2.4 Speichern der Verbindungsinformation

Bereits in Aufgabe 1 hast du dir Gedanken gemacht, wie man die Verbindungsinformation abspeichern könnte. Bei 1:1-, 1:N- und N:1-Beziehungen ist die einfachste Weise, eine der beiden Tabellen um ein Attribut zu erweitern. Darin speichert man den Schlüssel der zu verbindenden Zeile aus der anderen Tabelle. Man nennt diesen Schlüssel *Fremdschlüssel*, da es sich um einen Schlüssel aus einer „fremden“ Tabelle handelt.

ID	Vorname	Nachname	MenuID
1	Jonas	Allenmann	3
2	Elias	Wirth	1
3	Jonas	Kopp	1
4	Severin	Meier	NULL

Tabelle 3: Personentabelle inkl. bestelltem Menü

Aufgabe 4: Weshalb verwendet man wohl den Schlüssel und nicht beispielsweise den Namen des Menüs? Das wäre ja viel einfacher zu lesen!

Aufgabe 5: In welcher der beiden Tabellen müssen die Fremdschlüssel jeweils abgespeichert werden

a) bei einer 1:1-Beziehung?

b) bei einer 1:N-Beziehung?

c) bei einer N:1-Beziehung?

Begründe deine Antwort.

Etwas schwieriger sind die M:N-Beziehungen abzuspeichern. Egal, welche Tabelle man wählt, es müssten mehrere Fremdschlüssel gespeichert werden. Dies ist bei SQL-Tabellen jedoch keine sinnvolle Vorgehensweise, da grundsätzlich jedes Attribut genau ein Stück Information speichern soll, sonst werden die Abfragen mühsam und ineffizient. Stattdessen kreiert man eine neue Tabelle (vgl. Tabelle 4), welche in einer Zeile die Schlüssel beider verbundenen Einträge beinhaltet. So können beliebig viele Verbindungen gespeichert werden, ohne dass das Nur-Eine-Information-Gebot verletzt würde.

ID	PersonID	MenuID
1	1	1
2	1	3
3	2	1
4	2	2
5	2	3
...		

Tabelle 4: Verbindungstabelle Person-hat-Menü-gerne

In Tabelle 4 wurde jeder Verbindung explizit nochmals eine ID (d. h. ein Schlüssel) zugeordnet. Dabei wäre dies hier nicht nötig, da die Kombination aus PersonID und MenuID schon eindeutig ist. Unter solchen Umständen kann man die Kombination dieser Attribute als neuen Schlüssel verwenden. Dies nennt man allgemein einen *zusammengesetzten Schlüssel*. Sollte die Kombination dieser Attribute aber mehrfach vorkommen können, ist ein separater Schlüssel zwingend notwendig. Dies könnte beispielsweise dann der Fall sein, wenn man am zeitlichen Verlauf der Werte interessiert ist, und daher jedes Jahr erfasst, welche Person welches Menü gern hat. Selbstverständlich würde es dann Sinn machen, noch weitere Attribute wie z. B. das Jahr in der Verbindungstabelle zu speichern.

2.5 Relationen revisited

Vergleich mit Funktionen

Aus der Mathematik kennst du den Begriff der *Funktion* (manchmal auch *Abbildung* genannt). Auch sie ist eine Menge von Zuordnungen: Elementen x aus einer Definitionsmenge X werden Elemente y aus der Zielmenge Y zugeordnet (vgl. Abbildung 5). Bei der Funktion muss die Zuordnung zwingend *eindeutig* (oder präziser: *rechtseindeutig*[2]) sein, d. h., einem Element aus der Definitionsmenge darf maximal ein Element aus der Zielmenge zugeordnet werden. Umgekehrt darf ein Element aus der Zielmenge mehreren Elementen aus der Definitionsmenge zugeordnet sein (z. B. $y = 1$ in Abb. 5). Bei der Funktion handelt es sich also um eine N:1-Beziehung.

