

L2 Einfache Auswahlabfragen

SQL stellt eigentlich nur recht wenige Kommandos zur Verfügung. Die Flexibilität und Leistungsfähigkeit der Sprache kommt durch eine große Zahl von Klauseln und sogenannte Prädikate zustande. Das wohl am häufigsten benötigte Kommando ist SELECT. Auf dieses Kommando und einige seiner wichtigsten Klauseln wollen wir daher zuerst eingehen.

Das SELECT-Kommando

Der SELECT-Befehl stellt den zentralen SQL-Befehl dar. Mit SELECT ist es möglich, auch sehr komplexe Abfragen über mehrere Tabellen in wenigen Zeilen zu definieren. Die Minimal-Syntax des Befehls hat folgende Form:

```
SELECT Spalte1, Spalte2, ..., Spalte(n)
FROM Tabelle
```

Damit haben Sie praktisch das Grundmuster einer SQL-Abfrage, das zunächst zwei Schritte umfasst:

- 1 In der SELECT-Zeile werden die Spalten (Felder) aufgelistet, die in der Ergebnistabelle erscheinen sollen.
- 2 Die FROM-Klausel bestimmt die Tabelle, aus der die Werte stammen.

Später schließen sich dann weitere Schritte an, etwa für die Auswahl der Datensätze und die Sortierung. Wir kommen noch darauf zurück. Eine erste Anwendung auf unsere Kundentabelle könnte folgende Form haben:

```
SELECT KundenNr, Firma, Ort
FROM Kunden
```

Die Felder, die in der Ergebnistabelle erscheinen sollen, müssen durch Kommata separiert aufgelistet werden. Hinter dem letzten Feld erscheint jedoch kein Komma mehr. Die Auflistung

der Spalten hat zur Folge, dass die anderen Spalten in der Ergebnistabelle nicht berücksichtigt werden.

Mit dem Beispiel erhalten Sie eine Ergebnistabelle wie die aus Abbildung L2.1. Das Fenster ist etwas verkleinert, sodass in der Abbildung nicht alle Datensätze zu sehen sind. Die Abfrage liefert aber alle Datensätze, die in der Kundentabelle enthalten sind.

	KundenNr	Firma	Ort
	1	Maier KG	Dresden
	2	Meier GmbH	Osnabrück
	3	Mayer & Söhne GmbH	Leipzig
	4	Müller OHG	Bad Homburg
	5	Böger	Dresden
	6	Hannibal	Bad Essen

Abb. L2.1: Ergebnistabelle (Teilansicht)

Alle Spalten ausgeben

Sollen alle Spalten der betreffenden Tabelle ausgegeben werden, können Sie statt der Auflistung auch einfach das Sternchen verwenden. Mit der folgenden Anweisung erhalten Sie alle Spalten und alle Zeilen der Tabelle *Kunden*, also praktisch die ganze Tabelle:

```
SELECT * FROM Kunden
```

Allerdings kann das Sternchen auch zu Problemen führen. Das gilt beispielsweise, wenn die Tabelle auch Memofelder und Felder für Binärdaten, etwa Grafiken, enthält. Es kann daher vorteilhafter sein, immer eine Feldliste zu verwenden.

Verschiedene Schreibweisen

SQL ist keineswegs so vereinheitlicht, wie Sie das vielleicht erwarten würden. Nicht nur bei Zahl und Art der Sprachelemente müssen Sie mit Unterschieden rechnen. Auch die Verwendung von Tabellen und Feldnamen kann differieren. Gelegentlich werden Sie daher Anweisungen wie die folgende finden:

```
SELECT 'KundenNr', 'Firma'  
FROM Kunden
```

Die Einschließung der Spaltennamen in Anführungszeichen sollte nur mit wenigen SQL-Datenbanken funktionieren. Bei unseren Versuchen bestand nur eine Datenbank auf Anführungszeichen. Bei anderen Datenbanken oder auch bei der Einbettung des SQL-Strings in andere Programmiersprachen müssen Sie sogar mit einer Fehlfunktion rechnen. Access meldet in diesem Fall zwar keinen Fehler, erzeugt aber unsinnige Ergebnisse.

