

Aufgabe 4: Supermarkt

Eine Trennung der Dokumentation der Aufgabe in die Bereiche Lösungsidee, Programm-Dokumentation, Programm-Ablaufprotokoll und Programm-Text war bei dieser Aufgabe nicht möglich (zudem ja kein Programm zu schreiben war). Wir gehen erst auf das Modell der Datenbank ein und werden dann beschreiben, wie die Ausdrücke erstellt werden können und kommentieren zum Schluss noch den Geschäftsfall 4 und halten uns damit zumindest an die Reihenfolge der Teilaufgaben.

Zunächst haben wir ein Brainstorming gemacht uns überlegt welche Art von Objekten denn an den Geschäftsfällen beteiligt sind und haben schließlich auf dieser Basis ein Entity-Relationship-Modell entwickelt.

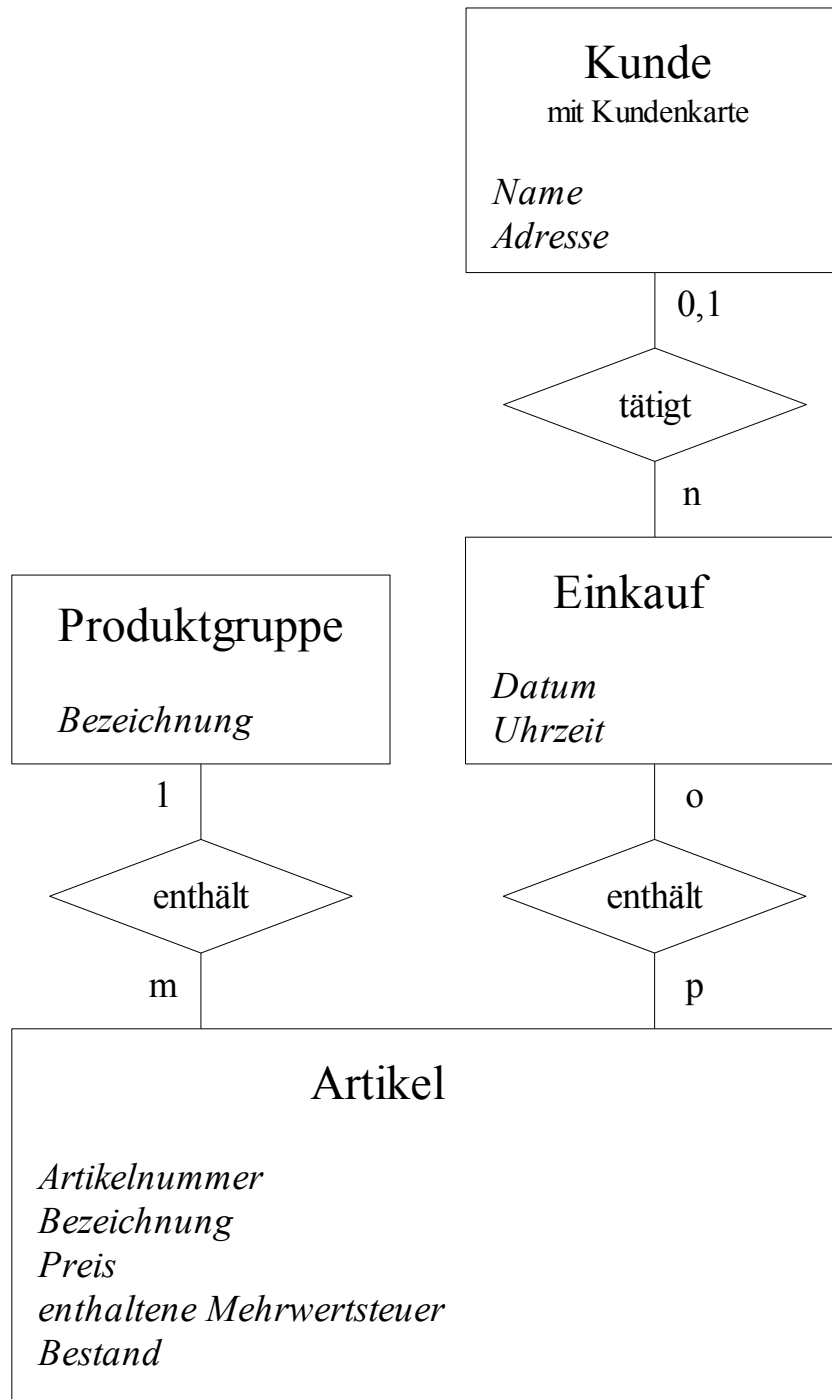
Hier ein Bild des ursprünglichen Schnabeltier-... ääh ER-Modells auf der Tafel unseres BWINF-Gruppenraumes.



Nein, natürlich müssen Sie die Daten nicht aus diesem Bild entnehmen, wir haben das für Sie in eine besser lesbare digitale Form umgewandelt, dazu bitte einmal dieses Blatt aus der Hand legen und Seite 2 in die Hand nehmen!

Entity-Relationship-Modell

Um die beteiligten Objekte und deren Beziehungen zueinander zu veranschaulichen, haben wir ein Entity-Relationship-Modell entwickelt. Die kursiv gedruckten Wörter sind dabei Attribute der in den Rechtecken dargestellten Entitäten.



