

SQL-Grundlagen

Was ist SQL	1
Die SQL-Anweisung	3
Spezielle Aufgaben lösen	9
SQL-Dialekte	14

Was ist SQL

SQL steht für »Structured Query Language«, was so viel wie strukturierte Abfragesprache bedeutet. Nebst der reinen Abfragen, also dem lesen der Daten, bestehen auch Möglichkeiten diese zu ändern bzw. neue hinzuzufügen. Im weiteren kann die Verwaltung der Datenbank, also das Anlegen, Erweitern usw. von Tabellen ebenfalls mit SQL bewerkstelligt werden.

Für Word Benutzer steht SQL vor allem im Zusammenhang mit dem Seriendruck und dem Einsatz der Database Feldfunktion von Interesse, sowie beim Datenaustausch über VBA. Words Benutzeroberfläche ermöglicht die einfache Auswahl von Spalten (Feldern), Zeilen (Datensätzen) und die Festlegung einer Sortierreihenfolge von Daten aus einer Datenquelle. Bei der Einbindungen mancher Datenquellen besteht die Möglichkeit, diese noch viel flexibler zu manipulieren, indem eigene SQL-Anweisungen erstellt werden, als Words Benutzerschnittstelle erlaubt.

In dieser Beilage stellen wir die Grundlagen der SQL Sprache vor, so dass Sie am Schluss im Stande sind, eigene SQL-Anweisungen zu erstellen, die den Seriendruck und Database Feldfunktion zu echten, vielseitigen Datenverwaltungswerkzeugen mutieren lassen.

Hinweis Beginn

Mehr über SQL können Sie in verschiedenen Fachbüchern lesen, sowie in der Access Hilfe. Sehr hilfreich ist auch die SQL-Ansicht in der Access Abfrage Fenster, um zu sehen, wie eine SQL-Anweisung aufgebaut wird.

Hinweis Ende

Die Diskussion basiert auf einem sehr einfachen Beispiel. Eine Datentabelle wie in **Tabelle.1** liegt in *SQL1.mdb* auf der CD vor, mit den Feldern »K« (eine ID-Nummer) und »T« (ein Ortsnamen) und »X« (ID-Nummer aus einer anderen Tabelle, *Tabelle_B*, mit Ländernamen).

Tabelle.1: Tabelle_A: Eine einfach Datentabelle

K	T	X
1	Sevilla	1
2	Madrid	1

K	T	X
3	Lissabon	2
4	Barcelona	1
5	Oporto	2
6	Milan	3

Wenn wir nur die Datensätze wollen, wo »K« größer ist als eins; sortiert nach Namen, in aufsteigender Reihenfolge, können wir dies problemlos in Words Abfrageoptionen festlegen. Anders sieht es jedoch aus, wenn zusätzlich nur Ortschaften mit 6 Buchstaben oder weniger im Namen übernommen werden sollen. Hier bietet uns die Abfrageoptionen keine Möglichkeit, ein solches Kriterium zu setzen. Wenn man jedoch die SQL Anweisung direkt bearbeitet, geht's:

```
SELECT [K], [T]
FROM `Tabelle_A`
WHERE LEN(T) <= 6
```

SQL ist eine äußerst leistungsfähige Sprache, die unter vielen Umständen nützlich ist

- Die Datenquelle stellt die Informationen nicht in der benötigten Form zur Verfügung
- Word hat keine integrierte Schnittstelle, um die Daten entsprechend zu formatieren

SQL ist vor allem unentbehrlich, wenn Sie

- Mit Datumsangaben rechnen müssen (ein Datum 10 Tage später als das in der Datensatz gespeicherte, beispielsweise)
- Zeichenketten auswerten oder manipulieren müssen
- Daten aus mehr als einer Quelle vereinen wollen.

Einschränkungen von SQL in Word

SQL in Word ist leider etwas beschränkt:

- Obwohl internationale ANSI Standards vorhanden sind, gibt es aus historischen Gründen mehrere »Dialekte« von SQL. Jede Datenquelle hat seinen eigenen; und manche sind leistungsfähiger als andere. Verschiedene Dialekte erreichen das gleiche Resultat auf unterschiedlichen Wegen. Man muss den passenden Dialekt für die einzelnen Datenquellen verwenden. Das kleine Beispiel oben funktioniert in Access (mit dem Jet Engine), aber in SQL Server müsste die letzte Zeile `WHERE LENGTH(T) <= 6` lauten.
- Einige Dialekte sind dermaßen rudimentär, dass sie nicht viel mehr als Words Benutzeroberfläche unterstützen. Stehen beispielsweise die Daten in einer Word Tabelle, ist Word auf seinen eigenen Dialekt angewiesen, der kein Äquivalent zu den Funktionen `Len` oder `Length` hat.
- Word akzeptiert keine SQL-Anweisung, die aus mehr als 512 Zeichen besteht. Auf dem ersten Blick erscheint das großzügig, aber viele Datenbankentabellen und -felder haben lange Bezeichnungen, die schnell 500 Zeichen belegen. (Die Grenze liegt in Database Feldfunktionen noch tiefer.)

Eine SQL-Anweisung in Word festlegen

Eine SQL-Anweisung für Daten, die in Word importiert werden, kann definiert

werden, entweder

- In der Anwendung, die die Daten zur Verfügung stellt. Dies ist nur möglich, wenn sie, wie Access oder SQL-Server, eine entsprechende Benutzerschnittstelle zur Verfügung stellt, um Abfragen bzw. Ansichten zu verwalten.

Hinweis Beginn

Access 2000 und später können sowohl Ansichten wie Abfragen enthalten. Word kann dazu in der Benutzeroberfläche nur über OLEDB (Word 2002) oder MS Query eine Verbindung erstellen. VBA bekündigt damit bei allen Verbindungsmethoden keine Mühe.

Hinweis Ende

- In Word selber. Für den Seriendruck kann sie in MS Query oder programmäßig (VBA) erstellt werden. Für Database Feldfunktionen in MS Query, oder direkt im Feldcode. Da MS Query ausschließlich mit ODBC arbeitet, müssten SQL-Anweisungen für andere Verbindungsmethoden in Database Feldfunktionen direkt im Feldcode definiert werden.

