3 Die Abfragesprache SQL

Access kennt zwei Methoden, um auf Daten zuzugreifen: das bewegungsorientierte Verfahren und das relationale Verfahren auf der Basis von SQL. Beim bewegungsorientierten Verfahren werden einzelne Datensätze einer Datentabelle bearbeitet, d.h., ein Datensatzzeiger deutet auf den Datensatz, der gelesen, verändert oder gelöscht werden soll. Um mehrere Datensätze zu bearbeiten, wird der Datensatzzeiger durch entsprechende Programmroutinen weiterbewegt.

Das bewegungsorientierte Zugriffsverfahren war lange Zeit, insbesondere auf dem PC, die von vielen Produkten bevorzugte Variante. Erfolgreiche Datenbanksysteme wie Borland dBase, Microsoft FoxPro oder Borland Paradox nutzen das bewegungsorientierte Verfahren.

Beim relationalen Zugriffsverfahren werden nicht einzelne Datensätze angesprochen, sondern mithilfe der Datenbankabfragesprache SQL abgefragt, sodass im Normalfall als Ergebnis eine Gruppe von Datensätzen zurückgeliefert wird. Der Zugriff mit SQL hat unter anderem den Vorteil, dass eine bessere Optimierung der Abfrage durch die Datenbank selbst vorgenommen werden kann. Die meisten der heute weit verbreiteten Datenbanken für Mainframe- oder UNIX-Rechner, wie IBM DB2, Oracle, Informix und viele andere, sind relationale Datenbanken, die über SQL abgefragt werden.

Access beherrscht beide Varianten der Verarbeitung, d.h., Sie können sowohl bewegungsorientiert als auch mit SQL auf Ihre Daten zugreifen. Der Zugriff mit SQL ist im Allgemeinen wesentlich effizienter und schneller, daher ist er dem bewegungsorientierten Verfahren vorzuziehen. In unseren Seminaren haben sich immer wieder Anwendungsentwickler über die zu langsame Verarbeitungsgeschwindigkeit von Access beschwert. In fast allen Fällen ließen sich die Geschwindigkeitsprobleme darauf zurückführen, dass die Entwickler die Leistung von SQL nicht genutzt hatten.

Wir möchten Ihnen empfehlen, sich mit der Datenbankabfragesprache SQL auseinander zu setzen, denn sie ermöglicht Ihnen die richtige Nutzung der Leistungen von Access. Insbesondere für die Programmierung mit Visual Basic sind SQL-Kenntnisse notwendig, denn hierbei werden oft die SQL-Kommandos direkt in die Programme eingebaut. In diesem Kapitel stellen wir Ihnen die Grundlagen von SQL vor und beschreiben die Einbindung in Access.

3.1 SQL – der Standard

SQL wurde vom »American National Standards Institute« (ANSI) normiert. Die ursprüngliche Normierung wurde im Laufe der Jahre weiterentwickelt. Es existieren heute mehrere durch Jahreszahlen gekennzeichnete Richtlinien: SQL-89, SQL-92, SQL-93 und SQL-99. Die verschiedenen Normvarianten differieren in Sprachumfang und Leistung.

Die verschiedenen Datenbankhersteller haben die SQL-Normen ganz oder teilweise in ihren Produkten umgesetzt. Leider haben sich die Datenbankanbieter bisher nicht auf eine einheitliche Linie festgelegt. Fast alle haben ihre Implementierung von SQL durch eigene Erweiterungen ergänzt, sodass sich die SQL-Varianten der einzelnen Produkte teilweise erheblich unterscheiden.

Microsoft Access 2002 unterstützt ANSI-89 SQL und ANSI-92 SQL. Beide Modi entsprechen weitestgehend den jeweiligen SQL-Level 1 Spezifikationen, sind mit ihnen aber nicht kompatibel. In der Microsoft Access-Hilfe werden für die von Microsoft verwendeten Modi die Bezeichnungen ANSI-89 SQL (oder auch Microsoft Jet SQL oder ANSI SQL) und ANSI-92 SQL benutzt. Die offiziellen Sprachstandards werden als ANSI-89 Level 1-Spezifikation und ANSI-92 Level 1-Spezifikation bezeichnet, was die Unterscheidung nicht eben leicht macht. Warum erwähnen wir noch SQL-89, das doch eine Untermenge von SQL-92 ist? Ganz einfach, Access 2002 arbeitet standardmäßig mit Jet-SQL, also ANSI-89 SQL. Erstellen Sie Access-Abfragen oder legen Sie die Datenherkunft von Formularen bzw. Berichten fest, so wird Jet-SQL eingesetzt. Auch alle Datenbankzugriffe über die Datenzugriffsschnittstelle »Data Access Objects«, DAO, verwenden diese SQL-Variante. Setzen Sie in Ihren Programmen die neuere Datenzugriffsschnittstelle ADO, »ActiveX Data Objects«, ein, so wird dort ANSI-92 SQL benutzt.

Allerdings ist es mit Access 2002 nun auch möglich, den ANSI SQL-Abfragemodus zu wechseln und standardmäßig auf ANSI-92-SQL umzuschalten. Dazu öffnen Sie über das Menü *EXTRAS* das Dialogfeld *Optionen* und setzten auf dem Registerblatt *Tabellen/Abfragen* unter *SQL-Server kompatible Syntax* ein Häkchen vor *In dieser Datenbank benutzen* und *Standard für neue Datenbanken*.

Beachten Sie aber, dass es nicht zu empfehlen ist, mit unterschiedlichen Abfragemodi erstellte Abfragen zu vermischen, da dadurch Laufzeitfehler oder unerwartete Ergebnisse auftreten können. Im Normalfall sollten Sie es bei den Standardeinstellungen belassen. Damit bleiben die Abfragen auch zu den Vorgängerversionen von Access kompatibel. Auch wenn Sie ANSI-92 SQL nicht zum Standard machen, können Sie über ADO trotzdem darauf zugreifen.

Seit der Vorgängerversion Access 2000 ermöglicht Microsoft mithilfe der Datenzugriffsschnittstelle ADO den Zugriff auf Datenbanken mithilfe von OLE DB-

Treibern (siehe Teil 7). Der OLE DB-Treiber für Access-Datenbanken, Version 4.0, ermöglicht den Einsatz von SQL-92. ADO kann nur aus VBA-Programmen oder mit so genannten Access-Projekten (siehe Kapitel 26) verwendet werden.

OLE DB ist die Spezifikation einer allgemeinen Schnittstelle zu beliebigen Datenquellen. Mit DAO wird auch ODBC (Open Database Connectivity) unterstützt. ODBC ist der Vorgänger von OLE DB.

Access ist über die ODBC-Schnittstelle in der Lage, auf SQL-Datenbanken wie IBM DB2, Oracle, Informix, mySQL und viele andere zuzugreifen. Allerdings ist der Zugriff auf SQL-89-Befehle beschränkt, es sei denn, man umgeht Access-SQL durch so genannte Pass-Through-Abfragen, in denen beliebige SQL-Kommandos erlaubt sind.

Erst mit OLE DB und ADO können Sie die volle Leistung der Server-Datenbanken und SQL-92 ausnutzen.

Der Sprachumfang von SQL setzt sich aus zwei Teilen zusammen: der »Data Definition Language« (DDL) und der »Data Manipulation Language« (DML). Mit DDL lassen sich Tabellenstrukturen anlegen, ändern und löschen sowie Indizes bestimmen. Die DML dient zur Abfrage der Daten bzw. zum Verändern und Löschen von Datenbeständen. In Access wird im allgemeinen nur mit der DML gearbeitet, da Tabellen und Indizes mit den Werkzeugen in Access erstellt werden.

3.2 SQL und Access-Abfragen

Der Access-Anwender hat im Normalfall wenig mit SQL zu tun, denn die Benutzerschnittstelle von Access verbirgt SQL hinter »Abfragen«. Eine Abfrage wird von Access immer in die Sprache SQL übersetzt. Sie können im Abfragefenster jederzeit auf die SQL-Darstellung umschalten bzw. die SQL-Befehle verändern, ergänzen oder sie in die Zwischenablage kopieren, um die Befehle in anderen Access-Programmteilen zu verwenden.

3.2.1 Query By Example

Die Benutzerschnittstelle zur Erstellung von Abfragen, die Entwurfsansicht, wird als »Query By Example« (QBE) bezeichnet. QBE ist sehr anwenderfreundlich, denn mithilfe von QBE können auch Anfänger und Ungeübte SQL-Abfragen erzeugen, ohne die Abfragesprache SQL selbst zu beherrschen.

Für den Programmierer hat QBE den Vorteil, dass sich schnell und frei von Schreibfehlern SQL-Abfragen erstellen lassen, die dann in Formularen, Berichten und Modulen verwendet werden können. Wir empfehlen Ihnen, alle SQL-Befehle als Abfragen zu testen und abzulegen, um aus Ihren Programmen auf die gespeicherten Abfragen zuzugreifen.

Es gibt allerdings einige SQL-Befehle, die nicht in der Entwurfsansicht eingegeben werden können. Diese müssen Sie daher direkt im SQL-Darstellungsfenster erfassen. Wir werden Ihnen die Sonderfälle im Laufe des Kapitels beschreiben.

Eine weitere Besonderheit von Access sind Nachschlagefelder, die wir für unsere Cocktail-Beispielanwendung auch vielfach eingesetzt haben. In Abfragen wertet Access die Nachschlagefelder aus, d.h., es wird immer der nachgeschlagene Wert gezeigt. Beispielsweise ist die *ZutatenNr* in der Tabelle *tblCocktailzutaten* eigentlich ein `Long Integer`, der auf die entsprechende Zutat verweist. Da die *ZutatenNr* als Nachschlagefeld vereinbart ist, wird die jeweilige Zutat aus *tblZutat* im Ergebnis einer Abfrage dargestellt. In den Beispielen dieses Kapitels werden wir die automatische Nachschlagefunktion vernachlässigen, da sonst viele Beispiele nicht zu überschauen wären.

3.2.2 SQL-Views

Access-Abfragen entsprechen teilweise den in den SQL-Standards definierten »Views«. Eine View ist eine Sicht auf die Daten von Tabellen. Dazu wird eine SQL-Abfrage unter einem Namen abgelegt. Auf eine View kann wie auf eine Tabelle zugegriffen werden.

Allerdings besteht ein erheblicher Unterschied zwischen Standard-SQL und Access, denn in Access ist es möglich, Datensätze einer Abfrage, auch wenn sie auf mehreren Tabellen oder anderen Abfragen basiert, zu verändern. In vielen anderen SQL-Datenbanken können immer nur Tabellen oder Abfragen bearbeitet werden, denen nur eine Tabelle zugrunde liegt. Microsoft nennt diese bearbeitbaren Abfragen »Dynasets«. Dynasets erleichtern die Arbeit mit SQL-Datenbanken erheblich.

Außerdem können Abfragen Parameter enthalten, die in reinen SQL-Views nicht enthalten sind.

3.3 Auswahlabfragen mit SELECT

Der folgende Abschnitt stellt Ihnen die vielfältigen Möglichkeiten der Auswahlabfragen mit dem SQL-Befehl `SELECT` vor. Sie werden am weitaus häufigsten eingesetzt und sind die Grundlage jeglicher Arbeit mit SQL.

3.3.1 Einfache SELECT-Abfragen

Mit `SELECT` werden Daten ausgewählt. Im einfachsten Fall besteht eine `SELECT`-Abfrage aus vier Teilen: dem Schlüsselwort `SELECT`, der Angabe der gewünschten Spalten der Tabelle, dem Schlüsselwort `FROM` und dem Namen einer Tabelle oder Abfrage.

```
SELECT * FROM tblCocktail;
```

Das Sternchen »*« dient dabei als Platzhalter für die Ausgabe aller Spalten der angegebenen Tabelle oder Abfrage, das Semikolon am Ende zeigt den Abschluss des Befehls an, der auch mehrzeilig angegeben werden kann. Das folgende Bild zeigt die SQL-Abfrage in der Entwurfsansicht.

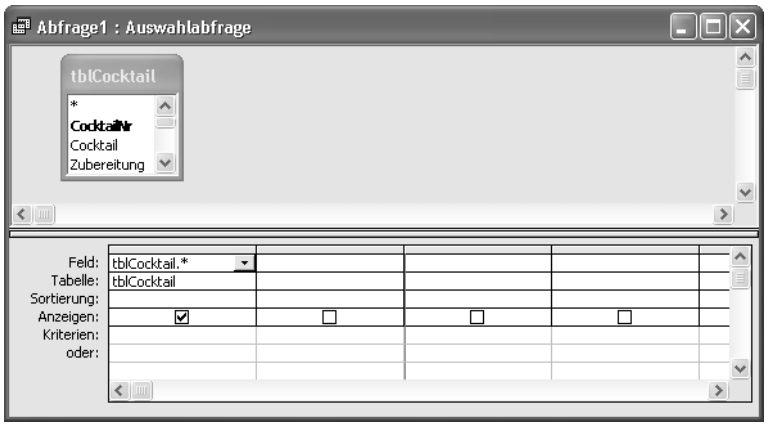


Bild 3.1: Entwurfsansicht

Mit der Schaltfläche *SQL-Ansicht* können Sie in die SQL-Darstellung wechseln, die im folgenden Bild mit dem SQL-Befehl zu sehen ist, den die oben abgebildete Abfrage erzeugt. Darin wird dem Sternchen automatisch die Tabellenbezeichnung vorangestellt.

Bild 3.2: SQL-Darstellung

Werden mehrere Spalten explizit ausgewählt, werden die Spaltenbezeichnungen mit vorangestelltem Tabellennamen aufgenommen. Der Tabellenname wird von Access immer vor der Feldbezeichnung angeordnet, unabhängig davon, ob die Option *ANSICHT Tabellennamen* für die Entwurfsansicht selektiert ist. Im folgenden Beispiel werden drei Spalten der Tabelle *tblCocktail* in die Abfrage aufgenommen.

```
SELECT tblCocktail.Cocktail, tblCocktail.Zubereitung,
tblCocktail.Alkoholgehalt
FROM tblCocktail;
```

Es können maximal bis zu 255 Spalten in einer Abfrage enthalten sein, wobei die Gesamtgröße des Ergebnisses nicht größer als ein GigaByte sein darf.

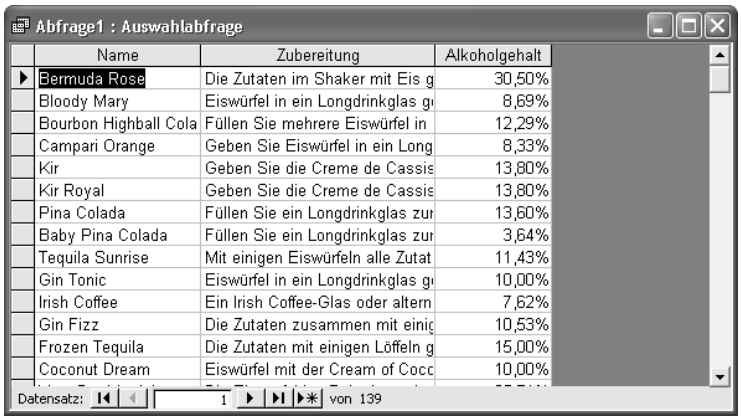


Bild 3.3: Ergebnisdarstellung

In der Ergebnisdarstellung werden die Bezeichnungen der Spalten ohne den Tabellennamen als Spaltenüberschriften verwendet. Sollten bei einer Abfrage mehrerer Tabellen die Feldbezeichnungen übereinstimmen, werden in den Spaltenüberschriften die Tabellennamen mit angezeigt.

Die Überschriften der Spalten können vom Benutzer neu benannt werden. Die neuen Bezeichnungen werden in SQL dem Tabellennamen mit dem Befehlswort `AS` wie in

```
SELECT tblCocktail.Cocktail AS Name, tblCocktail.Alkoholgehalt AS Prozent
FROM tblCocktail;
```

nachgestellt. Die neuen Namen werden als Alias-Namen bezeichnet. In der Entwurfsansicht werden Alias-Namen mit einem Doppelpunkt vorangestellt, wie es das nächste Bild illustriert.

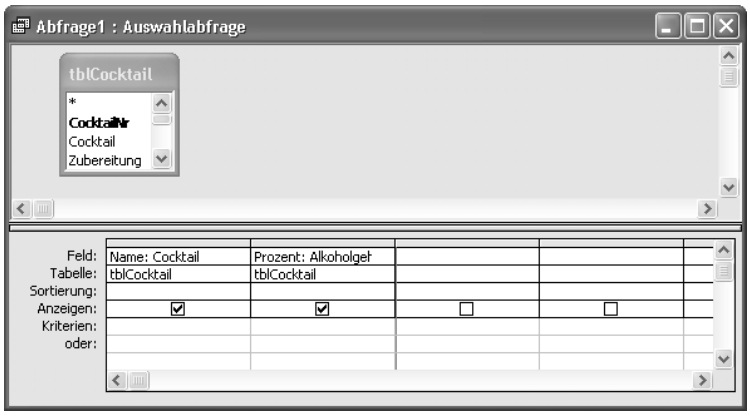


Bild 3.4: Vorangestellte neue Bezeichnungen

Für die Benennung von Datenfeldern erlaubt Access alle Zeichen bis auf den Punkt, das Ausrufezeichen, das Akzentzeichen und die eckigen Klammern. Um Feldnamen mit Leer- oder anderen Sonderzeichen in SQL-Befehlen darstellen zu können, werden diese von Access mit eckigen Klammern eingeschlossen.

```
SELECT  tblCocktail.[Dies ist ein Testfeld]
FROM tblCocktail;
```

Beachten Sie dabei, dass der Name der Tabelle außerhalb der eckigen Klammern vorangestellt wird. Auch Alias-Namen können entsprechend aufgebaut werden.

```
SELECT tblCocktail.Cocktail AS [Name des Cocktails]
FROM tblCocktail;
```

3.3.2 Daten sortieren

Mithilfe des `ORDER BY`-Befehls können Sie nach bis zu zehn Kriterien gleichzeitig sortieren. Die zu sortierenden Spalten lassen sich im Abfragefenster mit der Option *Sortierung Aufsteigend* oder *Absteigend* anordnen. In SQL beschreibt

```
SELECT tblCocktail.Cocktail
FROM tblCocktail
ORDER BY tblCocktail.Cocktail;
```

eine Sortierreihenfolge, womit die Spalte *Cocktail* aufsteigend sortiert wird. Eine absteigende Anordnung der Datensätze wird durch das SQL-Befehlswort `DESC`, als Abkürzung für »descending«, erreicht.

```
SELECT tblCocktail.Cocktail
FROM tblCocktail
ORDER BY tblCocktail.Cocktail DESC;
```

Standardmäßig wird aufsteigend sortiert. Sie können die aufsteigende Sortierung durch `ASC` für »ascending« explizit anzeigen.

Soll nach mehreren Spalten zur gleichen Zeit sortiert werden, werden die Spalten des Abfragefensters, für die eine Sortierreihenfolge angegeben ist, von links nach rechts hintereinander nach `ORDER BY` aufgeführt.

```
SELECT tblCocktail.Alkoholgehalt, tblCocktail.Cocktail
FROM tblCocktail
ORDER BY tblCocktail.Alkoholgehalt DESC , tblCocktail.Cocktail;
```

Viele andere SQL-Datenbanken verwenden auch die Schreibweise

```
SELECT tblCocktail.Alkoholgehalt, tblCocktail.Cocktail
FROM tblCocktail
ORDER BY 1 DESC, 2;
```

für die Sortierung. Dabei beschreiben die Ziffern, die wievielte Spalte sortiert werden soll. Prinzipiell können Sortierungen so auch in Access vorgegeben werden, allerdings ist die Darstellung im Abfragefenster nicht auf den ersten Blick einsichtig.

! Sortierungen: Normalerweise werden alle Tabellen nach dem Primärschlüssel sortiert, wenn Sie keinen speziellen Befehl zur Sortierung angeben. Das gilt für Abfragen, Formulare und Berichte. Allerdings ist Access hierbei inkonsequent, denn bei Auflistungen von Datensätzen in Listen- oder Kombinationsfeldern auf

Formularen und Berichten verwendet Access die Reihenfolge der Dateieingabe. Für die Steuerelemente muss dann nachträglich für die Datenherkunft eine Sortierung vereinbart werden.

Sortiergeschwindigkeit

Die richtige Auswahl der Felder, nach denen sortiert werden soll, hat maßgeblichen Einfluss auf die Ausführungsgeschwindigkeit von Abfragen. Sortieren Sie möglichst nur nach indizierten Feldern, denn bei allen anderen Spalten muss Access die Sortierung »ad hoc« durchführen. Bei kleineren Tabellen (< 500 Datensätze) ist das noch vertretbar, bei größeren steigt der Zeitbedarf erheblich.

Auf zwei unscheinbare Geschwindigkeitsfallen bei Abfragen, Sortierung und Filter sollten Sie besonders achten. In der Datenblattansicht kann mit den Schaltflächen *Aufsteigend sortieren* bzw. *Absteigend sortieren* nach jedem beliebigen Feld geordnet werden. Zusätzlich kann in der Datenblattansicht ein Filter vereinbart werden. Sortierung und Filter der Datenblattansicht werden von Access auf das Abfrageergebnis angewandt, d.h., zunächst wird die Abfrage durchgeführt und danach sortiert bzw. gefiltert. Während eine normale Abfrage von Access kompiliert wird, um ein Maximum an Geschwindigkeit zu erreichen, unterbleibt dies für die Einstellungen der Datenblattansicht.

Im Grunde genommen spricht nichts gegen eine nachträgliche Sortierung oder Filterung, die nur den Anwender betrifft, der sie durchführt. Allerdings werden das Sortierkriterium und die Filterbedingung mit der Abfrage gespeichert. Sie können die gespeicherten Kriterien im Dialogfeld *Abfrageeigenschaften* anschauen, welches Sie in der Entwurfsansicht über den Befehl *ANSICHT Eigenschaften* aufrufen.

