ppedv

TSQL – Labs - Lösungen

Übungen für TSQL Einsteiger
Kurs SQL Abfragesprache und Datenbankdesign

TSQL – Labs – Lösungen

Inhalt

Modul 1 – einfache Abfragen

Lösung zu Übung 1.1

```
SELECT      100 AS Zahl
     , 'Donaudampfschifffahrtsgesellschaft' AS Text
     , 400/2 AS Rechnung
```

Hinweise:

- Zahlen werden ohne Hochkomma geschrieben
- Text (Strings) werden unter Hochkomma gesetzt
- AS bedeutet, wir wollen diese Spalte in der Textausgabe mit folgender Überschrift versehen
- Die Spaltenüberschriften brauchen wir nicht unter Hochkommata zu setzen.
- Besteht eine gewünschte Spaltenüberschrift aus zwei oder mehr Wörtern, setzen wir sie in eckige Klammern, z.B.: [durchschnittliche Frachtkosten].

Lösung zu Übung 1.2

```
SELECT      CustomerID AS KundenID
          , CompanyName AS Firmenname
          , ContactName AS Kontaktperson
          , Phone AS Telefonnummer
FROM Customers
```

Hinweis: Wir müssen hier kein „AS“ verwenden; wenn wir es weglassen, bekommen wir die ursprünglichen (englischen) Spaltenüberschriften CustomerID, CompanyName, ContactName und Phone.

Lösung zu Übung 1.3

```
SELECT      ProductID AS ProduktID
     , ProductName AS Produktname
     , UnitPrice AS Stückpreis
FROM Products
```

Lösung zu Übung 1.4

```
SELECT       Freight AS Nettofrachtkosten
      , Freight*1.19 AS Bruttofrachtkosten
      , Freight*0.19 AS MwSt
FROM Orders
```

Modul 2 – String- und Datumsfunktionen

Lösung zu Übung 2.1

… hier gibt es gleich mehrere Möglichkeiten, z.B.:

```
-- Möglichkeit 1:
SELECT STUFF('1234567890', LEN('1234567890')-2, 3, 'xxx')
-- funktioniert unabhängig von der Länge!


-- Möglichkeit 2:
SELECT CONCAT(LEFT('1234567890', 7), 'xxx')
-- Achtung, funktioniert nur mit fixer Länge!

-- Möglichkeit 3:
SELECT CONCAT(LEFT('1234567890', LEN('1234567890')-3), 'xxx')
-- wie Möglichkeit 2, funktioniert aber für variable Länge


-- Möglichkeit 4:
SELECT REVERSE(STUFF(REVERSE('1234567890'), 1, 3, 'xxx'))
-- funktioniert auch bei variabler Länge!

-- zum Testen mit den Daten aus der Northwind-DB:
SELECT REVERSE(STUFF(REVERSE(Phone), 1, 3, 'xxx')) AS SecretNumber
           , Phone
           , CompanyName
FROM Customers
```

Lösung zu Übung 2.2

Auch hier gibt es mehrere Möglichkeiten, zwei davon hier:

```
-- Möglichkeit 1:
SELECT CONCAT(REPLICATE('x', LEN(Phone)-3), RIGHT(Phone, 3)) AS PhoneX
            , Phone
FROM Customers


-- oder mit Substring:
-- Möglichkeit 2:
SELECT CONCAT(REPLICATE('x', LEN(Phone)-3), SUBSTRING(Phone, LEN(Phone)-2, 3)) AS PhoneX
            , Phone
FROM Customers
```

Langsam aufgebaut kommen wir so zu dem Ergebnis:

```
-- langsamer Aufbau der verschachtelten Serverfunktionen (mit Testzahlen):
SELECT STUFF('1234567890', 1, 7, 'xxxxxxx')

-- wie ging noch mal REPLICATE:
SELECT REPLICATE('?', 3) -- ergibt: ???


-- in REPLICATE einsetzen:
SELECT STUFF('1234567890', 1, 7, REPLICATE('x', 7))


-- wie viele Zeichen sind es insgesamt:
SELECT LEN('1234567890') -- 10

-- wie kommen wir auf 7? Indem wir von LEN 3 abziehen:
SELECT LEN('1234567890') - 3


-- zusammenbauen:
-- überall, wo 7 steht, Berechnung einsetzen:
SELECT STUFF('1234567890', 1, (LEN('1234567890') - 3), REPLICATE('x',
(LEN('1234567890') - 3)))



-- mit DB:
SELECT   STUFF(Phone, 1, (LEN(Phone) - 3), REPLICATE('x', (LEN(Phone) - 3)))
       , Phone
FROM Customers
```

Lösung zu Übung 2.3

```
SELECT LEN('Wolfgang Amadeus Mozart') - CHARINDEX(' ',
REVERSE('Wolfgang Amadeus Mozart')) +1
```

Langsam aufgeschlüsselt:

```
-- Zeichenfolge umdrehen, denn CHARINDEX findet das erste Vorkommen
des gesuchten Zeichens:
SELECT REVERSE('Wolfgang Amadeus Mozart') -- trazoM suedamA gnagfloW


-- letztes Leerzeichen = 1. Leerzeichen im umgedrehten Text
SELECT CHARINDEX(' ', 'trazoM suedamA gnagfloW') -- 7

-- wie viele Zeichen sind es insgesamt:
SELECT LEN('Wolfgang Amadeus Mozart') -- 23

-- 23 - 7 = 16 (falsche Stelle wegen REVERSE)

-- 23 - 7 + 1 = richtige Stelle

-- einsetzen in diese Werte:

SELECT LEN('Wolfgang Amadeus Mozart') - CHARINDEX(' ',
REVERSE('Wolfgang Amadeus Mozart')) + 1 - 17
```

Bei „Wolfgang Amadeus Mozart“ kommt 17, bei „Georg Friedrich Händel“ 16 heraus.

Hinweis:

Achtung! In SQL beginnen wir bei 1 zu zählen! Programmierer müssen hier umdenken! (NICHT bei 0 zu zählen beginnen). Das „W“ in Wolfgang befindet sich an der 1. (ersten), NICHT an der 0. („nullten“) Stelle wie in einer Programmiersprache.

Lösung zu Übung 2.4

Eigener Geburtstag:

```
SELECT DATENAME(DW, '1981-04-22') AS [Leos Birthday]
```

Geburtstage Angestellte:

```
SELECT    DATENAME(DW, BirthDate) AS [Emp Birthday]
          , CONCAT(FirstName, ' ', LastName)
FROM Employees
```

Hinweise:

- DW steht für day of the week; damit kann man den Namen des Wochentages als Text ausgeben.
- Sie können händisch Ihr Geburtsdatum angeben.
- Achtung Schreibweise: Wie das Datum angegeben werden muss, ist systemabhängig! Möglicherweise müssen Sie das Datum in deutscher Schreibweise (dd.MM.yyyy) angeben. Dieses Problem haben wir nicht, wenn die Daten aus der DB kommen.

Lösung zu Übung 2.5

```
SELECT DATEDIFF(yyyy, '1977', GETDATE()) AS Movie
```

Hinweise:

- Sie können das Datum händisch eingeben.
- In diesem Fall ist nur die Differenz zwischen den Jahren gefragt; daher können sie Monat und Tag weglassen.
- DATEDIFF in Kombination mit yyyy erstellt nur eine Differenz zwischen den Jahren (gibt uns also kein exaktes Alter an)!