Anmerkung

Vor allem für Geschäftsfall 1 („Ausdruck eines Kassensbons“) musste auf spezielle Felder in Tabellen geachtet werden: laut Wikipedia gelten rechtlich die Bestimmungen einer Rechnung und so *musst* der Kassensbon bestimmte Daten enthalten. Der ein oder andere von uns möge jetzt übrigens den Unterschied zwischen einer Quittung und einer Rechnung kennen!

Im Folgenden sind die Beziehungen aus dem ER-Modell nochmals einzeln erläutert:

Beziehung Produktgruppe – Artikel

Eine Produktgruppe kann mehrere Artikel enthalten,
ein Artikel kann jedoch nur in einer Produktgruppe auftauchen. (1 : m)

Bei unseren Überlegungen standen wir vor dem Problem, was wir mit Produkten machen die in zwei Produktgruppen gehören. Als Paradebeispiel hierfür möchten wir das Schnabeltier anführen. Es gehört zwar unumstritten in die Kategorie Feinkosttheke, wenn man jedoch feiner gliedert in Säugetiere und Vögel müsste das Schnabeltier in beiden Gruppen vertreten sein! Beim Erstellen des Einkaufs und Errechnen des Bestands des Ladens oder ähnlichen Operationen könnte das Problem entstehen, dass das Schnabeltier doppelt berechnet wird. Die Lösung des Problems besteht darin das Schnabeltier eindeutig in eine Gruppe zuzuordnen. Da man es aber weder den Säugetieren noch den Vögeln eindeutig zuordnen kann, müssen wir eine neue Gruppe namens "Schnabeltiere" eröffnen. Leider besitzt diese Gruppe nun nur ein Element. Dies erscheint zunächst nicht problematisch, jedoch stellt sich die Frage ob es in Zukunft durch den Fortschritt der Wissenschaft nicht mehr und mehr nicht eindeutig zuordenbare Produkte geben wird. Als Beispiele seien nur der Tintenfischpapagei oder die Rattentaube anzuführen. Dies würde zu einer Inflation der Gruppen führen, was einen schließlich am Sinn der Gruppenzuordnung komplett zweifeln lässt. Diese inflationäre Entwicklung nennen wir das "Schnabeltiermodell".

Beziehung Einkauf – Artikel

Ein Einkauf kann mehrere Artikel enthalten,
ein Artikel kann aber auch bei mehreren Einkäufen gekauft werden. (o : p)

Beziehung Kunde mit Kundenkarte – Einkauf

Ein Kunde, der seine Einkäufe mit einer Kundenkarte bezahlt, kann mehrere Einkäufe tätigen,
ein Einkauf muss aber nicht zwangsweise von einem Kunden mit einer Kundenkarte stammen.
(0,1 : n)

Für die Datenbankstruktur heißt das jetzt, dass alle Entities als Tabellen umgesetzt werden können. Für die Beziehung Einkauf – Artikel ist allerdings eine extra Verknüpfungstabelle nötig, da die o : p- Beziehung in einer relativen Datenbank nicht anders realisierbar ist.

Um die Beziehungen zwischen den einzelnen Tabellen herzustellen werden IDs als Primär-beziehungsweise Fremdschlüssel verwendet.

Im Folgenden sind die Tabellen gelistet und Überlegungen dokumentiert:

Tabelle produktgruppe
enthält alle Produktgruppen

<i>Feldname</i>	<i>Datentyp</i>	<i>Funktion</i>
<u>P_ID</u>	integer	Primärschlüssel; ermöglicht Verknüpfung zur Tabelle artikel
bezeichnung	char	Bezeichnung der Produktgruppe

Tabelle artikel
enthält alle Artikel

<i>Feldname</i>	<i>Datentyp</i>	<i>Funktion</i>
<u>A_ID</u>	integer	Artikelnummer als Primärschlüssel; diese kann zum Beispiel die zugehörige EAN-13 oder EAN-8 des jeweiligen Produktes sein; für Obst und Gemüse lassen sich sonstige, beliebig lange Nummern festlegen ermöglicht Verknüpfung zur Verknüpfungstabelle
<u>P_ID</u>	integer	Fremdschlüssel; ermöglicht Verknüpfung zur Tabelle produktgruppe
bezeichnung	char	Bezeichnung des Artikels
preis	decimal(5,2)	Preis in Euro und Cent oder einer sonstigen auf dem Dezimalsystem basierende Währung (vgl. Maya-Zahlen)
mwst	integer	im Preis enthaltene Mehrwertsteuer in Prozent
bestand	integer	Gesamtbestand eines Artikels im Geschäft

Alternativ könnte man den Bestand evtl. auch in den Bestand im Verkaufsbereich und den Bestand im Lager speichern, dann wüsste man zum Beispiel auch noch, wann man wieder Waren aus dem Lager holen muss, um den Bestand im Verkaufsbereich wieder auszubauen (Auffüllen neuer Artikel im Verkaufsbereich, da sonst kann der Kunde nichts mehr kaufen kann, weil nichts mehr da ist). Für den Geschäftsfall 2 könnte man die beiden Bestände addieren und hat dann wieder den Gesamtbestand. Unterschreitet dieser einen bestimmten Wert, wird der Artikel mit ausgegeben. Ansonsten könnte man sich ggf. auch nur auf den Lagerbestand beziehen. Bei dieser Methode müsste man aber laufend wissen, wie viele Artikel sich im Verkaufsbereich befinden, um wieder nachfüllen zu können – dadurch würden wohl aufwendige neue Technologien (RFID-Tags o.ä.) nötig oder man müsste bei jedem Mal Auffüllen den Bestand an die aktuellen Daten manuell angleichen.