Bild 3.5: Dialogfeld Abfrageeigenschaften

Übrigens steckt eine weitere Fehlermöglichkeit in den Zeilen zu *Filter* und *Sortiert nach*. Wie in Bild 3.5 zu sehen, wird der Name der Abfrage mit in den Bedingungen aufgeführt. Benennen Sie die Abfrage um, bleibt hier der alte Name stehen und führt beim nächsten Aufruf des Filters zu einer Fehlermeldung.

3.3.3 Abfrageeinschränkungen mit WHERE

Mithilfe des SQL-Befehlswortes `WHERE` können Sie Ihre Abfragen einschränken. Die SQL-Abfrage

```
SELECT tblCocktail.CocktailNr, tblCocktail.Cocktail,
tblCocktail.Zubereitung, tblCocktail.Alkoholgehalt
FROM tblCocktail
WHERE (((tblCocktail.CocktailNr)=123));
```

gibt nur die Daten des Cocktails mit der *CocktailNr* 123 zurück. Die mehrfache Klammerung um die `WHERE`-Bedingung wurde von Access selbsttätig hinzugefügt und ist in diesem Fall überflüssig, aber unschädlich. Die Klammern können gelöscht werden, allerdings werden sie von Access wieder eingefügt. Im weiteren Verlauf des Kapitels haben wir auf die überflüssigen Klammern verzichtet, damit die SQL-Befehle übersichtlicher sind.

Die Spalten, die in der `WHERE`-Bedingung angegeben werden, müssen nicht zwangsläufig mit ausgegeben werden. Der Befehl

```
SELECT tblCocktail.Zubereitung, tblCocktail.Alkoholgehalt
FROM tblCocktail
WHERE tblCocktail.Cocktail="Kir Royal";
```

ist ausreichend. Im Abfragefenster wird die Abfrage wie in Bild 3.6 dargestellt, wobei kein Häkchen im Feld *Anzeigen* gesetzt wurde.

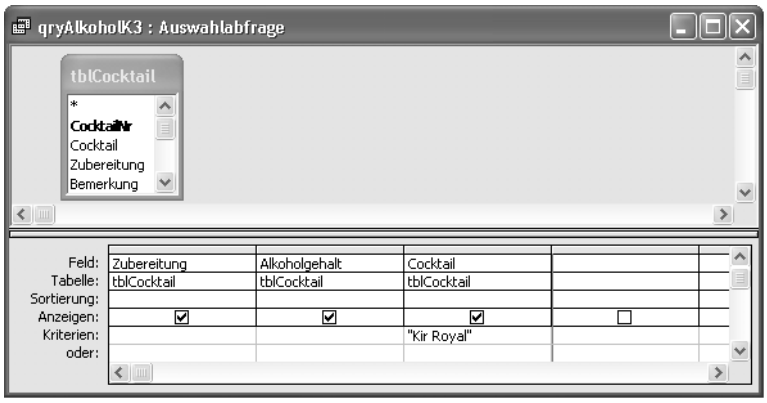


Bild 3.6: Abfrage nach dem Cocktailnamen

Vergleichsoperatoren

In einer `WHERE`-Klausel sind die Vergleichsoperatoren <, <=, >, >=, = und <> zulässig. Darüber hinaus stehen Ihnen die Operatoren `BETWEEN`, `LIKE` und `IN` zur Verfügung, die wir weiter unten beschreiben.

Bei allen Vergleichen müssen die Datentypen der Operanden miteinander verträglich sein, d.h., Sie können keine Zahl mit einem Text vergleichen. Allerdings können Sie alle Konvertierungsfunktionen von Access zur Umwandlung von Datentypen benutzen, wie wir es im weiteren Text erläutern.

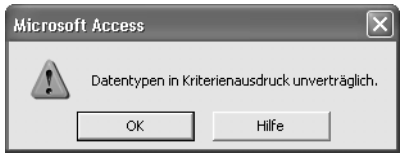


Bild 3.7: Warnmeldung bei unverträglichen Datentypen

Möchten Sie eine Liste erhalten, die die Namen aller Cocktails mit einem Alkoholgehalt von weniger als 15% aufführt, können Sie dazu die einfache Abfrage

```
SELECT  tblCocktail.Cocktail, tblCocktail.Alkoholgehalt
FROM tblCocktail
WHERE tblCocktail.Alkoholgehalt < 0.15;
```

verwenden. Im nächsten Bild ist die Definition zur Auswahl aller Drinks mit einem Alkoholgehalt zwischen 10% und 20% in der Entwurfsansicht zu sehen.

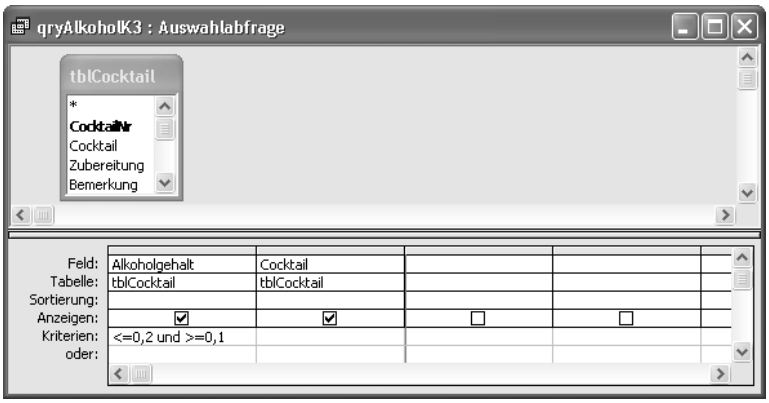


Bild 3.8: Alkoholgehalt zwischen 10% und 20%

Die Abfrage wird in SQL durch

```
SELECT tblCocktail.Cocktail, tblCocktail.Alkoholgehalt
FROM tblCocktail
WHERE tblCocktail.Alkoholgehalt <= 0.2
       AND tblCocktail.Alkoholgehalt >= 0.1;
```

umgesetzt. Die beiden Bedingungen wurden mit `AND` verbunden, d.h., ein Datensatz wird ausgewählt, wenn beide Bedingungen erfüllt sind. Über die Verknüpfung von Bedingungen mit `AND` und `OR` lesen Sie bitte im Folgenden den Abschnitt »Logische Operatoren«.

Zahlen und Datumswerte: In der Entwurfsansicht werden Zahlen und Datumswerte in dem in der Systemsteuerung vereinbarten Länderformat dargestellt, in unserem Fall in dem in Deutschland üblichen Format mit Dezimalkommas sowie Punkten als Datumstrennzeichen. In der SQL-Ansicht dagegen verwendet Access die englischen Schreibweisen, d.h., bei Zahlen wird der Dezimalpunkt verwendet, und Datumswerte wie 24.12.2002 werden zu 12/24/2002 umgesetzt.

Der BETWEEN-Operator

Das gleiche Ergebnis wie die zuletzt besprochene SQL-Abfrage liefert

```
SELECT tblCocktail.Cocktail, tblCocktail.Alkoholgehalt
FROM tblCocktail
WHERE tblCocktail.Alkoholgehalt BETWEEN 0.1 AND 0.2;
```

mit dem Operator `BETWEEN (ZWISCHEN)`. `BETWEEN` wird sehr oft zur Auswahl von Zeitabschnitten eingesetzt, beispielsweise um alle Cocktails zu ermitteln, die im November 2001 geändert wurden.

```
SELECT tblCocktail.Cocktail, tblCocktail.CocktailGeändert
FROM tblCocktail
WHERE tblCocktail.CocktailGeändert BETWEEN #11/1/2001# AND #11/30/2001#;
```

Die beiden Datumswerte werden in der SQL-Abfrage in der englischen Schreibweise dargestellt, obwohl sie im Abfragefenster im deutschen Format zu sehen wären. Möchten Sie die unterschiedliche Darstellung vermeiden, können Sie dazu die Funktion `DatWert()` bzw. englisch `DateValue()` verwenden. Die Funktion benötigt einen Parameter in Form einer Datumszeichenfolge. Die Zeichenfolge in der Funktion bleibt damit sowohl in der Entwurfs- als auch in der SQL-Ansicht unverändert.

```
SELECT tblCocktail.Cocktail, tblCocktail.CocktailGeändert
FROM tblCocktail
WHERE tblCocktail.CocktailGeändert BETWEEN DateValue("1.11.2001") AND
DateValue("30.11.2001");
```

Access-Funktionen: Der Einsatz von Access-Funktionen wie *DatWert()/ DateValue()* schränkt den Einsatz der SQL-Kommandos auf Access ein. Sie lassen sich ohne Modifikationen nicht mit anderen SQL-Datenbanksystemen verwenden. Prinzipiell können in Access-SQL alle Access-Funktionen, auch benutzerdefinierte, verwendet werden.

Der Operator LIKE

Ein häufig eingesetzter SQL-Operator ist `WIE` bzw. `LIKE`. Der Operator erlaubt die generische Suche nach Datensätzen. So ermittelt

```
SELECT tblCocktail.Cocktail, tblCocktail.Zubereitung
FROM tblCocktail
WHERE tblCocktail.Cocktail LIKE "B*";
```

alle Cocktails, deren Namen mit dem Buchstaben »B« beginnen. Access kennt zwei Platzhalterzeichen: das »*« für beliebig viele Zeichen und das »?« für ein beliebiges Zeichen. Die Bequemlichkeit des Einsatzes von Platzhalterzeichen geht in vielen Fällen auf Kosten der Geschwindigkeit, denn teilweise können `LIKE`-Bedingungen nur unzureichend von Access optimiert werden. Das gilt insbesondere, wenn die Bedingung mit einem »*« oder »?« beginnt, wie in der Form

```
SELECT tblCocktail.Cocktail, tblCocktail.Zubereitung
FROM tblCocktail
WHERE tblCocktail.Cocktail LIKE "*olada*";
```

Mit der Abfrage werden die im folgenden Bild gezeigten Cocktails ermittelt.

Wir empfehlen, `LIKE`-Bedingungen zu vermeiden, die mit Sternchen oder Fragezeichen beginnen, denn bei größeren Datenbeständen kann das Leistungsverhalten für den Benutzer inakzeptabel werden, da Access zur Lösung der Abfrage alle Datensätze einer Tabelle überprüfen muss.

Es empfiehlt sich daher, Spalten, die oft mit LIKE abgefragt werden, zu indizieren, da sie prinzipiell schneller abgefragt werden.

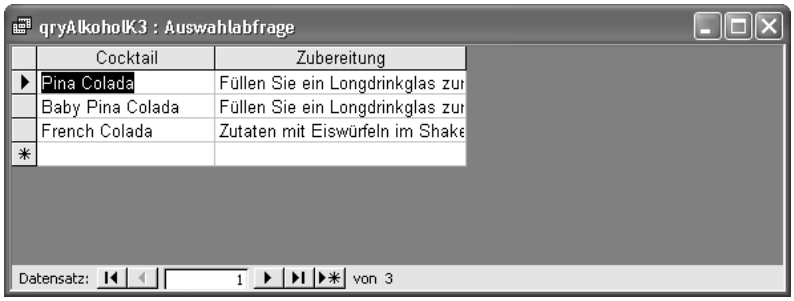


Bild 3.9: Ergebnis der Abfrage

! Unterschiedliche Platzhalterzeichen: Die in Microsofts Jet-SQL verwendeten Platzhalterzeichen entsprechen nicht der ANSI-SQL-Spezifikation. Die offiziellen SQL-Spezifikationen verwenden anstelle von »*« das Zeichen »%« bzw. »_« für »?«. Haben Sie die Access-Abfragen auf den ANSI-92 SQL-Standard (*EXTRAS Optionen SQL Server-kompatible Syntax (ANSI 92)*) umgestellt, so müssen die Platzhalterzeichen des Standards benutzt werden.

Mit `LIKE` werden normalerweise nur Zeichenfolgen verglichen, also Datenfelder vom Typ *Text* oder *Memo*. Die Abfrage von *Memo*-Feldern sollte jedoch vermieden werden, denn sie können nicht indiziert werden.

Prinzipiell ist es auch möglich, Felder vom Typ *Zahl, Währung* oder *Datum* mit `LIKE` auszuwerten, wie es die nächsten Beispiele zeigen. Der SQL-Befehl

```
SELECT tblCocktail.Cocktail, tblCocktail.CocktailNr
FROM tblCocktail
WHERE tblCocktail.CocktailNr LIKE "1*5";
```

ermittelt beispielsweise die Cocktails mit den Cocktailnummern 105, 115, 125 usw.

Als weiteres Platzhalterzeichen können Sie »#« verwenden, das für eine beliebige Ziffer von 0 bis 9 innerhalb einer Zeichenfolge steht. Mit der Bedingung

```
SELECT Gerätebezeichnung
FROM Inventarliste
WHERE Seriennummer LIKE "ABC###DE";
```

ließen sich beispielsweise alle Geräte mit den Seriennummern ABC000DE bis ABC999DE ermitteln.

Eine leistungsfähige Funktion bietet der Mustervergleich mit Zeichenlisten. Hierzu werden in eckigen Klammern die Zeichen oder Zeichenbereiche angegeben, die in einer Zeichenfolge erkannt werden sollen. Der SQL-Befehl

```
SELECT tblCocktail.Cocktail
FROM tblCocktail
WHERE tblCocktail.Cocktail LIKE "[ABC]*";
```

gibt als Ergebnis alle Cocktailnamen zurück, die mit A, B oder C beginnen. Das gleiche Resultat wird erreicht, wenn die Bedingung in `"[A-C]*"` umformuliert wird, dabei bezieht sich die Mustererkennung nur auf ein Zeichen. Die folgende Tabelle gibt einen Überblick über die verschiedenen Varianten zur Mustererkennung.

Wenn Bereiche angegeben werden, müssen sie in aufsteigender Reihenfolge festgelegt werden, `[9-0]` oder `[M-B]` sind also nicht zulässig.

Tabelle 3.1: Mustervarianten

Muster	Bedeutung
[A-Z]	Alle Großbuchstaben von A bis Z
[a-zA-Z]	Alle Groß- und Kleinbuchstaben von A bis Z
[0-9]	Alle Ziffern
[!A]	Alle Buchstaben außer A
[!A-C]	Alle Buchstaben bis auf A, B und C
[-0-9] oder [0-9-]	Um ein Minuszeichen zusätzlich zu den Ziffern zu erkennen, muss es ganz an den Anfang oder an das Ende der Musterfolge gestellt werden
[1-37-9]	Die Ziffern 1,2,3,7,8 und 9 werden erkannt

Der Operator IN

Mithilfe des Operators `IN` lässt sich überprüfen, ob der Inhalt eines Datenfeldes in einer vorgegebenen Liste von Werten vorkommt. Die SQL-Abfrage

```
SELECT tblCocktail.Cocktail
FROM tblCocktail
WHERE tblCocktail.Cocktail IN ("Alaska","Gin Fizz","Kir");
```

liefert alle Zeilen der Tabelle *tblCocktail*, deren Cocktailnamen in der IN-Liste auftauchen. Weitere Einsatzfälle des `IN`-Operators stellen wir Ihnen im Abschnitt »Unterabfragen« vor. In der Entwurfsansicht muss als Trennzeichen statt des Kommas ein Semikolon eingegeben werden.

Logische Operatoren

SQL kennt die logischen Operatoren `AND (UND)`, `OR (ODER)` und `NOT (NICHT)`, die in `WHERE`-Bedingungen verwendet werden können. Access-SQL erlaubt je nach Komplexität der Abfrage bis zu 40 `ANDs` in einer `WHERE`-Klausel. Die folgende Abfrage ermittelt alle Zutaten, die keine Spirituosen sind oder weniger als 20% Alkohol aufweisen.

```
SELECT tblZutat.Zutat, tblZutat.Alkoholgehalt, tblZutat.Art
FROM tblZutat
WHERE (NOT tblZutat.Art="Spirituose") OR
(tblZutat.Alkoholgehalt < 0.2);
```

Der Sonderfall NULL

Der Wert `NULL` zeigt die »Leere« eines Datenfeldes an. Ein Feld hat den Wert `NULL`, wenn es keinen definierten Inhalt hat. `NULL` darf nicht mit der Zahl 0, einer leeren Zeichenfolge "" oder einem Leerzeichen verwechselt werden. Der SQL-Befehl

```
SELECT tblCocktail.Cocktail, tblCocktail.Bemerkung
FROM tblCocktail
WHERE tblCocktail.Bemerkung IS NULL;
```

ermittelt alle Cocktails, für die das Bemerkungsfeld leer ist. `NULL` ist kein echter Wert und kann deshalb nicht direkt in einem Vergleich verwendet werden. Eine Bedingung wie `tblCocktail.Bemerkung = NULL` ist daher nicht korrekt. Access wandelt in manchen Fällen die fehlerhafte Bedingung um.

Die eigentlich nicht korrekte Bedingung `Feld <> NULL` wird von Access automatisch zu `Feld IS NOT NULL` (bzw. `IST NICHT NULL` in der Entwurfsansicht) umgesetzt.

Ohne doppelte Datensätze

Mithilfe des Prädikats `DISTINCT` kann die Ausgabe von mehrfach vorhandenen identischen Datensätzen unterdrückt werden. In der Entwurfsansicht rufen Sie über *ANSICHT Eigenschaften* das Dialogfeld *Abfrageeigenschaften* auf, in dem die Option *Keine Duplikate* eingeschaltet werden kann.

Der folgende SQL-Befehl listet alle Zutatennummern der Zutaten aller Cocktails auf. Jede Zutat wird nur einmal aufgeführt, auch wenn sie in mehreren Cocktails verwendet wird.

```
SELECT DISTINCT tblCocktailzutaten.ZutatenNr
FROM tblCocktailzutaten
ORDER BY tblCocktailzutaten.ZutatenNr;
```

! Verzicht auf DISTINCT: Sie können auf das Prädikat `DISTINCT` verzichten, wenn Sie unter den Ausgabefeldern Ihrer Abfrage den Primärschlüssel der Tabelle aufführen. Da ein Primärschlüssel immer eindeutig ist, benötigen Sie `DISTINCT` nicht, und die Abfrage wird schneller ausgeführt.

Die Besten

Das Prädikat `TOP` ermöglicht Ihnen, die ersten *n* bzw. die ersten *n* Prozent der mit einer Abfrage ermittelten Datensätze auszugeben. Die SQL-Abfrage

```
SELECT TOP 5 tblZutat.Zutat, tblZutat.Alkoholgehalt
FROM tblZutat
WHERE tblZutat.Alkoholgehalt > 0.2;
```

zeigt die ersten fünf Zutaten mit einem Alkoholgehalt von 20% und mehr an. Beachten Sie hierbei, dass eine Sortierung das Ergebnis beeinflussen kann. Die SQL-Abfrage oben wurde um die absteigende Sortierung nach dem Alkoholgehalt ergänzt. Die Abfrage

```
SELECT TOP 5 tblZutat.Zutat, tblZutat.Alkoholgehalt
FROM tblZutat
WHERE tblZutat.Alkoholgehalt > 0.2
ORDER BY tblZutat.Alkoholgehalt DESC;
```

ermittelt nun mehr als fünf Datensätze, nämlich die Zutaten mit den fünf höchsten unterschiedlichen Alkoholgehalten, so wie es das folgende Bild aufzeigt.

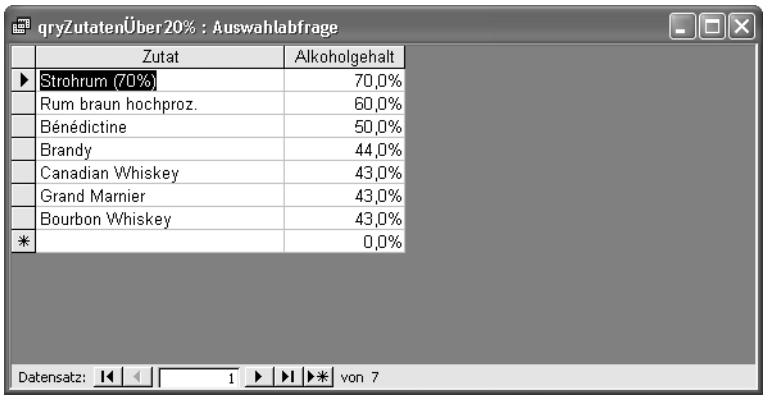


Bild 3.10: Spitzenwerte beim Alkoholgehalt

In der Entwurfsansicht stellen Sie das `TOP`-Prädikat mithilfe des entsprechenden Kombinationsfeldes auf der Symbolleiste oder der Abfrageeigenschaft *Spitzenwerte* ein. Standard-SQL verwendet hier im Gegensatz zu Access-SQL den Befehl `LIMIT TO nn ROWS`, um die Anzahl der Ergebniszeilen zu beschränken.

`NULL`-Werte werden vom `TOP`-Prädikat als kleinste Zahl bzw. als alphabetisch kleinste Zeichenfolge ausgewertet. Es empfiehlt sich in vielen Fällen, durch den Eintrag `IS NOT NULL` Nullwerte für die entsprechenden Spalten nicht mit auszuwerten.

Ausführungsberechtigung

Eine weitere Einstellung des Dialogfeldes *Abfrageeigenschaften* ist die *Ausführungsberechtigung*. Mit ihrer Hilfe bestimmen Sie, wer die Abfrage einsetzen darf. Selektieren Sie die Auswahl *Besitzer*, wird der SQL-Abfrage der Abschnitt `WITH OWNERACCESS OPTION` zugefügt.

```
SELECT tblZutat.Zutat, tblZutat.Alkoholgehalt
FROM tblZutat
WHERE tblZutat.Alkoholgehalt > 0.2
ORDER BY tblZutat.Alkoholgehalt DESC
WITH OWNERACCESS OPTION;
```

Diese Erweiterung ist Access-spezifisch und nicht kompatibel mit anderen SQL-Datenbanken. Zum Thema Berechtigungen für Besitzer und Benutzer in Access lesen Sie bitte Kapitel 24, »Datensicherheit«.

3.3.4 Verknüpfte Tabellen

Die Stärke relationaler Datenbanken liegt in der Möglichkeit der Verknüpfung von Tabellen. Die Verknüpfungen können in der Form von Beziehungen und von Nachschlagefeldern vorgegeben sein, sie können aber auch ad hoc in Abfragen definiert werden.