Lösung zu Übung 2.6

```
SELECT       FORMAT(BirthDate, 'd', 'de-de') AS BirthDate
           , FLOOR(DATEDIFF(DD, BirthDate, GETDATE())/365.25) AS EmpAge
           , CONCAT(FirstName, ' ', LastName) AS EmpName
FROM Employees
```

Hinweise:

- In der Möglichkeit oben wurde der Tag als Datumsintervall verwendet und die resultierenden Tage durch 365.25 (Schaltjahre!) dividiert. Damit keine Nachkommastellen herauskommen, wird jeder noch nicht vollständige Tag mit der Funktion FLOOR auf den nächstkleineren ganzzahligen Wert abgerundet.
- Braucht man es noch präziser, könnte man auch mit Stunden arbeiten (und wieder durch Stunden pro Jahr durchdividieren).

Lösung zu Übung 2.7

```
-- damit wir sehen, dass tatsächlich etwas passiert, machen wir erst
ein Experiment und machen den 1. Buchstaben klein, die anderen groß
-- dann "richtig"

SELECT      CONCAT(LOWER(LEFT(FirstName, 1)), UPPER(RIGHT(FirstName,
LEN(FirstName)-1))) AS FirstName
          , CONCAT(UPPER(LEFT(LastName, 1)), LOWER(RIGHT(LastName,
LEN(LastName)-1))) AS LastName
FROM Employees
```

Lösung zu Übung 2.8

```
SELECT      LEFT(ContactName, CHARINDEX(' ', ContactName)-1) AS
FirstName
          , RIGHT(ContactName, CHARINDEX(' ',
REVERSE(ContactName))-1) AS LastName
--        , RIGHT(ContactName, (LEN(ContactName) - CHARINDEX(' ',
ContactName))) AS LastName -- andere Möglichkeit
FROM Customers
```

Hinweise:

- In dieser DB gibt es keine Doppelnamen oder mehrere Vornamen, sonst würde unser Beispiel nicht funktionieren!
- Auch, wenn in einer Zeile Vor- und Nachname vertauscht wären, würde unser Ergebnis nicht mehr stimmen!

Modul 03 – WHERE-Klausel

Lösung zu Übung 3.1

```
SELECT      CustomerID
          , CompanyName
          , ContactName
          , Phone
FROM Customers
WHERE Country = 'France'
```

Hinweis: Die genauen Spalten sind für unsere Übung im Moment nicht relevant; sollten Sie mehr oder weniger als oben gewählt haben, ist das auch in Ordnung.

Lösung zu Übung 3.2

```
SELECT      CustomerID
          , CompanyName
          , ContactName
          , Phone
FROM Customers
WHERE Country = 'Argentina' AND City = 'Buenos Aires'
```

Hinweis:

Ganz ähnlich wie bei Übung 3.1, allerdings müssen wir hier ein AND verwenden. Beide Bedingungen (Argentina und Buenos Aires) *müssen* zutreffen. *Gäbe* es ein „Buenos Aires“, das nicht in Argentinien liegt, dann würde das von dieser Abfrage nicht erfasst werden.

Lösung zu Übung 3.3

```
SELECT      CustomerID
          , CompanyName
          , ContactName
          , Phone
FROM Customers
WHERE Country = 'Spain' OR Country = 'Portugal'
```

Oder für Fortgeschrittene:

```
SELECT       CustomerID
           , CompanyName
           , ContactName
           , Phone
FROM Customers
WHERE Country IN('Spain', 'Portugal')
```

Hinweise:

- Ganz ähnlich wie bei den Übungen 3.1 und 3.2, allerdings brauchen wir hier ein OR! Würden wir hier ein AND verwenden, müsste der Kunde zugleich in Portugal und Spanien ansässig sein.
- Fragen wir im WHERE Bedingungen für dieselbe Spalte mehrfach ab (soll also entweder A oder B oder C… zutreffen), dann können wir das auch mit einem **IN**('A', 'B', 'C') abfragen.

Lösung zu Übung 3.4

```
SELECT *
FROM Products
WHERE UnitsInStock > 100
```

Hinweis: SELECT * wählt alle Spalten der Tabelle aus. Das sollte in der Realität aber nicht verwendet werden; es könnte sich z.B. etwas an der Tabelle ändern, und dann funktioniert die Abfrage nicht mehr.

Auch wenn wir alle Spalten einer Tabelle ausgeben möchten, sollten sie einzeln angeführt werden. Strenggenommen wäre Folgendes korrekt:

```
SELECT       ProductID
           , ProductName
           , SupplierID
           , CategoryID
           , QuantityPerUnit
           , UnitPrice
           , UnitsInStock
           , UnitsOnOrder
           , ReorderLevel
           , Discontinued
FROM Products
WHERE UnitsInStock > 100
```

Lösung zu Übung 3.5

```
SELECT       ProductID
           , ProductName
           , UnitPrice
FROM Products
WHERE ProductID BETWEEN 10 AND 15
```

Hinweise:

- Bereiche (Wert liegt zwischen x und y) können wir auch mit BETWEEN abdecken.
- BETWEEN macht ein >= und <=, d.h. die angegebenen Grenzwerte sind noch enthalten.
- Wir könnten diese Übung auch mit >= und <= lösen, auch das wäre korrekt:

```
SELECT       ProductID
           , ProductName
           , UnitPrice
FROM Products
WHERE ProductID >= 10 AND ProductID <= 15
```

Lösung zu Übung 3.6

```
SELECT       ProductID
           , ProductName
           , UnitPrice
FROM Products
WHERE SupplierID IN(2,7,15)
```

Hinweise:

- Fragen wir im WHERE Bedingungen für dieselbe Spalte mehrfach ab (soll also entweder A oder B oder C… zutreffen), dann können wir das auch mit einem **IN**('A', 'B', 'C') abfragen.
- Da es sich bei der SupplierID um den Datentyp int handelt, und nicht um Text, setzen wir 2, 7 und 15 NICHT unter Hochkommata.
- Diese Abfrage könnte auch mit OR gelöst werden, dann haben wir aber ein bisschen mehr Tipparbeit:

```
SELECT       ProductID
           , ProductName
           , UnitPrice
FROM Products
WHERE SupplierID = 2 OR SupplierID = 7 OR SupplierID = 15
```

Lösung zu Übung 3.7

```
SELECT       ProductID
           , ProductName
           , UnitPrice
           , UnitsInStock
FROM Products
WHERE SupplierID IN(5, 10, 15) AND UnitsInStock > 10 AND UnitPrice <
100
```

Hinweise:

- Wir können mehrere Bedingungen aus mehreren, nicht unbedingt zusammengehörigen Spalten in einem WHERE abfragen. Werden diese mit einem AND verknüpft, müssen alle Bedingungen zutreffen.
- Die SupplierIDs 5, 10 und 15 können wir mit einem IN() zusammenfassen.