Tabelle kunden

enthält alle Kunden, die eine Kundenkarte besitzen

<i>Feldname</i>	<i>Datentyp</i>	<i>Funktion</i>
<u>K_ID</u>	integer	Primärschlüssel; ermöglicht Verknüpfung zur Tabelle einkauf
name	char	
strasse	char	Straße, Hausnummer
plz	integer	Postleitzahl
ort	char	Wohnort

Eventuell ließen sich noch weitere Daten des Kunden speichern, die relevant sind. Dazu könnten Geschlecht und Anrede zählen. Des Weiteren müsste man ggf. eine Kundenkarten-ID/ oder einen -Schlüssel generieren und könnte damit den Kunden per Kundenkarte eindeutig identifizieren. Wir speichern auf der Kundenkarte der Einfachheit wegen nur die K_ID.

Tabelle einkauf

enthält Grunddaten zu einem Einkauf

<i>Feldname</i>	<i>Datentyp</i>	<i>Funktion</i>
<u>E_ID</u>	integer	Primärschlüssel
<u>K_ID</u>	integer	Fremdschlüssel; ermöglicht Verknüpfung zur Tabelle kunden, wenn es sich um einen Kunden handelt, der mit Kundenkarte zahlt, sonst K_ID=0
stamp	integer	enthält Datum und Uhrzeit als UNIX-Timestamp

Eventuell ließe sich hier über Datentyp des Datums/Uhrzeit streiten (Datetime, UNIX-Timestamp oder anderes Format). Des Weiteren wäre es natürlich möglich, die Uhrzeit am Anfang *und* am Ende des Einkaufsⁿ zu speichern (somit wäre auch die Dauer des Vorgangs bekannt).

Wir haben uns nur für den Anfang entschieden. Zudem könnte man noch einen Kassierer/Mitarbeiter mit diesem Einkauf verknüpfen; alternativ auch eine Kasse. Beides wäre auch denkbar.

ⁿ*Anmerkung:* damit ist nicht die Zeit des Einkaufs selbst gemeint (Kunde betritt Supermarkt, schlendert durch die Regale und packt Waren in seinen Einkaufswagen,...), sondern bezieht sich auf den Vorgang an der Kasse (angefangen vom Scan des ersten Produktes bis zum Abschluss durch den Zahlvorgang des Kunden).

Weiterhin wäre es möglich die Wagennummer des genutzten Einkaufswagen zu speichern. Diese wird zum Beispiel bei Kaufland/Handelshof eingegeben. Ob damit jetzt allerdings „nur“ geschaut wird, wie oft die Einkaufswagen genutzt werden ist fraglich – theoretisch könnten die Einkaufswagen mit einer bestimmten Technologie auch den Weg des Kunden durch den Laden aufzeichnen und so ein Profil der Person erstellen (siehe Teilaufgabe 3 bezüglich der gläsernen Kunden) – bevorzugte Waren, bevorzugte Gänge,...

Laut Wikipedia benötigen wir eine Bonnummer: „eine fortlaufende Nummer mit einer oder mehreren Zahlenreihen, die zur Identifizierung der Rechnung vom Rechnungsaussteller einmalig vergeben wird (Rechnungsnummer)“. Der Einfachheit wegen nehmen wir die E_ID und drucken diese dann später auf den Bon!

Tabelle einkauf_artikel

Verknüpfungstabelle; hier werden alle vom Kunden gekauften Artikel in Verbindung mit dem richtigen Einkauf abgespeichert

Feldname	Datentyp	Funktion
E_ID	integer	Fremdschlüssel; ermöglicht Verknüpfung zur Tabelle einkauf
A_ID	integer	Fremdschlüssel; ermöglicht Verknüpfung zur Tabelle artikel
menge	integer	enthält die Menge eines Artikels (an einer bestimmten Position) in einem Einkauf; es könnte ein Artikel auch mehrmals gescaant werden, dann erfolgt ein neuer Datensatz und keine Änderung der Menge

Denkbar wäre in dieser Tabelle noch ein Primärschlüssel, der die Position auf dem Kassensbon angibt. Das ist nicht für die Erstellung eines Kassensbons nötig, sondern (nur) wenn man später mal den gleichen Bon nochmal ausdrucken möchte.

Spezialfall Obst und Gemüse: Artikelnummern

Aus der Wikipedia:

„Für Supermärkte oder andere Einzelhändler steht ein spezieller Ländercode zur Verfügung, um zur ausschließlichen internen Verwendung die vor Ort abgewogenen Lebensmittel mit einem Barcode versehen zu können.

- ◆ 2xx (anstatt der Ländernummer)
- ◆ Artikelnummer (4 Stellen)
- ◆ Gewicht, Menge oder Preis (5 Stellen)

Dieser Code wird vor allem für Obst und Gemüse sowie Fleisch- und Wurstwaren verwendet. Außerdem benutzen verschiedene Lebensmittel-Discounter, wie z .B. ALDI, diese geschäftsinternen EAN in der verkürzten EAN-8-Form.“

Wir folgen dieser Idee. Die Waage druckt gemäß des Preises einen EAN-Barcode, der auf die Ware aufgeklebt wird. Den Preis für die einzelnen Artikel bezieht die Waage vom Datenbankserver, der sie gespeichert hat. Die Verarbeitung der verschiedenen Obst- und Gemüsesorten (Artikelbezeichnungen) kann wiederum durch die Kasse geschehen. Dazu könnte in der Datenbank die oben genannte 4-stellige Artikelnummer verwendet werden mit welcher auch Kasse und Waage arbeiten. Das System kann ohne Schwierigkeiten auf die Wurst-, Fleisch- und Käsetheke erweitert werden.