Eckige Klammern in Access

Access kann Feldnamen auch in eckige Klammern setzen. Diese Option ermöglicht es, auch Sonderzeichen und Leerzeichen in Feldnamen zu verwenden. Wenn eine Tabelle beispielsweise Feldnamen (Spaltenbezeichnungen) wie die folgenden enthält, lassen sich diese in SQL-Kommandos normalerweise so nicht verwenden:

```
Straße  
Datum des Erstkontakts
```

Um solche Feldnamen in SQL-Anweisungen einsetzen zu können, müssen Sie unter Access eine SELECT-Anweisung wie folgt schreiben:

```
SELECT [Straße], [Datum des Erstkontakts]  
FROM Kunden
```

Beachten Sie, dass die vorstehende Anweisung mit unserer Beispieltabelle nicht funktioniert, weil diese keine entsprechenden Felder enthält. Bei anderen Datenbanken werden Sie Sonderzeichen und Leerzeichen ohnehin nicht einsetzen können.

Es dürfte daher die sicherste Methode sein, auf die genannten Zeichen und auch auf deutsche Umlaute zu verzichten, auch dann, wenn Sie nur mit Access arbeiten. Die »Straße« schreiben Sie folglich als *Strasse* und das Datum des Erstkontakts wird schlicht zum *Erstkontaktdatum* oder, noch einfacher, zum *Datum*. In der Programmierung sind häufig Unterstriche üblich,

Sonderzeichen

sodass unser Datumsfeld auch *Datum_des_Erstkontakts* heißen könnte. Diese Schreibweise funktioniert mit praktisch allen Datenbanken.

Vollständige Spaltenbezeichnungen

Wenn Sie Daten lediglich aus einer einzigen Tabelle auslesen, genügt die oben vorgestellte Schreibweise, bei der lediglich die Spaltennamen anzugeben sind. Die Tabelle wird ja schon in der FROM-Klausel bezeichnet. Bei Abfragen über mehrere Tabellen können aber gleiche Spaltennamen vorkommen. Sie müssen dann die Spalten mit dem kompletten Namen ansprechen. Diesen bilden Sie nach dem folgenden Schema:

Tabelle.Spalte

Aus dem Tabellen- und dem Spaltennamen *Kunden* und *Firma* wird so die folgende Zusammensetzung:

Kunden.Firma

Die schon weiter oben gezeigte Abfrage sollte dann wie folgt aussehen:

```
SELECT Kunden.KundenNr,  
       Kunden.Firma,  
       Kunden.Ort  
FROM Kunden
```

Tabellen- und Spaltennamen sind also durch einen Punkt zu verknüpfen. Die FROM-Klausel mit dem Tabellennamen ist aber auch weiterhin unverzichtbar.

Reihenfolge der Spalten

Sie sind nicht an die Reihenfolge der Spalten gebunden, die Sie in der Tabelle vorfinden. Die Reihenfolge, die Sie mit einer SQL-Abfrage erhalten, wird durch die Reihenfolge in der SQL-Anweisung bestimmt. Wenn Sie beispielsweise das Feld *Firma* an erster Stelle benötigen, ist eben eine SQL-Zeile wie die folgende erforderlich.

```
SELECT Firma, KundenNr, Ort
FROM Kunden
```

Verwenden Sie hingegen das Sternchen, erhalten Sie alle Felder in der Reihenfolge, in der diese auch in der Tabelle enthalten sind.

Datensätze auswählen

Mit der Feldliste der SELECT-Anweisung nehmen Sie eine Spaltenauswahl vor. Wollen Sie nur bestimmte Datensätze (Zeilen) ausgeben, müssen Sie eine Bedingung definieren und diese mit der WHERE-Klausel an die Abfrage anfügen. Eine Bedingung bezieht sich in der Regel auf den Inhalt einer Spalte. Die folgende Anweisung gibt beispielsweise nur Kunden aus, die aus dem Ort Dresden stammen:

```
SELECT * FROM Kunden
WHERE Ort = 'Dresden'
```

Das Feld *Ort* muss natürlich in der betreffenden Tabelle enthalten sein. Beachten Sie, dass der Wert der Bedingung in einfache Anführungszeichen gesetzt wird. Das gilt aber nur für Zeichenfolgen (Strings). Numerische Werte übergeben Sie ohne Anführungszeichen. Abbildung L2.2 zeigt das Ergebnis der Abfrage für unsere Beispieltabelle.