Lösung zu Übung 3.8

```
SELECT   OrderID
       , FORMAT(ShippedDate, 'd', 'de-de') AS Lieferdatum
       , FORMAT(RequiredDate, 'd', 'de-de') AS Wunschtermin
       , DATEDIFF(dd, RequiredDate, ShippedDate) AS Lieferverzögerung
FROM Orders
WHERE DATEDIFF(dd, RequiredDate, ShippedDate) IS NOT NULL

-- statt FORMAT können wir auch CONVERT verwenden, um das Datum in eine leserliche
Form zu bringen:
SELECT   OrderID
             , CONVERT(varchar, ShippedDate, 104) AS Lieferdatum
             , CONVERT(varchar, RequiredDate, 104) AS Wunschtermin
             , DATEDIFF(dd, RequiredDate, ShippedDate) AS Lieferverzögerung
FROM Orders
WHERE DATEDIFF(dd, RequiredDate, ShippedDate) IS NOT NULL
```

Hinweise:

- FORMAT verfügt über einen sogenannten Culture-Parameter. Den kennen wir auch von Webseiten, die in mehreren Sprachen verfügbar sind („de-de“, „en-gb“ usw.). Die Liste der Culture-Parameter gibt es in der Microsoft-Dokumentation: https://docs.microsoft.com/de-de/bingmaps/rest-services/common-parameters-and-types/supported-culture-codes
- CONVERT hingegen verwendet einen sogenannten Style-Parameter; hier geben wir über einen Zahlencode ein, in welcher Form die Ausgabe des Datums erfolgen soll. Eine Übersicht über die möglichen Style-Parameter gibt es in der Microsoft-Dokumentation: https://docs.microsoft.com/en-us/sql/t-sql/functions/cast-and-convert-transact-sql?view=sql-server-2017#date-and-time-styles
- In DATEDIFF verwenden wir in den allermeisten Fällen das ältere Datum als Startdatum und das jüngere/neuere Datum als Enddatum. Manchmal macht die Fragestellung aber einen großen Unterschied. In diesem Beispiel war die *Lieferverzögerung* gefragt. Hätte die Fragestellung aber

gelautet: *„Geben Sie an, wie viele Tage noch Zeit sind, um fristgerecht zu liefern“,* hätten wir Start- und Enddatum in der Abfrage genau umkehren müssen!

Modul 04 – Wildcards

Lösung zu Übung 4.1

```
SELECT       ProductID
           , ProductName
FROM Products
WHERE ProductName LIKE 'L%'
```

Hinweise:

- Die Suche ist nicht case-sensitiv. Gerade beim „L“ ist es aber hilfreich, den Großbuchstaben zwecks leichterer Lesbarkeit in der Abfrage zu verwenden.
- Falls Sie mehr als zwei Spalten ausgewählt haben, ist das in Ordnung (es waren keine speziellen Spalten gefragt).

Lösung zu Übung 4.2

```
SELECT       ProductID
           , ProductName
FROM Products
WHERE ProductName LIKE '%ost%'
```

Hinweis: Suchen wir nach Zeichen oder Zeichenfolgen innerhalb von Text, setzen wir ein %-Zeichen davor und dahinter.

Achtung:

Diese Art der Suche ist potenziell fehleranfällig. Falls es unsere Absicht war, nach „Osten“ zu suchen, so haben wir zwar den „Nord-**Ost** Matjeshering“ gefunden; aber auch z.B. in „Rhönbräu Kl**ost**erbier“ und „Thüringer R**ost**bratwurst“ findet sich die Zeichenfolge „ost“! Wenn wir hier nicht vorsichtig und wohlüberlegt suchen, laufen wir Gefahr, eine Ausgabe zu erzeugen, die nicht dem eigentlichen gewünschten Ergebnis entspricht.

Lösung zu Übung 4.3

```
-- Möglichkeit 1
SELECT       ProductID
           , ProductName
FROM Products
WHERE ProductName LIKE '[d-l]%' AND (ProductName LIKE '%[a-d]' OR
ProductName LIKE '%[m-o]')
-- ACHTUNG KLAMMERN! (Siehe Hinweise)


-- Möglichkeit 2 (kürzer)
SELECT       ProductID
           , ProductName
FROM Products
WHERE ProductName LIKE '[d-l]%' AND ProductName LIKE '%[a-d | m-o]'


-- Möglichkeit 3 (kürzeste Möglichkeit, alles zusammengefasst)
SELECT       ProductID
           , ProductName
FROM Products
WHERE ProductName LIKE '[d-l]%[a-d|m-o]'
```

Hinweis:

Bei Möglichkeit 1 müssen wir unbedingt Klammern setzen, sonst bekommen wir ein anderes Ergebnis heraus. Es würde dann Bedingung A AND B zusammengezogen und dann erst das OR gemacht. (Ähnlich wie bei „Punkt-vor-Strich-Rechnung“ gewinnt hier das AND gegenüber dem OR.)

Modul 05 – JOINS

Lösung zu Übung 5.1

```
SELECT       o.OrderID AS Bestellnummer
           , o.CustomerID AS Kundennummer
           , c.CompanyName AS Firmenname
           , o.Freight AS Frachtkosten
FROM Customers AS c INNER JOIN Orders AS o ON c.CustomerID = o.CustomerID
WHERE c.Country = 'Brazil'
```

Hinweise:

- Da der Name CustomerID in zwei Tabellen vorkommt, müssen wir dazusagen, aus welcher Tabelle wir diese Information beziehen wollen; bei den anderen ist es optional, *sollte* aber immer dabeistehen.
- In diesem Fall können wir die CustomerID entweder aus der Customers- oder der Orders-Tabelle abfragen; da die Daten hier übereinstimmen, spielt es in diesem konkreten Fall keine Rolle, für welche wir uns entscheiden.
- Damit wir uns Tipparbeit ersparen, dürfen wir die Tabellennamen innerhalb der Abfrage mit einem Kürzel abkürzen (in diesem Fall c für die Customers-Tabelle und o für die Orders-Tabelle).
- Das Kürzel für die Tabellennamen gilt nur innerhalb dieser Abfrage und wird wie ein Spaltenalias mit „AS“ vergeben.
- Das „AS“ dürfen wir auch weglassen. Bei den Spaltenalias sollten wir das „AS“ zwecks besserer Lesbarkeit verwenden, obwohl es weggelassen werden darf, bei den Tabellen kann es weggelassen werden, ohne die Lesbarkeit zu beeinträchtigen:

```
FROM Customers c INNER JOIN Orders o ON c.CustomerID = o.CustomerID
```

Lösung zu Übung 5.2

```
SELECT      p.ProductName
      , s.CompanyName
      , s.ContactName
      , s.Phone
FROM Products p INNER JOIN Suppliers s ON p.SupplierID = s.SupplierID
WHERE p.ProductName LIKE '%sauce%' OR p.ProductName LIKE N'%soße'
```

Hinweise:

- Wir brauchen einen JOIN der Tabellen Products (um den Produktnamen abfragen zu können) und Suppliers (um die Kontaktinformationen abfragen zu können).
- Wir verwenden einen INNER JOIN, denn wenn wir nachbestellen wollen, brauchen wir nur die Supplier, die uns schon Soßen geliefert haben.
- Wir verwenden ein LIKE und Wildcards, um mehrere Möglichkeiten abzudecken (also alle Produkte, bei denen im Namen „Sauce“ vorkommt – dann bekommen wir 2 Ergebnisse).
- Wollen wir auch die Produkte abdecken, bei denen im Namen „Soße“ vorkommt, bekommen wir ein Ergebnis mehr.

Lösung zu Übung 5.3

```
SELECT      o.OrderID
      , CONCAT(e.FirstName, ' ', e.LastName) AS FullName
FROM Orders o INNER JOIN Employees e ON o.EmployeeID = e.EmployeeID
WHERE OrderID IN(10251, 10280, 10990, 11000)
```

Hinweise:

- Mit CONCAT können wir Vor- und Nachname in einer Spalte als FullName ausgeben.
- Wir verwenden einen INNER JOIN, denn wir brauchen nur solche Angestellte, die schon Bestellungen bearbeitet haben.