Die SQL-Anweisung

Eine SQL-Anweisung besteht aus mehreren Elementen, oder Klauseln, die mit Schlüsselwörtern bestimmt werden. Der Seriendruck und die Database Feldfunktion befassen sich mit der Auswahl und Anzeigen von Daten, was nur einen kleinen Teil der SQL-Fähigkeiten beansprucht. SQL-Anweisungen können auch Daten ändern und hinzufügen, aber diese Aspekte liegen außerhalb des Bereichs dieser Diskussion. Wir werden nur Konzepte kurz vorstellen, die für die Datenauswahl von Interesse sind.

Select

Das Schlüsselwort `Select` heißt so viel wie »Auswahl« und ist grundlegend. Es eröffnet jede Anweisung, die in Word benutzt wird. Unmittelbar nach `Select` werden die Spalten (Felder) aufgelistet, deren Daten im Ergebnis stehen sollen.

Im allgemein, so lange die Feldnamen keine Leerzeichen oder Striche enthalten, genügt es, die Feldnamen, getrennt durch Kommas und ein Leerzeichen, aufzulisten. Feldnamen mit Leerzeichen oder Strichen müssen eindeutig gekennzeichnet werden, sei es mit Anführungszeichen (""), Akzent graves (`), oder eckigen Klammern ([]). Je nach Version von Word werden das eine oder das andere dieser Zeichen bei der Makroaufzeichnung oder in einer Database Feldfunktion von Word verwendet, aber nicht immer von ihm wieder erkannt. Wir empfehlen, in Database Feldfunktionen, Feldnamen mit eckigen Klammern zu umgeben.

Gelegentlich möchte man mit anderen Feldnamen arbeiten, als diejenige, die in der Datenquelle stehen. Im Beispiel am Anfang, sind »T« und »K« nicht besonders aussagekräftig. Mit dem Schlüsselwort `AS` kann ein »Alias« festgelegt werden:

```
SELECT [K] AS ID, [T] AS Ort.
```

Diese Alias-Namen werden in Word als Spaltenüberschriften erscheinen, statt den ursprünglichen.

Wenn alle Felder einer Tabelle in das Ergebnis zu übernehmen sind, kann ein

Sternchen – * – statt den Feldnamen – nach Select stehen: `SELECT *`.

In Theorie ist es möglich, mit `Select` und `AS` eine Tabelle zu wiedergeben, ohne sich auf eine Datenquelle zu berufen. Da Word aber immer eine Datenquelle erwartet, bleiben wir bei dieser Feststellung, und schauen, wie die Datenquelle angesprochen wird.

From

`From` spezifiziert, aus welcher Tabelle oder Abfrage die Daten genommen werden. Falls die Datenquelle eine Textdatei ist, würde der Dateiname hier stehen. Beispiel: `FROM C:\Data\MeineDaten.txt`. (Wir gehen davon aus, dass eine Verbindung – »Connection« – zur Datenquelle vorhanden ist; eine SQL-Anweisung stellt keine Verbindung her, sondern bedient sich immer einer vorhandenen.)

Im Beispiel am Anfang kommen die Daten aus einer Tabelle namens *Tabelle_A*. Um alle Felder aus dieser Tabelle zu übernehmen, würde die SQL-Anweisung so lauten:

```
SELECT *
FROM Tabelle_A
```

Hinweis Beginn

Es ist allgemein üblich, die Schlüsselwörter in SQL-Anweisungen groß zu schreiben, und für jede Klausel eine neue Zeile zu beginnen. Dies ist nicht zwingend, die entsprechende SQL-Anweisung ist jedoch einfacher zu lesen und verstehen.

Hinweis Ende

Daten aus mehr als einer Tabelle

Nehmen wir an, dass auch eine *Tabelle_B* vorhanden ist, wie in **Tabelle 2**.

Tabelle 2: Beispieltabelle *Tabelle_B*

K	T	X
1	Spanien	1
2	Portugal	2

Vielleicht würden Sie versuchen, die beiden Tabellen mit einer Anweisung wie folgt zu verbinden, in der Erwartung dadurch ein Resultat wie in **Tabelle 3** zu erhalten

```
SELECT *
FROM Tabelle_B, Tabelle_A
```

Tabelle 3: Dieses Ergebnis werden Sie nicht erhalten, sondern das in **Tabelle 4**

K	T	X
1	Sevilla	1
1	Spanien	1
2	Madrid	1
2	Portugal	2
3	Lissabon	2
4	Barcelona	1
5	Oporto	2
6	Milan	3

Das Ergebnis, jedoch fällt eher aus wie in **Tabelle 4**: ein Cartesian Produkt, wo jeder Eintrag in der einen Tabelle mit jedem Eintrag in der anderen verbunden wird.

Tabelle 4: Ein Cartesian Produkt

Tabelle B.K	Tabelle B.T	Tabelle B.X	Tabelle A.K	Tabelle A.T	Tabelle A.X
1	Sevilla	1	1	Spanien	1
1	Sevilla	1	2	Portugal	2
2	Madrid	1	1	Spanien	1
2	Madrid	1	2	Portugal	2
3	Lissabon	2	1	Spanien	1
3	Lissabon	2	2	Portugal	2
4	Barcelona	1	1	Spanien	1
4	Barcelona	1	2	Portugal	2
5	Oporto	2	1	Spanien	1
5	Oporto	2	2	Portugal	2
6	Milan	3	1	Spanien	1
6	Milan	3	2	Portugal	2

Wir stehen vor zwei Problemen: wie werden die Felder ausgewählt, wenn der gleiche Feldname in beiden Tabellen vorkommt und, wie werden die Tabellen verbunden, so dass das Endergebnis wie **Tabelle 5** aussieht.