Verknüpfungen mit zwei Tabellen

In Standard-SQL werden Verknüpfungen zwischen Tabellen mithilfe der `WHERE`-Bedingung aufgebaut. Eine einfache Verbindung zweier Tabellen hat die Form

```
SELECT Cocktail, Zubereitung, ZutatenNr, Menge
FROM tblCocktail, tblCocktailzutaten
WHERE tblCocktail.CocktailNr=tblCocktailzutaten.CocktailNr;
```

Die Tabellennamen wurden bei der Aufzählung der Spalten weggelassen, da sie nur bei nicht eindeutigen Spaltenbezeichnungen notwendig sind, allerdings fügt Access die Tabellennamen immer selbsttätig hinzu. Die Abfrage wird in der Entwurfsansicht wie im nächsten Bild dargestellt.

Access baut aufgrund der SQL-Abfrage keine Verbindungslinie zwischen den beiden Tabellen auf, sondern stellt die Beziehung über den Eintrag der Verknüpfungsbedingung als Kriterium her. Das Ergebnis der Abfrage erbringt die richtigen Datensätze, allerdings können diese nicht bearbeitet werden. Access stellt das Ergebnis nur als Snapshot und nicht als Dynaset dar.

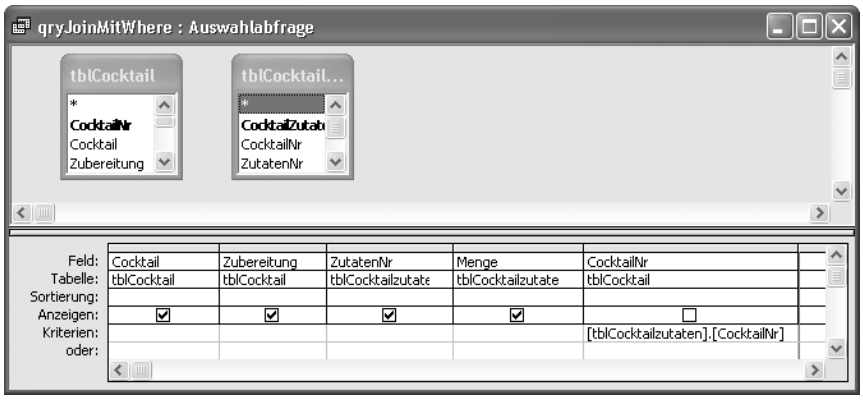


Bild 3.11: Einfache Verknüpfung in der Entwurfsansicht

Um die volle Leistung von Access im Hinblick auf Arbeitsgeschwindigkeit und bearbeitbare (»updatable«) Dynasets zu erhalten, müssen die Access-eigenen Verknüpfungsbefehle (`JOIN`) eingesetzt werden. Access wertet für die Verknüpfungen zwischen Tabellen die definierten Beziehungen im Dialogfeld *Beziehungen* (*EXTRAS Beziehungen*) aus.

Das folgende Beispiel soll den Sachverhalt erläutern. Wir haben dazu die Tabellen *tblCocktail* und *tblCocktailZutaten* in einen Abfrageentwurf aufgenommen, um die Zubereitung und die Zutaten aller Cocktails zu ermitteln. Für die beiden Tabellen ist eine 1:n-Beziehung mit referentieller Integrität definiert. Access zieht daher selbsttätig eine Verbindungslinie zwischen den beiden Tabellen, im vorliegenden Fall aufgrund der referentiellen Integrität eine dickere Linie mit den Bezeichnungen »1« und »∞«.

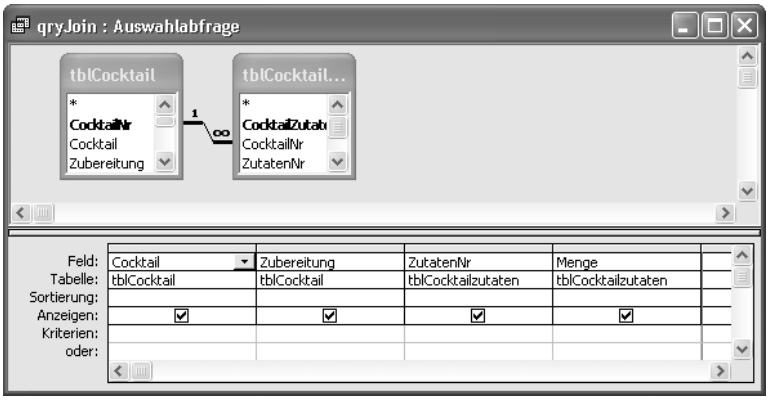


Bild 3.12: Zutaten und Zubereitung der Cocktails

Access wandelt die Festlegung der Entwurfsansicht um in den SQL-Befehl

```
SELECT tblCocktail.Cocktail, tblCocktail.Zubereitung,
tblCocktailzutaten.ZutatenNr, tblCocktailzutaten.Menge
FROM tblCocktail INNER JOIN tblCocktailzutaten ON tblCocktail.CocktailNr
= tblCocktailzutaten.CocktailNr;
```

Für die Verknüpfung wird das Befehlswort `INNER JOIN` eingesetzt. Mithilfe eines `INNER JOINs` werden Datensätze aus zwei Tabellen kombiniert, sobald übereinstimmende Werte in den Feldern der `ON`-Bedingung in beiden Tabellen gefunden werden. Durch den Einsatz von `INNER JOIN` erzeugt Access nach Möglichkeit bearbeitbare Dynasets.

Durch einen Doppelklick auf die Verbindungslinie zwischen den Tabellen in der Entwurfsansicht erhalten Sie das Dialogfeld *Verknüpfungseigenschaften*. Zusätzlich zu dem `INNER JOIN`, der mit der ersten Option des Dialogfeldes selektiert wird, kann Access zwei Inklusionsverknüpfungen (Outer Joins) erstellen. Der Unterschied zwischen Inner und Outer Join besteht darin, dass bei einem Inner Join in der Abfrage nur Datensätze erzeugt werden, für die in beiden Tabellen übereinstimmende Werte vorhanden sind. Für einen Outer Join hingegen werden alle Werte der einen Tabelle verwendet, falls in der zweiten Tabelle passende Werte vorhanden sind, werden sie dann ebenfalls aufgeführt.

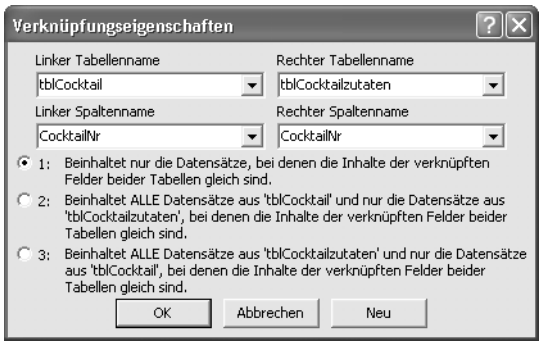


Bild 3.13: Dialogfeld Verknüpfungseigenschaften

So erzeugt die zweite Option der Verknüpfungseigenschaften eine linke Inklusionsverknüpfung; es werden also alle Datensätze der Tabelle links von dem Befehlswort `LEFT JOIN` und dazu nur die aus der rechten Tabelle ermittelt, die eine Entsprechung in der linken Tabelle haben.

```
SELECT tblCocktail.Cocktail, tblCocktail.Zubereitung,
tblCocktailzutaten.ZutatenNr, tblCocktailzutaten.Menge
```

```
FROM tblCocktail LEFT JOIN tblCocktailzutaten ON tblCocktail.CocktailNr =
tblCocktailzutaten.CocktailNr;
```

Für die dritte Option wird der Prozess umgekehrt: Der `RIGHT JOIN` nimmt alle Zeilen der rechten und nur die verknüpften Zeilen der Tabelle links des Befehls.

```
SELECT tblCocktail.Cocktail, tblCocktail.Zubereitung,
tblCocktailzutaten.ZutatenNr, tblCocktailzutaten.Menge
FROM tblCocktail RIGHT JOIN tblCocktailzutaten ON tblCocktail.CocktailNr
= tblCocktailzutaten.CocktailNr;
```

Die Felder, die für die Verknüpfung der Tabellen verwendet werden, in unserem Fall `tblCocktail.CocktailNr` und `tblCocktailzutaten.CocktailNr`, müssen vom gleichen Datentyp sein. Dabei ist zu beachten, dass bei Spalten vom Typ *Zahl* Gleitkommazahlen (`Single`, `Double`) und Ganzzahlen (`Byte`, `Integer`, `Long Integer`) nicht verknüpft werden können, sondern nur Gleitkomma- bzw. Ganzzahlen miteinander. *AutoWert*-Felder entsprechen `Long Integer`-Werten. Eine Verknüpfung von `Memo`- oder `OLE-Objekt`-Feldern ist nicht möglich.

Die Namen der Spalten für eine Verknüpfung müssen nicht übereinstimmen, d.h., es ist nicht zwingend notwendig, eine *CocktailNr* mit einer *CocktailNr* zu vergleichen, sondern Sie könnten, wenn Sie die Feldbezeichnungen beim Entwurf Ihrer Tabellen entsprechend gewählt haben, auch eine *CocktailNummer* mit einer *CNr* verknüpfen.

Eine typische Anwendung eines `RIGHT JOIN`-Befehls zeigt das nächste Beispiel. Es sollen alle Cocktails mit den dazugehörigen Gruppen ermittelt werden. Mit der SQL-Abfrage

```
SELECT tblCocktail.Cocktail, tblGruppe.Gruppe
FROM tblGruppe RIGHT JOIN tblCocktail ON tblGruppe.GruppeNr =
tblCocktail.GruppeNr
ORDER BY tblCocktail.Cocktail;
```

erhalten Sie wie gewünscht alle Cocktails mit ihrer Gruppenzugehörigkeit, die zudem nach den Cocktailnamen sortiert wurden.

Wäre die Abfrage mit einem `INNER JOIN` formuliert worden, hätte Access als Ergebnis nur die Cocktails ermittelt, für die eine Gruppe definiert ist.

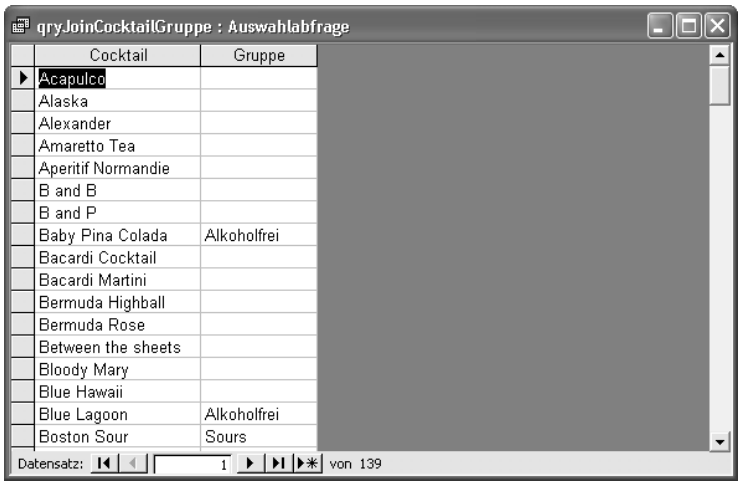


Bild 3.14: Alle Cocktails mit Gruppenzugehörigkeit

Drei und mehr Tabellen verknüpfen

Sie können bis zu 32 Tabellen miteinander verknüpfen. Im folgenden Beispiel wurden vier Tabellen verbunden, um zu einem Cocktail die Zutaten mit Mengenangabe und Einheit auszugeben.

```
SELECT tblCocktail.Cocktail, tblZutat.Zutat, tblCocktailzutaten.Menge,
tblEinheiten.Einheit
FROM tblZutat INNER JOIN (tblEinheiten INNER JOIN (tblCocktail INNER JOIN
tblCocktailzutaten ON tblCocktail.CocktailNr =
tblCocktailzutaten.CocktailNr) ON tblEinheiten.EinheitenNr =
tblCocktailzutaten.EinheitenNr) ON tblZutat.ZutatenNr =
tblCocktailzutaten.ZutatenNr;
```

Diese SQL-Abfrage erzeugt die folgende Entwurfsansicht.

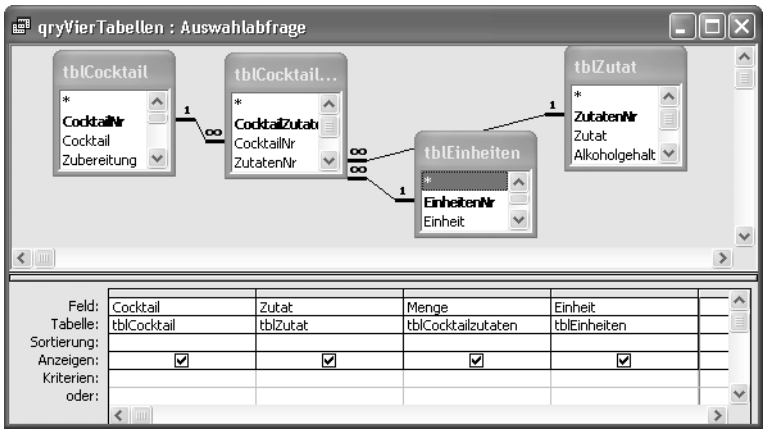


Bild 3.15: Vier verknüpfte Tabellen

Beachten Sie bei der Verknüpfung mehrerer Tabellen, dass `INNER JOINs` und `LEFT` bzw. `RIGHT JOINs` nicht beliebig verschachtelt werden können. Es ist möglich, einen `LEFT` oder `RIGHT JOIN` innerhalb eines `INNER JOINs` zu verwenden, aber nicht umgekehrt. Die SQL-Abfrage

```
SELECT tblCocktail.Cocktail, tblZutat.Zutat, tblCocktailzutaten.Menge,
tblEinheiten.Einheit
FROM tblZutat LEFT JOIN (tblEinheiten INNER JOIN (tblCocktail INNER JOIN
tblCocktailzutaten ON tblCocktail.CocktailNr =
tblCocktailzutaten.CocktailNr) ON tblEinheiten.EinheitenNr =
tblCocktailzutaten.EinheitenNr) ON tblZutat.ZutatenNr =
tblCocktailzutaten.ZutatenNr;
```

führt zu der Fehlermeldung im nächsten Bild, da die `INNER JOINs` innerhalb des `LEFT JOINs` angeordnet sind.

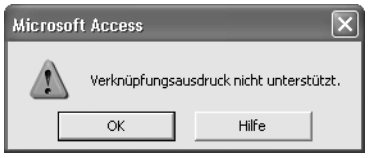


Bild 3.16: Warnhinweis bei fehlerhaften Verknüpfungen

Die folgende Abfrage soll alle Cocktails und die dazugehörigen Kategorien auflisten. Dabei soll aufgrund der *n:m*-Beziehung zwischen *tblCocktail* und *tblKategorie* ein Cocktail mehrfach ausgegeben werden, wenn er mehreren Kategorien

angehört. Die n:m-Beziehung wurde mithilfe der Tabelle *tblCocktailKategorien* aufgebaut.

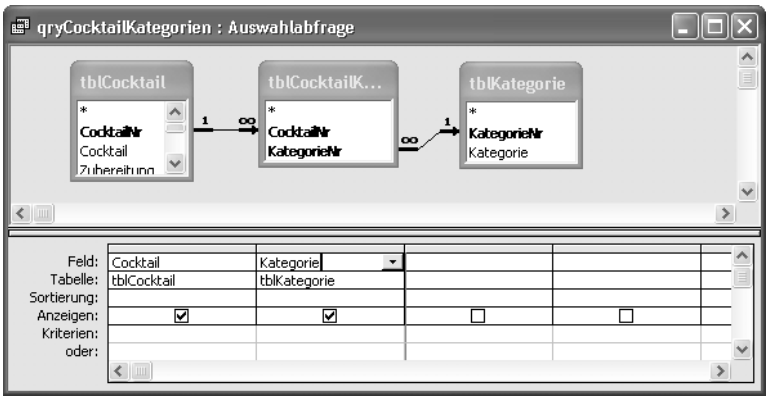


Bild 3.17: Verknüpfung einer n:m-Beziehung

Access wandelt die Abfragedefinition aus der Entwurfsansicht zum SQL-Befehl

```
SELECT tblCocktail.Cocktail, tblKategorie.Kategorie
FROM tblKategorie RIGHT JOIN (tblCocktail LEFT JOIN tblCocktailKategorie
ON tblCocktail.CocktailNr = tblCocktailKategorie.CocktailNr) ON
tblKategorie.KategorieNr = tblCocktailKategorie.KategorieNr;
```

`DISTINCTROW` ist eine Erweiterung von Access, die benötigt wird, um in bestimmten Fällen Updates in Dynasets zu erlauben. Wir möchten die Wirkung von `DISTINCTROW` in einem Beispiel erläutern. Wir haben dazu in der Entwurfsansicht die folgende Abfrage zusammengestellt, um alle Cocktails aufzulisten, die eine oder mehrere Zutaten mit einem Alkoholgehalt von 40% oder mehr enthalten.

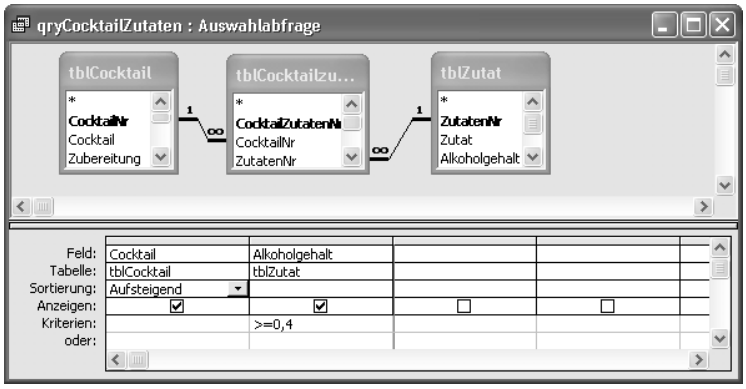


Bild 3.18: Ermittlung aller Cocktails mit hochprozentigen Zutaten

Der von Access erstellte SQL-Befehl hat die Form:

```
SELECT tblCocktail.Cocktail, tblZutat.Alkoholgehalt
FROM tblZutat INNER JOIN (tblCocktail INNER JOIN tblCocktailzutaten ON
tblCocktail.CocktailNr = tblCocktailzutaten.CocktailNr) ON
tblZutat.ZutatenNr = tblCocktailzutaten.ZutatenNr
WHERE tblZutat.Alkoholgehalt>=0.4
ORDER BY tblCocktail.Cocktail;
```

Das Abfrageergebnis führt Cocktails, die mehr als eine hochprozentige Zutat enthalten, mehrfach auf.

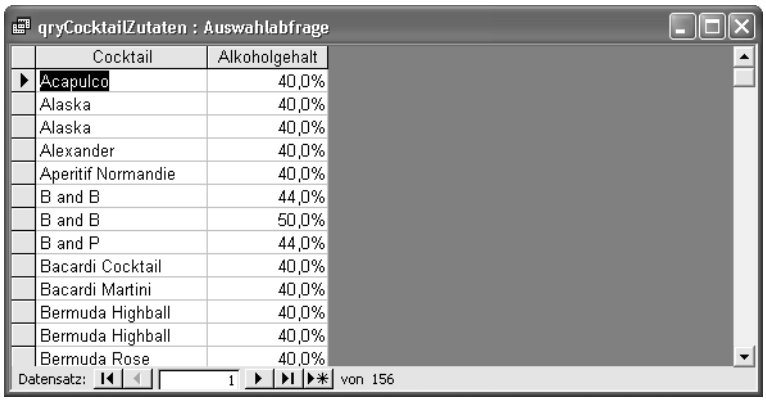


Bild 3.19: Ergebnis der Abfrage

Wird die SQL-Abfrage um das Prädikat `DISTINCT` (Abfrageeigenschaft *Keine Duplikate*) ergänzt, wird jeder Cocktail nur einmal gezeigt. Allerdings sind die Cocktailnamen nun nicht veränderbar, d.h., das Dynaset ist schreibgeschützt. Verändern Sie `DISTINCT` zu dem Access-eigenen Prädikat `DISTINCTROW`, werden hier die gleichen Ergebnisdaten ausgegeben wie mit `DISTINCT`, das Dynaset kann aber bearbeitet werden, d.h., in unserem Fall können die Namen der Cocktails modifiziert werden. (In den meisten Abfragen – nämlich in Abfragen mit mehreren Tabellen *und* mindestens einer Tabelle in der `FROM`-Klausel, die nicht in der `SELECT`-Anweisung genannt wird - erhält man die selben Ergebnisse.)

```
SELECT DISTINCTROW tblCocktail.Cocktail, tblZutat.Alkoholgehalt
FROM tblZutat INNER JOIN (tblCocktail INNER JOIN tblCocktailzutaten ON
tblCocktail.CocktailNr = tblCocktailzutaten.CocktailNr) ON
tblZutat.ZutatenNr = tblCocktailzutaten.ZutatenNr
WHERE tblZutat.Alkoholgehalt>=0.4
ORDER BY tblCocktail.Cocktail;
```

Sie können ebenso im Eigenschaftsfenster für *Eindeutige Datensätze* die Option `Ja` auswählen, um das Prädikat `DISTINCTROW` in der SQL-Abfrage zu erzeugen.

Bild 3.20: Nur eindeutige Datensätze zulassen

Selbstbezügliche Verknüpfungen

In einigen Fällen können Sie durch die Verknüpfung einer Tabelle mit sich selbst (self join) die gewünschten Ergebnisse erzielen. Wir möchten Ihnen ein Beispiel anhand der Zutatenliste vorstellen. Dort sind für viele der Zutaten Alternativen abgelegt, beispielsweise können Sie anstelle von Champagner auch Sekt verwenden. Dafür wird im Alternativfeld des Datensatzes zu Champagner die Zutatennummer von Sekt angegeben.

In unserem Beispiel soll bestimmt werden, welche Zutat als Alternative für eine andere Zutat vereinbart ist. In der Entwurfsansicht nahmen wir dazu die Tabelle *tblZutat* zweimal auf. Access benennt die zweite Zutatentabelle automatisch in *tblZutat_1* um. Mit der Maus wird eine Beziehungslinie zwischen *tblZutat.ZutatenNr* und *tblZutat_1.Alternativ* aufgebaut. Die Feldtypen der Felder *ZutatenNr* als *AutoWert* und *Alternativ* als *Long Integer* sind für eine Verknüpfung miteinander geeignet. Als Ausgabespalten selektierten wir aus beiden Tabellen das Feld *Zutat*.