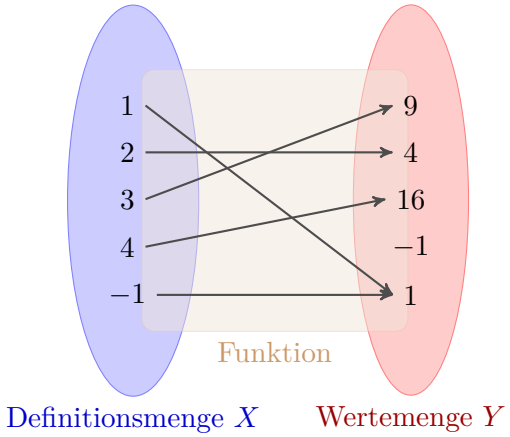


Abb. 5: Funktion

Die N:1-Multiplizität ist aber nicht die einzige Einschränkung, welche die Relation erfüllen muss, damit sie eine *Funktion* ist. Zusätzlich muss *jedes* Element aus der Definitionsmenge X einem y-Wert zugeordnet sein; es darf also kein Element x in X vorhanden sein, von dem kein Pfeil weggeht. Eine Relation, die diese Eigenschaft besitzt, nennt man *linkstotal*. [3]

Eine Funktion ist also eine rechtseindeutige und linkstotale Relation.

Weitere Eigenschaften

Es gibt noch weitere Eigenschaften, die eine Funktion oder Relation erfüllen kann:

- Eine Relation oder Funktion nennt man *injektiv* oder *linkseindeutig*, wenn jedes Element y aus der rechten Menge Y maximal einem Element x aus der linken Menge X zugeordnet wurde. Die Funktion in Abbildung 5 ist nicht injektiv, weil das Element $y = 1$ mehrfach zugeordnet wird.

[2] Es ist für jedes Element aus der rechten Menge eindeutig, welchem Element es aus der linken Menge zugeordnet wurde.

[3] Dies ist ein Unterschied zur N:1-Beziehung, wie wir sie in den vorangegangenen Kapiteln betrachtet haben: Dort darf es in der linken Menge Elemente geben, die nicht zugeordnet werden.

- Eine Relation oder Funktion nennt man *surjektiv* oder *rechtstotal*, wenn jedes Element y aus der rechten Menge Y mindestens einem Element x aus der linken Menge X zugeordnet wird. Die Funktion in Abbildung 5 ist nicht surjektiv, weil das Element $y = -1$ keinem Element x zugeordnet wurde.
- Eine Relation nennt man *bijektiv*, wenn sie sowohl links- als auch rechtstotal sowie links- als auch rechtseindeutig ist. Das heisst: Jedes Element x ist genau einem Element y zugeordnet, und jedes Element y ist genau einem Element x zugeordnet. Es gibt also eine *eineindeutige* Zuordnung (d. h. Eins-zu-Eins-Korrespondenz) für alle Elemente aus X und Y.

  Folglich ist eine Funktion bijektiv, wenn sie sowohl injektiv als auch surjektiv ist, da die anderen beiden Eigenschaften implizit bereits in der Eigenschaft „Funktion“ enthalten sind.

Aufgabe 6: Zähle für jede Relation aus Abbildung 6 auf, welche der folgenden Eigenschaften sie erfüllt: injektiv, rechtseindeutig, linkstotal, surjektiv, bijektiv. Was ist die Multiplizität? Ist es eine Funktion?

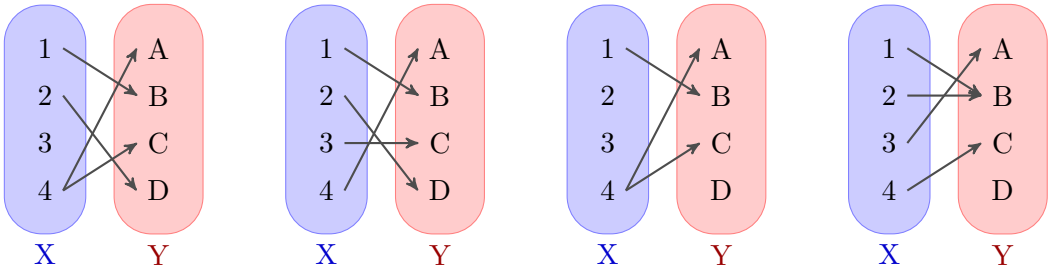
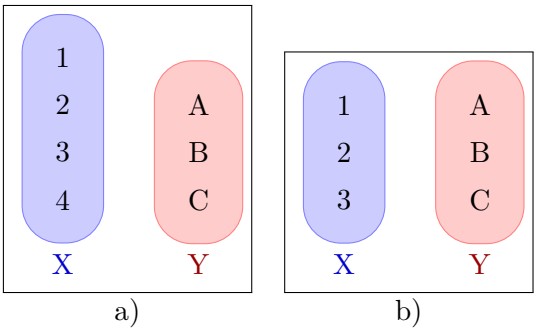


Abb. 6: Verschiedene Eigenschaften von Relationen

Aufgabe 7: Zeichne die Zuordnungen einer Relation, welche folgende Eigenschaften erfüllt:

a) Eine Funktion, die surjektiv, aber nicht injektiv ist.

b) Eine linkstotale Relation, welche sonst keine der oben kennen gelernten Eigenschaften erfüllt.