Tabelle 5: Das erwünschte Ergebnis

Stadt	Land
Sevilla	Spanien
Madrid	Spanien
Lissabon	Portugal
Barcelona	Spanien
Oporto	Portugal

Um einen Feldnamen eindeutig zu bestimmen, wird die Tabellenbezeichnung, gefolgt von einem Punkt, vor dem Feldnamen gestellt

```
SELECT Tabelle_A.T AS Stadt, Tabelle_B.T AS Land
FROM Tabelle_A, Tabelle_B
```

um das Ergebnis in **Tabelle 6** zu erhalten (was noch nicht dem Endresultat entspricht, aber ein gutes Stück näher ist).

Tabelle 6: Spalten mit den gleichen Namen eindeutig bestimmen

Stadt	Land
Sevilla	Spanien
Sevilla	Portugal
Madrid	Spanien
Madrid	Portugal
Lissabon	Spanien
Lissabon	Portugal
Barcelona	Spanien
Barcelona	Portugal
Oporto	Spanien
Oporto	Portugal
Milan	Spanien
Milan	Portugal

Wenn Sie aus irgendeinem Grund zwei Tabellen gleichen Namens haben, stehen Sie vor einem Problem. In diesem Fall müssen den Tabellen Alias-Namen zugewiesen werden:

```
SELECT A.T AS Stadt, B.T AS Land
FROM Tabelle_B A, Tabelle_B B
```

Wichtig Beginn

Das Schlüsselwort **AS** gilt nur für Feld Alias-Namen, und nicht für Tabellen Alias-Namen. Lediglich ein Leerzeichen trennt den Tabellennamen von seinem Alias.

Wichtig Ende

Alias-Namen für Tabellen haben eine wichtige Funktion: dadurch kann die Anzahl Zeichen in der SQL-Anweisung erheblich gekürzt werden. Stellen Sie sich vor, dass statt Tabellen die Daten in Textdateien wären. Die SQL Anweisung ohne Alias wäre:

```
SELECT c:\abc\Tabelle_a.txt.T, c:\abc\Tabelle_b.txt.T
FROM c:\abc\Tabelle_a.txt, c:\abc\Tabelle_b.txt
```

Aber mit Alias-Namen:

```
SELECT A.T1, B.T1
FROM c:\abc\Tabelle_a.txt A, c:\abc\Tabelle_b.txt B
```

Aus der Sicht von Word sind Tabellen Alias-Namen aus folgenden Gründen wichtig:

- Word und MS Query stellen standardmäßig Tabellen Alias-Namen vor Feldnamen, auch wenn es nicht nötig wäre. Es ist wichtig, dass Sie verstehen, was sie sind, wenn Sie eine SQL-Anweisung betrachten.
- Falls Sie eine SQL-Anweisung kürzen müssen, ist es am einfachsten, Alias-Namen einzusetzen, wo nötig, und sie zu entfernen, wo sie entbehrlich sind.

Um die zwei Tabellen so mit einander zu verbinden, dass sie das Ergebnis in **Tabelle 5** liefern, müssen wir uns zuerst mit der **Where** Klausel befassen.

Where

Mit dem **WHERE** Klausel werden Kriterien festgelegt. Auch komplexe Gebilden, die mit dem logischen **AND** und **OR** werden unterstützt. Beispiel:

```
SELECT A.K, A.T AS Name, A.X
FROM Tabelle_A A
WHERE A.K > 1 AND A.K < 5
```

gibt alle Datensätze zurück, wo die ID-Nummer zwischen 1 und 5 (aber nicht einschließlich) liegt, wie in **Tabelle 7**.

Tabelle 7: Nur Datensätze, wo der Wert in Spalte »K« zwischen 1 und 5 liegt, werden zurückgegeben

K	Name	X
2	Madrid	1
3	Lissabon	2
4	Barcelona	1

Das logische AND und logische OR

Die Bedeutung von **AND** und **OR** ist etwas verwirrend, weil sie scheinen, das Gegenteil zu machen, von dem, was die »Menschen-Logik« erwartet. **WHERE A.K > 1 AND A.K < 5** wählt *nicht* alle Datensätze, wo **K** größer ist, als eins, sowie alle Datensätze, wo **K** kleiner ist als fünf (was eigentlich keine Einschränkung wäre, da alle Werte damit berücksichtigt sind). Stattdessen werden alle Datensätze, wofür *beide* Kriterien »Wahr« sind, zurückgegeben.

Um Datensätze zu wählen, die mindestens einem mehrerer Kriterien entsprechen,

wird das logische OR eingesetzt. `WHERE A.K = 1 OR A.K = 5` würde alle Datensätze zurückgeben, wo der Wert im Feld A eins oder fünf ist, wie in **Tabelle 8**. (`WHERE A.K = 1 AND A.K = 5` würde keinen einzigen Datensatz zurückgeben, da kein Datensatz zwei verschiedenen Werte in einem Feld haben kann.)

Tabelle 8: Das Ergebnis vom logischen OR

K	T	X
1	Sevilla	1
5	Oporto	2

Die Bedingung eines Where Klauseln darf mehrere AND und OR Operatoren umfassen. Sobald dies der Fall ist, wird es unklar, in welcher Reihenfolge die Operationen auszuführen sind. Deshalb werden, genau wie beim Einsatz von mathematischen Operatoren, mit Klammern die Zusammenhänge festgelegt.

Beispiel: wir wollen, wie in **Tabelle 9**, alle Datensätze, wo der Ort (T) mit dem Buchstabe »B« oder »M« anfängt; in beiden Fällen soll der Wert des ID Felds (K) vier oder weniger betragen:

`WHERE (A.T LIKE "B*" OR A.T LIKE "M*") AND A.K<=4.`

Zuerst werden alle Datensätze, die mit B oder M anfangen gewählt, und aus diesen nur diejenigen weitergegeben, die eine ID-Nummer von 4 oder weniger.

Hinweis Beginn

Da in einer Database Feldfunktion die SQL-Anweisung in Anführungszeichen steht, müssen statt Anführungszeichen innerhalb der Anweisung Apostrophen benutzt werden: `WHERE (A.T LIKE 'B*' OR A.T LIKE 'M*') AND A.K<=4`. Bitte beachten Sie, dass Apostrophen (') und das Akzent grave (`), das die Database Feldfunktion um Tabellennamen verlangt, *nicht* das gleiche Zeichen ist.