KundenNr	Firma	Name	Vorname	Strasse	Ort
1	Maier KG	Maier	Klaus	Maierweg 17	Dresden
5	Böger	Böger	Herbert	Großer Kamp	Dresden

Abb. L2.2: Ergebnistabelle mit ausgewählten Datensätzen

Nur Datensätze, die der Bedingung genügen, werden in die Ergebnistabelle aufgenommen. Da unsere Tabelle lediglich zwei Kunden aus Dresden enthält, liefert die Ergebnistabelle auch nur zwei Datensätze.

Das folgende Beispiel verwendet einen Größer-Vergleich. Es liefert nur Datensätze an, die im Umsatzfeld einen Wert größer als 1000 aufweisen.

Die WHERE-Klausel

```

SELECT Firma, Name, Ort, Umsatz
FROM Kunden
WHERE Umsatz > 1000

```

Das Ergebnis dieser Abfrage zeigt Abbildung L2.3. Die meisten unserer Kunden (7 von 9) erfüllen die Bedingung und landen folglich in der Ergebnistabelle.

	Firma	Name	Ort	Umsatz
	Maier KG	Maier	Dresden	2.312,33 €
	Meier GmbH	Müller	Osnabrück	22.728,18 €
	Mayer & Söhne GmbH	Schulze	Leipzig	15.004,55 €
	Böger	Böger	Dresden	98.333,56 €
	Hannibal	Kunz	Bad Essen	4.200,00 €
	Wünsche	Wünsche	Berlin	1.200,00 €
	Rumsfeld	Rumsfeld	Freiburg	2.000,00 €

Abb. L2.3: Ergebnistabelle für Größer-Vergleich

Eine Bedingung kann sehr komplex sein und sich über mehrere Felder erstrecken. Zu diesem Zweck stellt SQL Operatoren und Funktionen zur Verfügung, auf die wir später noch eingehen werden.

Verwendung von Datentypen

Jede Programmiersprache kennt eine Unzahl von Datentypen. Da die Datenverwaltung die zentrale Aufgabe einer Datenbank bildet, sind unter SQL eher noch mehr Datentypen zu finden. Wir werden die üblicherweise unterstützten Typen später noch ausführlich vorstellen. An dieser Stelle sollen uns einige Basistypen genügen, die Tabelle L2.1 zusammenfasst.

Typ	Bezeichnung	Bedeutung
Text	CHAR	Wird für Textfelder mit bis zu 255 Zeichen verwendet. Die Bezeichnung des Typs kann je nach Datenbanksystem variieren. So bezeichnet Access diesen Typ beispielsweise als TEXT.

Typ	Bezeichnung	Bedeutung
Ganzzahl	INTEGER	Wird für Ganzzahlen verwendet. Der Wertebereich kann sich von Datenbank zu Datenbank unterscheiden.
Dezimalzahl	DOUBLE	Wird für Fließkommazahlen verwendet.
Datum	DATE	Wird für Datumswerte verwendet.

Tab. L2.1: Die wichtigsten SQL-Datentypen

Kenntnisse über den Datentyp benötigen Sie nicht nur bei der Erzeugung von neuen Tabellen. Wie wir schon gesehen haben, ist der Typ auch bei der Verwendung der WHERE-Klausel regelmäßig zu berücksichtigen. Beachten Sie auch, dass die Bezeichnungen nicht immer einheitlich sind. So finden Sie den Typ CHAR unter Access auch als TEXT.