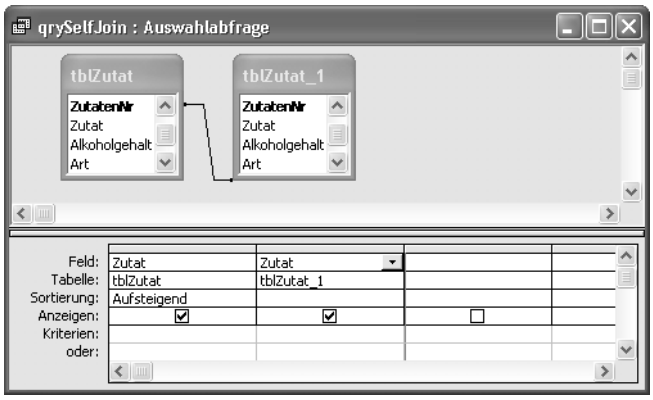


Bild 3.21: Entwurfsansicht für Liste mit Alternativzutaten

Aus der Festlegung in der Entwurfsansicht resultiert der folgende SQL-Befehl:

```
SELECT tblZutat.Zutat, tblZutat_1.Zutat
FROM tblZutat INNER JOIN tblZutat AS tblZutat_1 ON tblZutat.ZutatenNr =
tblZutat_1.Alternativ
ORDER BY tblZutat.Zutat;
```

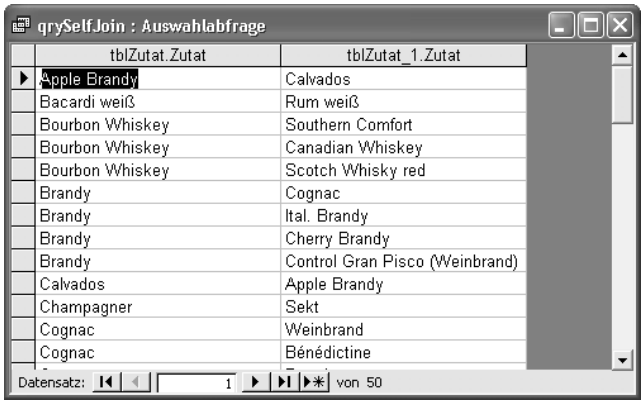


Bild 3.22: Liste der Alternativzutaten

In der Tabelle *tblZutat* ist festgelegt, dass für jede Zutat nur eine Alternative erfasst werden kann. Theoretisch hätten wir auch eine Tabelle mit Alternativen anlegen und diese mit der Zutatenliste verknüpfen können. Da aber die meisten Zutaten mit nur einer Alternative auskommen, haben wir den Aufwand vermieden. Sollten Sie nun fragen, wie es dazu kommt, das in Bild 3.22 für Bourbon Whiskey mehrere Alternativen aufgeführt sind, dann lautet die Antwort, dass Bourbon Whiskey die Alternative der rechts aufgeführten Zutaten ist.

Stellen Sie sich vor, Sie möchten einen Cocktail mixen, in dem Cognac vorkommt, den Sie leider nicht im Hause haben. Die für die Zutat »Cognac« definierte Alternative ist Brandy, der aber leider auch nicht in Ihrer Hausbar vorrätig ist. Abhilfe könnte die folgende SQL- Abfrage

```
SELECT tblZutat.Zutat
FROM tblZutat AS tblZutat_1 INNER JOIN tblZutat ON tblZutat_1.ZutatenNr =
tblZutat.Alternativ
WHERE tblZutat_1.Zutat = "Cognac"
ORDER BY tblZutat.Zutat;
```

schaffen, die alle Zutaten ermittelt, für die Cognac als Alternative eingetragen ist. Vielleicht findet sich dann eine dieser Spirituosen in Ihrem Bestand.

Verknüpfungen mit anderen Operatoren

Access ist in der Lage, Verknüpfungen mit den Operatoren >, >=, <, <= und <> aufzubauen, wie es im Standard für SQL-89 bzw. SQL-92 vorgesehen ist. Verknüpfungen mit anderen Operatoren als dem Gleichheitszeichen werden verhältnismäßig selten eingesetzt. Darüber hinaus wird ihr Einsatz in Access dadurch erschwert, dass Access sie in der Entwurfsansicht nicht darstellen kann.

Wir möchten mithilfe einer »>«-Verknüpfung eine Abfrage erstellen, die zu einem bestimmten Cocktail alle Cocktails auflistet, deren Alkoholgehalt geringer ist.

```
SELECT tblCocktail.Cocktail, tblCocktail.Alkoholgehalt,
tblCocktail_1.Cocktail, tblCocktail_1.Alkoholgehalt
FROM tblCocktail INNER JOIN tblCocktail AS tblCocktail_1 ON
tblCocktail.Alkoholgehalt > tblCocktail_1.Alkoholgehalt
ORDER BY tblCocktail.Cocktail,tblCocktail_1.Alkoholgehalt Desc;
```

Zur Lösung der SQL-Abfrage wird die erste Zeile der ersten Tabelle genommen und alle Zeilen werden zum Ergebnis hinzugefügt, für die der Alkoholgehalt der ersten Tabelle größer ist als der Gehalt der zweiten Tabelle. Danach wird der Vorgang von der zweiten bis zur letzten Zeile der ersten Tabelle fortgeführt.

Wir haben die SQL-Abfrage für das folgende Bild durch die Bedingung

```
WHERE tblCocktail.Cocktail = "Alaska"
```

weiter eingeschränkt, sodass das folgende Ergebnis entstand.

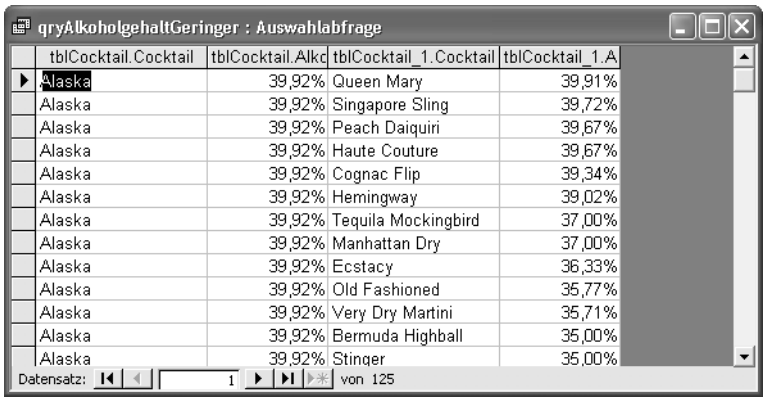


Bild 3.23: Ergebnisdarstellung der »Größer als«-Verknüpfung

Dass in den ersten Zeilen auch Cocktails mit 40% Alkoholgehalt zu sehen sind, liegt nur an den aufgerundeten Werten, die hier angezeigt werden. Wenn Sie die Prozentzahlen mit 2 Nachkommastellen anzeigen lassen, sehen Sie, dass alle in der Abfrage aufgeführten Cocktails tatsächlich einem geringeren Alkoholgehalt als der Alaska-Cocktail haben.

In der Praxis ist die Qualität von Verknüpfungen mit Nicht-Gleich-Operatoren schwer zu sichern, da die Ergebnisse oft schwer vorhersehbar sind. In vielen Fachbüchern wird von der Verwendung abgeraten, denn in den meisten Fällen lassen sich die Ergebnisse auch anders erreichen.

Das Ergebnis unseres Beispiels wäre auch mit der SQL-Abfrage

```
SELECT tblCocktail.Cocktail, tblCocktail.Alkoholgehalt,
tblCocktail_1.Cocktail, tblCocktail_1.Alkoholgehalt
FROM tblCocktail, tblCocktail AS tblCocktail_1
WHERE tblCocktail.Cocktail)="Alaska" AND
tblCocktail_1.Alkoholgehalt<[tblCocktail].[Alkoholgehalt]
ORDER BY tblCocktail.Cocktail, tblCocktail_1.Alkoholgehalt DESC;
```

zu erreichen, die im Gegensatz zu der zuvor besprochenen in der Access-Entwurfsansicht darstellbar ist.

3.3.5 Funktionen

Standard-SQL kennt fünf eingebaute Standardfunktionen, die als Aggregatfunktionen bezeichnet werden: Mittelwert (`Avg`), Anzahl (`Count`), Summe (`Sum`), Maxi-

malwert (`Max`) und Minimalwert (`Min`). Access fügt den Standardfunktionen noch weitere hinzu, die Sie der Tabelle 3.2 entnehmen können.

SQL-Aggregatfunktionen

In SQL können Sie nur die englischen Funktionsbeschreibungen verwenden, während in der Entwurfsansicht sowohl die deutsche als auch die englische Schreibweise erlaubt ist.

Tabelle 3.2: SQL-Aggregatfunktionen

Funktion	Bedeutung	Standard-SQL
Avg([Spalte]) Mittelwert([Spalte])	Mittelwert aller Spaltenwerte verschieden von Null	Ja
Count([Spalte]) Anzahl([Spalte])	Anzahl der Spaltenwerte verschieden von Null	Ja
Count(*) Anzahl(*)	Anzahl der Spaltenwerte inklusive der Nullwerte	Ja
Min([Spalte])	Kleinster Spaltenwert verschieden von Null	Ja
Max([Spalte])	Größter Spaltenwert verschieden von Null	Ja
Sum([Spalte]) Summe([Spalte])	Summe der Spaltenwerte verschieden von Null	Ja
First([Spalte]) ErsterWert([Spalte])	Spaltenwert der ersten Zeile des Ergebnisses, kann Null sein	Nein
Last([Spalte]) LetzterWert([Spalte])	Spaltenwert der letzten Zeile des Ergebnisses	Nein
StDev([Spalte]) StAbw([Spalte])	Standardabweichung einer Stichprobe der Spaltenwerte	Nein
StDevP([Spalte]) StdAbwG([Spalte])	Standardabweichung der Grundgesamtheit der Spaltenwerte	Nein
Var([Spalte]) Varianz([Spalte])	Varianz der Stichprobe der Spaltenwerte	Nein
VarP([Spalte]) VarianzG([Spalte])	Varianz der Grundgesamtheit der Spaltenwerte	Nein

Die Aggregatfunktionen lassen sich in der Entwurfsansicht einfach anwenden. Schalten Sie mit *ANSICHT Funktionen* bzw. durch die Schaltfläche mit dem Summenzeichen in der Symbolleiste die zusätzliche Zeile *Funktion* im unteren Bereich der Entwurfsansicht ein. Für die Darstellung im nächsten Bild haben wir die

Funktion *Summe* für die Spalte *Menge* selektiert. Die Ausführung der Abfrage würde den aufsummierten Wert aller Mengenangaben der Tabelle *tblCocktailZutaten* ergeben.

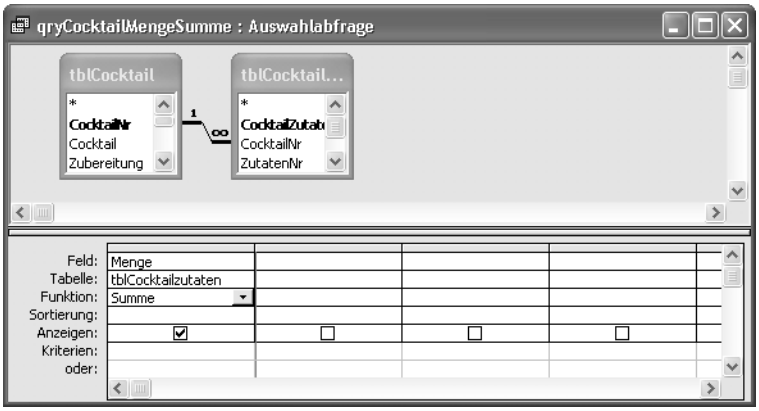


Bild 3.24: Einfache Summe

Damit die Auswertung etwas sinnvoller wird, haben wir zusätzlich die im folgenden Bild gezeigte Bedingung eingefügt. Für sie wurde die Funktion *Bedingung* angewählt. Normalerweise wird für ein neu in den unteren Entwurfsteil aufgenommenes Feld automatisch die Funktion *Gruppierung* eingestellt, die wir aber erst im nächsten Abschnitt des Kapitels besprechen.

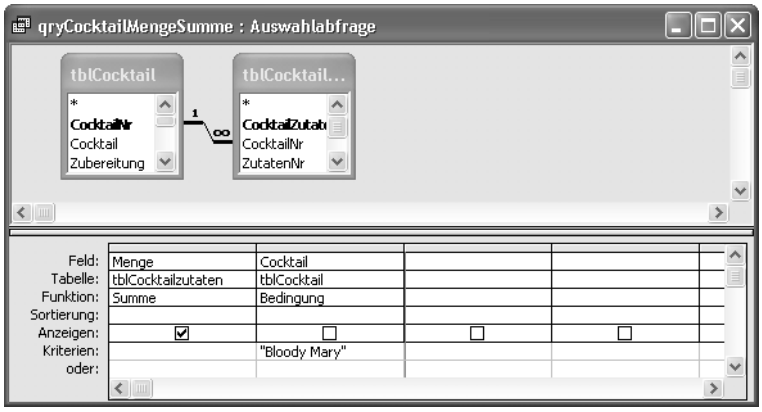


Bild 3.25: Erweiterte Abfrage

Das Ergebnis der Abfrage ist die Flüssigkeitsmenge für eine »Bloody Mary«, die das vorliegende Rezept ergibt. Zur Vereinfachung haben wir übrigens die Einheiten der Mengenangaben nicht berücksichtigt.

Bild 3.26: Ergebnisdarstellung

In der SQL-Darstellung wird die Anweisung zur Summenbildung durch

```
SELECT Sum(tblCocktailzutaten.Menge) AS [Summe von Menge]
FROM tblCocktail INNER JOIN tblCocktailzutaten ON tblCocktail.CocktailNr
= tblCocktailzutaten.CocktailNr
WHERE tblCocktail.Cocktail = "Bloody Mary";
```

umgesetzt. Übrigens ließe sich die SQL-Anweisung in der Entwurfsansicht auch in der in Bild 3.27 gezeigten Weise ohne die Zeile *Funktion* definieren. Das Ergebnis ist in beiden Fällen gleich.

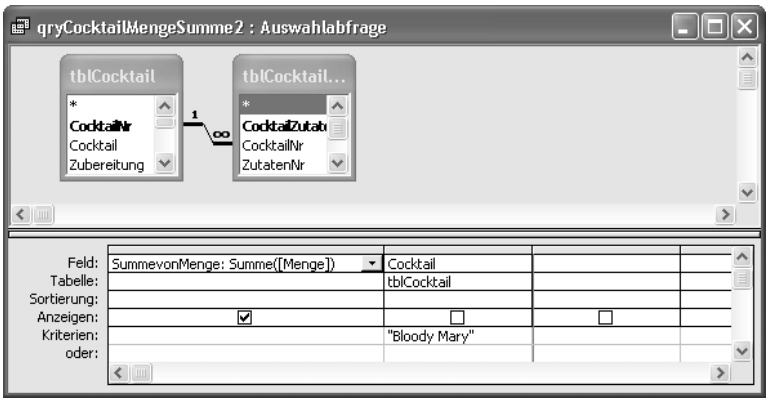


Bild 3.27: Umgeschriebene Summenanweisung

! Aggregatfunktionen ohne GROUP: Verwenden Sie Aggregatfunktionen ohne eine `GROUP`-Anweisung (siehe Abschnitt 3.3.6, »Daten gruppieren«), dann dürfen Ausgabefelder mit und ohne Aggregatfunktion nicht gemeinsam verwendet werden.

Behandlung von Nullwerten

Alle Funktionen berücksichtigen nur Datensätze, bei denen der jeweilige Feldinhalt verschieden von NULL ist. Mit `COUNT(tblCocktail.Cocktail)` werden nicht

alle Cocktails gezählt, sondern nur die, bei denen `tblCocktail.Cocktail` ungleich NULL ist. In der folgenden Entwurfsansicht sind drei Varianten der `COUNT()`-Funktion dargestellt.

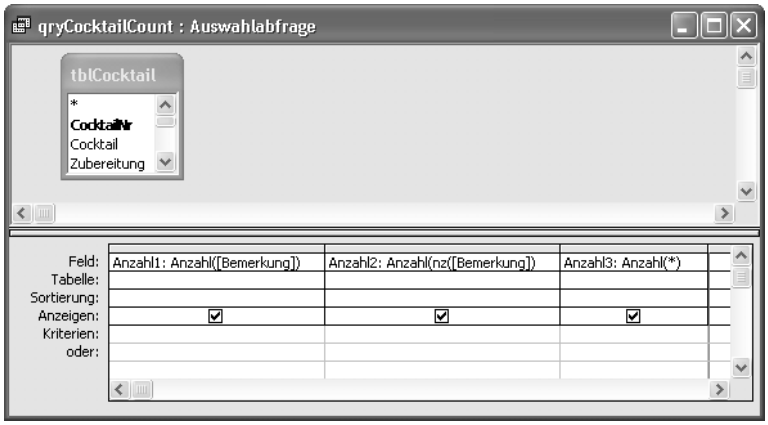


Bild 3.28: Drei Varianten »Anzahl«

Die Abfrage führt zu der im nächsten Bild gezeigten Ergebnisdarstellung.

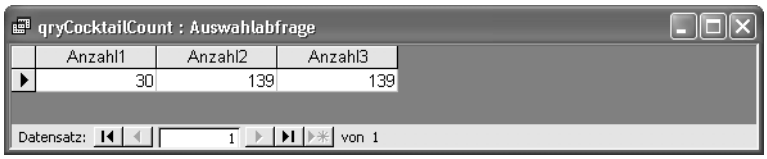


Bild 3.29: Ergebnisse der Zählfunktionsvarianten

In der Spalte `Anzahl1` wurde die Anzahl der Datensätze ermittelt, für die eine Bemerkung erfasst wurde. In `Anzahl2` und `Anzahl3` steht jeweils die gesamte Anzahl der Datensätze der Tabelle *tblCocktail*. Die SQL-Darstellung der Abfrage

```
SELECT Count([Bemerkung]) AS Anzahl1, Count(Nz([Bemerkung])) AS Anzahl2,
Count(*) AS Anzahl3
FROM tblCocktail;
```

zeigt die drei `COUNT()`-Varianten. Die Varianten `Count(Nz([Bemerkung])` und `Count(*)` führen zum gleichen Ergebnis, da durch die Verwendung von `Nz()` auch alle Nullwerte mitgezählt werden. Durch die Access-Funktion `Nz()` werden Nullwerte zum Wert 0 umgesetzt. Damit werden sie von `COUNT()` mitgezählt. `Count(*)` berücksichtigt automatisch alle Nullwerte, ist jedoch wesentlich schneller in der Ausführung. Daher empfehlen wir Ihnen, diese Variante prinzipiell zur

Ermittlung der Gesamtzahl der Resultatzeilen einer Abfrage zu verwenden. Die Abfrage

```
SELECT Count(*) AS Gesamtzahl
FROM tblCocktail
WHERE tblCocktail.Cocktail LIKE "C*";
```

beispielsweise gibt die Anzahl der Cocktails zurück, die mit »C« beginnen.

Haben Sie in Ihren Tabellen eine Spalte mit dem Datentyp *Ja/Nein* eingerichtet, können Sie mit der Summenfunktion die Anzahl der Ja- bzw. Nein-Werte zählen. Ja-Werte werden Access-intern als -1, Nein-Werte als 0 dargestellt. Angenommen, in einer Tabelle mit Rechnungen wäre ein Ja/Nein-Feld *Bezahlt* definiert, und mit einer Abfrage soll die Zahl der bezahlten bzw. unbezahlten Rechnungen ermittelt werden. Die bezahlten Rechnungen werden mit

```
SELECT Sum(Abs([Bezahlt])) AS [Bezahlte Rechnungen]
FROM tblRechnungen;
```

herausgesucht. Die Funktion `Abs()` gibt den Absolutwert einer Zahl zurück, also den Wert ohne Vorzeichen. Der Absolutwert von der *Ja/Nein*-Spalte *Bezahlt* ist entweder 1 oder 0. Die unbezahlten Rechnungen ermittelt die Abfrage

```
SELECT Sum(Abs(NOT[Bezahlt])) AS [Bezahlte Rechnungen]
FROM tblRechnungen;
```

Diese Art des Umgangs mit Ja/Nein-Werten funktioniert mit Access, aber nicht unbedingt mit anderen Datenbanken, denn die interne Darstellung von Ja/Nein-Werten ist unterschiedlich. Das gleiche Ergebnis erzielen Sie übrigens mit

```
SELECT Count(*) AS [Bezahlte Rechnungen]
FROM tblRechnungen
WHERE tblRechnungen.Bezahlt = True;
```

wobei die `WHERE`-Klausel alternativ auch

```
WHERE tblRechnungen.Bezahlt = Yes;
```

oder

```
WHERE tblRechnungen.Bezahlt;
```

lauten könnte.

3.3.6 Daten gruppieren

Auch Ergebnisdaten einer Abfrage lassen sich gruppieren und mit verschiedenen Funktionen auswerten. Durch die Gruppierung können die im vorherigen Abschnitt beschriebenen Aggregatfunktionen noch effektiver genutzt werden.

Die Ergebnismenge einer gruppierten Abfrage ist immer »read only«, d.h., Sie können keine Veränderung an den Daten vornehmen.

Der GROUP BY-Befehl

Als erstes einfaches Beispiel soll die Anzahl der Zutaten für jeden Cocktail bestimmt werden. Die Abfrage hat in der Entwurfsansicht das im folgenden Bild dargestellte Aussehen. Dabei wurde die Bezeichnung des Cocktails als Gruppierungskriterium gewählt.