Hinweis Ende

Das Schlüsselwort `LIKE` weist SQL an, ein Mustervergleich auf den darauf folgenden Ausdruck auszuführen. Das Sternchen (*) bedeutet – im ANSI Standard 89 – »eine unbestimmte Anzahl von Zeichen«, wie schon in den Kapiteln 7 und 10 beschrieben. Wenn Sie ein System einsetzen, das ANSI 92 benutzt, muss hier statt des Sternchens ein Prozentzeichen (%) stehen.

Tabelle 9: Gruppierte AND und OR Operatoren

K	T	X
2	Madrid	1
4	Barcelona	1

Mehrere Tabellen mit einander verknüpfen

Es ist auch möglich, die Werte zweier Spalten, oder Spalten aus verschiedenen Tabellen zu vergleichen.

Wie anfangs erwähnt, enthält das Feld »X« der Tabelle_A (Ortsnamen) eine ID-Nummer aus einer anderen Tabelle, der Tabelle_B (Länder). Dadurch besteht zwischen den zweien eine Beziehung. Über diese Beziehung ist es möglich, herauszufinden, in welchem Land eine Stadt ist: Man sucht den Eintrag in Tabelle_A.X in der Spalte K der Tabelle_B, und findet das Land in der Spalte T für diese Zeile.

Der Eintrag in Spalte X für Barcelona ist »1«. In Tabelle_B steht neben dem »1« in Spalte T »Spanien«.

Diese Beziehung kann in der SQL-Anweisung folgendermaßen ausgedrückt werden:

```
SELECT A.T AS Stadt, B.T AS Land
FROM Tabelle_A A, Tabelle_B B
WHERE A.X = B.K
```

Die das gewünschte Ergebnis in Tabelle 5 zurück gibt.

Diese Art von Verknüpfung wird ein »Equijoin« genannt. Nur Datensätze, wo der Eintrag in einer Tabelle einen genau gleichen Wert in der entsprechenden Spalte der anderen Tabelle beträgt, werden zurückgegeben. Wenn, wie »Milan« in unserem Beispiel, keinen »Partner« in der zweiten Tabelle vorhanden ist, wird der Datensatz nicht berücksichtigt.

Die meisten SQL Dialekte unterstützen zudem noch eine andere Art von »Join« (Verknüpfung), wo alle Datensätze einer der Tabellen durchgegeben werden. Es gibt das »Left Join«, sowie das »Right Join«, wobei »Left« (links) und »Right« (rechts) sich auf den Seiten neben diesem Operator beziehen. Dazu kommt das Schlüsselwort ON, wonach der Vergleich angegeben wird (statt des Schlüsselworts WHERE).

Beispiel: Alle Datensätze aus Tabelle_A werden zurückgegeben, weil »Tabelle_A A« sich links von »LEFT JOIN« befindet:

```
SELECT A.T AS Stadt, B.T AS Land
FROM Tabelle_A A LEFT JOIN Tabelle_B B ON A.X = B.K
```

Tabelle 10: Alle Datensätze aus Tabelle_A, auch wenn in Tabelle_B kein Land mit dem Eintrag in Spalte X vorhanden ist

Stadt	Land
Sevilla	Spanien
Madrid	Spanien
Lissabon	Portugal
Barcelona	Spanien
Oporto	Portugal
Milan	

Vorteile vom »Left Join« oder »Right Join« gegenüber einem »Equijoin« sind:

- Nur so können alle Datensätze aus einer von zwei Tabellen angezeigt werden (und damit wird es klar, welche Tabelle »Vorrang« hat).
- Es kann die Ausführung der Abfrage beschleunigen, wenn viele Datensätze in den Tabellen vorhanden sind.
- Den WHERE Klausel wird dadurch verständlicher.

Hinweis Beginn

»Join« Ausdrücke können ineinander verschachtelt werden, was zu komplexen SQL-Anweisungen führen kann. Wir werden in dieser kurzen Einführung nicht näher darauf eingehen. Solche SQL-Anweisungen überschreiten schnell Words 512 Zeichen Limite.

Hinweis Ende

ORDER BY

Diese Klausel bestimmt die Reihenfolge der Datensätze, nach einem oder mehreren Spalten, in auf- oder absteigender Reihenfolge. Um beispielsweise die Tabelle nach Land, dann nach Ort in aufsteigender Reihenfolge zu sortieren:


```
SELECT B.T AS Land, A.T AS Stadt
FROM Tabelle_A A LEFT JOIN Tabelle_B B ON A.X = B.K
ORDER BY B.T, A.T
```

Tabelle 11: Datensätze nach Land, dann Ort sortieren

Land	Stadt
	Milan
Portugal	Lissabon
Portugal	Oporto
Spanien	Barcelona
Spanien	Madrid
Spanien	Sevilla

Durch hinzufügen von »ASC« bzw. »DESC« kann eine auf- bzw. absteigende Reihenfolge festgelegt werden. Die folgende Anweisung sortiert die Länder in absteigender, während die Städte noch in aufsteigender Reihenfolge erscheinen.

```
SELECT B.T AS Land, A.T AS Stadt
FROM Tabelle_A A LEFT JOIN Tabelle_B B ON A.X = B.K
ORDER BY B.T DESC, A.T
```

Nicht alle Dialekte von SQL unterstützen die Verwendung von Feld Alias-Namen in der ORDER BY Klausel. Manche akzeptieren jedoch eine Ganzzahl, die die Position eines Felds in der Auflistung repräsentiert. Das obige Beispiel würde dann lauten:

```
SELECT B.T AS Land, A.T AS Stadt
FROM Tabelle_A A LEFT JOIN Tabelle_B B ON A.X = B.K
ORDER BY 1 DESC, 2
```

Spezielle Aufgaben lösen

Es gibt Anliegen bei der Übergabe von Daten an Word, die zuerst ohne VBA unlösbar erscheinen. Unterstützt der SQL-Dialekt jedoch die richtigen Befehle, können sie mit einer Anweisung ziemlich einfach gelöst werden.

