1 Vorstellung Kursbeispiel

Dieses Kapitel basiert auf den datenbanktheoretischen Grundlagen des Kapitel 1 und stellt die im Kurs verwendete Testdatenbank vor. Weiterhin soll grob skizziert werden, wie man in der Praxis von den Fachbereichsanforderungen hin zu einem Datenbankmodell kommt.

Fachliche Rahmenbedingungen

Die Schmidt-Imbissgruppe möchte ihre **Kunden** und **Bestellungen** in Zukunft in einer zentralen Datenbank erfassen, um die Daten vernünftig auswertbar zu machen. Derzeit hat die Firma 4 Filialen in *Hamburg, Kiel, Lübeck und Flensburg*. Es werden verschiedene Produkte verkauft, die sich zu Produktgruppen bündeln lassen. Dieser Fakt soll auch unbedingt abgebildet werden, damit bei steigender Produktzahl in Zukunft der Überblick nicht verloren geht. Jedes Produkt hat derzeit einen Preis.

Die Kunden können sich ihre Bestellung entweder selbst direkt in der Filiale abholen oder sich beliefern lassen. Dazu hat die Schmidt-Imbissgruppe derzeit *5 Fahrer, 2 Köche und 3 Verkäufer*, die die Bestellungen annehmen und bearbeiten. Die Mitarbeiter erhalten ein *monatliches Grundgehalt* und darüber hinaus eine *Provision*, die sich bei den Fahrern anhand der ausgelieferten Produkte bemisst. Bei den Verkäufern anhand der Bestellungen. Die Köche beziehen derzeit ausschließlich ein Festgehalt.

Weiterhin sollen folgende Attribute bei den Kunden auswertbar sein:

- *PLZ und ORT*
- *Geschlecht*
- *Geburtsdatum*

Ein Kunde kann darüber hinaus *mehrere Bestellungen* aufgeben. Jede dieser *Bestellung wird von einem Verkäufer bearbeitet*.