3 Tabellen verbinden mit JOIN

Wenn wir wissen wollen, welche Menüs Elias Wirth mag, oder welches Menü Severin Meier bestellt hat, müssen wir die Information aus mehreren Tabellen zusammensuchen.

Aufgabe 8: Beantworte folgende Fragen unter Verwendung der Tabellen 2, 3 und 4.

a) Wie heisst das Menü, welches Jonas Allenmann bestellt hat?

b) Gib an, wie alle Menüs heissen, die Elias mag.

c) Hat Jonas Allenmann ein Menü bestellt, das er mag?

3.1 Verwendung von JOIN

Bei kleinen und wenigen Tabellen findet man die gewünschten Informationen einigermassen schnell. Mühsamer wird es bei grösseren Datenmengen und vielen involvierten Tabellen. SQL bietet hier jedoch gute Unterstützung. Mit dem Befehl `JOIN` können Tabellen verbunden werden. Eine SQL-Abfrage, welche die Antwort zu a) liefert, würde wie folgt aussehen:

```
1 SELECT Speisekarte.Menu
2 FROM Personen
3         JOIN Speisekarte ON Personen.MenuID = Speisekarte.ID
4 WHERE Personen.Nachname="Allenmann"
```

Abfrage 1: Bestellung von Jonas Allenmann

Im Detail erklärt:

- Die Struktur `SELECT ... FROM ... WHERE ...` bleibt grundsätzlich erhalten. Im `FROM`-Teil können beliebig viele Tabellen mit Hilfe von `JOIN` angegeben werden.
- Zeile 3, `JOIN <Tabelle> ON <Bedingungen>`: Hier wird angegeben, welche Tabellen verbunden werden sollen, und wie das genau geschehen soll.
  Das bedeutet konkret in diesem Beispiel: Eine Zeile aus der Personentabelle (d. h. eine Person) soll mit einer Zeile aus der Menütabelle (d. h. ein Menü) verbunden werden, so dass die ID des Menüs mit dem Wert der MenuID der Person übereinstimmt.

- Bezeichnung der Attribute (Zeilen 1, 3, und 4): Weil die Informationen von verschiedenen Tabellen zusammengefügt werden, muss man das gewünschte Attribut exakt benennen. So ist beispielsweise der Bezeichner ID nicht eindeutig – soll das Attribut `ID` der Personentabelle oder dasjenige der Menütabelle verwendet werden? Um Eindeutigkeit zu schaffen, stellt man `<Tabellenname>.` vor das gewünschte Attribut. Man nennt dies den *voll qualifizierten Namen*.

  Die Spalte `Menu` ist allerdings eindeutig, da es diesen Namen in beiden Tabellen nur einmal gibt. Daher könnte man beispielsweise den Tabellenbezeichner `Speisekarte.` in Zeile 1 weglassen, so dass nur noch `SELECT Menu` stehen bleibt, und die SQL-Abfrage würde immer noch funktionieren.

3.2 Aliasse

Informatikerinnen und Informatiker lieben kompakte Schreibweisen. Statt den vollen Tabellennamen immer auszuschreiben, kann man innerhalb der Abfrage mit dem Schlüsselwort `AS` eine Abkürzung, ein so genanntes *Alias* einführen. Abfrage 2 zeigt nochmals die obige Abfrage unter Verwendung von Aliassen. Die Personentabelle kann fortan mit `p`, die Speisetabelle mit `s` referenziert werden. `p.Nachname` ist also gleichbedeutend mit `Personen.Nachname` oder `s.ID` mit `Speisetabelle.ID`.