Duplizierte Datensätze ausschließen

Eine Frage, die wir regelmäßig in den Newsgroups sehen, ist, wie man doppelt vorkommende Datensätze unterdrückt. Manche Dialekte haben das Schlüsselwort DISTINCT, das diese Aufgabe übernimmt:

```
SELECT DISTINCT A.X
FROM TABELLE_A A
```

würde die drei Zeilen in **Tabelle 12** zurückgeben, statt sechs Zeilen, wie der Fall wäre ohne Distinct.

Tabelle 12: Duplikate unterbinden

X
1
2
3

Berechnungen im Endergebnis

Je nach SQL-Dialekt und Verbindungsmethode stehen verschiedenen Funktionen

zur Verfügung, die für die Datenmanipulation oder für Berechnungen zur Verfügung stehen. Sie finden am Ende dieser Kapitel einige aufgelistet, in **Tabelle 18** und **Tabelle 19**. Schauen wir uns einige Beispiel an.

Den Feldinhalt abkürzen

Word hat bekanntlich keine Feldfunktionen, die Zeichenketten bearbeiten. Es ist nicht möglich, beispielsweise, nur die ersten drei Buchstaben eines Ausdrucks anzuzeigen. Alle Daten, die über eine ODBC Verbindung kommen, können mit der LEFT Funktion gekürzt werden:

```
SELECT A.T AS Stadt, UCase(Left([B].[T],3)) AS Land
FROM Tabelle_A A, Tabelle_B B
WHERE A.K = B.K
```

Hinweis Beginn

Diese Funktion steht nur für DDE oder OLEDB Verbindungen zur Verfügung, wenn die Funktion Teil des SQL-Dialekts ist. Dies ist beispielsweise der Fall für Jet (Access) und SQL-Server der Fall. Stehen die Daten jedoch in einer Word-Tabelle, die mit Words Konvertierungsfiler importiert werden, wird LEFT nicht erkannt.

Hinweis Ende

Tabelle 13: Nur die ersten drei Buchstaben des Landes anzeigen

Stadt	Land
Sevilla	SPA
Madrid	POR

Nur das erste Wort

Noch ein Beispiel der Zeichenmanipulation übergibt alle Zeichen, bis zum ersten Leerschlag. Da ein Feld unter Umständen kein Leerschlag enthalten dürfte, muss dem Feldinhalt einen hinzugefügt werden:

```
SELECT LEFT([Kundenname], INSTR([Kundenname] + ' ', ' ') - 1) AS Name
```

Datumsangaben berechnen

Ähnlich sieht die Lage bei Berechnungen mit Datumsangaben aus. Words Feldfunktionen können nur begrenzt damit umgehen, wie im Kapitel 8 beschrieben. Wir sehen häufig die Frage, wie in einem Schreiben einem Datum zehn, vierzehn oder 30 Tagen hinzugefügt werden können.

Das Jet Engine, das Access zum Grunde liegt, stellt einige Datumsberechnungsfunktionen zur Verfügung, wie DateDiff und DateAdd. Um ein Datum von heute in 10 Tagen anzuzeigen, würde die SQL-Anweisung so aussehen:

```
SELECT B.*, DateAdd("d",10,Date()) AS Verfall
FROM Tabelle_B B
```

Bitte bemerken Sie, dass »Date()« kein Feld in der Tabelle ist, sondern auch eine Funktion, die das heutige Datum zurückgibt. »"d"« steht für »Days« (Tage) und legt fest, das Tagen (und nicht etwa Monaten, Jahren oder Wochen) dem Datum hinzugezählt werden.

Je nach Datenquelle, stehen weniger leistungsfähige oder benutzerfreundliche Möglichkeiten zur Verfügung. Für eine Textdatei haben wir wieder nur über eine

ODBC Verbindung Gelegenheit, Datumsberechnungen in der SQL-Anweisung auszuführen. Um 10 Tagen einem Datum hinzu zu addieren, das von der Datenquelle übergeben wird, kann man sich die Funktion CDate bedienen. Hier ist es nicht möglich, das Zeitintervall festzulegen; eine Ganzzahl entspricht einem Tag:

```
SELECT (CDate([KurzDatum])+10) AS Plus10, [LangDatum]
FROM DataImport.txt
```

Europäische Datumsangaben in Word 2002

Wie schon im Kapitel 10 diskutiert, verwechselt der OLEDB Provider für das Jet Engine (Access und Excel) die Tages- und Monatszahlen von Datumsangaben, so fern beide 12 oder weniger betragen. Wir haben gezeigt, wie Sie in der Datenquelle die Abfrage ändern können. Falls die Quellabfrage nicht geändert werden kann, steht Ihnen über VBA die gleiche Möglichkeit zur Verfügung. Dann würden Sie im SQLStatement Argument der OpenDataSource Methode oder für die QueryString Eigenschaft eine SQL-Anweisung wie diese festlegen:

```
SELECT *, Format([RechnungsDatum]), "d-mmm-yyyy") AS SD_Datum
FROM Rechnungen
WHERE RechnungsDatum > #1/1/2002# AND RechnungsDatum < #1/1/2003#
Hinweis Beginn
```

Meistens ist es notwendig, Datumsangaben in Kriterien ausdrücklich als Datumsangaben zu bezeichnen. Die meisten SQL-Dialekte erkennen das »#« Zeichen. Ferner sollen sie im nordamerikanischen Format vorliegen.

Hinweis Ende

Aggregationsfunktionen: Count, Sum, Avg, usw

Berechnungen können auch auf Gruppen von Datensätzen ausgeführt werden. Dafür stellen viele Datenquellen Aggregationsfunktionen zur Verfügung.

Wollen Sie beispielsweise die Anzahl Städte in Tabelle_A für jedes Land in Tabelle_B wissen, würde die SQL-Anweisung so aussehen:

```
SELECT B.T AS Land, COUNT(A.T) AS Anzahl
FROM TABELLE_A A RIGHT JOIN TABELLE_B B ON A.X = B.K
GROUP BY B.T
```

Tabelle 14: Die Anzahl Städte mit der Funktion Count ermitteln

Land	Anzahl
Portugal	2
Spanien	3

Je nach SQL-Dialekt, stehen andere (oder keine) Aggregationsfunktionen zur Verfügung. Die meisten Dialekte (die von ODBC unterstützte, inklusiv) erkennen Count, Sum, Avg (average), Min (minimum), sowie Max (maximum).