Lösung zu Übung 5.4

```
SELECT      o.OrderID
          , o.CustomerID
          , c.CompanyName
          , p.ProductName
          , od.Quantity
FROM Customers c INNER JOIN Orders o ON c.CustomerID = o.CustomerID
                 INNER JOIN [Order Details] od ON od.OrderID = o.OrderID
                       INNER JOIN Products p on p.ProductID = od.ProductID
WHERE p.ProductName LIKE '%chai%'
```

Hinweise:

- Wir verwenden INNER JOINS, denn wir brauchen nur die Kunden, die schon etwas bestellt haben (nämlich Chai Tee), und nur die Produkte, die schon bestellt wurden (wieder Chai Tee).
- Beim Verwenden von mehreren JOINS kommen die weiteren Tabellen zu den bereits verknüpften hinzu. Wenn es jemandem hilft, die Lesbarkeit zu verbessern, dürften die JOINS auch mit Klammern versehen werden:

```
FROM ((Customers c INNER JOIN Orders o ON c.CustomerID = o.CustomerID)
                 INNER JOIN [Order Details] od ON od.OrderID = o.OrderID)
                       INNER JOIN Products p on p.ProductID = od.ProductID
```

- Die JOINS untereinander zu schreiben und einzurücken hat keine Auswirkung auf die Funktionalität, sondern dient lediglich der besseren Lesbarkeit und Übersichtlichkeit.
- In diesem Fall gibt es nur ein Produkt namens „Chai“. Mit '%chai%' hätten wir aber auch andere Varianten von Chai-Tee abgedeckt, etwa, wenn es einen „Vanilla Chai“ oder einen „Premium Chai Tee“ gäbe.
- Vorsicht mit Zeichenfolgen und Wildcards: Gäbe es ein anderes Produkt, das die Zeichenfolge „chai“ enthält, ohne etwas mit Tee zu tun zu haben, so würde auch dieses ausgegeben werden.

Lösung zu Übung 5.5

```
SELECT       o.OrderID
           , c.CustomerID
           , c.CompanyName
           , p.ProductName
           , od.Quantity
FROM Customers c INNER JOIN Orders o ON c.CustomerID = o.CustomerID
                    INNER JOIN [Order Details] od ON od.OrderID = o.OrderID
                        INNER JOIN Products p ON p.ProductID = od.ProductID
WHERE ProductName LIKE '%bier%'
                        OR ProductName LIKE '%lager%'
                             OR ProductName LIKE '%ale'
ORDER BY Quantity DESC, CompanyName -- ASC
```

Hinweise:

- JOINS wie in Übung 5.4
- Achtung auf die Aufgabenstellung: Wenn Sie statt '%ale' (soll mit „ale“ enden) '%ale%' (soll „ale“ enthalten) abgefragt haben, bekommen Sie auch Ergebnisse zurück, die mit Bier nichts zu tun haben.
- Im ORDER BY geben wir an, wonach die Ausgabe geordnet werden soll. Der Default ist ASC (ascending); wenn wir möchten, dass die Ausgabe alphabetisch oder vom kleinsten zum größten Wert geordnet wird, müssen wir keinen Zusatz zum Spaltennamen hinzufügen.
- Wenn wir vom größten zum kleinsten Wert oder in umgekehrter alphabetischer Reihenfolge ordnen wollen, verwenden wir den Zusatz DESC (descending, absteigend).

Lösung zu Übung 5.6

```
SELECT   CONCAT(e1.FirstName, ' ', e1.LastName) AS Mitarbeiter
       , e1.EmployeeID AS Mitarbeiternummer
       , CONCAT(e2.FirstName, ' ', e2.LastName) AS Vorgesetzter
       , e2.EmployeeID AS ChefID -- oder e1.ReportsTo
FROM Employees e1 LEFT JOIN Employees e2 ON e1.ReportsTo = e2.EmployeeID
```

Hinweise:

- Mit CONCAT können wir Vor- und Nachname in einer Spalte ausgeben.
- Welches ALIAS wir für die Tabellen vergeben, ist egal, solange wir selbst uns damit auskennen. Wir könnten auch die erste Tabelle z.B. „emp“ und die hypothetische zweite Tabelle „boss“ nennen:

```
FROM Employees emp LEFT JOIN Employees boss ON emp.ReportsTo = boss.EmployeeID
```

- Wenn auch die Employees, die keinen Vorgesetzten haben (entweder, weil keiner zugewiesen ist oder, so wie hier, weil sie selbst der Chef sind) ausgegeben werden sollen, verwenden wir einen OUTER JOIN, hier einen LEFT JOIN.

- Wenn der Chef selbst oder solche Angestellte, die keinen Vorgesetzten haben (Teamleiter, noch nicht zugeordnet, …) NICHT mit ausgegeben werden sollen, verwenden wir einen INNER JOIN.
- Beachten Sie: wenn wir ein CONCAT verwenden, steht in der Spalte „Vorgesetzter“ beim Chef nicht „NULL“ wie in der ID-Spalte daneben, denn in der Spalte steht jetzt das Leerzeichen aus dem CONCAT!

Lösung zu Übung 5.7

```
SELECT       a.CompanyName AS Kunde
      , a.City AS Stadt
      , b.CompanyName AS Mitfahrgelegenheit
      , b.City AS Kontrollfeld
FROM Customers a INNER JOIN Customers b ON a.City = b.City
WHERE a.CustomerID != b.CustomerID
ORDER BY a.City, a.CompanyName
```

Hinweise:

- Wir erstellen den Selfjoin auf der Spalte City, denn wir wollen ja die herausfinden, die in der hypothetischen zweiten Tabelle in derselben Stadt wohnen, wie die in der eigentlichen Customers-Tabelle.
- Wir verwenden einen INNER JOIN; hier haben wir keine, wo es keine Übereinstimmungen gibt.
- Mit `WHERE a.CustomerID != b.CustomerID` schließen wir aus, dass z.B. „ALFKI“ neben „ALFKI“ (also sich selbst) steht.

Modul 06 – DISTINCT, TOP, ORDER BY

Lösung zu Übung 6.1

```
-- teuerstes Produkt
SELECT TOP 1
                  ProductID
                , ProductName
                , UnitPrice
FROM Products
ORDER BY UnitPrice DESC

-- günstigstes Produkt
SELECT TOP 1
                  ProductID
                , ProductName
                , UnitPrice
FROM Products
ORDER BY UnitPrice
```

Hinweis:

Was in der ersten Zeile steht, ist immer abhängig vom ORDER BY. Wollen wir also das teuerste Produkt mittels TOP-Befehl ausgeben, drehen wir einfach die Sortierreihenfolge mit DESC um.

Lösung zu Übung 6.2

```
SELECT TOP 10 PERCENT WITH TIES
                          p.ProductName
                        , od.Quantity
FROM [Order Details] od INNER JOIN Products p ON od.ProductID =
p.ProductID
ORDER BY od.Quantity DESC
```

Hinweise:

- Top % müssen als TOP 10 PERCENT ausgeschrieben werden.
- Ohne WITH TIES würden wir 216 Zeilen zurückbekommen. Sehen wir uns Zeile 216 an, sehen wir, die Quantity darin ist 50. Mit WITH TIES bekommen wir alle anderen, die auch 50 Stück bestellt haben, mit ausgegeben. (Immer abhängig davon, wonach geordnet wurde.)