In SQL ist es bei *Aliassen für Tabellennamen* sogar erlaubt, das `AS`-Schlüsselwort wegzulassen. Vergleicht man Abfrage 2 mit Abfrage 3, sieht man, dass die `AS`-Schlüsselwörter in den Zeilen 2 und 3 fehlen. Andere Aliasse müssen jedoch immer mit `AS` eingeführt werden.

```
1 SELECT *
2 FROM Personen AS p
3         JOIN Speisekarte AS s ON p.MenuID = s.ID
4 WHERE p.Nachname="Allenmann"
```

Abfrage 2: Verwendung von Aliassen

3.3 Funktionsweise von JOIN

Beim Joinen von Tabellen erzeugt SQL intern zuerst eine temporäre Tabelle, in der alle zugeordneten Einträge zu neuen Zeilen zusammengefasst werden (vgl. Tabelle 5). Die Zeilen können dadurch recht viele Attribute enthalten. Auf dieser Tabelle sind dann wieder alle üblichen Operationen möglich – Selektionen (`WHERE`), Projektion (`SELECT`), Aggregationen (z. B. `COUNT`), usw.

ID	Vorname	Nachname	MenuID	ID	Menu	Preis
1	Jonas	Allenmann	3	3	Schnitzel mit Pommes	23
2	Elias	Wirth	1	1	Spaghetti	21
3	Jonas	Kopp	1	1	Spaghetti	21

Tabelle 5: Join von Person und Menü (über p.MenuID = s.ID)

Dieses interne Resultat können wir anzeigen, indem wir in obiger Abfrage bei der Projektion (`SELECT`) keine Einschränkung vornehmen sowie die Selektion (`WHERE`) weglassen. Wir erhalten Abfrage 3.

```
1 SELECT *
2 FROM Personen p
3         JOIN Speisekarte s ON p.MenuID = s.ID
```

Abfrage 3: Join von Person und Menü (über p.MenuID = s.ID)

Aufgabe 9:

a) Was passiert wohl, wenn man in Abfrage 3 Zeile 3 ersetzt durch `JOIN Speisekarte AS s ON p.ID = s.ID`? Schreibe die erwartete Ausgabe auf.

b) Kontrolliere deine Antwort, indem du die Abfrage auf der Datenbank restaurant.db ausführst.

3.4 Mehrdeutigkeiten

Was passiert wohl, wenn die `ON`-Bedingung beim Join kein eindeutiges Resultat liefert?

Um das herauszufinden machen wir einen Test. Der Preis der Menüs ist z. B. kein eindeutiges Kriterium zur Bestimmung eines Menüs, da Cordon bleus und Schnitzel mit Pommes gleich viel kosten. In Abfrage 4 wird dies ausgenutzt, die Verbindung geschieht über `p.MenuID + 20` und `s.preis`. Wir suchen also alle Personen-Menü-Kombinationen, bei denen der Preis des Menüs um 20 höher ist als die bestellte MenuID (was in der realen Welt keine besondere Bedeutung hat).

```
1 SELECT *
2 FROM Personen p
3         JOIN Speisekarte s ON p.MenuID + 20 = s.Preis
```

Abfrage 4: Mehrdeutiger Join über MenuID und Preis

Das Resultat dieser Abfrage ist in Tabelle 6 angegeben. Es fällt auf, dass Jonas Allenmann zweimal erscheint, einmal gepaart mit dem Cordon bleu und einmal mit Schnitzel mit Pommes.

ID	Vorname	Nachname	MenuID	ID	Menu	Preis
1	Jonas	Allenmann	3	5	Cordon bleu	23
1	Jonas	Allenmann	3	3	Schnitzel mit Pommes	23
2	Elias	Wirth	1	1	Spaghetti	21
3	Jonas	Kopp	1	1	Spaghetti	21

Tabelle 6: Mehrdeutiger Join über MenuID und Preis

SQL listet also bei Mehrdeutigkeiten alle möglichen Kombinationen auf. Dies ist ein allgemeines Prinzip in SQL: Wann immer es mehrere Möglichkeiten gibt um eine Anfrage korrekt zu beantworten, werden alle passenden Kombinationen ausgegeben.

Das macht Sinn: Würde bei der temporären internen Tabelle eine Kombination fehlen, und man sucht mit einem Filter (`WHERE`) genau nach dieser Kombination, erhielte man ein leeres Resultat!

Aufgabe 10:

a) Es gibt noch eine weitere Mehrdeutigkeit in Tabelle 6 versteckt. Findest du sie? Unterscheidet sich diese prinzipiell von derjenigen auf den ersten beiden Zeilen?

b) Vertausche die Reihenfolge der beiden Tabellen im `FROM`-Bereich und führe die Abfrage auf der Datenbank restaurant.db aus. Inwiefern verändert das die Ausgabe? Ist das ein prinzipieller Unterschied zu vorher?