Numerische Datentypen

Alle Zahlenangaben werden häufig als numerische Typen bezeichnet. Das ist auch ganz korrekt, hilft aber dennoch nicht weiter. Gerade bei den numerischen Typen finden sich unzählige Sonderformen. So werden zunächst ganzzahlige Typen unterschieden, die es selbst wieder in verschiedenen Ausprägungen gibt. Die Unterschiede beziehen sich auf den Wertebereich, den der jeweilige Typ darstellen kann. Wir haben in Tabelle 1 nur den Typ INTEGER aufgelistet, der lediglich Werte zwischen -32.768 und +32.767 akzeptiert. Das gilt jedoch nur für Access.

Andere Datenbanksysteme definieren für den Typ INTEGER auch einen anderen Wertebereich. MySQL und andere Datenbanken unterscheiden zudem zwischen mehreren INTEGER-Typen. Einige Beispiele:

TINYINT (-128 bis 127)

SMALLINT (-32.768 bis 32.767)

MEDIUMINT (-8.388.608 bis 8.388.607)

Unterschiedliche Wertebereiche

Ähnlich problematisch ist der Umgang mit Fließkommazahlen. Dazu gehört auch der Typ *Währung*, den wir für das Feld *Umsatz* verwenden. Solche Typen können Dezimalstellen enthalten, die dann auch bei Abfragen zu berücksichtigen sind. Das folgende Beispiel zeigt, wie Sie Dezimalzahlen in einer WHERE-Klausel behandeln:

```
SELECT Firma, Ort, Umsatz
FROM Kunden
WHERE Umsatz > 100.55
```

Dezimalpunkt statt Komma

Für Dezimalzahlen ist also ein Dezimalpunkt erforderlich, nicht das im deutschsprachigen Raum übliche Dezimalkomma. Das folgende Beispiel variiert die zuletzt gezeigte Anweisung:

```
SELECT Firma, Umsatz
FROM Kunden
WHERE Umsatz >= 2000
```

Das Ergebnis der letzten Abfrage für die von uns verwendeten Daten zeigt Abbildung L2.4.

	Firma	Umsatz
	Maier KG	2.312,33 €
	Meier GmbH	22.728,18 €
	Mayer & Söhne GmbH	15.004,55 €
	Böger	98.333,56 €
	Hannibal	4.200,00 €
	Rumsfeld	2.000,00 €

Abb. L2.4: Firmen mit einem Umsatz gleich oder größer 2000

Der Umgang mit Datentypen, insbesondere mit numerischen Typen, ist nicht ganz einfach. Sie müssen sich also schon darum kümmern, welche Datentypen von Ihrem Datenbanksystem unterstützt werden und wie Sie diese sinnvoll einsetzen.

Autowert

Viele Datenbanken unterstützen inzwischen eine Variante des numerischen Typs, die in Access *Autowert* (in anderen Daten-

banksystemen *AutoIncrement*) genannt wird. Dabei handelt es sich um einen ganzzahligen Typ, der automatisch vom Datenbanksystem als fortlaufende Nummer vergeben wird. Der Typ *Autowert* wird häufig für Felder wie Kunden-, Artikel- oder Personalnummer verwendet.

Ein solches Feld enthält immer eindeutige Einträge, weil das Datenbanksystem schon dafür sorgt, dass der gleiche Wert nicht zweimal vergeben wird. Wenn Sie diesen Typ beispielsweise für die Kundennummer verwenden, müssen Sie aber akzeptieren, dass die Nummer automatisch verwaltet wird. Sie haben keine Möglichkeit, den Wert zu ändern. Die Einträge in *Autowert*-Feldern bleiben nach der Zuweisung jedoch unverändert, sodass Sie für jeden Kunden eine feste Nummer erhalten. In Abfragen können Sie diese daher auch als Kriterium einsetzen:

```
SELECT Firma, Ort, Umsatz
FROM Kunden
WHERE KundenNr = 3
```

Den Kunden, den Sie damit ermitteln, werden Sie immer unter dieser Nummer finden. Das gilt auch, wenn Sie davorliegende Datensätze löschen.