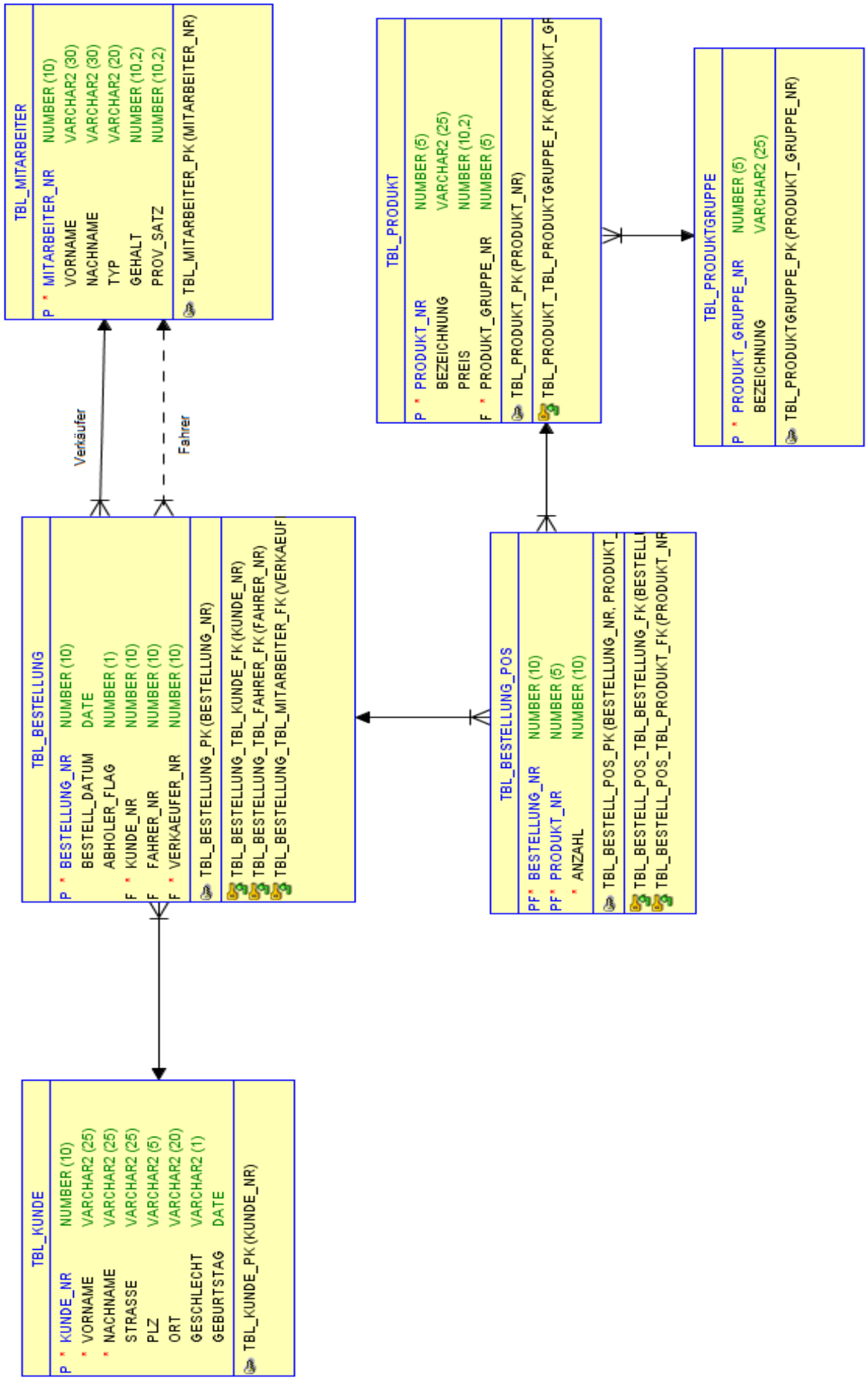

Abb.1: Kurs-Datenmodell

2 Analytische Funktionen

2.1 Grundlegende Struktur

Dieses Kapitel behandelt die sog. analytischen Funktionen, die es ermöglichen dynamische Aggregate zu berechnen. Wir nehmen ein einfaches Beispiel. Und zwar möchten wir eine Liste erzeugen, die für das Jahr 2014 die Umsätze pro Produkt darstellt. Darüber hinaus möchten wir den Anteil am Gesamtumsatz berechnen. Mit den bisherigen Mitteln müssten wir zwei Unterabfragen entwickeln, eine für die Produktumsätze und eine für den Gesamtumsatz, da es sich jeweils um unterschiedliche Gruppierungsebenen handelt. Diese beiden Unterabfragen müssen dann mittels Kreuzprodukt miteinander verbunden werden und in der Hauptabfrage kann dann der Umsatzanteil berechnet werden. Das Ganze könnte dann folgendermaßen aussehen:

```
SELECT  p.PRODUKT_NR,
        p.BEZEICHNUNG,
        ges.UMSATZ_GES,
        SUM(bp.ANZAHL*p.PREIS) AS UMSATZ_PROD,
        round(SUM(bp.ANZAHL*p.PREIS)/ges.UMSATZ_GES*100,1) AS ANTEIL

FROM TBL_PRODUKT p
 JOIN TBL_BESTELLUNG_POS bp ON p.PRODUKT_NR=bp.PRODUKT_NR
 CROSS JOIN (
        SELECT SUM(bp.ANZAHL*p.PREIS) AS UMSATZ_GES
         FROM TBL_PRODUKT p
            JOIN TBL_BESTELLUNG_POS bp ON p.PRODUKT_NR=bp.PRODUKT_NR
        ) ges

GROUP BY p.PRODUKT_NR,
            p.BEZEICHNUNG,
            ges.UMSATZ_GES

ORDER BY ANTEIL DESC
```

Ergebnis:

	Nr	Produkt	Umsatz	Umsatz Ges.	Anteil
1	1	Rumpsteak	4546,15	20097,45	22,6
2	2	Grillteller	3303,95	20097,45	16,4
3	6	Pizza Spezial	1612,4	20097,45	8
4	7	Pizza Vital	1363	20097,45	6,8
…					

Mit Hilfe der analytischen Funktionen kann man das Ganze etwas einfacher gestalten. Und zwar folgendermaßen:

```
SELECT  DISTINCT
      p.PRODUKT_NR,
      p.BEZEICHNUNG,

      SUM(bp.ANZAHL*p.PREIS) OVER( PARTITION BY p.PRODUKT_NR) AS UMSATZ_PRD,
      SUM(bp.ANZAHL*p.PREIS) OVER() AS UMSATZ_GES,

      round(SUM(bp.ANZAHL*p.PREIS) OVER( PARTITION BY p.PRODUKT_NR)/
            SUM(bp.ANZAHL*p.PREIS) OVER()*100,1)  AS UMSATZ_ANTEIL

FROM TBL_PRODUKT p
     JOIN TBL_BESTELLUNG_POS bp ON p.PRODUKT_NR=bp.PRODUKT_NR

ORDER BY UMSATZ_ANTEIL DESC
```

Ergebnis:

	Nr	Produkt	Umsatz	Umsatz Ges.	Anteil
1	1	Rumpsteak	4546,15	20097,45	22,6
2	2	Grillteller	3303,95	20097,45	16,4
3	6	Pizza Spezial	1612,4	20097,45	8
4	7	Pizza Vital	1363	20097,45	6,8