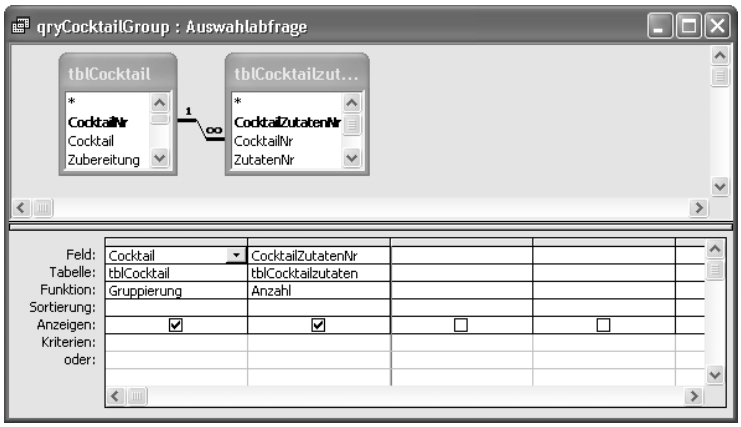


Bild 3.30: Ermittlung der Anzahl der Zutaten

Access erzeugt aus der Entwurfsansicht die SQL-Abfrage

```
SELECT tblCocktail.Cocktail, Count(tblCocktailzutaten.CocktailZutatenNr)
AS AnzahlvonCocktailZutatenNr
FROM tblCocktail INNER JOIN tblCocktailzutaten ON tblCocktail.CocktailNr
= tblCocktailzutaten.CocktailNr
GROUP BY tblCocktail.Cocktail;
```

Hierbei wird die Spaltenbezeichnung `AnzahlvonCocktailZutatenNr` automatisch eingesetzt. Das Ergebnis der Abfrage präsentiert das folgende Bild.

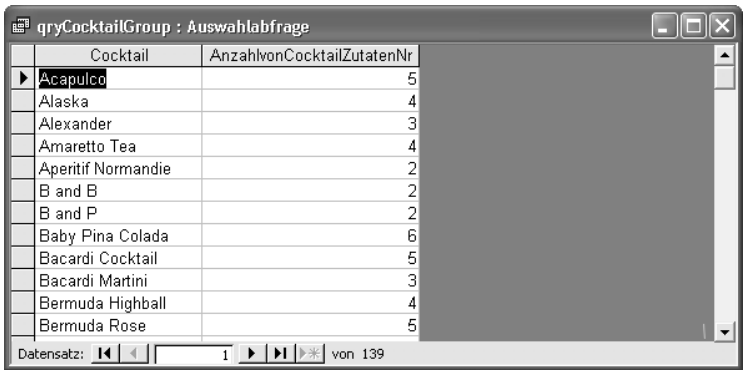


Bild 3.31: Anzahl der Zutaten pro Cocktail

Durch den Gruppierungsbefehl `GROUP BY` wird die Aggregatfunktion `COUNT()` für jeden Cocktail bestimmt, d.h., es werden alle Zutaten für eine bestimmte Cocktailnummer gezählt. Durch die Gruppierung ist es möglich, eine normale Ausgabespalte und eine Spalte mit Aggregatfunktion gleichzeitig auszugeben. Allerdings dürfen nur diejenigen Spalten, die hinter der `GROUP BY`-Klausel stehen, als Ausgabefelder benannt werden. Durch den `GROUP BY`-Befehl wird nicht automatisch nach den Gruppierungsfeldern sortiert. Hierfür müssen die Spalten in einer `ORDER BY`-Klausel aufgeführt werden.

Das nächste Beispiel summiert die Mengenangaben für jeden Cocktail aus der Tabelle *tblCocktailzutaten*. Die Cocktails werden über ihre Cocktailnummer bestimmt. Zur Vereinfachung haben wir die Einheiten der Mengenangaben nicht berücksichtigt.

```
SELECT tblCocktailzutaten.CocktailNr, Sum(tblCocktailzutaten.Menge) AS
SummevonMenge
FROM tblCocktailzutaten
GROUP BY tblCocktailzutaten.CocktailNr;
```

Die HAVING-Klausel

Mithilfe des SQL-Befehls `HAVING` lassen sich gruppierte Daten einschränken. Die `HAVING`-Klausel wird wie ein `WHERE`-Befehl verwendet, bezieht sich aber immer nur auf die Gruppen des `GROUP BY`-Befehls.

Zwei Beispiele sollen die Möglichkeiten der `HAVING`-Klausel verdeutlichen. Zuerst soll für alle Cocktails, die mehr als fünf Zutaten haben, die Anzahl der Zutaten und die Menge der Flüssigkeit bestimmt werden, wobei die Einheiten der Mengenangaben vernachlässigt werden sollen. Access zeigt die Einstellungen in der Entwurfsansicht in der im nächsten Bild präsentierten Form.

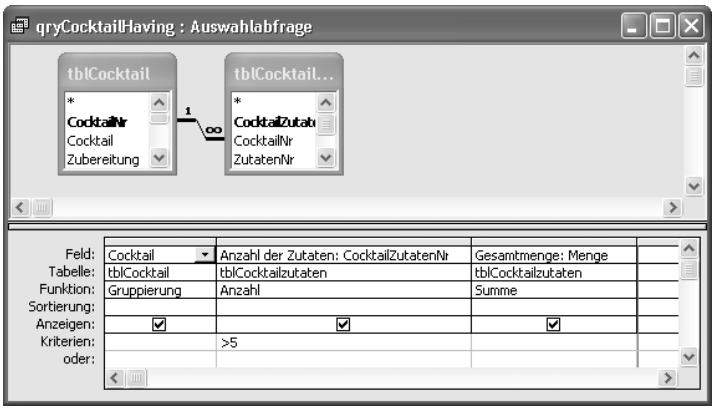
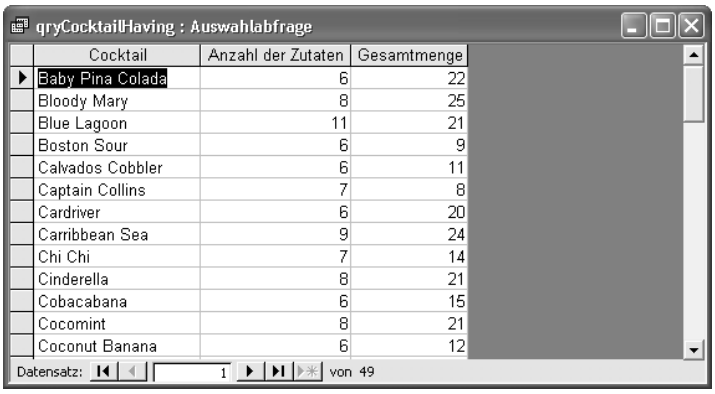


Bild 3.32: Cocktails mit mehr als fünf Zutaten

Das Ergebnis der Abfrage wird in Bild 3.33 dargestellt, wobei wir, wie in der Entwurfsansicht zu sehen, eigene Spaltenüberschriften festgelegt haben.



Cocktail	Anzahl der Zutaten	Gesamtmenge
Baby Pina Colada	6	22
Bloody Mary	8	25
Blue Lagoon	11	21
Boston Sour	6	9
Calvados Cobbler	6	11
Captain Collins	7	8
Cardriver	6	20
Carribbean Sea	9	24
Chi Chi	7	14
Cinderella	8	21
Cobacabana	6	15
Cocomint	8	21
Coconut Banana	6	12

Bild 3.33: Ergebnis bei Verwendung der Having-Klausel

Zu diesem Ergebnis führte folgende von Access aus der Entwurfsansicht umgesetzte SQL-Abfrage:

```
SELECT tblCocktail.Cocktail, Count(tblCocktailzutaten.CocktailZutatenNr)
AS [Anzahl der Zutaten], Sum(tblCocktailzutaten.Menge) AS Gesamtmenge
FROM tblCocktail INNER JOIN tblCocktailzutaten ON tblCocktail.CocktailNr
= tblCocktailzutaten.CocktailNr
GROUP BY tblCocktail.Cocktail
HAVING Count(tblCocktailzutaten.ZutatenNr) > 5;
```

Die COUNT()-Funktion der HAVING-Klausel bezieht sich auf jede durch den GROUP BY-Befehl erstellte Gruppe, d.h. die Cocktailzutaten werden für jede Cocktailnummer gruppiert und mit HAVING ausgewertet.

Die SQL-Abfrage lässt sich noch mit einer WHERE-Bedingung ergänzen. WHERE bezieht sich immer auf die gesamte Tabelle, d.h. WHERE wird vor der Bildung der Gruppen angewendet. Die folgende Abfrage ermittelt wiederum die Anzahl der Zutaten und die Gesamtmenge für jeden Cocktail mit mehr als fünf Zutaten, aber diese Spalten werden nur für Cocktails ermittelt, deren Bezeichnung mit dem Buchstaben »C« beginnt.

```
SELECT tblCocktail.Cocktail, Count(tblCocktailzutaten.ZutatenNr) AS
[Anzahl der Zutaten], Sum(tblCocktailzutaten.Menge) AS Gesamtmenge
FROM tblCocktail INNER JOIN tblCocktailzutaten ON tblCocktail.CocktailNr
= tblCocktailzutaten.CocktailNr
WHERE tblCocktail.Cocktail LIKE "C*"
GROUP BY tblCocktail.Cocktail
HAVING Count(tblCocktailzutaten.ZutatenNr) > 5;
```

In die Entwurfsansicht wird die WHERE-Klausel durch eine weitere Spalte aufgenommen, deren Ausgabe unterdrückt ist. Als *Funktion* wird *Bedingung* gewählt, um die Bedeutung der Spalte anzuzeigen.

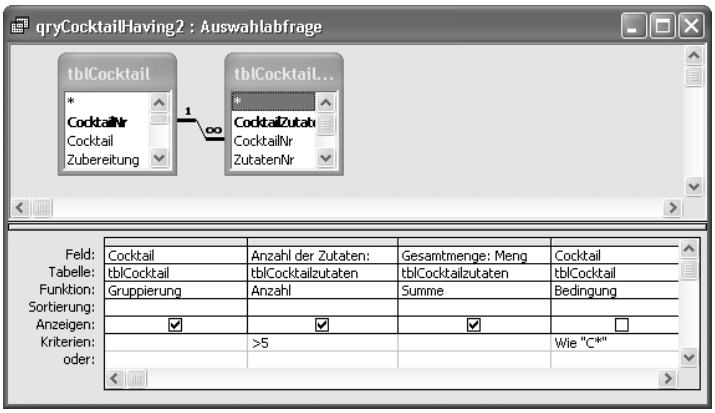


Bild 3.34: Um neue Bedingung erweiterte Entwurfsansicht

Das Resultat der Abfrage zeigt das folgende Bild. In unserer Cocktail-Datenbank haben sich neun Drinks mit mehr als fünf Zutaten gefunden, die mit »C« beginnen. Beachten Sie, dass immer zuerst die WHERE-Bedingung, dann die GROUP BY-Klausel und erst danach die HAVING-Bedingung ausgewertet wird.

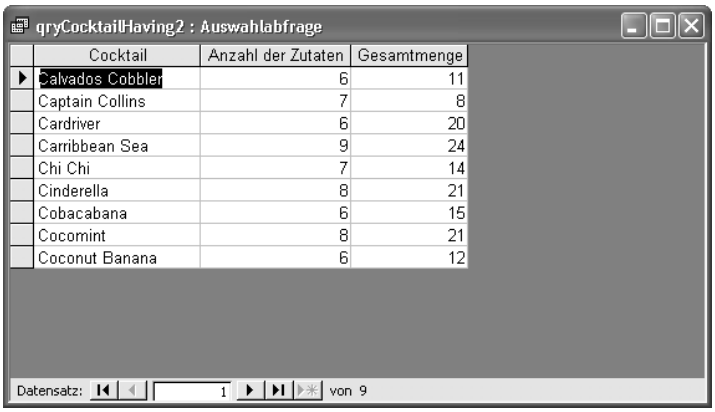


Bild 3.35: Alle Cocktails, die mit »C« beginnen und mehr als fünf Zutaten haben

Theoretisch können Sie eine `HAVING`-Klausel auch ohne `GROUP BY` einsetzen. Die Bedingung wird dann auf eine Gruppe, nämlich die gesamte Abfrage, angewandt.

3.3.7 Berechnete Felder

In Abfragen können sowohl in den Ausgabespalten als auch in den Bedingungen Berechnungen enthalten sein. Wir möchten uns in diesem Abschnitt mit berechneten Ausgabefeldern beschäftigen. Die gleichen Regeln und Möglichkeiten lassen sich auf errechnete Bedingungen übertragen.

Einfache Berechnungen

Im ersten Beispiel soll die Mengenangabe von Cocktailzutaten in Zentiliter (cl) umgerechnet werden. In den meisten Mixanleitungen sind Zentiliter die übliche Maßeinheit, aber einige Rezepte, insbesondere solche aus angelsächsischen Ländern, verwenden andere Maße. In der Tabelle *tblEinheiten* werden die Einheiten und ein entsprechender Umrechnungsfaktor in Zentiliter aufgeführt. In einer Abfrage sollen für alle Cocktailzutaten die Mengen in Zentiliter angegeben werden. Dazu muss die Menge mit dem Umrechnungsfaktor multipliziert werden. In der im Bild dargestellten Entwurfsansicht wurde die `cl:[Menge]*[Umrechnung_cl]` als Umrechnung angegeben. Der Text vor dem Doppelpunkt bezeichnet die Überschrift der Spalte. Vergeben Sie keine eigenen Spaltenüberschriften, so nennt Access die berechneten Spalten `Ausdr1`, `Ausdr2` usw.

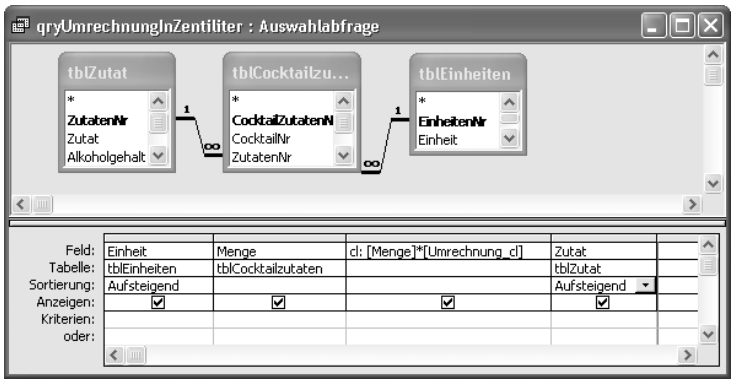


Bild 3.36: Berechnung der Menge in Zentiliter

Das Ergebnis der Abfrage zeigt die folgende Abbildung, wobei wir die Abfrage zur besseren Übersichtlichkeit nach der Bezeichnung der Einheit sortiert haben.

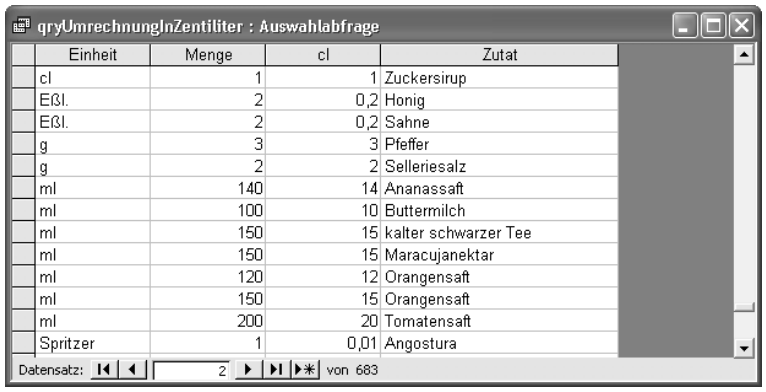


Bild 3.37: Ergebnis der Einheitenumrechnung

In der SQL-Darstellung werden Feldnamen in Berechnungen von Access automatisch in eckige Klammern gesetzt. Sollte die von Ihnen gewählte Spaltenüberschrift für das berechnete Feld Leerzeichen enthalten, wird die Bezeichnung ebenfalls in eckige Klammern eingeschlossen, beispielsweise `... AS [Menge in cl]`.

```
SELECT tblEinheiten.Einheit, tblCocktailzutaten.Menge,
[Menge]*[Umrechnung_cl] AS cl, tblZutat.Zutat
FROM tblEinheiten INNER JOIN (tblZutat INNER JOIN tblCocktailzutaten ON
tblZutat.ZutatenNr = tblCocktailzutaten.ZutatenNr) ON
tblEinheiten.EinheitenNr = tblCocktailzutaten.EinheitenNr
ORDER BY tblEinheiten.Einheit, tblZutat.Zutat;
```

Nullwerte: Hat eine der Spalten in einer Berechnung den Inhalt `NULL`, ist das Ergebnis der Kalkulation ebenfalls `NULL`. Um dies zu vermeiden, können Sie die einzelnen Felder in der Berechnung in die Funktion `Nz()` einschließen, wie beispielsweise in `Nz([Menge])*Nz([Umrechnung_cl])`. `Nz()` wandelt Nullwerte zu 0 um, sodass Kalkulationen ausgeführt werden, auch wenn eines der beteiligten Felder `NULL` ist.

Berechnete Spalten können in anderen Spalten weiter verrechnet werden. Im folgenden Beispiel wird im Feld `Alkohol` der Gesamtalkoholgehalt eines Cocktails kalkuliert, indem die errechneten Spalten `AlkMenge` und `Gesamtmenge` dividiert werden.

```
SELECT tblCocktail.Cocktail,
Sum([Menge]*[tblZutat].[Alkoholgehalt]*[Umrechnung_cl]) AS AlkMenge,
Sum([Menge]*[Umrechnung_cl]) AS Gesamtmenge, [AlkMenge]/[Gesamtmenge] AS
Alkohol
FROM tblZutat INNER JOIN (tblEinheiten INNER JOIN (tblCocktail INNER JOIN
tblCocktailzutaten ON tblCocktail.CocktailNr =
tblCocktailzutaten.CocktailNr) ON tblEinheiten.EinheitenNr =
tblCocktailzutaten.EinheitenNr) ON tblZutat.ZutatenNr =
tblCocktailzutaten.ZutatenNr
GROUP BY tblCocktail.Cocktail
ORDER BY tblCocktail.Cocktail;
```

In der Entwurfsansicht wird die SQL-Abfrage wie in Bild 3.38 dargestellt. In der Zeile *Funktion* im unteren Teil der Entwurfsansicht wurde für das Feld *Cocktail* *Gruppierung* gewählt, für die drei kalkulierten Spalten vereinbarten wir *Ausdruck*.

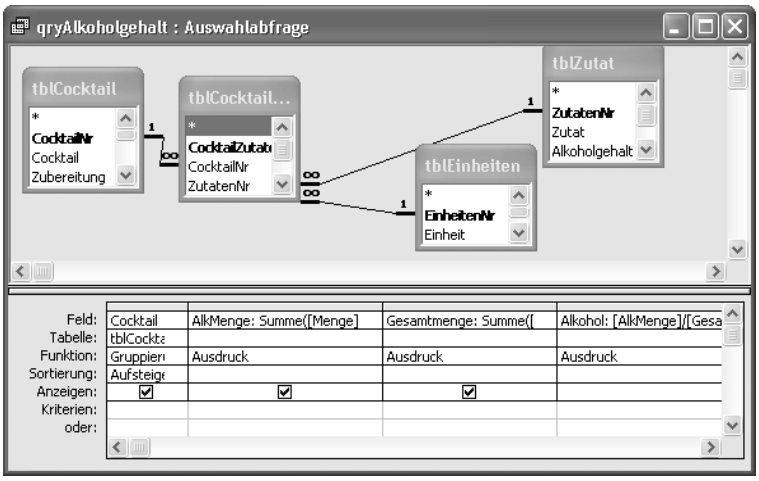


Bild 3.38: Entwurfsansicht zur Ermittlung des Gesamtalkoholgehalts pro Cocktail

Das Ergebnis der Abfrage zeigt, dass Access intern mit doppelter Genauigkeit rechnet, da die Menge als `Double`-Wert in der Tabelle *tblCocktailZutaten* definiert ist. In der Datenblattdarstellung werden von Access alle Nachkommastellen angezeigt.

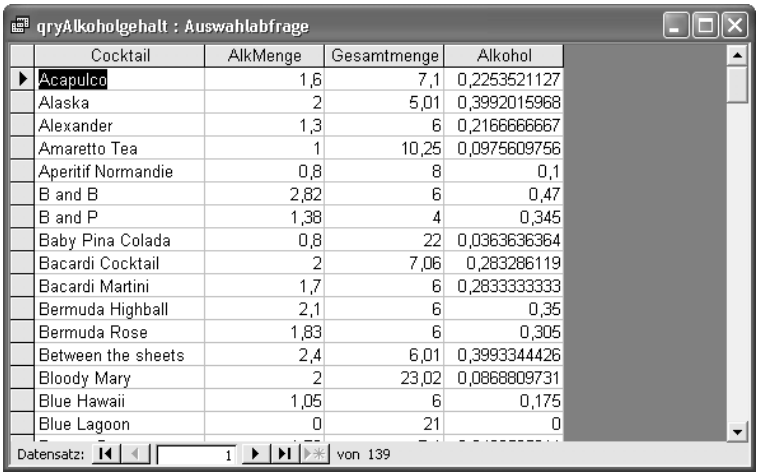


Bild 3.39: Errechneter Gesamtalkoholgehalt der Cocktails

Einsatz von Access-Funktionen

In den meisten Fällen wird eine Formatierung der dargestellten Werte, z.B. mit nur einer Nachkommastelle, in einem Formular oder einem Bericht vorgenom-

men. Mithilfe der Access-Funktion `FORMAT()` können Sie eine Formatierung schon für das Abfrageergebnis vornehmen. In der SQL-Abfrage wird dazu die Anweisung

```
[AlkMenge]/[Gesamtmenge] AS Alkohol
```

zu

```
FORMAT([AlkMenge]/[Gesamtmenge],"0.0") AS Alkohol
```

ergänzt. Die Funktion `FORMAT()` benötigt zwei Parameter, den Wert und die Formatierungsanweisung. Informationen zur Format-Funktion finden Sie in Kapitel 7, »VBA-Funktionen«. Beachten Sie bitte, dass bei der Eingabe der `FORMAT()`-Funktion in der Entwurfsansicht im Gegensatz zur SQL-Funktion die Form `FORMAT(Feld;"0,0")`, verwendet werden muss, also mit deutschen Trenn- und Dezimalzeichen.