Sie sollten sich vorerst merken, dass der Typ *Autowert* immer eindeutig ist. Auch wenn Sie einen Datensatz löschen, wird dieser Wert anschließend nicht neu vergeben. Er ist dann mit dem Datensatz untergegangen.

Der Datumstyp

Sehr wichtig ist natürlich der Datumstyp. Dieser verlangt in *WHERE*-Klauseln nach einer besonderen Behandlung, wobei einzelne Datenbanksysteme auch noch unterschiedliche Angaben erwarten. Das folgende Beispiel funktioniert so nur mit Access:

```
SELECT Firma, Ort, Umsatz, Datum
FROM Kunden
WHERE Datum = #12/15/07#
```

**Autowert ist
eindeutig**

Sie müssen also nicht nur Nummernzeichen für die Einschließung verwenden, sondern auch ein im deutschsprachigen Raum unübliches Trennzeichen (/) sowie das Format MM/TT/JJ. Erst damit erhalten Sie ein Ergebnis wie das aus Abbildung L2.5. Alternativ besteht auch die Möglichkeit, den Bindestrich (-) als Trennzeichen einzusetzen.

#12-15-07#

Der im deutschsprachigen Raum übliche Punkt funktioniert jedoch nicht. Die Jahresangabe kann natürlich auch vierstellig erfolgen.

	Firma	Ort	Umsatz	Datum
	Maier KG	Dresden	2.312,33 €	15.12.2007
	Mayer & Söhne GmbH	Leipzig	15.004,55 €	15.12.2007
	Böger	Dresden	98.333,56 €	15.12.2007

Abb. L2.5: Ergebnis einer Abfrage mit Datumsvergleich

Andere Datenbanksysteme akzeptieren die von Access erwartete Darstellung nicht. Kurz: Datumswerte sind in SQL-Abfragen grundsätzlich recht problematisch. Im Zusammenhang mit Datumsfunktionen stellen wir diesen Typ noch ausführlich vor.

SELECT-Prädikate

Der SELECT-Befehl kann zusätzlich drei Prädikate verwenden: ALL, DISTINCT und DISTINCTROW. Das Prädikat ALL bestimmt, dass alle Datensätze angezeigt werden, auch solche, die identische Zeilen liefern. Dieses Prädikat ist praktisch voreingestellt, sodass Sie normalerweise darauf verzichten können.

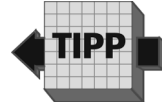
DISTINCT

Mit dem Prädikat DISTINCT unterdrücken Sie gleiche Zeilen in der Ergebnistabelle. Das folgende Beispiel zeigt dann gleiche Firmennamen nur einmal an.

```
SELECT DISTINCT Firma
FROM Kunden
```

Wie Sie aus Abbildung L2.6 ersehen, unterschlägt die erste Abfrage (mit DISTINCT) eine Maier KG. Da Sie auf diese Art bestimmte Informationen unterdrücken, sollten Sie DISTINCT nur sehr zurückhaltend einsetzen. In der Regel wird dieses Prädikat nicht benötigt.

Um das gleiche Ergebnis wie in Abbildung L2.6 zu erhalten, müssen Sie unsere Beispieltabelle um eine zusätzliche Maier KG ergänzen.



Firma	Firma
Böger	Maier KG
Hannibal	Maier KG
Maier KG	Meier GmbH
Mayer & Söhne GmbH	Mayer & Söhne GmbH
Meier GmbH	Müller OHG
Müller & Söhne	Böger
Müller OHG	Hannibal
Rumsfeld	Wünsche
Schulze	Rumsfeld
	Schulze

Abb. L2.6: Ergebnistabelle mit und ohne DISTINCT-Prädikat

Ein Nebeneffekt, der bei der Verwendung von DISTINCT auftritt, ist die Sortierung der Datensätze. Ohne das Prädikat verwendet Access für die Anzeige der Ergebnistabelle die Reihenfolge, in der die Datensätze eingegeben wurden.