Hinweis Beginn

Der Seriendruck erkennt Access Abfragen nicht, die solchen Funktionen enthalten. Dieser Umstand verhindert jedoch nicht deren Gebrauch in den SQLStatement Argumenten oder QueryString Eigenschaften von VBA. Sie können auch in Database Feldfunktionen benutzt werden.

Hinweis Ende

Die GROUP BY Klausel

So bald eine SQL-Anweisung eine Aggregatsfunktion enthält, muss sie in der GROUP BY Klausel jedes Feld auflisten, die nicht mit einer solchen Funktion berechnet wird. Im obigen Beispiel wird nach »Land« gruppiert.

Es ist auch möglich, nur das Funktionsergebnis zurückzugeben, ohne andere Felder mit einzubeziehen und gruppieren.

Die HAVING Klausel

Die HAVING Klausel wird mit Group By verwendet, um Kriterien zu setzen; sie erfüllt die Funktion der WHERE Klausel, also, für die gruppierten Datensätze. Anders ausgedrückt: WHERE legt fest, welche Datensätze aus der Datenbank gewählt werden; HAVING legt fest, welche Zeilen im Ergebnis erscheinen. Beispiel:

```
SELECT B.T AS Land, Count(A.T) AS Anzahl
FROM TABELLE_A A LEFT JOIN TABELLE_B B ON A.X = B.K
GROUP BY B.T
HAVING Count(A.T) < 3
```

zeigt nur die Funktionsergebnisse an, wo die Anzahl Datensätze in jeder Gruppe weniger als 3 sind, wie **Tabelle 15** veranschaulicht.

Tabelle 15: Das Ergebnis der Aggregatsfunktion Count mit Having Klausel

Land	Anzahl
	1
Portugal	2

Notice that the use of the LEFT join means that cities with no corresponding country are counted in a single group with a NULL country name.

Hinweis Beginn

Die Klausel einer SQL-Anweisung müssen in einer bestimmten Reihenfolge erscheinen, wie folgt: SELECT FROM WHERE GROUP BY HAVING ORDER BY. Falls es sich um eine UNION Abfrage handelt, bezieht sich ORDER BY auf die ganze Abfrage, und erscheint am Schluss.

Hinweis Ende

UNION: Tabellen an einander reihen

Im Kapitel 10 wurde eine Union Abfrage in Access erstellt, um eine Anzahl »leere« Datensätze für fehlende Etiketten festzulegen. Hier stellen wir etwas mehr Theorie vor. Diese Funktionalität wird nicht sehr oft benötigt, aber es kommt doch immer wieder vor, dass zwei unabhängige Datenquellen (Excel Tabellen oder Textdateien) mit der gleichen Struktur für den Seriendruck oder die Database Feldfunktion vorübergehend »zusammengeführt« werden sollen.

Nicht alle SQL-Dialekte unterstützen das Schlüsselwort UNION, aber die Microsoft Text ODBC-Treiber und Jet Provider tun es. Die Funktionalität steht also für die meisten Datenquellen zur Verfügung. In **Tabelle 16** sehen Sie ein Beispiel: zwei mit Tabzeichen getrennten Textdateien, eine in englischer, die zweite in deutscher Sprache, wurden zusammen gefügt.

Tabelle 16: Mit dem Schlüsselwort UNION Daten aus zwei Tabellen in einer zusammenbringen

Vorname	Name	Ort
Bill	Gates	Redmond
Peter	Jamieson	London
John	McGhie	Melbourne

Vorname	Name	Ort
Cindy	Meister	Zürich

Die SQL-Anweisung besteht, im Grunde genommen, aus zwei an einander gereihten Select Anweisungen:

```
SELECT A.[Vorname], A.[Name], A.[Ort]
FROM BspA4U02.txt A
UNION
SELECT [FirstName], [LastName], [City]
FROM BspA4U01.txt
ORDER BY 2
```

Folgende Punkte sind zu bemerken:

- Die Tabellen müssen die gleiche Struktur haben: die Spalten (Felder) im Ergebnis müssen die gleiche Datentypen haben, und in der gleichen Reihenfolge aufgelistet sein.
- Nur die Feldnamen der ersten Tabelle erscheinen im Ergebnis.
- Es ist nicht erheblich, ob die Feldnamen übereinstimmen. Im Beispiel waren die Feldnamen der zweiten Tabelle »FirstName«, LastName« und »City«. Wichtig ist nur die Reihenfolge.
- Die Datensätze werden nicht einfach nach einander aufgelistet; sie werden sortiert. Wenn man keine Sortierreihenfolge festlegt, werden sie nach einer internen Regelung sortiert (meistens einem Primärschlüssel oder der ersten Spalte).
- Sind mehrere gleiche Datensätze vorhanden, erscheint der Eintrag nur einmal. Wenn Sie auch Duplikate sehen möchten, probieren Sie mit dem Schlüsselwort UNION ALL.

ALL wird nicht von allen SQL-Dialekten unterstützt. In diesem Fall kann jedem Select-Ausdruck einer Spalte mit einem festgelegten Wert hinzugefügt werden, so dass die Datensätze der verschiedenen Tabellen sich ganz sicher unterscheiden. Es hat gleichzeitig der Vorteil, dass man auf einem Blick erkennen kann, aus welcher Tabelle die Daten stammen, wie aus **Tabelle 17** ersichtlich:

```
SELECT 1 AS N, A.[Vorname], A.[Name], A.[Ort]
FROM BspA4U02.txt A
UNION
SELECT 2 AS N, [FirstName], [LastName], [City]
FROM BspA4U01.txt
ORDER BY 2
```

Tabelle 17: Die erste Spalte wurde ausschließlich in der SQL-Anweisung definiert und stellt sicher, dass Datensätze mit den gleichen Daten aus verschiedenen Tabellen einmalig sind

N	Vorname	Name	Ort
2	Bill	Gates	Redmond
1	Cindy	Meister	Zürich
2	John	McGhie	Melbourne
1	Peter	Jamieson	London

SQL-Dialekte

Wie eingangs erwähnt, haben Datenquellen ihre eigenen SQL-Dialekte. Die grundlegende Struktur und Schlüsselwörter haben sie alle gemeinsam, aber die Syntax für spezielle Aufgaben kann sich unterscheiden. Vom besonderen Interesse sind für die Zusammenarbeit mit Word Funktionen, wie Len, Left, CDate, usw.