Lösung zu Übung 6.3

```
SELECT TOP 3
       CONCAT(FirstName, ' ', LastName) AS EmpName
     , Title
     , FORMAT(HireDate, 'd', 'de-de') AS HireDate
FROM Employees
ORDER BY HireDate -- ASC
```

Hinweise:

- Vor- und Nachname in einer Spalte ausgeben mit CONCAT
- Datum mit FORMAT und Culture-Parameter oder mit CONVERT und Style-Parameter in sinnvolles Ausgabeformat bringen
- ORDER BY HireDate ASC – damit ordnen wir in aufsteigender Reihenfolge, vom kleinsten zum größten Ergebnis. Da wir diejenigen ausgeben wollen, die schon am längsten beim Unternehmen sind, brauchen wir die mit der kleinsten (= am längsten zurückliegenden) Jahreszahl.
- ASC können wir dazuschreiben, dürfen wir aber auch weglassen, da es sich dabei um den Default handelt. (DESC wäre genau umgekehrt, descending, also vom größten zum kleinsten Wert geordnet.)

Modul 07 – Aggregatfunktionen

Lösung zu Übung 7.1

```
SELECT COUNT(ProductID) AS [Anzahl Produkte]
FROM Products
```

Hinweis:

Wir können alle Zeilen in der Tabelle Produkte zählen, denn in jeder Zeile steht genau ein Produkt; oder, wir zählen wie hier die Anzahl der ProductIDs. Da die ProductID eindeutig ist, gibt es genau eine pro Produkt; zählen wir, wie viele es davon gibt, wissen wir, wie viele Produkte wir in der DB haben.

Lösung zu Übung 7.2

```
SELECT COUNT(DISTINCT Country) AS Länderanzahl
FROM Customers
```

Hinweise:

- Hier müssen wir ein DISTINCT verwenden!
- Neben jedem Kunden steht auch ein Land; machen wir einfach nur ein COUNT(Country), zählen wir jeden Eintrag in der Spalte Country, und bekommen so viele Einträge zurück, wie es Kunden gibt!
- Das kann auch leicht zu falschen/falsch interpretierten Ergebnissen führen:
- Würden wir z.B. ein COUNT(Region) machen, bekämen wir weniger Treffer zurück, als wir Kunden haben. Das könnte man leicht (fälschlich!) so interpretieren, dass wir diese Anzahl an Regionen haben. Tatsächlich gibt es aber viele Kunden, bei denen keine Region eingetragen ist, der Inhalt des Feldes also NULL ist. Auch hier müssten wir ein COUNT(DISTINCT Region) machen, um zum gewünschten Ergebnis zu kommen.

Lösung zu Übung 7.3

```
SELECT   AVG(p.UnitPrice) AS [Durchschnittspreis/Anbieter]
      , s.SupplierID
      , s.CompanyName
FROM Products p INNER JOIN Suppliers s ON p.SupplierID = s.SupplierID
GROUP BY s.SupplierID, s.CompanyName
ORDER BY AVG(p.UnitPrice)
```

Hinweise:

- Wir brauchen einen INNER JOIN der Tabellen Products und Suppliers, da wir den UnitPrice aus den Produkten und den CompanyName aus den Suppliers ausgeben wollen.
- Ob wir die SupplierID aus der Products- oder der Suppliers-Tabelle auswählen, spielt für diese Abfrage keine Rolle; aber wir müssen uns für eine Tabelle entscheiden, da der Name nicht eindeutig ist.
- Da es keine doppelten CompanyNames gibt, sind in diesem Fall auch diese eindeutig; wir bekommen kein anderes Ergebnis, wenn wir nur SupplierID oder zusätzlich auch den CompanyName abfragen.
- Wäre der CompanyName nicht gefragt gewesen, wäre kein JOIN notwendig gewesen, da die SupplierID allein auch in der Products-Tabelle vorkommt.
- Im ORDER BY können wir die Aggregatfunktion oder auch das dafür angegebene ALIAS verwenden.

Lösung zu Übung 7.4

```
SELECT   SUM(Freight) AS [Summe Frachtkosten/Kunde]
       , CustomerID
       , ShipCountry
FROM Orders
WHERE YEAR(ShippedDate) = 1996
GROUP BY CustomerID, ShipCountry
ORDER BY [Summe Frachtkosten/Kunde] DESC
```

Hinweise:

- „Pro“ Kunde im jeweiligen Land: Diese beiden Spalten stehen auch im SELECT und müssen mit GROUP BY gruppiert werden.
- Nur die Frachtkosten, die im Jahr 1996 entstanden sind, können wir mittels WHERE abfragen. Dabei hilft uns auch die Datumsfunktion YEAR, um nur das Jahr des jeweiligen Datums abfragen zu können.
- ORDER BY ist der letzte Ausdruck, den wir am Schluss des gesamten Statements verwenden.
- Im ORDER BY können wir entweder das Spaltenalias verwenden (wie oben), oder noch einmal SUM(Freight) schreiben.

Lösung zu Übung 7.5

```
SELECT       COUNT(CustomerID) AS CustomerCount
           , Country
FROM Customers
GROUP BY Country
HAVING COUNT(CustomerID) > 5
ORDER BY CustomerCount DESC
```

Hinweise:

- Wir brauchen hier kein DISTINCT im Count, denn die CustomerID ist eindeutig (es können keine Mehrfacheinträge weggekürzt werden).
- Wir möchten Kunden pro Land ausgeben, daher müssen wir mit GROUP BY Country gruppieren.
- WHERE funktioniert nicht, wenn wir mit einem Wert vergleichen wollen, der erst durch eine Aggregatfunktion berechnet werden soll; daher verwenden wir ein HAVING.
- Es soll in absteigender Reihenfolge geordnet werden, daher: DESC.

Lösung zu Übung 7.6

```
SELECT       CONCAT(FirstName, ' ',  LastName) AS FullName
           , COUNT(o.OrderID) AS NumberOfOrders
FROM Orders o INNER JOIN Employees e ON o.EmployeeID = e.EmployeeID
GROUP BY LastName, FirstName
HAVING COUNT(o.OrderID) > 70
ORDER BY NumberOfOrders
```

Hinweise:

- FirstName und LastName und Leerzeichen können wir mittels CONCAT in einer Spalte ausgeben.
- Wir brauchen kein DISTINCT, denn die OrderID ist eindeutig (es können keine Mehrfacheinträge weggekürzt werden).
- Wir verwenden einen INNER JOIN, denn uns interessieren nur solche Mitarbeiter, die schon Bestellungen bearbeitet haben (bzw. sogar nur die, die mehr als 70 bearbeitet haben).
- Im CONCAT stecken zwei Spaltennamen; wir müssen über beide gruppieren.
- Wir verwenden HAVING, denn wir wollen mit einem Wert vergleichen, der erst durch eine Aggregatfunktion berechnet werden muss.

Lösung zu Übung 7.7

```
SELECT       LastName
           , COUNT(OrderID) AS NumberOfOrders
FROM Orders o INNER JOIN Employees e ON o.EmployeeID = e.EmployeeID
WHERE LastName IN('Leverling', 'Peacock')
GROUP BY LastName
HAVING COUNT(OrderID) > 100
```

Hinweise:

- Wir brauchen kein DISTINCT im COUNT; die OrderID ist eindeutig und daher können keine Mehrfacheinträge weggekürzt werden.
- Wir verwenden einen INNER JOIN, denn uns interessieren hier nur zwei Mitarbeiter, die schon Bestellungen bearbeitet haben, bzw. nur dann, wenn sie über eine bestimmte Anzahl bearbeitet haben.
- Wir haben hier ein WHERE, um die Namen der Angestellten abzufragen; wir können sie einzeln mit einem OR dazwischen abfragen oder sie in einem IN zusammenfassen.
- Wir verwenden ein HAVING, um mit einem Wert vergleichen zu können, der erst in einer Aggregatfunktion berechnet werden muss.
- In der Realität würden wir eher nach der Mitarbeiternummer als nach dem Namen suchen, denn theoretisch könnten wir mehrere Mitarbeiter mit diesen (Nach-)Namen haben.