4 Varianten von JOIN

Es gibt verschiedene Varianten des JOIN-Befehls. Wir wollen ein paar davon kennen lernen.

4.1 NATURAL JOIN

In Aufgabe 9 haben wir die Tabellen über die beiden gleichnamigen `ID`-Attribute verbunden. Das war nicht sehr sinnvoll im Restaurant-Beispiel, aber es gibt durchaus Anwendungen, wo das nützlich ist. Betrachte z. B. folgende Tabellen:

Personal-nummer	Vorname	Nachname
43596	Jakob	Zindel
49832	Herbert	Hug
47493	Susanne	Schmid

Tabelle 7: Lehrpersonen

Kurs-nummer	Fach	Klasse	Personal-nummer
98435	Deutsch	1c	43596
98439	Deutsch	2b	43596
98478	Mathematik	1a	49832
98483	Mathematik	3a	49832
98488	Mathematik	3c	47493
98494	Informatik	2b	47493
98497	Informatik	3a	47493

Tabelle 8: Kurse

Wenn wir den Namen der Person wissen wollen, welche einen bestimmten Kurs unterrichtet (die Verbindungsinformation ist in der Tabelle „Kurse“ in der Spalte „Personalnummer“ enthalten), lässt sich das bequem mit einem `NATURAL JOIN` lösen:

```
1 SELECT Kurse.Fach, Kurse.Klasse, Lehrperson.Nachname
2 FROM Lehrpersonen NATURAL JOIN Kurse
```

Abfrage 5: NATURAL JOIN

Beachte, dass nach dem `JOIN` ... kein `ON` ... benutzt wurde. `NATURAL JOIN` macht die Verbindung automatisch über die gleichnamigen Attribute. Hier werden also diejenigen Zeilen zusammengefügt, welche dieselbe Personalnummer haben.

Typischerweise gibt es nur ein gleichnamiges Attribut, über das verbunden wird. Sind jedoch mehrere gleichnamige Attribute vorhanden, müssen die Zeileneinträge in allen gleichnamigen Attributen übereinstimmen, damit die Zeilen mit `NATURAL JOIN` verbunden werden.

4.2 LEFT und RIGHT JOIN

Aufgabe 11: Im Resultat der Abfrage 3 (Seite 13) fehlt eine Person, nämlich Severin Meier.

a) Weshalb fehlt Severin Meier?

Aufgrund dieser Eigenschaft wird der „nackte“ `JOIN`-Befehl manchmal auch `INNER JOIN` genannt.

b) Abfrage 6 liefert das Resultat in Tabelle 9. Was macht also der Befehl `LEFT JOIN`?

```
1 SELECT *
2 FROM Personen
3         LEFT JOIN Speisekarte ON Personen.MenuID = Speisekarte.ID
```

Abfrage 6: LEFT JOIN von Person und Menü

ID	Vorname	Nachname	MenuID	ID	Menu	Preis
1	Jonas	Allenmann	3	3	Schnitzel mit Pommes	23
2	Elias	Wirth	1	1	Spaghetti	21
3	Jonas	Kopp	1	1	Spaghetti	21
4	Severin	Meier	NULL	NULL	NULL	NULL

Tabelle 9: LEFT JOIN von Person und Menü

Aufgabe 12:

a) Es gibt in einigen Datenbanksystemen auch noch den `RIGHT JOIN`-Befehl. Was könnte wohl seine Rolle sein? Beschreibe deine Idee in Worten.

Wie sähe das Resultat der Abfrage 6 aus, wenn der `LEFT JOIN`-Befehl durch `RIGHT JOIN` ersetzt würde? Schreibe die Kombinationen aus den Nachnamen und Menüs auf, die du in der Ausgabe erwartest.

b) Das Fehlen des `RIGHT JOIN`-Befehls ist keinerlei Einschränkung für die möglichen SQL-Abfragen (und natürlich deren Resultate). Weshalb ist das wohl so?

c) Ersetze in Abfrage 6 den `LEFT JOIN`-Befehl durch `RIGHT JOIN` und führe die Abfrage auf der Datenbank restaurant.db aus. Falls die Abfrage einen Fehler im Stil von „RIGHT and FULL OUTER JOINs are not currently supported“ generiert, unterstützt dein Datenbanksystem keine `RIGHT JOIN`s. Das ist aber nicht weiter schlimm: Nutze die Idee aus Aufgabe b), um den Befehl umzuschreiben und trotzdem ausführen zu können.