Sortiereffekt

Sinnvoller ist die Verwendung von DISTINCT bei der Ausgabe der Orte, in der Kunden vorhanden sind. In der Auswahl erscheint dann nicht hundertmal der Ort *Leipzig*. Die entsprechende Abfrage hat folgende Form:

```
SELECT DISTINCT Ort
FROM Kunden
```

Wollen Sie die Anzeige von identischen Zeilen aus der Datenbank unterdrücken, müssen Sie DISTINCTROW verwenden. DISTINCTROW funktioniert üblicherweise nur bei Verwendung von

Feldern aus mehreren Tabellen. Durch die Verknüpfung der Tabellen können sich identische Zeilen ergeben, deren gleichzeitige Anzeige sich dann durch dieses Prädikat verhindern lässt.

Für unsere ersten Übungen ist `DISTINCTROW` daher ohne Bedeutung. Sofern Sie SQL-Anweisungen in einer anderen Programmiersprache einsetzen, kann es sinnvoll sein, auch das `ALL`-Prädikat zu verwenden. Ein Beispiel:

```
SELECT ALL Firma, Ort  
FROM Kunden
```

Bei der Programmpflege sehen Sie dann sofort, dass hier auch alle Datensätze gemeint sind.

Aliasnamen verwenden

Spaltennamen sind gelegentlich sehr lang und enthalten nicht selten auch Leer- oder Sonderzeichen. In diesen und anderen Fällen ist es oft sinnvoll, eigene Namen, sogenannte Aliasnamen, zu verwenden. In der Ergebnistabelle werden die betreffenden Spalten dann mit den Aliasnamen angezeigt. Einen Alias definieren Sie mit der optionalen Klausel `AS`, sodass sich folgende Syntax ergibt:

```
... Feldname AS NeuerName, ...
```

Ein Beispiel:

```
SELECT Firma AS Kunde,  
       Ort AS Firmensitz  
FROM Kunden
```

Das vorstehende Beispiel erzeugt für das Feld *Firma* den Alias *Kunde* und für *Ort* den Alias *Firmensitz*. Abbildung L2.7 zeigt das Ergebnis der Abfrage als Access-Tabelle.

In der Ergebnistabelle können Sie jetzt nur noch über die Bezeichnungen *Kunde* und *Firmensitz* auf die beiden Datenspalten zugreifen. Diese Funktion wird Ihnen bei der direkten Anwendung von SQL-Befehlen nicht sehr nützlich erscheinen. Das ändert sich jedoch, wenn Sie SQL-Anweisungen in andere

Programmiersprachen einbetten. Dort werden die Namen der Ergebnisspalten für die Weiterverarbeitung benötigt.

	Kunde	Firmensitz
	Maier KG	Dresden
	Meier GmbH	Osnabrück
	Mayer & Söhne GmbH	Leipzig
	Müller OHG	Bad Homburg
	Böger	Dresden
	Hannibal	Bad Essen
	Wünsche	Berlin
	Rumsfeld	Freiburg

Abb. L2.7: Ergebnistabelle mit Aliasnamen (Ausschnitt)

Beachten Sie, dass ein Aliasname für jede Spalte separat vergeben werden muss. Spalten, für die Sie keinen Alias vergeben, werden mit ihrem ursprünglichen Namen angezeigt. Ein Beispiel:

```
SELECT Firma AS Kunde,  
       PLZ,  
       Ort AS Firmensitz,  
       Strasse  
FROM Kunden
```

Hilfreich ist die Aliasbezeichnung auch, wenn Sie mehrere Spalten der Ursprungstabelle zu einer Spalte zusammenfassen, etwa wie im folgenden Beispiel (nur Access):

```
SELECT Firma & ', ' & Ort & ', ' & Strasse AS Kunde  
FROM Kunden
```

In diesem Beispiel setzen wir drei Spalten der Ursprungstabelle zu einer Spalte in der Ergebnistabelle zusammen. Ohne Alias würde das betreffende Datenbanksystem nicht wissen, welchen Namen es für die einzelne Ergebnisspalte vergeben soll. Schließlich sind drei Namen zu berücksichtigen.