…

Die Ergebnisse beider Abfragen sind identisch. Ansonsten ist die zweite Abfrage deutlich kompakter und übersichtlicher. Wenn man sich die Ausführungspläne in der Datenbank anguckt, sieht man, dass bei dem zweiten Statement nur einmal auf beide Tabellen zugegriffen werden muss. Bei dem ersten Statement jeweils zweimal (1x bei der Berechnung der Gesamt-Summe und 1x bei der Berechnung der Produkt-Summen). Abfragen mit analytischen Funktionen können also deutlich performanter sein. Kommen wir nun zu den einzelnen Elementen. Die Grundlegende Struktur für eine analytische Funktion sieht folgendermaßen aus:

<Funktion> **OVER**(**PARTITIONY BY** <Spalte1>, <Spalte2>, ….)

Es können diverse Funktionen verwendet werden. Zunächst einmal können sämtliche Aggregatsfunktionen ***(SUM, Count, MIN, MAX, …)*** auch als analytische Funktion verwendet werden. Darüber hinaus gibt es noch zahlreiche weitere Funktionen, die wir im weiteren Verlauf noch kennen lernen werden. Mit Hilfe des Schlüsselworts ***OVER(….)*** wird festgelegt, dass es sich um eine analytische Funktion handeln soll. Innerhalb der Klammern kann dann noch das Schlüsselwort ***PARTITION BY*** gefolgt von 1..n Spalten angegeben werden. Dadurch wird für diese Berechnung die Gruppierungsebene definiert. Wird es weggelassen, berechnet man sozusagen eine Gesamtsumme über alle Datensätze.

Bezogen auf das obige Beispiel bedeutet das, dass wir zwei analytische Funktionen definiert haben. Eine, um den Gesamtumsatz zu berechnen. Dort fehlt innerhalb der OVER Klausel die Definition einer Gruppierung. Die zweite Berechnung arbeitet mit einer Gruppierung nach der PRODUKT_NR. Somit wird jeweils eine Summe pro Produkt-Nr berechnet. Mit PARTITION BY wird zwar die Gruppierung für eine bestimmte Berechnung definiert, nicht jedoch für die gesamte Abfrage. In dem obigen Beispiel wurde das Schlüsselwort DISTINCT verwendet, damit man pro Produkt nur noch eine Zeile erhält. Es wurden somit die Duplikate ausgeschlossen.

Man kann mit den analytischen Funktionen auch weiterrechnen, wie das Beispiel der Berechnung des Umsatzanteils demonstriert. Dazu verwendet man einfach die Ausdrücke der analytischen Funktionen in den weitergehenden Berechnungen. Man kann in der gleichen Abfrage nicht auf den Spalten-Alias zugreifen, um die Berechnungen übersichtlicher zu gestalten.

Übung 1: Erstellen Sie eine Liste der Monate in 2013 und berechnen Sie den Monatsumsatz mit Hilfe einer analytischen Funktion.

Übung 2: Erstellen Sie eine Abfrage, die den Produktumsatz für jedes Produkt im 2. Quartal 2013 berechnet und den zugehörigen Produktgruppen-Umsatz. Berechnen Sie den Anteil des Produktumsatzes am Produktgruppenumsatz.

Übung 3: Erstellen Sie eine Liste der Monate für das 1. Quartal 2013. Stellen Sie pro Monat den Umsatz dar. Dazu den Anteil am Quartalsumsatz und den größten und kleinsten Tagesumsatz innerhalb des jeweiligen Monats.

Innerhalb der OVER-Klausel kann noch die ***ORDER BY*** Klausel angegeben werden. Diese ist wichtig, wenn es sich um eine analytische Funktion handelt, bei der die Reihenfolge entscheidend ist. Das soll einmal an Beispiel der ***ROW_NUMBER*** Funktion dargestellt werden. Diese liefert die Nummer der Zeile zurück. Je nachdem wie ich die Zeilen sortiere, erhalten die Zeilen unterschiedliche Zeilennummern. Dazu das folgende Beispiel:

```
SELECT  DISTINCT
      p.PRODUKT_NR,
      p.BEZEICHNUNG,

      Row_Number() Over( ORDER BY PRODUKT_NR DESC) AS RowNumber
FROM TBL_PRODUKT p
ORDER BY PRODUKT_NR ASC;
```

Ergebnis:

	Nr	Bezeichnung	RowNumber
1	1	Rumpsteak	16
2	2	Grillteller	15
3	3	Pizza Salami	14
4	4	Pizza Hawai	13
5	5	Pizza Thunfisch	12
…			

Wie man sieht, wurde die Liste nach Produkt-Nr aufsteigend sortiert. Die Berechnung der Row-Number wurde dann aber auf Grundlage einer Sortierung nach Produkt-Nr absteigend durchgeführt.