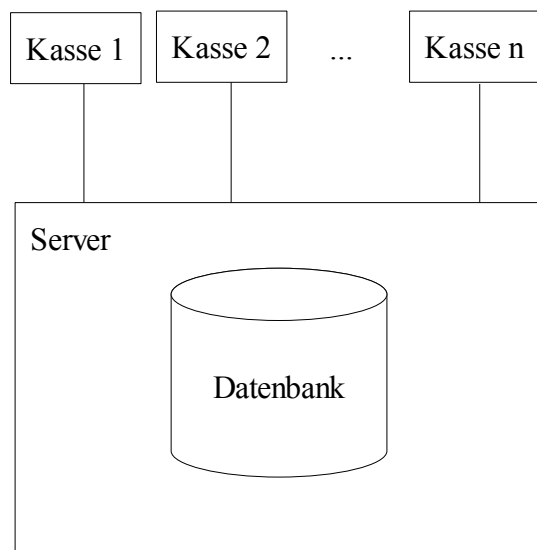


Preisschild für Obst mit einem EAN-Barcode (aus Wikipedia)

Das Kassensystem

Um die Geschäftsprozesse verwirklichen zu können brauchen wir noch etwas mehr als ein bloßes Datenbanklayout. Wir stellen einen Server auf: dieser dient uns als Datenbankserver und verwaltet unsere Daten, die wir behandeln möchten. Hier werden also alle Artikel, Produktgruppen, Kunden, Einkäufe und weitere Daten gespeichert. Außerdem werden sie von dort abgerufen und darauf verändert.

Da wir allerdings keine statischen Daten haben wollen und auch einen funktionierenden Bezahlvorgang sicherstellen wollen, brauchen wir noch unsere Kassen. Diese werden über Datenleitungen mit unserem Server verbunden.



Das System könnte noch durch eine Anbindung an einen zentralen Server (anderer Ort) erweitert werden. Von diesem Server könnten zum Beispiel für eine ganze Ladenkette die Preise für manche (oder sogar alle) Artikel zentral festgelegt werden.

Geschäftsfall 1

Der Einkauf eines Kunden mit dem Ausdruck des Kassensbons soll wie folgt bewältigt werden:

Aus der Sicht der Datenverarbeitung

- (1) Der Kunde legt seine Waren vom Einkaufswagen auf das Band der Kasse.
- (2) Der erste Artikel ist beim Kassierer angekommen. Dieser scannt den Barcode am Artikel oder gibt die eindeutige Artikelnummer ein sowie ggf. die Menge, die vom Artikel gekauft wird (wenn abweichend von 1). Mit dieser A_ID ist der Artikel also eindeutig identifiziert worden. Die Kasse selber weiß allerdings nichts mit dieser Nummer anzufangen.
- (3) Mit der Identifizierung des *ersten* Produktes wurde ein Einkauf begonnen. Die Kasse meldet das an den Server. Dieser generiert einen Eintrag in der Tabelle `einkauf` und sendet die automatisch erzeugt E_ID an die Kasse zurück. Die Kasse „weiß“ jetzt, für welchen Einkauf sie zuständig ist. (Eigentlich muss sie das nicht wissen – es genügt, wenn der Datenbankserver das weiß und den Einkauf der Kasse zuordnen kann!)
- (4) Daraufhin schickt die Kasse eine erneute Anfrage an den Server – diesmal soll der eingelese Artikel in die Liste der Artikel dieses Einkaufs aufgenommen werden. Dazu wird die A_ID an den Server übermittelt. Dieser prüft, ob der Artikel existiert. Ist das der Fall fügt er einen Eintrag in die Tabelle `einkauf_artikel` mit der vorher entsprechend übermittelten Menge ein. Der Bestand des Artikels wird in der Tabelle `artikel` entsprechend vermindert. Die Kasse erhält die Bezeichnung des Artikels und den Preis zurück.

Druck des Kassensbons

Es wird der Kopf des Kassensbons gedruckt. Zuerst kommt ggf. das Logo, dann folgen Name und Adresse unseres Unternehmens, wenn gewünscht noch Telefon- und Faxnummer.

```
Supermarkt Kundenfreund
Karl Koch Platz 23
33699 Bielefeld
Tel. 05205/547410
```

Druck des aktuell eingelesenen Artikels mit der zugehörigen Menge. Beträgt die Menge eines Artikels, der gekauft wird mehr als 1, so wird der Stückpreis mit der Menge multipliziert.

Artikel mit Menge $n = 1$

```
Rottaler Eistee LE      1,15
```

Artikel mit Menge $n > 1$ (hier $n = 2$):

```
Rottaler Eistee LE      2x 1,15
                        2,30
```

- (5) Der Kassierer zieht den nächsten Artikel über den Scanner (oder tippt die Artikelnummer ein, etc.). Auch dieser Artikel wird überprüft, ein Eintrag in der Tabelle einkauf_artikel erstellt und die Daten zurück an die Kasse gesandt.

Dieser Vorgang wiederholt sich so lange, bis alle Artikel des Einkäufers von der Kasse registriert worden sind und im Datenbankserver in Form von Bits & Bytes ihren Platz gefunden haben.