Wenn Sie eine Anwendung entwickeln, die mit mehreren Datenquellen zusammen genutzt werden könnte, kann diese Vielfalt an Dialekten recht mühsam sein. Wird in einer SQL-Anweisung Len eingebaut, aber die gewählte Datenquelle erkennt nur Length, erhält der Benutzer eine Fehlermeldung. Da eine der Vorteile von ODBC eine gemeinsame Schnittstelle für alle Datenquellen sein soll, wurde ODBC mit einigen »Escape sequences« ausgestattet, um dieses Problem entgegenzuwirken. Betrachten Sie folgende SQL-Anweisung:

```
SELECT A.K, A.T, A.X
FROM Tabelle_A A
WHERE { fn len(A.T) } > 2
```

{ fn len(A.T) } ist eine solche »Escape Sequence«. Der ODBC Treiber einer Datenbank ersetzt sie bei der Ausführung der SQL-Anweisung mit seiner eigenen, entsprechenden Funktion. Der Jet ODBC-Treiber (für Access), würde an dieser Stelle Len einsetzen, während der für SQL-Server Length nehmen würde.

Wenn Sie diese Methode in Betracht ziehen, bedenken Sie, dass

- Nicht alle Funktionen einer Datenquelle eine entsprechende »Escape Sequence« haben. DateAdd ist beispielsweise nicht darunter zu finden.
- Nicht alle ODBC-Treiber eine eigene, entsprechende Funktion für alle verfügbaren »Escape Sequences« haben. TimeStampDiff wird von SQL-Server aber nicht vom Jet (Access) erkannt.
- Nur ODBC mit »Escape Sequences« arbeitet. Es gibt für OLEDB keinen Äquivalent.

Die **Tabelle 18** und die **Tabelle 19** listen in der ersten Spalte die verfügbare ODBC { fn Funktion } Funktionen »Escape Sequences« auf. In den übrigen Spalten können Sie entnehmen, ob diese von ODBC-Treibern gängiger Datenbanken erkannt werden, und welche eigene, äquivalente Funktion vorhanden ist. Wenn ein Eintrag mit Klammern () umgeben ist, bedeutet dies, dass die eigene Funktion nicht die gleiche, sondern nur eine ähnliche Wirkung hat.

Tabelle 18: ODBC Funktionen für SQL-Server, Jet und Visual FoxPro, die in SQL-Anweisungen eingesetzt werden können

Standard ODBC Funktion	Gültig für Microsoft SQL Server?	Microsoft SQL Server Äquivalent	Gültig für Jet?	Jet-eigene Äquivalent	Gültig für VFP?	VFP eigene Äquivalent
Zeichenkettenfunktionen						
ASCII	✓	ASCII	✓	ASC	✓	ASC
CHAR	✓	CHAR	✓	CHR	✓	CHR
CHAR_LENGTH, CHARACTER_LENGTH	✗	✗	✗	✗	✗	✗

Standard ODBC Funktion	Gültig für Microsoft SQL Server?	Microsoft SQL Server Äquivalent	Gültig für Jet?	Jet-eigene Äquivalent	Gültig für VFP?	VFP eigene Äquivalent
CONCAT	✓	+ Operator	✓	+ Operator	✓	+ Operator
DIFFERENCE	✓	DIFFERENCE	✗	✗	✓	DIFFERENCE
INSERT	✓	STUFF	✗	✗	✓	STUFF
LCASE	✓	LOWER	✓	LCASE	✓	LOWER
LEFT	✓	LEFT	✓	LEFT	✓	LEFT
LENGTH	✓	LENGTH	✓	LEN	✓	LEN
LOCATE	✓	CHARINDEX	✓	INSTR	✗	AT
LTRIM	✓	LTRIM	✓	LTRIM	✓	LTRIM
OCTET_LENGTH	✓	✗	✗	✗	✗	✗
POSITION	✗	CHARINDEX	✗	INSTR	✗	AT
REPEAT	✓	REPLICATE	✗	✗	✓	REPLICATE
REPLACE	✓	REPLACE	✗	✗	✓	STRTRAN
RIGHT	✓	RIGHT	✓	RIGHT	✓	RIGHT
RTRIM	✓	RTRIM	✓	RTRIM	✓	RTRIM
SOUNDEX	✓	SOUNDEX	✗	✗	✓	SOUNDEX
SPACE	✓	SPACE	✓	SPACE	✓	SPACE
SUBSTRING	✓	SUBSTRING	✓	MID	✓	SUBSTR
UCASE	✓	UPPER	✓	UCASE	✓	UPPER
Einfache (nicht trigonometrische) numerische Funktionen						
ABS	✓	ABS	✓	ABS	✓	ABS
CEILING	✓	CEILING	✓	✗	✓	CEILING
FLOOR	✓	FLOOR	✓	✗	✓	FLOOR
MOD	✓	✗	✓	✗	✓	MOD
POWER	✓	POWER	✓	^ Operator	✓	^ Operator
ROUND	✓	ROUND	✓	ROUND	✓	ROUND
SQRT	✓	SQRT	✓	SQR	✓	SQRT
TRUNCATE	✓	✗	✗	(INT)	✗	(INT)
Zeit, Datums- und Intervalfunktionen						
CURRENT_DATE	✓	✗	✗	DATE	✗	DATE
CURRENT_TIME	✓	✗	✗	TIME	✗	TIME
CURRENT_TIMESTAMP	✓	GETDATE	✗	NOW	✗	DATETIME
CURDATE	✓	GETDATE	✓	DATE	✓	DATE
CURTIME	✓	✗	✓	TIME	✓	TIME
DAYNAME	✓	✗	✓ (nur neuere Versionen)	✗	✓	CDOW
DAYOFMONTH	✓	DAY	✓	DAY	✓	DAY