Separate Aliasnamen für jede Spalte

Alias für kombinierte Spalten

	Kunde
	Maier KG, Dresden, Maierweg 17
	Meier GmbH, Osnabrück,
	Mayer & Söhne GmbH, Leipzig, Kleine Straße 23
	Müller OHG, Bad Homburg, Müllergasse 123
	Böger, Dresden, Großer Kamp
	Hannibal, Bad Essen,

Abb. L2.8: Zusammengesetzte Spalten mit Aliasnamen

Das Datenbanksystem erzeugt dann eine künstliche Spaltenbezeichnung, die wenig Aussagekraft hat. Erst mit dem Alias bestimmen Sie selbst, wie die Ergebnisspalte heißen soll.

Den Alias benötigen Sie auch, wenn Sie Funktionen verwenden und Datensätze gruppieren. Er wird uns daher im folgenden Text noch häufiger begegnen. Beachten Sie auch die eingefügten Kommata, diese werden als Zeichenfolgen in einfachen Anführungszeichen übergeben. Unsere Zeichenfolge besteht lediglich aus dem Komma und einem Leerzeichen. Sie können aber auch andere alphanumerische Zeichen verwenden.

Verknüpfungsoperator

Nebenbei haben wir hier schon einen Operator, den sogenannten Verknüpfungs- bzw. Verkettungsoperator (& oder auch +) kennengelernt. Das Pluszeichen dient eben nicht nur der Addition, sondern ermöglicht in Access auch die Verknüpfung von Zeichenketten. MySQL-Datenbanken verwenden dafür allerdings eine Funktion. Wir kommen noch darauf zurück.

Ausgabe sortieren

Sortierfeld bestimmen

Die Datensätze der Ergebnistabelle liegen zunächst in der Sortierung vor, in der diese in die Ursprungstabelle eingegeben wurden. Mit der Klausel ORDER BY lässt sich jedoch jede Spalte als Sortierfeld verwenden. Dabei bestimmen die Ergänzungen ASC und DESC, ob auf- oder absteigend sortiert werden soll. Standardmäßig ist eine aufsteigende Sortierung (ASC = ascending) vorgesehen. Das folgende Beispiel sortiert nach der Spalte *Firma*:

```
SELECT KundenNr, Firma, Ort
FROM Kunden
ORDER BY Firma
```

Es ist also nicht die Reihenfolge der Felder in der SELECT-Zeile, welche die Sortierung vorgibt.

KundenNr	Firma	Ort
5	Böger	Dresden
6	Hannibal	Bad Essen
10	Maier KG	Leipzig
1	Maier KG	Dresden
3	Mayer & Söhne GmbH	Leipzig
2	Meier GmbH	Osnabrück

Abb. L2.9: Nach dem Feld *Firma* sortierte Ergebnistabelle

Eine absteigende Sortierung kann für Datumsfelder sinnvoll sein. Die Datensätze mit den neuesten Datumswerten werden dann zuerst angezeigt. Sie müssen dazu das Schlüsselwort DESC verwenden:

```
SELECT KundenNr, Firma, Ort, Datum
FROM Kunden
ORDER BY Datum DESC
```

Auch bei numerischen Werten, etwa bei unserem Feld *Umsatz*, kann eine absteigende Sortierung gewünscht sein. Die Kunden mit den höchsten Umsätzen werden dann zuerst angezeigt:

```
SELECT KundenNr, Firma, Ort, Umsatz
FROM Kunden
ORDER BY Umsatz DESC
```

Die Anweisung lässt sich natürlich noch durch eine WHERE-Klausel ergänzen. So liefert die folgende Abfrage nur die Umsätze der Kunden aus Dresden:

```
SELECT KundenNr, Firma, Ort, Umsatz
FROM Kunden
WHERE Ort = 'Dresden'
ORDER BY Umsatz DESC
```

Beachten Sie unbedingt die Reihenfolge: Die ORDER BY-Klausel kommt ganz zum Schluss. Sie gehen immer in folgenden Schritten vor:

- 1** Liste der Spalten definieren
- 2** Bedingung (WHERE-Klausel) für Auswahl der Datensätze definieren
- 3** Sortierung bestimmen

Erst muss die komplette Abfrage definiert sein, inklusive der Bedingung, dann kann sortiert werden.