- (6) Ist der letzte Artikel bearbeitet worden, wird der Kauf abgeschlossen. Der zu zahlende Preis wird jetzt errechnet und ausgegeben (Display + Kassenbon).

- (7) Der Kassierer nennt den zu zahlenden Betrag woraufhin der Kunde seinen Einkauf bezahlt. Der Kassierer tippt den erhaltenen Betrag in die Kasse. Die Kasse berechnet das nötige Rückgeld welches der Kassierer an den Kunden aushändigt (natürlich nur bei Barzahlung).

Druck aller weiteren Artikel mit ihren zugehörigen Mengen.

Rottaler Eistee PE	3x 1,15	
		3,45
Alufolie		1,35
Joghurt natur		0,14
Pril Spülmittel		1,15
Joghurt natur		0,14
Joghurt natur		0,14
Knorr Salatkrönung		0,99
Maggi Bratensaft		3,35
Presse-Erzeugnis		5,20
Schnabeltier		6,00
Bananen (Konserve)		3,14

Der zu zahlende Betrag wird ausgedruckt:

zu zahlen	EUR	23,42
-----------	-----	-------

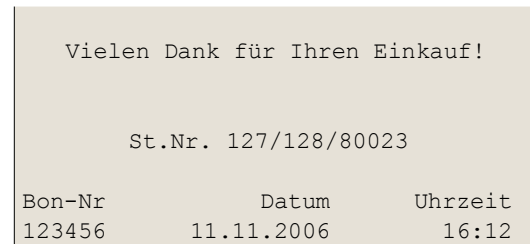
Erhaltenen Betrag und Rückgeld für den Kunden ausdrucken:

Bargeld	EUR	30,00
Rückgeld	EUR	6,58

Der Kassenbon enthält des Weiteren die im gezahlten Betrag enthaltene Mehrwertsteuer aufgeschlüsselt nach den Steuersätzen sowie den Netto-Betrag.

Mwst 16,0% aus 18,22		2,51
Nettobetrag		15,71
MwSt 7,0% aus 5,20		0,34
Nettobetrag		4,86

Zum Schluss kommen noch ein paar Infos auf den Kassenzettel: Datum und Uhrzeit des Einkaufs, die Bonnummer, Steuernummer oder Umsatzsteuer-Identifikationsnummer.



Vielen Dank für Ihren Einkauf!

St.Nr. 127/128/80023

Bon-Nr	Datum	Uhrzeit
123456	11.11.2006	16:12

Zusätzlich könnte der Bon jetzt noch enthalten an welcher Kasse der Einkauf getätigt wurde und wer der Kassierer war. Dazu müsste unser Datenbanklayout allerdings noch erweitert werden oder es wird einzig und alleine für diesen Ausdruck der Kasse entnommen („Login“ des Kassierers an der Kasse).

Weitere Möglichkeiten, den Kassenbon (größtenteils unnötig) in die Länge zu ziehen: Angabe der Ladenöffnungszeiten, weitere Servicrufnummern, Reklame für Artikel der Hausmarke, Hinweise auf die Kundenkarte, der Satz „Umtausch/Reklamation nur mit Kassenbon“, Leerzeilen,...

Zwischen Schritt (1) und (6) hätte jederzeit die Identifizierung des Kunden als Kunde der mit einer Kundenkarte bezahlt erfolgen können. Dazu könnte zum Beispiel die Kundenkarte, die mit einem bestimmten Barcode bedruckt ist, ausgelesen werden – die Interpretation erfolgt dann durch Kasse/Server. Alternativ ginge auch eine mit RFID-Tag ausgestattete Kundenkarte.

Der zum Einkauf gehörende Eintrag in der Tabelle einkauf wird editiert – die K_ID in der Tabelle wird von 0 zu der dem Kunden zugehörigen K_ID geändert und der Einkauf somit mit dem Kunden verknüpft.

Offen blieb die Frage, ob jeder Artikel einzeln zu kennzeichnen ist, welchen Mehrwertsteuersatz er enthält. Ist dies nötig, könnte das über ein bestimmtes Schema passieren. So könnten alle Artikel, die 16 % enthalten ungekennzeichnet bleiben, diejenigen, die nur 7 % enthalten etwa mit einem Stern (*). Andere Möglichkeiten sind natürlich auch denkbar (beispielsweise E oder V).

Außerdem haben wir uns nicht festgelegt, ob die Summe von der Kasse oder dem Server berechnet werden soll. Beides wäre denkbar. Wir tendieren dazu, dass die Kasse diese Aufgabe übernimmt, da der Server so nicht zusätzlich Rechenleistung abgeben muss.

„Zwischenwort“

Generell ist bei der Dokumentation der folgenden drei Geschäftsfälle zu beachten:

Wir haben uns mit unserem erstellten Datenbanklayout befasst und Abfragen für die jeweiligen Geschäftsfälle geschrieben. Diese Abfragen wurden in SQL verfasst. Sie sollten generell ANSI-konform sein; sind sie es nicht, so beziehen sie sich auf MySQL.

Bei vielen der hier aufgeführten Abfragen wurde ein `SELECT *` verwendet. Diese Art der Abfrage ist offensichtlich nicht performant. Es werden allerdings eigentlich *nie* alle Felder benötigt. In den finalen Abfragen haben wir uns aber meistens auch auf die wesentlichen Felder konzentriert. Die Nennung aller Felder hätte die Abfragen hier teilweise noch unübersichtlicher gemacht und wurden deshalb auf das `SELECT *` zusammengefasst.