Prinzipiell können fast alle Access-Funktionen in Abfragen eingesetzt werden. Beachten Sie aber, dass bei ODBC-Zugriffen die Funktionen lokal in Access abgearbeitet werden.

Benutzerdefinierte Funktionen

Möglich sind auch benutzerdefinierte Funktionen, die in Visual Basic als Access-Module erfasst werden. Müssen Sie z.B. in vielen Abfragen mit der Mehrwertsteuer kalkulieren, ist es sinnvoll, für diese Aufgabe eine eigene Funktion in der Form

```
Function Mwst(ByVal curNetto As Currency) As Currency
    Const conMwstSatz = 0.16
    Mwst = curNetto * (1 + conMwstSatz)
End Function
```

zu schreiben, sodass Sie einen geänderten Steuersatz gegebenenfalls nur in der Funktion anpassen müssen, nicht aber in allen Abfragen. Sollten Sie mit Visual Basic noch nicht vertraut sein, erläutern wir die Details dieser und anderer Funktionen in Kapitel 6, »Einführung in Visual Basic«.

In einer Abfrage würde die Funktion wie folgt eingesetzt werden:

```
SELECT tblHausbar.ZutatenNr, tblHausbar.Menge, Mwst([Einkaufspreis]) AS
Brutto
FROM tblHausbar;
```

Vergleiche mit IIF()

Eine in Abfragen sehr hilfreiche Funktion ist `IIF()`, ausgesprochen »inline IF«. Die Funktion besitzt drei Argumente: `IIF(Bedingung, Wahr-Teil, Falsch-Teil)`. In der Entwurfsansicht heißt die Funktion `WENN()`. Beachten Sie, dass in der Entwurfsansicht die Argumente mit Semikolon getrennt werden müssen.

Die folgende SQL-Abfrage gibt in Abhängigkeit von der Menge, die das jeweilige Cocktailrezept ergibt, einen Text aus, wobei wir zur Vereinfachung der Abfrage die verschiedenen Mengeneinheiten vernachlässigt haben.

```
SELECT tblCocktail.Cocktail,
IIf(Sum([Menge])>20,"Mehr als ein Glas","Ein Glas") AS Rezept
FROM tblCocktail INNER JOIN tblCocktailzutaten ON tblCocktail.CocktailNr
= tblCocktailzutaten.CocktailNr
GROUP BY tblCocktail.Cocktail;
```

`IIF()`-Funktionen lassen sich auch verschachteln, beispielsweise lässt sich die `IIF()`-Funktion aus der obigen SQL-Abfrage wie folgt erweitern:

```
IIf(Sum([Menge])>20,IIF(Sum([Menge])>40,"Mehr als zwei Gläser","Mehr als
ein Glas"),"Ein Glas")
```

Verkettung von Zeichenfolgen

Mit den Operatoren »&« und »+« fügen Sie mehrere Zeichenfolgen zu einer Zeichenkette zusammen, beispielsweise erzeugen

```
tblCocktail.Cocktail & " - " & tblCocktail.Zubereitung
```

oder

```
tblCocktail.Cocktail + " - " + tblCocktail.Zubereitung
```

ein berechnetes Feld, in dem die Bezeichnung des Cocktails, ein Bindestrich und die Zubereitung zu einer Zeichenfolge verkettet werden. Der Unterschied zwischen den Operatoren »&« und »+« liegt in der Behandlung von Nullwerten. Während mit dem »&«-Zeichen Nullwerte als leere Zeichenfolgen "" aufgefasst werden, ist das Ergebnis einer »+«-Verknüpfung `NULL`, wenn eine der Teilzeichenfolgen `NULL` ist.

Die Access-Operatoren

In der folgenden Tabelle sind der Vollständigkeit halber die in Access einsetzbaren Operatoren aufgeführt. Beachten Sie bitte, dass im SQL-Standard nicht alle Operatoren nutzbar sind, die Access anbietet.

Tabelle 3.3: Operatoren

Operator	Bedeutung	Bemerkung
+	Addition	
-	Subtraktion	
*	Multiplikation	
/	Division	
\	Ganzzahlige Division	Die Operanden werden vor der Division in Byte, Integer- oder Long Integer-Wert umgewandelt und gerundet. Das Ergebnis ist ganzzahlig vom Typ Byte, Integer oder Long Integer.
^	Potenzierung	
Mod	Modulo	Gibt den Rest einer ganzzahligen Division zurück. Fließkommaoperanden werden zu ganzen Zahlen gerundet. Das Ergebnis ist ein Wert vom Typ Byte, Integer oder Long Integer.

3.4 Parameterabfragen

Abfragen mit Parametern sind eine Erweiterung des SQL-Standards durch Access. Parameter ermöglichen die Eingabe von Werten während der Auswertung der SQL-Abfrage, ohne den SQL-Text oder den Entwurf der Abfrage zu verändern. Sie sind ein gängiges Mittel innerhalb von Access, den Benutzer nach Eingaben zu fragen. Parameter können sowohl in der `WHERE`-Bedingung als auch in der `HAVING`-Klausel eingesetzt werden.

3.4.1 Einfache Parameter

In der Entwurfsansicht lassen sich Parameter durch Texte in eckigen Klammern hinter *Kriterium* definieren. Die Texte dürfen jedoch nicht mit Feldnamen der Tabelle oder der Tabellen übereinstimmen. Alle Bezeichnungen, die Access nicht als Feldnamen interpretiert, werden als Parameter angesehen. Aus diesem Grund entdeckt man in der Regel Schreibfehler in Feldnamen nur sehr langsam: Sie werden von Access als Parameter abgefragt. Die Länge des Parametertextes ist nicht begrenzt, wie Sie im nächsten Bild sehen können.

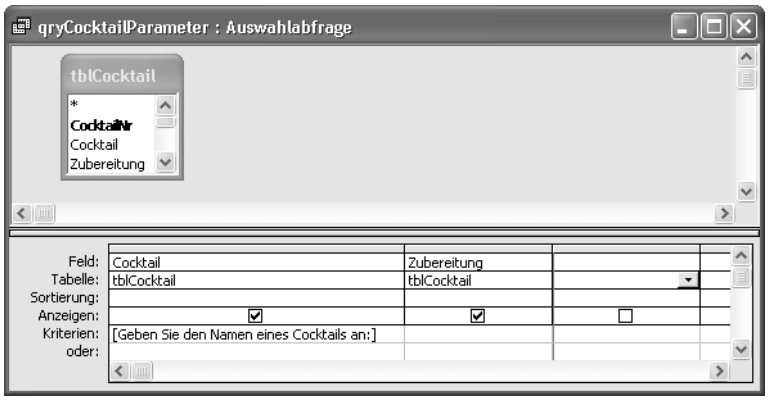


Bild 3.40: Vereinbarung eines Parameters

Die Umsetzung zu SQL hat das folgende Aussehen:

```
SELECT tblCocktail.Cocktail, tblCocktail.Zubereitung
FROM tblCocktail
WHERE tblCocktail.Cocktail = [Geben Sie den Namen eines Cocktails an:];
```

Wird die Abfrage ausgeführt, erscheint das folgende Dialogfeld, um einen Wert für den Parameter entgegenzunehmen.

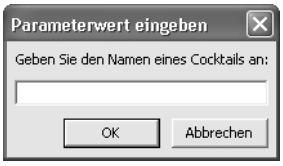


Bild 3.41: Dialogfeld für Parameter

Sollten Sie mehrere Parameter in einer Abfrage vereinbart haben, werden sie nacheinander in jeweils eigenen Dialogfeldern abgefragt. Bei der Eingabe findet keine Überprüfung des Datentyps statt. Im folgenden Beispiel ist die Spalte *tblCocktail.CocktailErfasst* vom Typ *Datum/Zeit*:

```
SELECT tblCocktail.Cocktail, tblCocktail.Zubereitung
FROM tblCocktail
WHERE tblCocktail.CocktailErfasst = [Geben Sie ein Datum an:];
```

Die Abfrage kann nur ausgewertet werden, wenn Sie im Parameterdialogfeld ein Datum angeben. Geben Sie Werte an, die Access nicht als Datum oder Zeit interpretieren kann, erhalten Sie die im nächsten Bild abgebildete Warnmeldung.

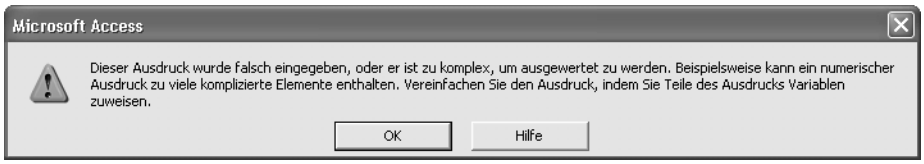


Bild 3.42: Fehlermeldung bei Typproblem

3.4.2 Vordefinierte Parameter

Sie können Access zu einer Typüberprüfung bei der Eingabe von Parametern veranlassen, indem Sie die Parameter vordefinieren. Rufen Sie dazu in der Entwurfsansicht über den Menübefehl *Parameter* im Menü *ABFRAGE* das in Bild 3.43 gezeigte Dialogfeld auf. Geben Sie in der Spalte *Parameter* die von Ihnen gewünschte Parameterbezeichnung ein, und wählen Sie dazu rechts einen entsprechenden *Felddatentyp*.

Bild 3.43: Dialogfeld Abfrageparameter

In der SQL-Darstellung wird die Parameterdefinition dem SQL-Befehl vorangestellt:

```
PARAMETERS [Geben Sie ein Datum an:] DateTime;
SELECT tblCocktail.Cocktail, tblCocktail.Zubereitung
FROM tblCocktail
WHERE tblCocktail.CocktailErfasst = [Geben Sie ein Datum an:];
```

Wird die Abfrage ausgeführt, erhalten Sie zur Eingabe das entsprechende Dialogfeld. Entspricht Ihre Eingabe nicht dem für den Parameter vereinbarten Felddatentyp, erhalten Sie die im nächsten Bild vorgestellte Fehlermeldung. Bestätigen Sie die Fehlermeldung, werden Sie erneut nach dem Parameter gefragt.

Bild 3.44: Fehlermeldung bei fehlerhafter Parametereingabe

Mehrere vordefinierte Parameter werden im SQL-Befehl durch Kommas getrennt und mit einem Semikolon abgeschlossen. Die folgende `PARAMETERS`-Anweisung zeigt alle vordefinierbaren Felddatentypen. Als Parameter-Namen in den eckigen Klammern haben wir die Bezeichnungen gewählt, die im Dialogfeld *Abfrageparameter* (Bild 3.43) in der Spalte *Felddatentyp* angeboten werden.

```
PARAMETERS [Ja/Nein] Bit, [Byte] Byte, [Integer] Short, [Long Integer]
Long, [Währung] Currency, [Single] IEEESingle, [Double] IEEEDouble,
[Datum/Uhrzeit] DateTime, [Binär] Binary, [Text] Text ( 255 ), [OLE-
Objekt] LongBinary, [Memo] Text, [Replikations-ID] Guid, [Decimal]
Decimal, [Wert] Value;
```

3.4.3 Benutzerdefiniertes Formular zur Parametereingabe

Besser und für den Anwender bequemer ist der Einsatz eines Formulars zur Eingabe der Parameter einer Abfrage. Anhand eines einfachen Beispiels möchten wir das Zusammenspiel zwischen Formular und Abfrage zeigen. Dazu werden wir zunächst eine Abfrage definieren, die den Verweis auf das Formular enthält und dann ein Formular erstellen, das über eine Schaltfläche die Abfrage aufruft.

Die Abfrage besteht aus dem SQL-Befehl

```
SELECT tblCocktail.Cocktail, tblCocktail.Zubereitung
FROM tblCocktail
WHERE tblCocktail.Cocktail LIKE [forms].[frmParaEingabe].[txtParameter] &
"*";
```

der in der Entwurfsansicht – wie im nächsten Bild gezeigt – dargestellt wird.

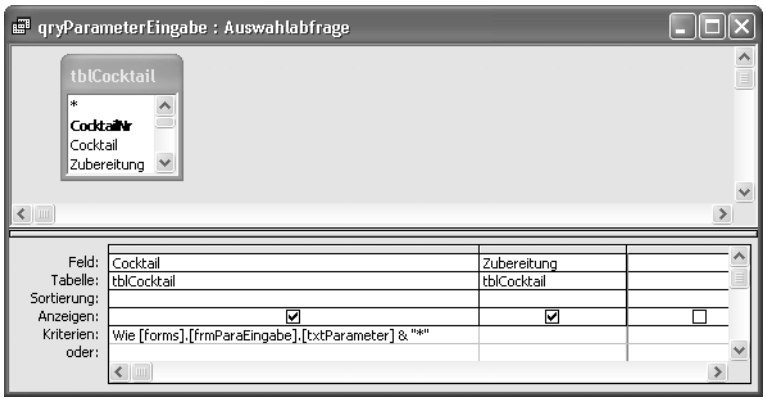


Bild 3.45: Entwurfsansicht

In die `WHERE`-Bedingung wurde der Parameter in der Form

```
LIKE [forms].[frmParaEingabe].[txtParameter]  & "*"
```

aufgenommen, d.h., der Parameter stammt aus einem Formular (`[forms]`) mit dem Namen `frmParaEingabe` und dort aus dem Textfeld `txtParameter`. Hinter den Parameter platzieren wir aus Gründen der Bequemlichkeit das Platzhalterzeichen »*«.

Auf die Schreibweise für den Zugriff auf Formularfelder `[forms].[Formularname].[Feldname]` werden wir in Kapitel 14, »Formulare«, noch ausführlich eingehen.

Abschließend wird die Abfrage gespeichert. Der Parameter in der Abfrage bezieht sich nun auf ein Feld in einem Formular, das noch nicht existiert. Prinzipiell können Sie die Abfrage aber auch jetzt schon einsetzen, denn der Parameter, der ja noch nicht in einem Formular zu finden ist, wird von Access in einem normalen Parameterdialogfeld abgefragt.

Im nächsten Schritt wird ein Formular angelegt. Das Formular wird als ungebundenes Formular erstellt, d.h., es wird keine Tabelle oder Abfrage als Grundlage für die Daten des Formulars genutzt. In Kapitel 14, »Formulare«, finden Sie die entsprechenden Informationen zu gebundenen und ungebundenen Formularen.

Bild 3.46 zeigt das fertige Formular, auf dem ein Textfeld zur Eingabe und zwei Befehlsschaltflächen zum Aufruf der Abfrage und zum Schließen des Formulars angeordnet sind.

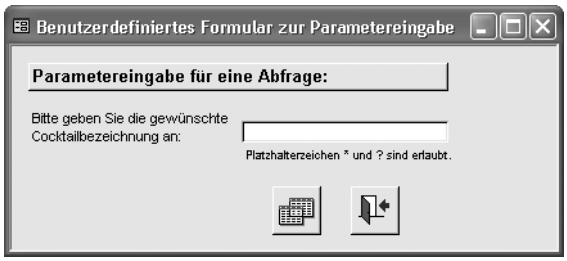


Bild 3.46: Fertiges Parametereingabeformular

Das Textfeld bekam den Namen *txtParameter*, so wie wir ihn schon in der oben erstellten Abfrage vorgesehen hatten. Das Formular selbst speicherten wir unter dem Namen *frmParaEingabe*.

Die beiden Befehlsschaltflächen sind mit dem Befehlsschaltflächen-Assistenten erzeugt worden. Die linke zeigt das Ergebnis der Parameterabfrage in Tabellenform, die rechte schließt das Formular.

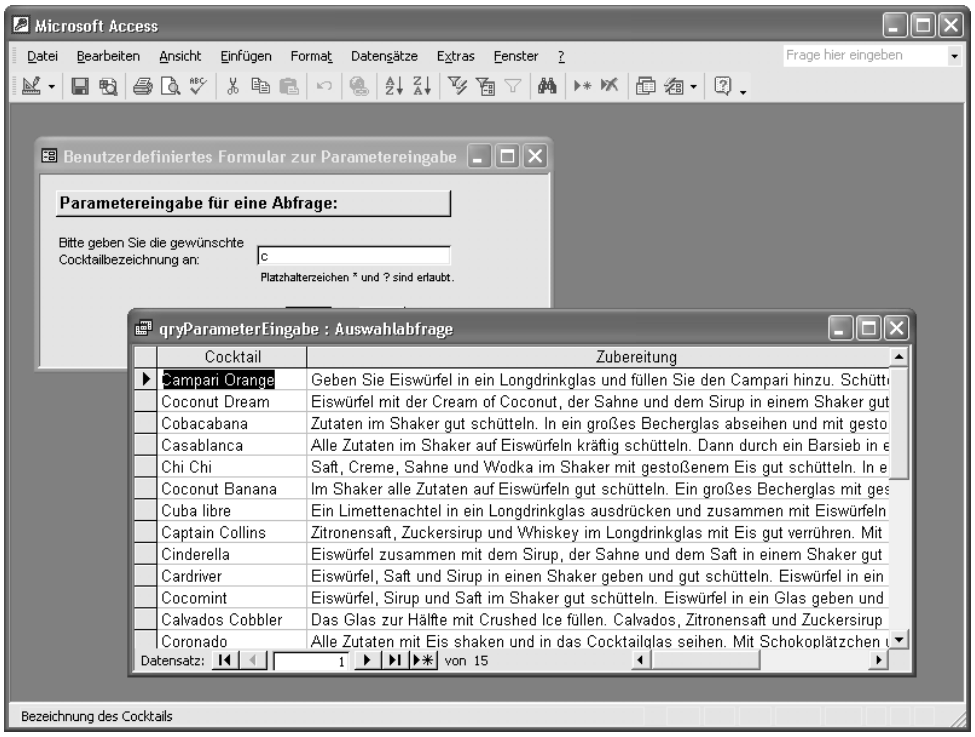


Bild 3.47: Parametereingabeformular und Abfrageergebnis

3.4.4 Von-Bis-Abfragen mit Parametern

Zum Schluss des Abschnitts über Parameter noch ein Tipp für den Einsatz von Parametern mit dem `BETWEEN...AND`-Operator. Stellen Sie sich vor, der Benutzer sollte die Möglichkeit erhalten, alle Cocktails zu ermitteln, die mit den Buchstaben C, D, E und F beginnen. In der Abfrage werden dazu die Parameter `[Von:]` und `[Bis:]` abgefragt.

```
SELECT tblCocktail.Cocktail, tblCocktail.Zubereitung
FROM tblCocktail
WHERE tblCocktail.Cocktail BETWEEN [Von:] AND [Bis:];
```

Um die Cocktails herauszufinden, die mit C, D, E oder F beginnen, muss der Anwender für den ersten Parameter ein »C« und für den zweiten ein »G« eingeben. Für viele Anwender scheint dies unlogisch, obwohl es lexikalisch korrekt ist, denn hätte der zweite Parameter ein »F« zum Inhalt, wären Drinks wie »Frozen Tequila« nicht im Abfrageergebnis enthalten, denn »Fr...« kommt alphabetisch nach «F«. Schwierig für den Anwender wird es insbesondere dann, wenn die Liste auch die Drinks mit «Z« enthalten soll. Welcher Buchstabe kommt nach »Z«?

Der folgende Trick schafft Abhilfe: An den zweiten Parameter wird das in der alphabetischen Sortierung größte Zeichen angehängt. Das letzte Zeichen der ASCII-Tabelle kann über die Access-Funktion `Chr()` als `Chr(255)` ermittelt werden. In der deutschen Schreibweise der Entwurfsansicht wird die Funktion als `Zchn()` benannt. Die SQL-Abfrage erhält damit das folgende Aussehen:

```
SELECT tblCocktail.Cocktail, tblCocktail.Zubereitung
FROM tblCocktail
WHERE tblCocktail.Cocktail BETWEEN [Von:] AND [Bis:] & Chr(255);
```

3.5 Unterabfragen

Unterabfragen sind `SELECT`-Abfragen innerhalb von `SELECT`-Abfragen. Das bedeutet, es wird eine `SELECT`-Abfrage verwendet, um Werte und Bedingungen für eine andere zu finden. Access erlaubt, die `SELECTs` bis zu 50 Ebenen tief zu schachteln.

3.5.1 Unterabfragen mit dem IN-Operator

Im ersten Beispiel sollen alle Cocktails gefunden werden, die mehr als fünf Zutaten haben. Wir haben die Aufgabenstellung schon mit einer Gruppierung im Abschnitt 3.3.6, »Daten gruppieren«, in »Die HAVING-Klausel« gelöst. Im Prinzip können solche Fragestellungen sowohl mit verknüpften Tabellen (joins) als auch mit Unterabfragen beantwortet werden.

In der SQL-Abfrage

```
SELECT tblCocktail.Cocktail
FROM tblCocktail
WHERE tblCocktail.CocktailNr
In(SELECT tblCocktailZutaten.Cocktailnr FROM tblCocktailZutaten GROUP BY
tblCocktailZutaten.Cocktailnr HAVING count(*) > 5 );
```

wird für die `WHERE`-Bedingung eine Menge von Cocktailnummern mithilfe des in Klammern eingeschlossenen `SELECT`s ermittelt. Der erste `SELECT` zeigt dann die Bezeichnung eines Cocktails, wenn sich die Cocktailnummer in der Ergebnismenge der Unterabfrage befindet. Der Operator `IN` führt die Überprüfung durch, ob `tblCocktail.CocktailNr` in der Ergebnismenge vorhanden ist.

In der Entwurfsansicht wird die Abfrage wie im folgenden Bild dargestellt. Dabei wird die Unterabfrage in der Entwurfsansicht direkt in die Kriterienzeile eingegeben. Es hat sich bewährt, die Unterabfrage zuerst in einem eigenen Entwurfsansichtsfenster zu erstellen und zu testen. Anschließend kann der SQL-Text in die Zwischenablage kopiert und dann als Unterabfrage eingefügt werden.