Lösung zu Übung 8.1

```
CREATE VIEW v_Rechnungsdaten
AS
SELECT       c.CustomerID
           , c.CompanyName
           , c.Address
           , c.PostalCode
           , c.City
           , c.Country
           , o.OrderID
           , o.ShippedDate
           , o.Freight
           , od.Quantity
           , od.UnitPrice
           , od.Discount
           , p.ProductID
           , p.ProductName
FROM Customers c INNER JOIN Orders o ON c.CustomerID = o.CustomerID
                 INNER JOIN [Order Details] od ON o.OrderID = od.OrderID
                     INNER JOIN Products p ON od.ProductID = p.ProductID
```

Hinweise:

- Wenn Sie nicht exakt die gleichen Spalten ausgewählt haben, ist das ok. Ziel der Übung war das Erstellen einer View, nicht bestimmte Spalten ins SELECT zu schreiben.
- Die View mit allen enthaltenen Informationen können Sie nun verwenden wie eine Tabelle.

Lösung zu Übung 8.2

```
CREATE PROC p_Customers_Cities @City nvarchar(50)
AS
SELECT       c.CustomerID
           , c.CompanyName
           , c.PostalCode
           , c.Address
           , c.City
           , c.Country
                , o.OrderID
     , o.OrderDate
FROM Customers c INNER JOIN Orders o ON c.CustomerID = o.CustomerID
WHERE City = @City
GO



CREATE TABLE #test (
                     CustomerID nchar(5)
                     , CompanyName nvarchar(50)
                     , PostalCode nvarchar (50)
                     , Address nvarchar (100)
                     , City nvarchar (50)
                     , Country nvarchar (50)
                     , OrderID int
                     , OrderDate date
                   )


INSERT INTO #test
EXEC p_Customers_Cities @City = 'Buenos Aires'


SELECT * FROM #test ORDER BY OrderID
```

Hinweise:

- Der Datentyp der Variable, in die wir die Stadt eingeben möchten, muss nicht die exakte Anzahl an Zeichen vorgeben, die auch die entsprechende Spalte hat (es dürfen auch mehr sein).
- Wenn Sie mehr oder weniger Spalten gewählt haben, ist das in Ordnung (es waren keine genauen Spalten in der Aufgabenstellung gefragt).
- In diesem Fall ist das „GO“ empfehlenswert! Würden wir das, was wir ausführen wollen, nicht in kleinen „Happen“ markieren und einzeln ausführen, sondern alles auf einmal, wären die darauffolgenden Anweisungen noch Teil der Prozedur! Damit können Endlosschleifen erzeugt werden!
- Mit `EXEC p_Customers_Cities @City = 'Buenos Aires'` können wir die Kundeninformationen der Kunden aus Buenos Aires abrufen.

- Fortgeschrittenenteil: Der Prozedur selbst können wir kein ORDER BY anhängen. Eine Möglichkeit, die Informationen nachträglich noch zu sortieren wäre, eine temporäre Tabelle zu erstellen und die Ergebnisse der Prozedur darin zu hinterlegen. Dann können die Daten aus der temporären Tabelle abgefragt und beliebig geordnet werden.

Lösung zu Übung 8.3

```
-- Sicherstellen, dass die richtige DB verwendet wird:
USE TestDB

-- Prozedur erstellen
CREATE PROCEDURE p_raise
AS
UPDATE TestEmployees
SET Salary = (CASE
                WHEN Salary >= 10000 THEN Salary
                WHEN Salary > (SELECT AVG(Salary) FROM
TestEmployees) THEN Salary * 1.1
                WHEN Salary < (SELECT AVG(Salary) FROM
TestEmployees) THEN Salary * 1.2
                ELSE Salary
           END)
GO


-- Prozedur ausführen:
EXEC p_raise

-- Ergebnis testen:
SELECT       EmployeeID
           , LastName
           , Salary

FROM TestEmployees
```

Hinweise:

- Beim Erstellen der Prozedur können wir PROCEDURE ausschreiben oder mit PROC abkürzen.
- Wir müssen das durchschnittliche Gehalt erst berechnen. Das können wir mit einer Subquery erreichen.
- Verwenden Sie ein GO! Damit stellen Sie sicher, dass nicht versehentlich das Ausführen der Prozedur Teil derselben wird.
- Welche Spalten Sie zum Testen auswählen, bleibt Ihnen überlassen, aber Salary muss enthalten sein, damit Sie sehen können, wie sich die Prozedur auf die Gehälter Ihrer Mitarbeiter auswirkt.

Modul 09 – Setoperatoren

Lösung zu Übung 9.1

```
SELECT       CustomerID AS ContactID
           , CompanyName
           , ContactName
           , ContactTitle
           , Phone
           , 'C' AS Category
FROM Customers
UNION ALL
SELECT       CAST(SupplierID AS nvarchar)
           , CompanyName
           , ContactName
           , ContactTitle
           , Phone
           , 'S'
FROM Suppliers
ORDER BY ContactID
```

Hinweise:

- Wir verwenden hier ein UNION ALL. Wir wollen alle Kunden und Anbieter ausgeben, und hier können keine Mehrfacheinträge vorhanden sein (CompanyName und IDs sind eindeutig). UNION würde uns in diesem Fall das gleiche Ergebnis liefern, allerdings ist UNION ALL schneller, weil im Hintergrund keine Überprüfung stattfinden muss, ob es einen Eintrag schon gibt. Bei so kleinen Datenmengen wie in der Northwind-DB wird der Unterschied kaum auffallen, da er sich im Millisekundenbereich bewegt; bei größeren Datenmengen kann es aber einen signifikanten Performanceunterschied ausmachen.
- JOINS wären für diese Aufgabenstellung nicht zielführend. Zum einen würden wir mit einem JOIN keine Liste in der Ausgabe erstellen, sondern mehrere Spalten nebeneinander; zum anderen müssten fünf Tabellen verwendet werden, von denen drei gar nicht gebraucht werden (also unsinnig viel Daten angefasst werden, was die Abfrage wesentlich verlangsamen würde).
- In die selbsterstellte Spalte „Category“ können wir einfach Text hineinschreiben. Beim SELECT aus der Customers-Tabelle fügen wir ein „C“ ein, beim SELECT aus der Suppliers-Tabelle ein „S“. Die Spaltenüberschrift „Category“ müssen wir nur im ersten SELECT angeben, sie gilt für alle Informationen, die in diese Spalte geschrieben werden.
- Beim Hinzufügen der CustomerID und SupplierID mit der neuen Spaltenüberschrift „ContactID“ kommt es vermutlich zu einer Fehlermeldung: `“Conversion failed when converting the nvarchar value 'ALFKI' to data type int.“` Das liegt daran, dass die CustomerID ein nchar-Datentyp und die SupplierID ein int ist. Der Server versucht, den nchar in einen int umzuwandeln, und scheitert; wir müssen den int explizit zu einem nvarchar konvertieren (mit CAST oder CONVERT).
- Ausgabereihenfolge: Ohne unser ORDER BY kommen zuerst die Customers, dann die Suppliers; nach unserem ORDER BY ist es genau umgekehrt. Wir verwenden für unser ORDER BY das im

ersten SELECT vergebene ALIAS (ContactID). Wenn wir nach CompanyName ordnen, werden Customer- und Supplier-Kontakte in gemischter Reihenfolge ausgegeben.