Sortierung über mehrere Spalten

Grundsätzlich besteht auch die Möglichkeit, über mehrere Spalten zu sortieren. Die einzelnen Spalten sind dann durch Kommata zu trennen. Die zweite Spalte kommt jedoch nur zum Zuge, wenn im ersten Sortierfeld identische Einträge enthalten sind:

```
SELECT KundenNr, Firma, Ort
FROM Kunden
ORDER BY Ort, Firma
```

Das vorstehende Beispiel sortiert zunächst nach der Spalte *Ort*. Nur wenn hier identische Einträge vorliegen, wird das Feld *Firma* berücksichtigt. Sie können natürlich auch die Reihenfolge in der Feldliste entsprechend ändern, damit die Priorität des ersten Sortierfelds sichtbar wird. Notwendig ist das jedoch nicht. Zudem lässt sich für jedes Feld die Sortierrichtung separat vorgeben. So können Sie nach dem Feld *Ort* und innerhalb des Ortes nach den höchsten Umsätzen sortieren:

```
SELECT Firma, Ort, Umsatz
FROM Kunden
ORDER BY Ort ASC, Umsatz DESC
```

Für unsere Beispieldaten erhalten Sie damit eine Ausgabe wie die aus Abbildung L2.10.

	Firma	Ort	Umsatz
	Hannibal	Bad Essen	4.200,00 €
	Müller OHG	Bad Homburg	450,00 €
	Wünsche	Berlin	1.200,00 €
	Böger	Dresden	98.333,56 €
	Maier KG	Dresden	2.312,33 €

Abb. L2.10: Sortierung nach Ort (aufsteigend) und Umsatz (absteigend)

Grundsätzlich besteht auch die Möglichkeit, über mehr als zwei Feldern zu sortieren. Das kann beispielsweise erforderlich werden, wenn Sie auf Basis einer Abfrage komplexe Reports (Berichte) erstellen wollen. Wir werden in dieser Einführung aber nicht darauf eingehen.

Zahl der Datensätze vorgeben

Eine restriktive WHERE-Klausel schränkt die Datensatzauswahl schon weitgehend ein. Sie haben damit aber noch keine Möglichkeit, die Zahl der Datensätze genau vorzugeben. Um nur eine bestimmte Zahl von Datensätzen zu erhalten, müssen Sie eine zusätzliche Klausel verwenden. Allerdings ist dieses Sprachelement nicht einheitlich geregelt. Unter MySQL ist dafür die Klausel LIMIT zuständig, während Access dafür den Begriff TOP verwendet. Das folgende Beispiel funktioniert unter Access. Es liefert nur die ersten drei Datensätze:

```
SELECT TOP 3 Firma, Ort, Umsatz
FROM Kunden
ORDER BY Umsatz DESC
```

Die Klausel macht normalerweise nur Sinn, wenn Sie die Datensätze sortieren oder mit einer WHERE-Klausel versehen. Abbildung L2.11 zeigt das Ergebnis der Abfrage.

	Firma	Ort	Umsatz
	Böger	Dresden	98.333,56 €
	Meier GmbH	Osnabrück	22.728,18 €
	Mayer & Söhne GmbH	Leipzig	15.004,55 €

Abb. L2.11: Auswahl mit der TOP-Klausel

Die Anwendung von TOP und LIMIT unterscheidet sich erheblich. TOP folgt direkt auf das Schlüsselwort SELECT und kennt nur einen Parameter: die Zahl der Datensätze. LIMIT wird als letzte Klausel eingesetzt und kann zwei Parameter aufnehmen, den ersten und den letzten Satz. Wenn es sich bei der Tabelle *Kunden* um eine MySQL-Tabelle handelt, erhalten Sie die ersten drei Datensätze mit der folgenden Anweisung:

```
SELECT * FROM Kunden
ORDER BY Umsatz
LIMIT 3
```

Wollen Sie einen Bereich vorgeben, sind die beiden Werte nach der LIMIT-Klausel durch Kommata zu trennen.