Geschäftsfall 2

Für den Geschäftsfall 2 muss man sich die Anforderungen nochmal ansehen:
„Ausdruck einer Liste aller Artikel, deren Bestand einen Wert unterschreitet“

Die Frage, die sich bei der Lösung dieser Aufgabe stellte ist, was denn darunter verstanden wird, wenn der Bestand „einen Wert unterschreitet“: bezieht sich dieser Wert auf alle Artikel oder ist dieser Wert für alle Artikel unterschiedlich? Da wir diese Frage nicht eigenständig beantworten können, betrachten wir beide Sichtweisen:

Sichtweise 1:

Bestand aller Artikel unterschreitet *einen* bestimmten Wert (artikelunabhängig)

Es handelt sich um eine einfache Abfrage. In SQL formuliert könnte diese folgendermaßen aussehen:

```
SELECT * FROM artikel WHERE bestand < x
```

Wobei für x eine beliebige natürliche Zahl eingesetzt werden kann. Dieser Wert könnte entweder als Konstante in einem Programm abgespeichert werden (x ist festgelegt) oder als variable Eingabe erfolgen (x wird festgelegt, kann sich ändern).

Sichtweise 2:

Bestand aller Artikel unterschreitet einen bestimmten Wert (artikelabhängig)

Auch aus dieser Sichtweise handelt es sich noch um eine einfache Abfrage. Der Wert, der gespeichert wird, wobei ein Artikel dessen Bestand kleiner ist, ausgegeben werden soll, nennen wir Mindestbestand. Wir müssen jetzt für jeden Artikel einen eigenen Mindestbestand speichern – hierzu müssen wir die Tabellenstruktur der Tabelle artikel um das Feld min_bestand (beliebige Bezeichnung) erweitert werden (Datentyp: Integer).

Die Abfrage in SQL sieht jetzt so aus:

```
SELECT * FROM artikel WHERE bestand < min_bestand
```

Eventuell wäre für beide Sichtweisen eine Verknüpfung mit der Produktgruppen-Tabelle sinnvoll, zudem könnte man die Ergebnisse noch sortieren. Beispielsweise nach der Bezeichnung der Artikel (alphabetisch aufsteigend) und/oder nach Produktgruppen.

Geschäftsfall 3

Fest steht, dass für diesen Geschäftsfall die Daten eingeschränkt werden müssen – und zwar auf einen festen Zeitraum („innerhalb eines Monats“). Betroffen von dieser zeitlichen Eingrenzung ist die Tabelle einkauf. In dieser Tabelle ist im Feld stamp der Zeitpunkt des Einkaufs als UNIX-Timestamp angegeben.

Um jetzt alle Einkäufe in diesem Zeitraum auszugeben ist folgende SQL-Abfrage nötig:

```
SELECT * FROM einkauf
WHERE stamp >= erster_stamp_des_monats AND
      stamp <= letzter_stamp_des_monats
```

Oder auch gleichbedeutend:

```
SELECT * FROM einkauf
WHERE stamp BETWEEN erster_stamp_des_monats AND
      letzter_stamp_des_monats
```

Wir wollen allerdings nicht alle Einkäufe des Monats auflisten, sondern eine Hitliste der Artikel erstellen. Wir verknüpfen also die Tabelle einkauf mit der Tabelle einkauf_artikel.

```
SELECT * FROM einkauf_artikel AS v
JOIN einkauf AS e ON v.E_ID = e.E_ID
WHERE e.stamp BETWEEN erster_stamp_des_monats AND
      letzter_stamp_des_monats
```

Wir erhalten also jetzt alle Einkäufe des Monats mit ihren jeweiligen gekauften Artikeln ausgegeben. Die Informationen sind allerdings zum einen zu viel: wir müssen nicht wissen zu welchem Einkauf die Artikel gehören – wir wüssten nur sehr gerne welche Artikel wie oft gekauft wurden und ändern unsere Abfrage ab:

```
SELECT *, SUM(v.menge) AS anzahl FROM einkauf_artikel AS v
JOIN einkauf AS e ON v.E_ID = e.E_ID
WHERE e.stamp BETWEEN erster_stamp_des_monats AND
      letzter_stamp_des_monats
GROUP BY v.A_ID
```

Jetzt erhalten wir endlich die Anzahl wie oft der Artikel gekauft wurde. Zuerst gruppieren wir die eingekauften Artikel nämlich alle über die A_ID und fassen somit die Artikel zu einem Eintrag zusammen. Mit Hilfe der Aggregatfunktion SUM() bilden wir die Summe aller Mengen eines Produktes und erhalten so die Gesamtmenge für den Kauf eines bestimmten Produktes.