In manchen Datenbanksystemen wird der `LEFT JOIN`-Befehl mit `LEFT OUTER JOIN` bezeichnet, und analog `RIGHT JOIN`-Befehl mit `RIGHT OUTER JOIN`. Nicht alle Datenbanksysteme unterstützen den `RIGHT JOIN`-Befehl.

4.3 OUTER JOIN

Nachfolgend wird ein Beispiel für `OUTER JOIN` gegeben. Abfrage 7 liefert das Resultat in Tabelle 10.

```
1 SELECT *
2 FROM Personen
3         OUTER JOIN Speisekarte ON Personen.MenuID = Speisekarte.ID
```

Abfrage 7: OUTER JOIN von Person und Menü

ID	Vorname	Nachname	MenuID	ID	Menu	Preis
1	Jonas	Allenmann	3	3	Schnitzel mit Pommes	23
2	Elias	Wirth	1	1	Spaghetti	21
3	Jonas	Kopp	1	1	Spaghetti	21
4	Severin	Meier	NULL	NULL	NULL	NULL
NULL	NULL	NULL	NULL	2	Ravioli	19
NULL	NULL	NULL	NULL	4	Pilzrisotto	18
NULL	NULL	NULL	NULL	5	Cordon bleu	23

Tabelle 10: OUTER JOIN von Person und Menü

Aufgabe 13:

a) Beschreibe, was `OUTER JOIN` macht.

b) Wenn man die `ON`-Bedingung in Abfrage 7 weglässt, ist die Ausgabe 20 Zeilen lang. Welche Zeilen werden da wohl verbunden? Schreibe die ersten 7 Zeilen auf.
Mit welchem bereits kennen gelernten Prinzip kann dieses Verhalten erklärt werden?

In manchen Datenbanksystemen wird der `OUTER JOIN`-Befehl mit `FULL OUTER JOIN` bezeichnet. Nicht alle Datenbanksysteme unterstützen den `OUTER JOIN`-Befehl.

4.4 Übersicht über alle JOIN-Varianten

In Tabelle 11 wird für den `INNER JOIN`-Befehl gezeigt, welche Einträge aus den Tabellen verwendet werden: Nämlich diejenigen, bei welchen eine Zuordnung zweier Zeilen aus beiden Tabellen erfolgreich war. War die Zuordnung nur in einer der beteiligten Tabellen möglich, wird dieser Eintrag nicht ausgegeben. Interpretiert man die Tabellen als Mengen, bezeichnet `INNER JOIN` also die Schnittmenge dieser Mengen.

Aufgabe 14: Vervollständige die Übersichtstabelle, indem du die entsprechenden Bereiche markierst.

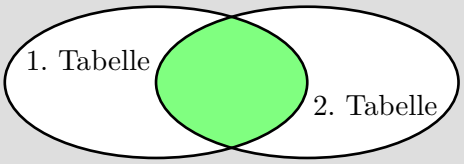
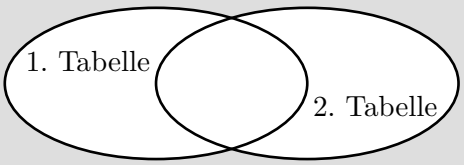
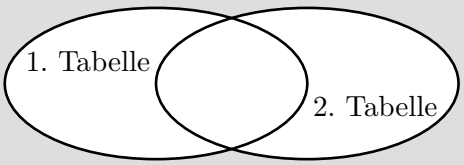
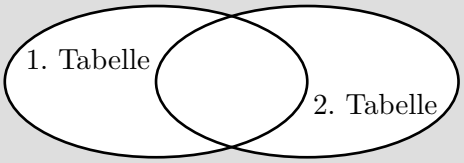
Befehl	Diagrammdarstellung
`JOIN` `INNER JOIN`	
`LEFT JOIN` `LEFT OUTER JOIN`	
`RIGHT JOIN` `RIGHT OUTER JOIN`	
`OUTER JOIN` `FULL OUTER JOIN`	

Tabelle 11: Übersicht über die verschiedenen JOIN-Varianten.

Die Option `NATURAL` kann jedem `JOIN`-Befehl vorangestellt werden.