Lösung zu Übung 9.2

```
SELECT TOP 1
            OrderID
          , Freight
          , 'niedrigster Wert' AS Anmerkung
INTO #niedrigster_Wert
FROM Orders
ORDER BY Freight
```

```
SELECT TOP 1
            OrderID
          , Freight
          , 'höchster Wert' AS Anmerkung
INTO #hoechster_Wert
FROM Orders
ORDER BY Freight DESC
```

```
SELECT *
FROM #niedrigster_Wert
UNION ALL
SELECT *
FROM #hoechster_Wert
```

Hinweise:

- Wenn wir auch die OrderID ausgeben wollen, können wir nicht mit einer Aggregatfunktion (MIN oder MAX) arbeiten, denn das wäre dann der höchste bzw. niedrigste Frachtkostenwert pro OrderID (also wieder genau die Frachtkosten).
- Mit den bisher gelernten Mitteln haben wir die Möglichkeit, TOP 1 zu verwenden, einmal nach Frachtkosten (Freight) absteigend geordnet (höchste Frachtkosten) und einmal aufsteigend (niedrigste Frachtkosten). In beiden Fällen brauchen wir aber ein unterschiedliches ORDER BY.
- Wir können die so erzeugten Ergebnisse in temporären Tabellen zwischenspeichern und diese mit UNION ALL verbinden.
- Wir verwenden UNION ALL, denn hier gibt es nur zwei (unterschiedliche) Ergebnisse und somit brauchen keine doppelten ausgeschlossen werden (UNION macht auch ein DISTINCT).

Modul 10 – Subqueries (Unterabfragen)

Lösung zu Übung 10.1

```
SELECT      OrderID
     , Freight
FROM Orders
WHERE Freight > (SELECT AVG(Freight) FROM Orders)
ORDER BY Freight DESC
```

Hinweise:

- Wir berechnen die durchschnittlichen Frachtkosten in einer Subquery; damit können wir dann die Frachtkosten unserer eigentlichen Abfrage vergleichen.
- Versuchen Sie, Freight direkt mit AVG(Freight) zu vergleichen – wir bekommen eine Fehlermeldung:

"An aggregate may not appear in the WHERE clause unless it is in a subquery […]"

- Um vom größten zum kleinsten Wert in absteigender Reihenfolge zu ordnen, brauchen wir ein DESC (descending – absteigend).

Lösung zu Übung 10.2

```
SELECT      SupplierID
     , CompanyName
     , ContactName
     , Country
FROM Suppliers
WHERE Country = (SELECT Country FROM Suppliers WHERE SupplierID = 2)
```

Hinweis: Wir müssen zuerst herausfinden, aus welchem Land der Supplier mit der ID 2 stammt. Wir könnten dann händisch das aus der Abfrage resultierende Land in die andere Abfrage einsetzen, oder wir kombinieren diesen Vorgang zu einer Unterabfrage.

Lösung zu Übung 10.3

```
SELECT  CONCAT(TitleOfCourtesy, ' ', FirstName, ' ', LastName) AS FullName
      , FORMAT(HireDate, 'd', 'de-de') AS HireDate
FROM Employees
WHERE YEAR(HireDate) =
      (SELECT YEAR(HireDate) FROM Employees WHERE CONCAT(TitleOfCourtesy, ' ',
FirstName, ' ', LastName) = 'Mr. Robert King')
```

Hinweise:

- Mit CONCAT können wir die Ergebnisse aus mehreren Spalten in einer Spalte zusammenfassen (mit Leerzeichen dazwischen).
- Mit FORMAT (oder CONVERT und Style-Parameter) können wir das Datum in ein leserliches Format bringen.
- Normalerweise würden wir die MitarbeiterID überprüfen, nicht Name und Titel (es könnte theoretisch mehrere Mr. Robert Kings geben).
- Statt YEAR(HireDate) wäre auch DATEPART(year, HireDate) möglich (exakt gleiches Ergebnis).
- In diesem Beispiel wird Robert King selbst auch mit ausgegeben. Will man ihn weglassen: `AND CONCAT(TitleOfCourtesy, ' ', FirstName, ' ', LastName) != 'Mr. Robert King'`, bzw. wir könnten auch seine ID herausfinden und diese ausschließen. Das wäre auch in einer zweiten Unterabfrage möglich (vorausgesetzt es gibt in dem Unternehmen nur einen Mr. Robert King): `AND EmployeeID!= (SELECT EmployeeID FROM Employees WHERE CONCAT(TitleOfCourtesy, ' ', FirstName, ' ', LastName) = 'Mr. Robert King')`

Lösung zu Übung 10.4

```
SELECT      EmployeeID
          , CONCAT(FirstName, ' ', LastName) AS FullName
          , Salary
FROM TestEmployees
WHERE Salary > (SELECT Salary FROM TestEmployees WHERE EmployeeID = 8)
```

Hinweis: Vor- und Nachname können wir mittels CONCAT in einer Spalte ausgeben.

Lösung zu Übung 10.5

```
SELECT      EmployeeID
          , CONCAT(FirstName, ' ', LastName) AS FullName
          , Salary
          , Country
FROM TestEmployees
WHERE Salary IN (SELECT MIN(Salary) FROM TestEmployees GROUP BY Country)
```

Hinweise:

- Vor- und Nachname können wir mittels CONCAT in einer Spalte ausgeben.
- Hier vergleichen wir mit mehreren Ländern – daher verwenden wir ein IN()!

Lösung zu Übung 10.6

```
SELECT       EmployeeID
      , CONCAT(FirstName, ' ', LastName) AS FullName
      , Salary
      , Country
FROM TestEmployees
WHERE Salary BETWEEN (SELECT MIN(Salary) FROM TestEmployees) AND
3000
ORDER BY Salary -- ASC
```

Hinweise:

- Vor- und Nachname können wir mittels CONCAT in einer Spalte ausgeben.
- Wir können mit >= und <= oder mit BETWEEN den Bereich zwischen dem niedrigsten Gehalt und 3000 abfragen.
- Da aufsteigend geordnet werden soll, müssen wir den Zusatz ASC nicht angeben (Default), dürfen aber, wenn wir unbedingt wollen.

Modul 11 – CASE

Lösung zu Übung 11.1

```
SELECT
                DISTINCT Country
                , CASE
                            WHEN Country IN('Germany', 'Austria', 'France') THEN 'EU'
                            WHEN Country IN('Argentina', 'US', 'Brazil') THEN 'nicht
EU'
                            WHEN Country = 'UK' THEN 'Brexit'

                            ELSE 'unbekannt'
                      END AS [Zugehörigkeit]
FROM Customers
ORDER BY [Zugehörigkeit]
```

Hinweise:

- Wir können entweder jedes Country einzeln abfragen und jedes Mal „EU“ oder „nicht EU“ dazuschreiben, oder die Länder in einem IN() zusammenfassen.
- Auch der Spalte, die dynamisch mit CASE befüllt wird, kann natürlich eine Spaltenüberschrift erhalten (Zugehörigkeit).
- Im ORDER BY können wir dieses Alias verwenden, um danach zu sortieren (natürlich wollen wir hier nicht noch einmal den gesamten CASE-Ausdruck hinschreiben!).