Die Daten sind jetzt zwar nicht mehr zu viel, aber immer noch zu wenig, denn aus den A_IDs kann der normal sterbliche Geschäftsleiter oder Mitarbeiter (sei er noch der größte Computernerd, -geek oder -freak) nicht viel anfangen – denn jetzt müsste er für jeden Artikel dessen Bezeichnung nachschauen und dann noch nach den Produktgruppen sortieren. Das wäre sehr umständlich (außerdem würde er direkten Zugang zum Datenbanklayout/-inhalt benötigen)... also erweitern wir unsere Abfrage und machen sie *noch* komplexer:

```
SELECT v.A_ID, SUM(v.menge) AS anzahl, a.bezeichnung,  
       p.bezeichnung  
FROM einkauf_artikel AS v  
JOIN einkauf AS e ON v.E_ID = e.E_ID  
JOIN artikel AS a ON v.A_ID = a.A_ID  
JOIN produktgruppe AS p ON a.P_ID = p.P_ID  
WHERE e.stamp BETWEEN erster_stamp_des_monats AND  
       letzter_stamp_des_monats  
GROUP BY v.A_ID  
ORDER BY p.P_ID, a.bezeichnung
```

Das geschulte Informatikerauge hat natürlich noch den Überblick. Trotzdem kurz zur Erklärung.
Wir haben unsere Abfrage um die beiden Zeilen

```
JOIN artikel AS a ON v.A_ID = a.A_ID  
JOIN produktgruppe AS p ON a.P_ID = p.P_ID
```

erweitert.

Diese stellen weitere Verknüpfen dar und zwar mit der Tabelle der Artikel und der Tabelle der Produktgruppen. Somit können wir über unseren Artikel noch mehr sagen – zum Beispiel können wir uns jetzt die Produktbezeichnung ausgeben lassen. Doch damit nicht genug – um nach Produktgruppen sortieren zu können müssen wir diese den Artikeln auch noch zurodnen. Die P_ID aus der Tabelle artikel würde uns schon reichen, wenn wir allerdings die passende Bezeichnung noch dazu wollen, dann verknüpfen wir eben noch mit der Produktgruppentabelle.

Des Weiteren könnte Ihnen die letzte Zeile aufgefallen sein:

```
ORDER BY p.P_ID, a.bezeichnung
```

Damit erhalten wir – unschwer zu erkennen – eine Sortierung nach der Produktgruppe (p.P_ID) und innerhalb der Produktgruppe nehmen wir noch eine (alphabetische) Sortierung nach Artikelbezeichnungen (a.bezeichnung) durch (optional, aber eventuell hilfreich für die Übersicht).

Die abgefragten Felder wurden auf das Wesentliche beschränkt. Sie könnte natürlich noch durch gewünschte Informationen (wie Preis oder Bestand) erweitert werden!

Und wie ein Wunder... die Abfrage funktioniert sogar!

Wie es allerdings mit der Performance aussieht konnten wir nicht testen. Dazu fehlten uns leider die großen Datenmengen und die nötige Zeit.

Geschäftsfall 4

Der Geschäftsfall 4 beschäftigt sich mit dem Ausdruck von Adressartikeln für den Versand eines Werbebriefes an Kunden, die mit Kundenkarte bezahlt und viel Wein gekauft haben.

Um Adressetiketten ausdrucken zu können müssen erst einmal die Kunden gefunden werden, die „viel“ Wein gekauft haben. Hier stellt sich die Frage, was denn unter „viel“ zu verstehen ist. Ist eine absolute Menge Wein (sprich Artikel in Einkäufen des Kunden) gemeint – oder relativ zu anderen Kunden? Ist der Zeitraum beschränkt oder offen?

Am besten schauen wir erstmal in die Tabelle `einkauf_artikel` und verknüpfen mit der Tabelle `einkauf` und beschränken dabei die Einkäufe auf diejenigen, die von einem Kunden, der mit Kundenkarte bezahlt hat stammen:

```
SELECT * FROM einkauf_artikel AS v
JOIN einkauf AS e ON v.E_ID = e.E_ID
WHERE e.K_ID <> 0
```

Allerdings haben wir jetzt mal wieder nur unsere `A_IDs` können aber weiterhin nicht viel damit anfangen. Wir verknüpfen mit der Tabelle `artikel`. Jetzt können wir uns nämlich wieder die Artikelbezeichnungen ausgeben lassen – außerdem können wir alle Produktgruppen, die nicht der Produktgruppe „Wein“ entsprechen dadurch aussortieren:

```
SELECT * FROM einkauf_artikel AS v
JOIN einkauf AS e ON v.E_ID = e.E_ID
JOIN artikel AS a ON v.A_ID = a.A_ID
WHERE e.K_ID <> 0 AND a.P_ID = produktgruppen_ID_von_Wein
```

Problematischerweise müssen wir für das Aussortieren allerdings die `P_ID` der Produktgruppe „Wein“ kennen. Tun wir das nicht müssen wir auch noch mit der Tabelle `produktgruppe` verknüpfen und anhand der Bezeichnung aussortieren:

```
SELECT * FROM einkauf_artikel AS v
JOIN einkauf AS e ON v.E_ID = e.E_ID
JOIN artikel AS a ON v.A_ID = a.A_ID
JOIN produktgruppe AS p ON a.P_ID = p.P_ID
WHERE e.K_ID <> 0 AND p.bezeichnung = "Wein"
```

Der Übersicht wegen nehmen wir an, dass man die `P_ID` von „Wein“ kennt. Da wir jetzt alle relevanten Daten haben – bis auf die Kundenadresse, die ja eines der wichtigsten Dinge darstellt, müssen wir noch mit der Kundentabelle verknüpfen:

```
SELECT * FROM einkauf_artikel AS v
JOIN einkauf AS e ON v.E_ID = e.E_ID
JOIN artikel AS a ON v.A_ID = a.A_ID
JOIN kunden AS k ON e.K_ID = k.K_ID
WHERE e.K_ID <> 0 AND a.P_ID = P_ID_von_Wein
```

Die Kunden sind jetzt allerdings mehrfahrgelistet – das ist nicht sehr performant und natürlich auch

nicht gewollt. Wir gruppieren jetzt anhand der Kunden (K_ID) und lesen die Anzahl der von ihnen gekauften Artikel der Produktgruppe „Wein“ neben ihrer Anschrift aus:

```
SELECT name, strasse, plz, ort, SUM(menge) AS anzahl
FROM einkauf_artikel AS v
JOIN einkauf AS e ON v.E_ID = e.E_ID
JOIN artikel AS a ON v.A_ID = a.A_ID
JOIN kunden AS k ON e.K_ID = k.K_ID
WHERE e.K_ID <> 0 AND a.P_ID = P_ID_von_Wein
GROUP BY e.K_ID
ORDER BY anzahl DESC
```

Die Ausgabe sortieren wir noch nach der Gesamtmenge von gekauftem Wein der jeweiligen Kunden von viel nach wenig.