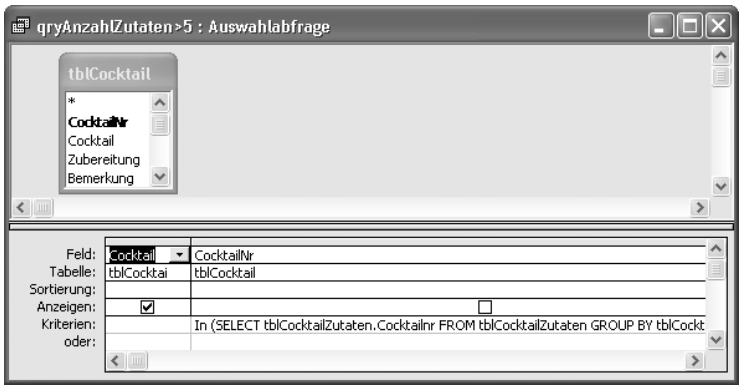


Bild 3.48: Darstellung der Unterabfrage in der Entwurfsdarstellung

Beachten Sie bitte, dass die Unterabfrage nur ein Ausgabefeld haben darf, da in der `WHERE`-Bedingung nur jeweils ein Wert verglichen wird. Haben Sie irrtümlich

mehrere Ausgabespalten definiert, wird Access mit der folgenden Fehlermeldung reagieren. Mehrere Ausgabefelder sind nur in Zusammenhang mit dem Operator `EXISTS` erlaubt, der weiter unten besprochen wird.

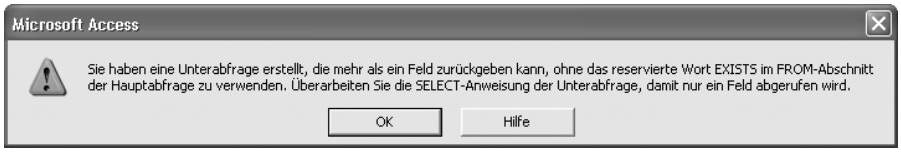


Bild 3.49: Fehlermeldung bei mehr als einem Ausgabefeld

Der `IN`-Operator wird auch im nächsten Beispiel genutzt, um alle Cocktails zu finden, die ohne die Zutat Gin gemixt werden. Dazu spürt die Unterabfrage die Nummern aller Cocktails auf, die Gin als Zutat verwenden. Durch die Anwendung von `NOT IN` in der `WHERE`-Bedingung werden als Resultat der Gesamtabfrage alle Drinks ausgegeben, die nicht in der Ergebnismenge der Unterabfrage vorkommen.

```
SELECT tblCocktail.Cocktail
FROM tblCocktail
WHERE tblCocktail.Cocktail NOT IN (SELECT DISTINCTROW
tblCocktail.Cocktail
FROM tblCocktail RIGHT JOIN (tblZutat LEFT JOIN tblCocktailzutaten ON
tblZutat.ZutatenNr = tblCocktailzutaten.ZutatenNr) ON
tblCocktail.CocktailNr = tblCocktailzutaten.CocktailNr
WHERE tblZutat.Zutat = "Gin");
```

Access verwendet Unterabfragen im »Abfrage-Assistenten zur Duplikatssuche«, der Ihnen im Auswahlmenü bei der Erstellung neuer Abfragen angeboten wird. Wir haben mithilfe des Assistenten eine Abfrage erstellt, die die Cocktail-Tabelle auf doppelte Cocktailnamen überprüft.

```
SELECT tblCocktail.Cocktail, tblCocktail.CocktailNr
FROM tblCocktail
WHERE tblCocktail.Cocktail IN (SELECT [Cocktail] FROM [tblCocktail] As
Tmp GROUP BY [Cocktail] HAVING Count(*)>1 )
ORDER BY tblCocktail.Cocktail;
```

Die Abfrage ergibt eine Liste der Cocktailbezeichnungen, die doppelt vorkommen. Die Unterabfrage gruppiert dazu die Cocktails nach `tblCocktail.Cocktail` und ermittelt für jede Gruppe die Anzahl der Datensätze mit `Count(*)`.

3.5.2 Unterabfragen mit einem Ergebniswert

Die Unterabfrage, die wir im nächsten Beispiel beschreiben möchten, ermittelt alle Cocktails, deren Alkoholgehalt kleiner als der durchschnittliche Alkoholgehalt aller Cocktails ist. Dazu wird zunächst eine Unterabfrage zusammengestellt, die den durchschnittlichen Gehalt an Alkohol ermittelt. Dieser Wert wird dann in einem `SELECT` zur Bestimmung der entsprechenden Cocktails verwandt.

```
SELECT tblCocktail.Cocktail, tblCocktail.Alkoholgehalt
FROM tblCocktail
WHERE tblCocktail.Alkoholgehalt < (SELECT Avg(tblCocktail.Alkoholgehalt)
FROM tblCocktail)
```

Die Rückgabemenge der Unterabfrage besteht hierbei nur aus einem Wert, mit dem der Vergleich der `WHERE`-Bedingung durchgeführt wird. Sie müssen sicherstellen, dass die Unterabfrage auch wirklich nur einen Wert als Resultat zurückliefert, denn sonst meldet Access einen Fehler mit dem im folgenden Bild gezeigten Dialogfeld.

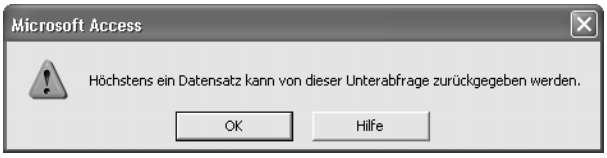


Bild 3.50: Fehlermeldung für Unterabfragen

Möchten Sie im Ergebnis der Abfrage zusätzlich erfahren, wie hoch der durchschnittliche Alkoholgehalt eigentlich war, erweitern Sie die Abfrage um ein entsprechendes Ausgabefeld.

```
SELECT tblCocktail.Cocktail, tblCocktail.Alkoholgehalt,
(SELECT Avg(tblCocktail.Alkoholgehalt) FROM tblCocktail) AS Mittelwert
FROM tblCocktail
WHERE tblCocktail.Alkoholgehalt<(SELECT Avg(tblCocktail.Alkoholgehalt)
FROM tblCocktail);
```

Unterabfragen können sowohl in Bedingungen als auch im Ausgabebereich eingesetzt werden.

3.5.3 Korrelierte Unterabfragen

Bei korrelierten Unterabfragen enthält die innere `SELECT`-Anweisung eine Spalte, deren Werte in der äußeren `SELECT`-Anweisung festgelegt sind.

Die folgende Abfrage zeigt eine einfache Unterabfrage, in der alle Cocktails bestimmt werden, in denen die Zutat mit der ZutatenNr 50 verwandt wird.

```
SELECT tblCocktail.Cocktail FROM tblCocktail
WHERE tblCocktail.CocktailNr IN (SELECT tblCocktailzutaten.CocktailNr
FROM tblCocktailzutaten
WHERE tblCocktailzutaten.ZutatenNr = 50);
```

Die gleiche Abfrage lässt sich auch als korrelierte Unterabfrage schreiben:

```
SELECT tblCocktail.Cocktail FROM tblCocktail
WHERE 50 IN (SELECT tblCocktailzutaten.ZutatenNr FROM tblCocktailzutaten
WHERE tblCocktail.CocktailNr = tblCocktailzutaten.CocktailNr);
```

3.5.4 Unterabfragen mit EXISTS

Mithilfe des EXISTS-Operators kann die Differenz bzw. mit NOT EXISTS der Durchschnitt zweier Tabellen bestimmt werden. Die Differenz zweier Tabellen besteht aus den Zeilen der ersten Tabelle, die nicht in der zweiten Tabelle auftreten. Der Durchschnitt zweier Tabellen beinhaltet alle Zeilen, die sowohl in der ersten als auch in der zweiten Tabelle vorkommen.

```
SELECT tblCocktail.Cocktail FROM tblCocktail
WHERE EXISTS (SELECT tblCocktailzutaten.ZutatenNr FROM tblCocktailzutaten
WHERE tblCocktail.CocktailNr = tblCocktailzutaten.CocktailNr
AND tblCocktailzutaten.ZutatenNr = 50);
```

3.6 UNION-Abfragen

UNION-Abfragen vereinigen die Ergebnisse zweier SELECT-Abfragen zu einem Resultat. Für die Definition von UNION-Abfragen müssen die SQL-Befehle direkt im SQL-Fenster eingegeben werden. Rufen Sie das entsprechende Fenster in der Entwurfsansicht mit *ABFRAGE SQL-spezifisch Union* auf. Im folgenden Bild ist eine UNION-Abfrage abgebildet, die eine Liste aller Cocktailbezeichnungen und Zutaten liefert. Durch das Befehlswort UNION werden zwei SELECT-Abfragen miteinander verbunden. Für eine erfolgreiche Verbindung zweier Abfragen müssen die Anzahl der Felder und die entsprechenden Feldtypen übereinstimmen.

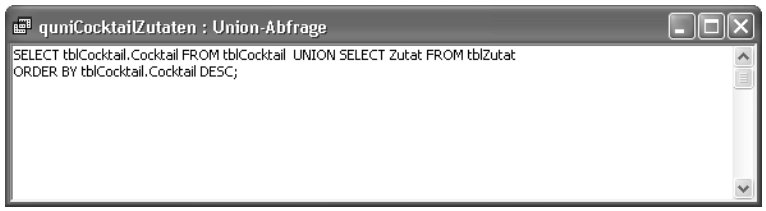


Bild 3.51: Definition der UNION-Abfrage

Soll die Abfrage die Daten sortiert ausgeben, können Sie ein Sortierkriterium mithilfe von `ORDER BY` festlegen, wobei nur die Felder der ersten `SELECT`-Abfrage als Sortierfelder benutzt werden können. Entsprechend sortiert

```
SELECT tblCocktail.Cocktail FROM tblCocktail  UNION SELECT tblZutat.Zutat
FROM tblZutat ORDER BY tblCocktail.Cocktail DESC;
```

wie Sie im nächsten Bild sehen können, die Cocktailbezeichnungen und Zutaten durcheinander, aber in umgekehrter Reihenfolge.

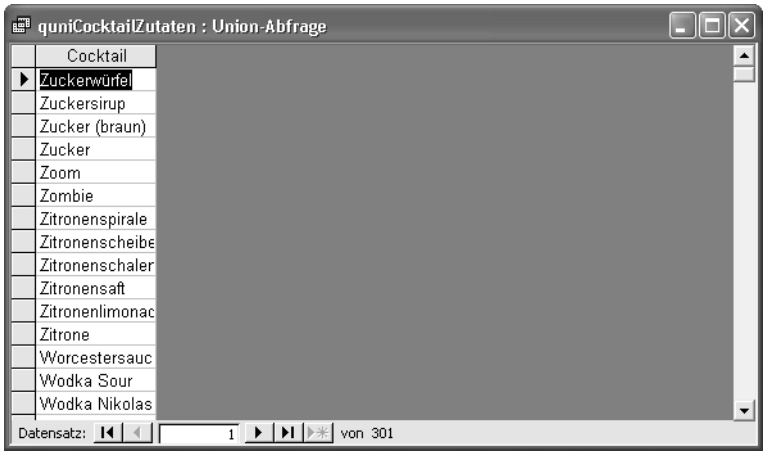


Bild 3.52: Ergebnis der UNION-Abfrage

Eine interessante `UNION`-Anwendung, die wir später in Kapitel 14, »Formulare«, einsetzen, möchten wir Ihnen noch vorstellen. Im folgenden Beispielformular wird eine Liste mit Cocktails gezeigt. Sie können ein oder mehrere Cocktailrezepte ausdrucken lassen, indem Sie die entsprechenden Cocktails selektieren. Der erste Eintrag der Liste »*** Alle Cocktails ***« ermöglicht den Ausdruck aller Rezepte.

Bild 3.53: Druckauswahlformular

Da dieser Eintrag nicht als Datensatz in der zugrunde liegenden Tabelle *tblCocktail* vorliegt, wird er mithilfe der `UNION`-Abfrage

```
SELECT tblCocktail.Cocktail, tblCocktail.Cocktailnr FROM tblCocktail
UNION SELECT "*** Alle Cocktails ***",0 FROM tblCocktail
ORDER BY tblCocktail.Cocktail;
```

für die Liste erzeugt. Die `UNION`-Abfrage zeigt, dass zur Erzeugung der Liste mit einem Trick gearbeitet wird. Die zweite `SELECT`-Abfrage liefert nur einen Datensatz mit zwei Feldern, nämlich `"*** Alle Cocktails ***"` und `0`. Beide basieren nicht auf der angegebenen Tabelle *tblCocktail*. Eine Tabelle muss aber in der `FROM`-Klausel angegeben werden, da sonst die Abfrage nicht bearbeitet werden kann. Der Wert `0` kommt als Cocktailnummer in *tblCocktail* nicht vor, deshalb wurde der Wert zur Erkennung des speziellen Eintrags gewählt.

Die Abfrage hat mit unseren Beispieldaten das im folgenden Bild gezeigte Ergebnis.

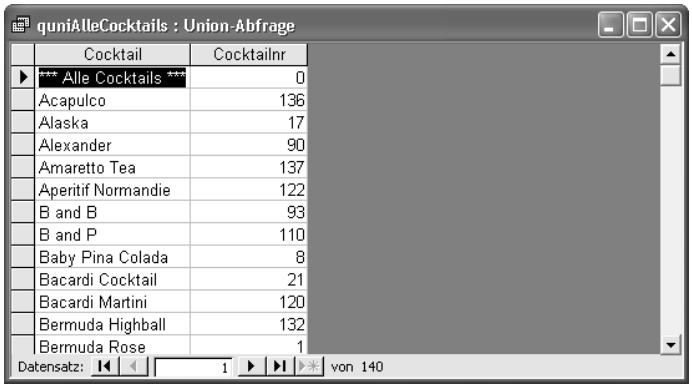


Bild 3.54: Ergebnis der geänderten UNION-Abfrage

3.7 Kreuztabellenabfragen

Eine Spezialität von Access sind Kreuztabellenabfragen. Mit ihrer Hilfe können Datenbestände schnell nach verschiedenen Kriterien abgefragt werden. Bei der Erstellung von Kreuztabellen unterstützt Sie ein Assistent. Wir möchten Ihnen im folgenden Abschnitt den Einsatz von Kreuztabellen in zwei Beispielen zeigen und die SQL-Erweiterungen in Access für Kreuztabellen ansprechen.

3.7.1 Cocktails in Gruppen und Kategorien

Im ersten Beispiel soll die Zugehörigkeit der Cocktails zu verschiedenen Gruppen und Kategorien dargestellt werden. Dazu ermittelt die SQL-Abfrage

```
SELECT DISTINCTROW tblCocktail.Cocktail, tblKategorie.Kategorie,
tblGruppe.Gruppe
FROM tblKategorie RIGHT JOIN ((tblGruppe RIGHT JOIN tblCocktail ON
tblGruppe.GruppeNr = tblCocktail.GruppeNr) LEFT JOIN tblCocktailKategorie
ON tblCocktail.CocktailNr = tblCocktailKategorie.CocktailNr) ON
tblKategorie.KategorieNr = tblCocktailKategorie.KategorieNr;
```

zunächst das im nächsten Bild gezeigte Ergebnis. In der ersten Spalte sind alle Cocktails aufgeführt, in den weiteren Spalten die entsprechenden Kategorien und Gruppen.

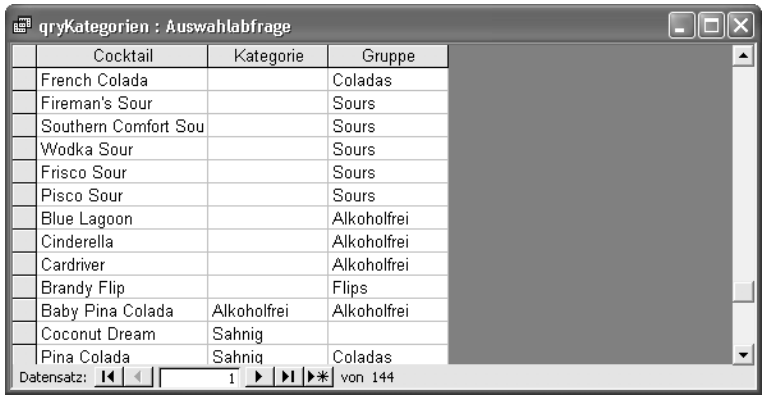
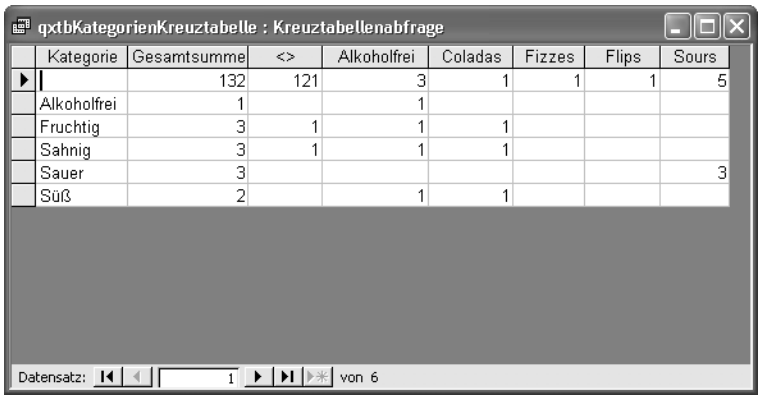


Bild 3.55: Ergebnis der Abfrage

Möchten Sie nun wissen, wie viele Cocktails welchen Kategorien und Gruppen zugeordnet sind, können Sie dafür eine Kreuztabelle einsetzen. Sie erstellen eine Kreuztabelle am einfachsten mit dem Kreuztabellenabfrage-Assistenten auf der

Grundlage der eben erstellten Abfrage. Das folgende Bild stellt das Ergebnis unserer Kreuztabelle dar.



Kategorie	Gesamtsumme	<>	Alkoholfrei	Coladas	Fizzes	Flips	Sours
	132	121	3	1	1	1	5
Alkoholfrei	1		1				
Fruchtig	3	1	1	1			
Sahnig	3	1	1	1			
Sauer	3						3
Süß	2		1	1			

Bild 3.56: Ergebnis der Kreuztabellenabfrage

In der Entwurfsansicht wird für Kreuztabellen eine zusätzliche Zeile *Kreuztabelle* eingefügt, die die Funktion des Felds innerhalb der Kreuztabelle beschreibt.

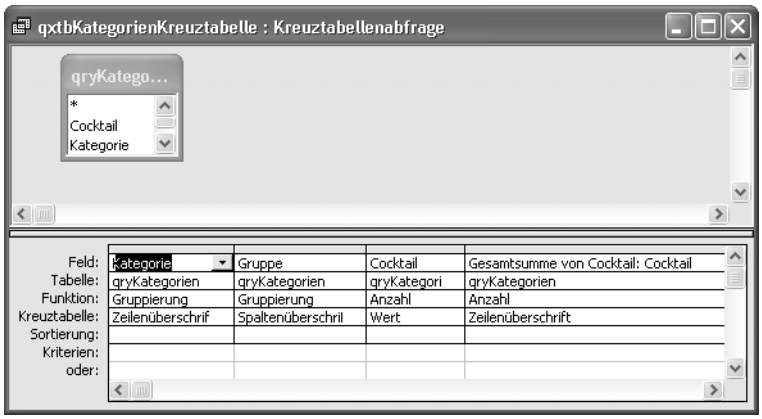


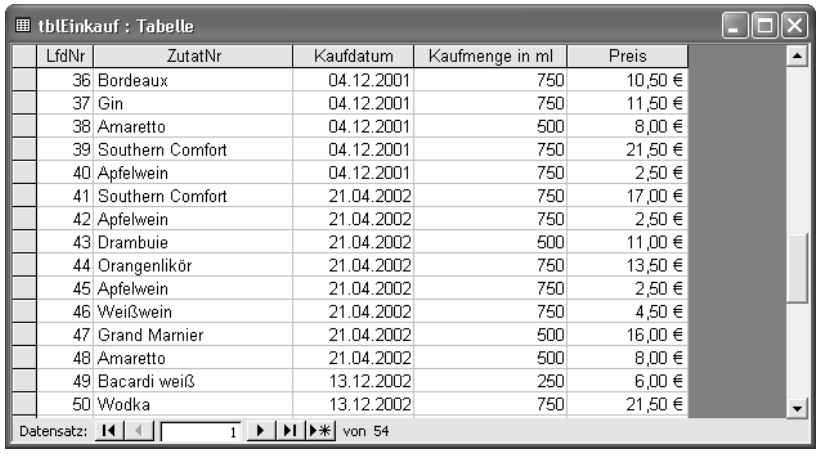
Bild 3.57: Entwurfsansicht der Kreuztabellenabfrage

Die Definition der Entwurfsansicht wird wie folgt in SQL umgeformt: Hinter dem Access-spezifischen SQL-Befehl `TRANSFORM` wird das Feld benannt, das als *Wert* in der Tabelle errechnet werden soll. Die *Zeilenüberschriften* legen Sie hinter `SELECT` fest. Die Spaltenüberschriften vereinbart der Access-eigene SQL-Befehl `PIVOT`.

```
TRANSFORM Count(qryKategorien.Cocktail) AS AnzahlvonCocktail
SELECT qryKategorien.Kategorie, Count(qryKategorien.Cocktail) AS
[Gesamtsumme von Cocktail]
FROM qryKategorien
GROUP BY qryKategorien.Kategorie
PIVOT qryKategorien.Gruppe;
```

3.7.2 Der Verbrauch pro Jahr

Für das folgende Beispiel soll ausgerechnet werden, welche Beträge pro Jahr für Spirituosen zum Mischen ausgegeben wurden. Dazu wurde zunächst eine neue Tabelle *tblEinkauf* angelegt, die neben dem Einkaufsdatum und der eingekauften Menge auch den Preis der gekauften Spirituosen aufzeigt.