Lösung zu Übung 11.2

Lösung der ersten Teilaufgabe:

```
SELECT c.CustomerID AS Kundennummer
      , c.CompanyName AS Firmenname
      , YEAR(o.OrderDate) AS Bestelljahr
      , CAST(SUM(od.Quantity*od.UnitPrice*(1-Discount)) AS
decimal(10, 2)) AS Einkaufssumme
      , CASE
            WHEN SUM(od.Quantity*od.UnitPrice*(1-Discount)) >= 500
              THEN 'Premium'
            WHEN SUM(od.Quantity*od.UnitPrice*(1-Discount)) < 500
              THEN 'Standard'
                 ELSE 'unbekannt'
            END AS Kundenstatus
FROM Customers c LEFT JOIN Orders o ON c.CustomerID = o.CustomerID
            LEFT JOIN [Order Details] od ON o.OrderID = od.OrderID
GROUP BY c.CustomerID, c.CompanyName, YEAR(o.OrderDate)
ORDER BY Einkaufssumme
```

Hinweise:

- Wenn wir mit einem INNER JOIN arbeiten, bekommen wir in der Ausgabe nur die Kunden, die schon etwas bestellt haben; wenn wir mit einem OUTER JOIN (in diesem Fall: LEFT JOIN) arbeiten, werden auch die Kunden ausgegeben, die noch nichts bestellt haben.
- Uns interessiert nur das Bestelljahr, nicht das genaue Bestelldatum bis hin zur (nicht eingetragenen) Millisekunde; daher schneiden wir das Jahr mit der Funktion YEAR(OrderDate) oder DATEPART(year, OrderDate) aus.
- Die Einkaufssumme ergibt sich folgendermaßen: UnitPrice*Quantity ist der jeweilige Rechnungsposten (hat jemand 3 Stück Käse bestellt, dann Käsepreis*3). Wir möchten aber die Rechnungssumme berechnen; hat also jemand 3 Stück Käse und 2 Flaschen Champagner bestellt, dann 3*Käsepreis + 2*Champagnerpreis. Davon ziehen wir noch den jeweiligen Discount ab, indem wir mit (1-Discount) multiplizieren.
- Wollen wir die Rechnungssumme pro Kunde und Jahr, müssen wir über diese Spalten gruppieren (GROUP BY).
- Der Firmenname soll mit ausgegeben werden; das verfälscht unsere Ausgabe nicht, obwohl dann auch über CompanyName gruppiert werden muss, da ja exakt genauso viele CompanyNames wie CustomerIDs vorhanden sind (pro Firma gibt es eine ID).
- 236 Zeilen Ausgabe kommen daher, dass wir über das Bestelljahr gruppiert haben. Zwar gibt es nur 91 Kunden, viele davon haben aber mehr als einmal (und manche in mehr als einem Jahr) eine Bestellung getätigt.

Lösung der zweiten Teilaufgabe:

```
SELECT    c.CustomerID AS Kundennummer
                , c.CompanyName AS Firmenname
                , YEAR(o.OrderDate) AS Bestelljahr
                , CAST(SUM(od.Quantity*od.UnitPrice*(1-Discount)) AS decimal(10, 2)) AS Einkaufssumme
                , CASE
                                WHEN SUM(od.Quantity*od.UnitPrice*(1-Discount)) >= 500 THEN 'Premium'
                                WHEN SUM(od.Quantity*od.UnitPrice*(1-Discount)) < 500 THEN 'Standard'
                                ELSE 'unbekannt'
                        END AS Kundenstatus
FROM Customers c INNER JOIN Orders o ON c.CustomerID = o.CustomerID
                        INNER JOIN [Order Details] od ON o.OrderID = od.OrderID
WHERE YEAR(OrderDate) = 1996 -- 1997 -- 1998
GROUP BY c.CustomerID, c.CompanyName, YEAR(o.OrderDate)
ORDER BY Einkaufssumme
```

Hinweise:

- Im Unterschied zur ersten Teilaufgabe ändert sich nur, dass wir nun im WHERE die Jahre 1996, 1997 und 1998 abfragen.
- Falls es jemand ausprobiert und keine anderen Ergebnisse zurückbekommen hat: Es sind nur Bestellungen für die Jahre 1996, 1997 und 1998 vorhanden.
- In diesem Fall macht es keinen Unterschied, ob wir mit INNER oder OUTER JOIN gearbeitet haben, da wir auf das Bestelldatum einschränken. Auch, wenn wir mit OUTER JOIN gearbeitet haben, werden die Kunden, die noch nichts bestellt haben, nun nicht angezeigt, da bei ihnen kein Bestelldatum vorhanden ist.

Lösung der dritten Teilaufgabe:

```
SELECT    c.CustomerID AS Kundennummer
                , c.CompanyName AS Firmenname
                , YEAR(o.OrderDate) AS Bestelljahr
                , CAST(SUM(od.Quantity*od.UnitPrice*(1-Discount)) AS decimal(10, 2)) AS Einkaufssumme
                , CASE
                                WHEN SUM(od.Quantity*od.UnitPrice*(1-Discount)) >= 500 THEN 'Premium'
                                WHEN SUM(od.Quantity*od.UnitPrice*(1-Discount)) < 500 THEN 'Standard'
                                ELSE 'unbekannt'
                        END AS Kundenstatus
FROM Customers c LEFT JOIN Orders o ON c.CustomerID = o.CustomerID
                                        LEFT JOIN [Order Details] od ON o.OrderID = od.OrderID
WHERE YEAR(OrderDate) IS NULL
GROUP BY c.CustomerID, c.CompanyName, YEAR(o.OrderDate)
ORDER BY Einkaufssumme
```

Hinweise:

- Um hier etwas in der Ausgabe angezeigt zu bekommen, müssen wir mit einem OUTER JOIN (hier: LEFT JOIN) arbeiten, um die beiden Kunden auszugeben, die noch nichts bestellt haben.
- Im Gegensatz zur zweiten Teilaufgabe ändert sich hier nur, dass wir statt einer Jahreszahl IS NULL abfragen.
- Achtung: Bei NULL können wir kein = Zeichen verwenden, sondern müssen IS NULL ausschreiben!

Modul 12 – CREATE, INSERT, UPDATE, DELETE

Lösung zu Übung 12.1

```
CREATE DATABASE TestDB

USE TestDB

SELECT *
INTO TestEmployees
FROM Northwind.dbo.Employees

ALTER TABLE TestEmployees
ADD Salary money

-- entweder für jeden einzeln zuweisen (9 Mitarbeiter)
-- UPDATE TestEmployees
-- SET Salary = 2500
-- WHERE EmployeeID = 1

-- oder UPDATE mit CASE:
UPDATE TestEmployees
SET Salary = (CASE
                WHEN EmployeeID = 1 THEN 2500
                WHEN EmployeeID = 2 THEN 8000
                WHEN EmployeeID = 3 THEN 1800
                WHEN EmployeeID = 4 THEN 5000
                WHEN EmployeeID = 5 THEN 3200
                WHEN EmployeeID = 6 THEN 3100
                WHEN EmployeeID = 7 THEN 2300
                WHEN EmployeeID = 8 THEN 2800
                WHEN EmployeeID = 9 THEN 3000
                ELSE Salary
            END)



SELECT      EmployeeID
          , CONCAT(FirstName, ' ', LastName) AS EmpName
          , Salary
FROM TestEmployees
```

Hinweise:

- Wenn wir Informationen aus einer anderen Datenbank „ausborgen“ (hier aus der Northwind-DB), dann müssen wir Datenbank und Schema angeben (Northwind.dbo).
- Da es sich in diesem Fall nur um 9 Angestellte handelt, können wir entweder die Werte einzeln zuweisen, indem wir im WHERE die EmployeeID abfragen, oder alle Gehälter auf einmal mit CASE zuweisen.

- ELSE Salary bedeutet, dass für den Fall, dass es einen Angestellten gibt, dessen ID wir hier nicht angeführt haben, sein Gehalt gleichbleibt (für unseren hypothetischen 10. Angestellten wäre dann noch kein Gehalt eingetragen und der Wert in dieser Spalte wäre NULL).