Doch wie war das? Wir wollten ja eigentlich „nur“ die Kunden, die viel Wein gekauft haben! Dazu könnten wir beispielsweise erst ausrechnen, wie viel Wein von einem Kunden (der mit Kundenkarte bezahlt, oder allgemein allen Kunden) gekauft wird, die durchschnittlich gekaufte Menge Wein berechnen und all denjenigen anschreiben, die „überdurchschnittlich *viel*“ Wein gekauft haben. Ansonsten könnten wir einen konstanten Wert annehmen – es gibt unzählige Möglichkeiten, wie man hier vorgehen könnte!

Wie leistungsfähig/schnell diese Abfrage ist, ist leider auch unklar! Optimierungen wären sicherlich denkbar.

Bewertung des Geschäftsfall 4

Die Anforderungen für den Geschäftsfall 4 sind, mit der Beschränkung auf die Produktgruppe „Wein“, bereits sehr speziell gehalten. Es ließe sich ohne größere Schwierigkeiten ein umfangreicheres System entwickeln, das jede beliebige Produktgruppe einbezieht, oder das für jeden Kunden, der über eine Kundenkarte verfügt, einen Werbebrief erstellt, der auf dessen speziellen Vorlieben und Abneigungen zugeschnitten ist. Ob der Kunde solch einen Werbebrief überhaupt wünscht, oder ihn nur als störend empfinden würde (vgl. Spam), sei nun erstmal dahingestellt.

Durch Auslesen der Einkaufstabelle, die für jeden Kunden mit Kundenkarte existiert (andere seien für diesen Geschäftsfall vernachlässigbar) ließe sich durch die jeweilige Anzahl an gekauften Artikeln bzw. an gekauften Artikeln in einer Produktgruppe und die Gesamtzahl an Einkäufen ein Durchschnittswert ermitteln (beispielsweise 3 Flaschen Wein pro Einkauf) Nennen wir diese Variable kurz μ .

Soll nun ein Werbebrief für eine gewisse Produktgruppe verschickt werden, können alle Kunden bei denen μ für eben diese Gruppe einen gewissen Wert überschreitet ausgegeben werden. Zusammen mit den kundenspezifischen Adressdaten, lassen sich auch gleich Adressaufkleber ausdrucken, was den gesamten Vorgang so einfach wie möglich gestaltet.

Ein eher technisches Problem wirft die Einteilung der Artikel in die verschiedenen Produktgruppen auf. So müsste, um dem Kunden einen möglichst zutreffenden Werbebrief zu schicken, die

Einteilung möglichst detailliert sein. So wäre „Wein“ nur sehr grob, da nicht jeder Kunde der Weißwein mag, auch gerne Rotwein trinkt, geschweige denn Portwein oder Rosé etc. Um dem gerecht zu werden müssten Ressourcen eingesetzt werden, die kleinere Betriebe wesentlich überstrapazieren würden.

Diese Geschäftspraktik wirft allerdings einige Fragen bezüglich der moralischen Vertretbarkeit auf. Eine Sammlung von kundenspezifischen Daten, die deren Kaufverhalten so exakt dokumentieren wäre ein großer Einschnitt in die Privatsphäre des Kunden, ein weiterer Schritt zum „gläsernen Kunden“. Niemand kann garantieren, dass diese sensiblen Daten auch wirklich in den Händen des Geschäftes/Unternehmens bleiben. Allein schon der Gewinn, der durch den Verkauf solcher Daten an Dritte erzielt werden könnte, wäre Anreiz genug die Persönlichkeitsrechte der Kundschaft mit Füßen zu treten. Dies ist bereits bei so genannten Bonuspunktprogrammen zu beobachten. Des Weiteren könnte es auch zu einem Missbrauch durch Nachrichten- und Geheimdienste kommen, so könnte die CIA, oder der BND über halb-legale oder illegale Kanäle Zugriff auf diese Daten bekommen und diese zu ihrem „Krieg gegen den Terrorismus“ ge- bzw. missbrauchen. Schnell würde aus einem ahnungslosen Kunden der gerne Schnabeltiere aus dem arabischen Raum bezieht ein Terrorverdächtiger.

Zusammenfassend lässt sich sagen, dass es sich zwar hierbei um eine wohlgemeinte Idee handelt, die sowohl dem Geschäft als auch, in gewisser Weise, dem Kunden dienen soll, doch schlussendlich vor allem Dritten zugute kommt. In diesem Sinne ist die Empfehlung der Datenschutzbeauftragten der Schnabeltiere featuring Hr. Höllwerber powered by knoob.de: *Man solle es doch unterlassen Wein in gläserne Kunden zu füllen.*