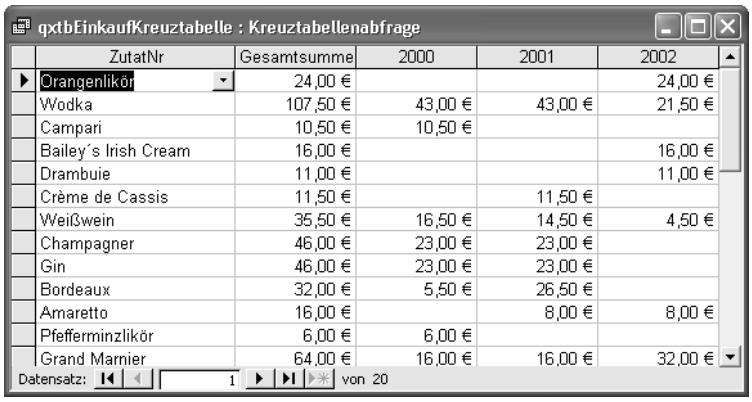
tblEinkauf : Tabelle

LfdNr	ZutatNr	Kaufdatum	Kaufmenge in ml	Preis
36	Bordeaux	04.12.2001	750	10,50 €
37	Gin	04.12.2001	750	11,50 €
38	Amaretto	04.12.2001	500	8,00 €
39	Southern Comfort	04.12.2001	750	21,50 €
40	Apfelwein	04.12.2001	750	2,50 €
41	Southern Comfort	21.04.2002	750	17,00 €
42	Apfelwein	21.04.2002	750	2,50 €
43	Drambuie	21.04.2002	500	11,00 €
44	Orangenlikör	21.04.2002	750	13,50 €
45	Apfelwein	21.04.2002	750	2,50 €
46	Weißwein	21.04.2002	750	4,50 €
47	Grand Marnier	21.04.2002	500	16,00 €
48	Amaretto	21.04.2002	500	8,00 €
49	Bacardi weiß	13.12.2002	250	6,00 €
50	Wodka	13.12.2002	750	21,50 €

Datensatz: 1 von 54

Bild 3.58: Tabelle tblEinkauf

Daraus wurde mit dem Kreuztabellenabfrage-Assistenten eine neue Kreuztabelle erstellt, die für die einzelnen Spirituosen pro Jahr den ausgegebenen Betrag darstellt sowie für jede eingekaufte Spirituose die Summe über die Jahre 2000 bis 2002 errechnet.



qxtbEinkaufKreuztabelle : Kreuztabellenabfrage

ZutatNr	Gesamtsumme	2000	2001	2002
Orangenlikör	24,00 €			24,00 €
Wodka	107,50 €	43,00 €	43,00 €	21,50 €
Campari	10,50 €	10,50 €		
Bailey´s Irish Cream	16,00 €			16,00 €
Drambuie	11,00 €			11,00 €
Crème de Cassis	11,50 €		11,50 €	
Weißwein	35,50 €	16,50 €	14,50 €	4,50 €
Champagner	46,00 €	23,00 €	23,00 €	
Gin	46,00 €	23,00 €	23,00 €	
Bordeaux	32,00 €	5,50 €	26,50 €	
Amaretto	16,00 €		8,00 €	8,00 €
Pfefferminzlikör	6,00 €	6,00 €		
Grand Marnier	64,00 €	16,00 €	16,00 €	32,00 €

Datensatz: 1 von 20

Bild 3.59: Kreuztabelle zum Bestimmen des Verbrauchs an Spirituosen pro Jahr

In der Entwurfsansicht sehen Sie, wie hierbei für die Jahresspalten die Format-Funktion eingesetzt wird.

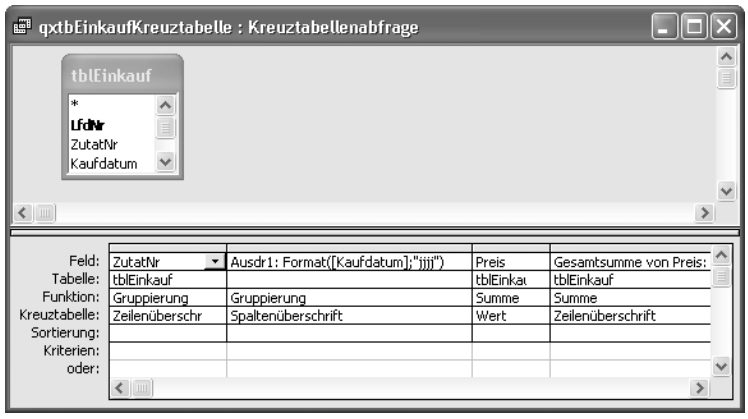


Bild 3.60: Entwurfsansicht der Kreuztabelle

Die entsprechende SQL-Anweisung sieht folgendermaßen aus:

```
TRANSFORM Sum(tblEinkauf.Preis) AS SummevonPreis
SELECT tblEinkauf.ZutatNr, Sum(tblEinkauf.Preis) AS [Gesamtsumme von
Preis]
FROM tblEinkauf
GROUP BY tblEinkauf.ZutatNr
PIVOT Format([Kaufdatum],"yyyy");
```

3.8 Ein komplexes Beispiel

Die einfache Fragestellung: »Welche Cocktails kann ich mit den Zutaten in meiner Hausbar mixen?« führt zu einer komplexen Abfrage. Wir möchten Ihnen im Folgenden zwei Varianten zur Lösung des Problems vorstellen. Im ersten Fall wurde die Fragestellung mit einer Access-typischen Lösungsvariante beantwortet. Für die zweite Variante wurden Unterabfragen eingesetzt.

Beide Lösungen arbeiten mit dem gleichen Grundschema. Ein Cocktail lässt sich dann mit den Zutaten der Hausbar mixen, wenn die Anzahl der Zutaten für den Cocktail gleich der Anzahl der Zutaten der Hausbar ist. Allerdings darf dabei nicht die Gesamtzahl der Zutaten der Hausbar verwendet werden, sondern es werden durch eine Verknüpfung mit der Cocktailzutatenliste nur die tatsächlich benötigten Zutaten ausgewertet. Dabei sollen auch die in der Hausbar noch vorhandenen Mengen berücksichtigt werden.

Beide Lösungen sind, zumindest mit dem kleinen Datenbestand der Cocktail-Datenbank, ungefähr gleich schnell.

Wir verwenden die Abfragen in einem Beispiel in Kapitel 14, »Formulare«. Dort wird in einem Formular zwischen der Anzeige aller Cocktails und der Anzeige nur der Cocktails, die mit den Zutaten der Hausbar gemixt werden können, umgeschaltet.

3.8.1 Aufeinander aufbauende Abfragen

Zur Beantwortung der Fragestellung sind für den ersten Ansatz drei aufeinander aufbauende Abfragen notwendig. Die erste Abfrage

```
SELECT Count(*) AS [ZAnzahl], tblCocktailzutaten.CocktailNr
FROM tblCocktailzutaten
GROUP BY tblCocktailzutaten.CocktailNr;
```

ermittelt die Anzahl der Zutaten pro Cocktail. Wir verwenden dabei aus Leistungsgründen die Funktion `Count(*)`, alternativ könnte `Count([tblCocktailzutaten.CocktailZutatenNr])` eingesetzt werden. Die Abfrage wurde als »qryHausbar1« abgespeichert.

Die zweite Abfrage, abgelegt unter dem Namen »qryHausbar2«, zählt ebenfalls die Anzahl der Zutaten eines Cocktails, allerdings mit der Bedingung, dass sich die entsprechenden Zutaten in der Hausbar befinden.

```
SELECT Count(*) AS [ZAnzahl], tblCocktailzutaten.CocktailNr
FROM tblCocktailzutaten INNER JOIN tblHausbar ON
tblCocktailzutaten.ZutatenNr = tblHausbar.ZutatenNr
GROUP BY tblCocktailzutaten.CocktailNr;
```

Diese Einschränkung wird mithilfe einer Verknüpfung zwischen den Tabellen *tblCocktailZutaten* und *tblHausbar* definiert. Die Definition der Einschränkung ist der entscheidende Teil der Abfrage, denn hier wird die eigentliche Verbindung zwischen Hausbar und Cocktail hergestellt.

Wir haben die zweite Abfrage um eine Prüfung ergänzt, ob die in der Hausbar vorhandene Menge überhaupt ausreicht. Hierzu wurde eine `WHERE`-Klausel hinzugefügt, die allerdings sehr aufwändig ist. Aufgrund der verschiedenen Einheiten für die Mengenangaben ist eine Umrechnung der Einheiten erforderlich, um die Mengen vergleichen zu können. Das führt dazu, dass die Einheitentabelle zweimal, als *tblEinheiten* und als *tblEinheiten_1*, mit der Zutaten- und der Hausbartabelle verknüpft wird. Um Nullwerte in der `WHERE`-Klausel abzufangen und zu behandeln, haben wir die Funktion `Nz()` eingesetzt, die Nullwerte zur Zahl 0 umsetzt.

```
SELECT Count(*) AS [ZAnzahl], tblCocktailzutaten.CocktailNr
FROM tblEinheiten INNER JOIN (tblEinheiten AS tblEinheiten_1 INNER JOIN
(tblCocktailzutaten INNER JOIN tblHausbar ON tblCocktailzutaten.ZutatenNr
= tblHausbar.ZutatenNr) ON tblEinheiten_1.EinheitenNr =
tblCocktailzutaten.EinheitenNr) ON tblEinheiten.EinheitenNr =
tblHausbar.EinheitenNr
WHERE Nz([tblCocktailzutaten].[Menge]) *
Nz([tblEinheiten_1].[Umrechnung_cl]) <= Nz([tblHausbar].[Menge]) *
Nz([tblEinheiten].[Umrechnung_cl])
GROUP BY tblCocktailzutaten.CocktailNr;
```

In der Entwurfsansicht ist die Abfrage leichter zu übersehen.

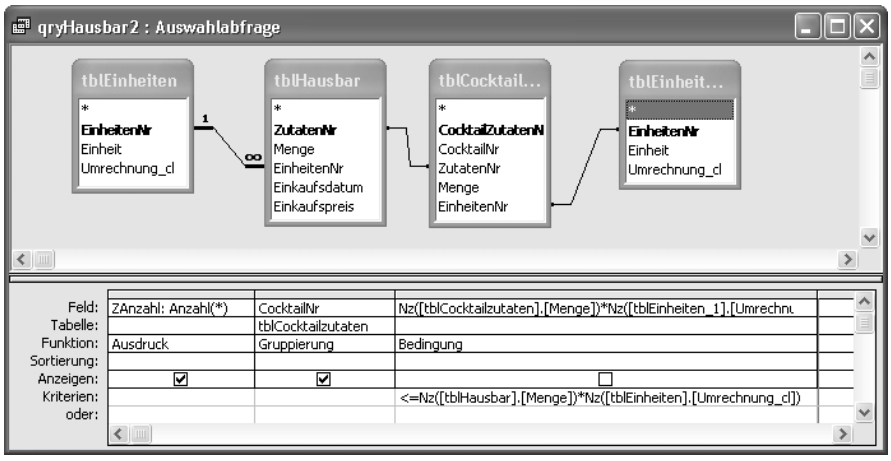


Bild 3.61: Abfrage »qryHausbar2«

Im letzten Schritt wird ermittelt, für welche Cocktails die Anzahl der benötigten Zutaten und die Anzahl der auch in der Hausbar vorhandenen Zutaten gleich ist.

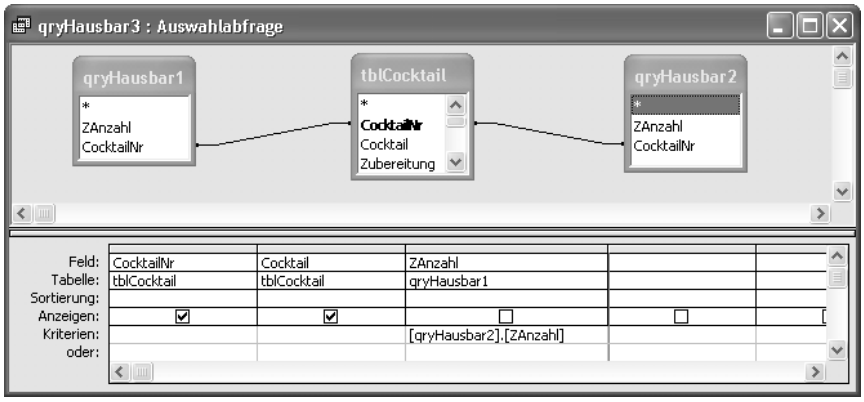


Bild 3.62: Abfrage »qryHausbar3«

Access setzt die Entwurfsansicht zu dem folgenden SQL-Befehl um.

```
SELECT tblCocktail.CocktailNr, tblCocktail.Cocktail
FROM (tblCocktail INNER JOIN qryHausbar1 ON tblCocktail.CocktailNr =
qryHausbar1.CocktailNr) INNER JOIN qryHausbar2 ON tblCocktail.CocktailNr
= qryHausbar2.CocktailNr
WHERE [qryHausbar1].[ZAnzahl]=[qryHausbar2].[ZAnzahl];
```

3.8.2 Mit Unterabfragen

Die Fragestellung, welche Cocktails mit den Zutaten aus der Hausbar möglich sind, lässt sich auch mit einer einzigen Abfrage lösen. Das Grundprinzip ist gleich, d.h., sowohl in der Cocktailzutaten-Tabelle als auch in der Hausbar wird die Anzahl der Zutaten bezogen auf einen bestimmten Cocktail ermittelt.

In der `WHERE`-Klausel werden die Ergebnisse zweier `SELECT`-Abfragen verglichen. Vereinfacht dargestellt hat die `WHERE`-Klausel die Form

```
WHERE (SELECT Anz. Cocktailzutaten) = (SELECT Anz. Hausbarzutaten)
```

In den `SELECTs` wird jeweils die Bedingung

```
WHERE tblCocktailzutaten.Cocktailnr = tblCocktail.CocktailNr
```

verwandt, die eine korrelierte Verknüpfung der Unterabfragen aufbaut. Ausführlich lässt sich die Abfrage

```
SELECT tblCocktail.CocktailNr, tblCocktail.Cocktail
FROM tblCocktail
WHERE
(SELECT Count(*) AS ZAnzahl
FROM tblCocktailzutaten INNER JOIN tblHausbar ON
tblCocktailzutaten.ZutatenNr = tblHausbar.ZutatenNr
WHERE tblCocktailzutaten.CocktailNr = tblCocktail.CocktailNr
GROUP BY tblCocktailzutaten.CocktailNr)
=
(SELECT Count(*) AS ZAnzahl
FROM tblCocktailzutaten
WHERE tblCocktailzutaten.Cocktailnr = tblCocktail.CocktailNr);
```

schreiben, wobei keine Mengen und Einheiten berücksichtigt wurden.

Mit der Berücksichtigung von Mengen und Einheiten wird die Abfrage ziemlich kompliziert und lässt sich nur schwer lesen.

```
SELECT tblCocktail.CocktailNr, tblCocktail.Cocktail
FROM tblCocktail
WHERE
(SELECT Count(*) AS ZAnzahl
FROM tblEinheiten INNER JOIN (tblEinheiten AS tblEinheiten_1 INNER JOIN
(tblCocktailzutaten INNER JOIN tblHausbar ON tblCocktailzutaten.ZutatenNr
= tblHausbar.ZutatenNr) ON tblEinheiten_1.EinheitenNr =
```

```
tblCocktailzutaten.EinheitenNr) ON tblEinheiten.EinheitenNr =
tblHausbar.EinheitenNr
WHERE tblCocktailzutaten.Cocktailnr = tblCocktail.CocktailNr and
(Nz([tblCocktailzutaten].[Menge])*Nz([tblEinheiten_1].[Umrechnung_cl]))<=
Nz([tblHausbar].[Menge])*Nz([tblEinheiten].[Umrechnung_cl]))
=
(SELECT Count(*) AS ZAnzahl
FROM tblCocktailzutaten
WHERE tblCocktailzutaten.CocktailNr = tblCocktail.CocktailNr);
```

3.9 Zugriff auf andere Datenbanken

Der Access-SQL-Dialekt ermöglicht es Ihnen, direkt auf Tabellen und Abfragen anderer Access- oder ODBC-Datenbanken zuzugreifen (ODBC erläutern wir Ihnen in Kapitel 25, »Client/Server-Verarbeitung«).

In der Entwurfsansicht von Abfragen können Sie die Eigenschaften *Quelldatenbank* und *Quellverbindung* setzen. Dort geben Sie entweder unter *Quelldatenbank* Pfad und Namen einer Access-Datenbank an oder legen unter *Quellverbindung* die Verbindungsdaten beispielsweise zu einer ODBC-Datenbank fest.

Wird als Quelldatenbank beispielsweise C:\PREISLISTE.MDB angegeben, könnte eine Auswahlabfrage auf die Tabelle *tblPreise* wie folgt aussehen:

```
SELECT *
FROM tblPREISE
IN 'C:\PREISLISTE.MDB';
```

Es ist allerdings für Sie übersichtlicher und zudem von der Verarbeitungsleistung schneller, wenn Sie auf Tabellen anderer Datenbanken über verknüpfte Tabellen zugreifen, wie es in Kapitel 23, »Anwendungsentwicklung«, gleich zu Beginn beschrieben wird.

3.10 Zusammenfassung des SELECT-Befehls

Der SELECT-Befehl ist der am häufigsten verwendete SQL-Befehl. Er ist aber sehr vielseitig und wird dadurch unübersichtlich. Wir haben aus diesem Grund ein Syntax-Diagramm zusammengestellt, mit dessen Hilfe er leicht zu generieren ist.

Um das Schema leichter lesen zu können, zeigt die folgende Übersicht die verwendeten Symbole.

Beginn des Befehls

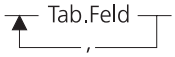
Kommando geht über die Zeile

Ende des Befehls

Wenn sich ein Schlüsselwort unterhalb der Linie befindet, handelt es sich um ein optionales Schlüsselwort.

Befinden sich mehrere Schlüsselwörter übereinander unter der Linie, kann eines davor ausgewählt werden. Ist eines der Schlüsselwörter unterstrichen, handelt es sich dabei um den Standardwert, der automatisch verwendet wird, auch wenn sich keines der Schlüsselwörter in der Befehlsfolge befindet.

Befindet sich ein Schlüsselwort einer Auswahl auf der Linie, so bedeutet das, dass eines der Schlüsselwörter der Auswahl selektiert werden muss.



Ein Pfeil, der zurück an den Anfang eines Arguments zeigt, gibt an, dass das Argument wiederholt werden kann. Das Zeichen, das Sie auf dem Rückwärtspfeil finden, ist als Trennzeichen zwischen den Argumenten zu verwenden.

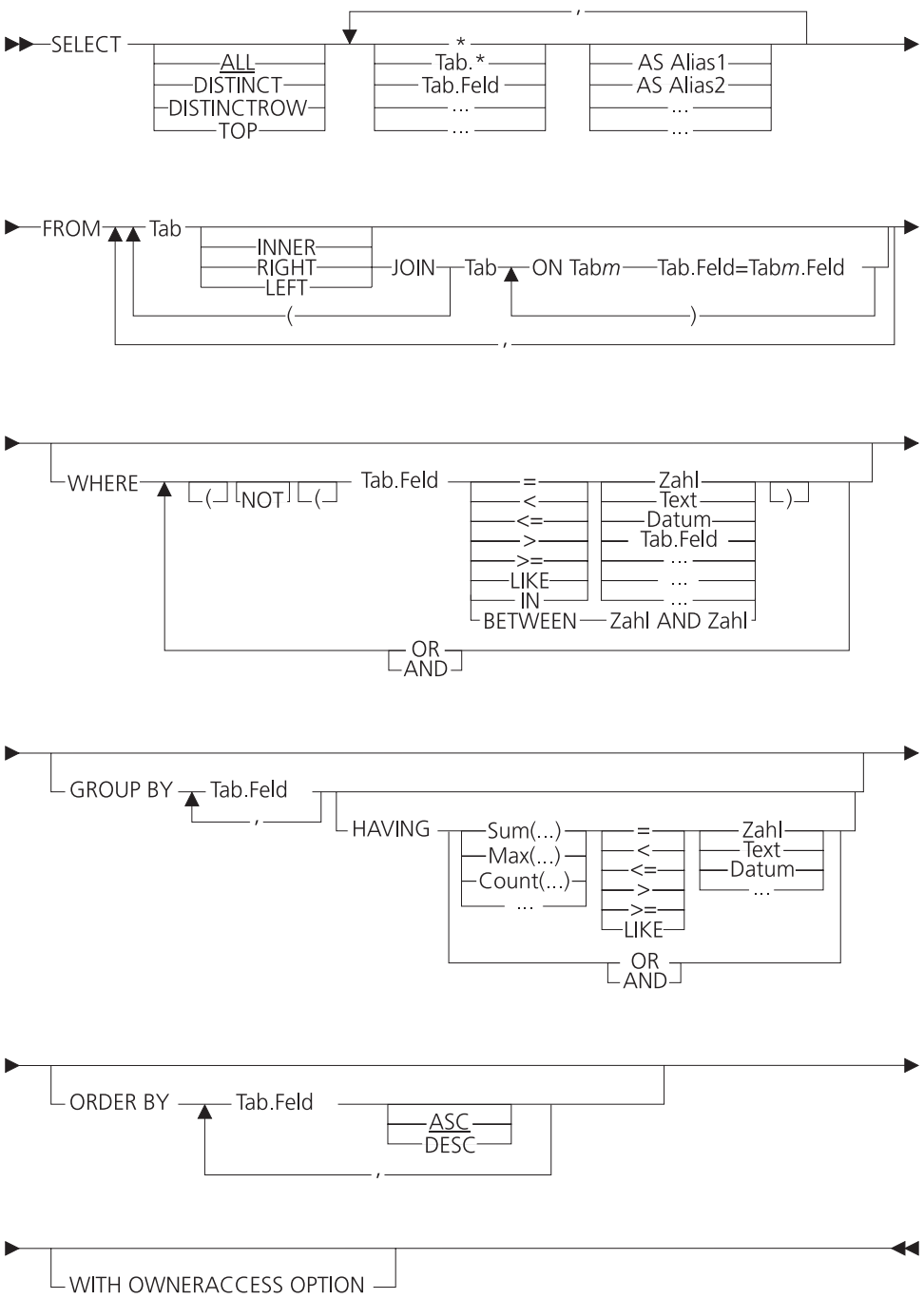


Bild 3.63: Syntax-Diagramm für den SQL-Befehl SELECT