Standard ODBC Funktion	Gültig für Microsoft SQL Server?	Microsoft SQL Server Äquivalent	Gültig für Jet?	Jet-eigene Äquivalent	Gültig für VFP?	VFP eigene Äquivalent
DAYOFWEEK	✓	✗	✓	✗	✓	✗
DAYOFYEAR	✓	✗	✓	✗	✗	✗
EXTRACT	✓	DATEPART	✗	✗	✗	✗
HOUR	✓	✗	✓	HOUR	✓	HOUR
MINUTE	✓	✗	✓	MINUTE	✓	MINUTE
MONTH	✓	MONTH	✓	MONTH	✓	MONTH
MONTHNAME	✓	✗	✓ (nur neuere Versionen)	✗	✓	CMONTH
NOW	✓	GETDATE	✓	NOW	✓	DATETIME
QUARTER	✓	✗	✓	✗	✗	✗
SECOND	✓	✗	✓	SECOND	✓	SEC
TIMESTAMPDIFF	✓	DATEDIFF	✗	✗	✗	✗
WEEK	✓	✗	✓	✗	✓	✗
YEAR	✓	YEAR	✓	YEAR	✓	YEAR
System Funktionen						
DATABASE	✓	DB_NAME	✗	✗	✗	DBC
IFNULL	✓	ISNULL	✗	(IIF)	✓	(IIF)
USER	✓	USER_NAME	✗	✗	✗	✗
Datentyp Konversionsfunktion						
CONVERT	✓	CAST, CONVERT	✗	✗	✗	✗

Tabelle 19: ODBC Funktionen für Oracle und MySQL, die in SQL-Anweisungen eingesetzt werden können

Standard ODBC Funktion	Gültig für Oracle?	Oracle Äquivalent	Gültig für MySQL	MySQL Äquivalent
Zeichenkettenfunktionen				
ASCII	✓	ASCII	✓	ASCII
CHAR	✓	CHR	✓	CHAR
CHAR_LENGTH, CHARACTER_LENGTH	✗	✗	✓	CHAR_LENGTH, CHARACTER_LENGTH
CONCAT	✓	CONCAT, operator	✓	CONCAT
DIFFERENCE	✗	✗	✗	✗
INSERT	✓	✗	✓	INSERT
LCASE	✓	LOWER, NLS_LOWER	✓	LCASE, LOWER
LEFT	✓	SUBSTR	✓	LEFT
LENGTH	✓	LENGTH	✓	LENGTH

Standard ODBC Funktion	Gültig für Oracle?	Oracle Äquivalent	Gültig für MySQL	MySQL Äquivalent
LOCATE	✗ (always returns 0)	INSTR	✓	LOCATE, INSTR
LTRIM	✓	LTRIM	✓	LTRIM
OCTET_LENGTH	✗	✗	✓	OCTET_LENGTH
POSITION	✗	INSTR	✓	POSITION, INSTR
REPEAT	✓	✗	✓	REPEAT
REPLACE	✓	REPLACE	✓	REPLACE
RIGHT	✓	SUBSTR	✓	RIGHT
RTRIM	✓	RTRIM	✓	RTRIM
SOUNDEX	✓	SOUNDEX	✓	SOUNDEX
SPACE	✗	✗	✓	SPACE
SUBSTRING	✓	SUBSTR	✓	SUBSTRING, MID
UCASE	✓	UPPER	✓	UCASE UPPER
Einfache (nicht trigonometrische) numerische Funktionen				
ABS	✓	ABS	✓	ABS
CEILING	✓	CEIL	✓	CEILING
FLOOR	✓	FLOOR	✓	FLOOR
MOD	✓	MOD	✓	MOD
POWER	✓	POWER	✓	POWER, POW
ROUND	✓	ROUND	✓	ROUND
SQRT	✓	SQRT	✓	SQRT
TRUNCATE	✓	TRUNC	✓	TRUNCATE
Zeit, Datums- und Intervalfunktionen				
CURRENT_DATE	✗	✗	✓	CURRENT_DATE, CURDATE
CURRENT_TIME	✗	✗	✓	CURRENT_TIME, CURTIME
CURRENT_TIMESTAMP	✗	SYSDATE	✓	CURRENT_TIMESTAMP , SYSDATE, NOW
CURDATE	✓ (returns 00:00:00 time)	✗	✓	CURDATE
CURTIME	✓ (returns date and time)	✗	✓	CURTIME
DAYNAME	✗	✗	✓	DAYNAME

Standard ODBC Funktion	Gültig für Oracle?	Oracle Äquivalent	Gültig für MySQL	MySQL Äquivalent
DAYOFMONTH	✓	✗	✓	DAYOFMONTH
DAYOFWEEK	✓	✗	✓	DAYOFWEEK, WEEKDAY
DAYOFYEAR	✓	✗	✓	DAYOFYEAR
EXTRACT	✗	TO_CHAR, TO_DATE	✓	EXTRACT
HOURL	✓	✗	✓	HOURL
MINUTE	✓	✗	✓	MINUTE
MONTH	✓	✗	✓	MONTH
MONTHNAME	✗	✗	✓	MONTHNAME
NOW	✓	(SYSDATE) – i.e. date and time	✓	NOW
QUARTER	✓	✗	✓	QUARTER
SECOND	✓	✗	✓	SECOND
TIMESTAMPDIFF	✗	✗	✗	✗
WEEK	✓	✗	✓	WEEK
YEAR	✓	✗	✓	YEAR
Systemfunktionen				
DATABASE	✓	✗	✓	DATABASE
IFNULL	✓	ISNULL	✓	IFNULL
USER	✓	USER_NAME	✓	USER, SYSTEM_USER, SESSION_USER
Konvertierungsfunktion				
CONVERT	✓	CAST, CONVERT	✗	✗