2

SQL-SELECT-Grundlagen

Bisher haben Sie die Struktur der relationalen Datenbanken kennengelernt und erfahren, welchen Bezug diese Struktur zur AS/400 hat. Nun erhalten Sie Informationen über den Kern von SQL: Die SELECT- Anweisung. SELECT ist der wichtigste und nützlichste Teil von SQL, der verwendet wird, um Daten von einer AS/400-Tabelle zu einem Zielprogramm abzurufen. SELECT ist einfach erlernbar, aber schwer zu beherrschen, weil die Möglichkeiten für die Zusammensetzung von Anweisungen unendlich groß sind.

Sobald Sie die Grundlagen der SELECT-Anweisungen beherrschen, können Sie diese Daten von Ihrer AS/400 in Ihre Programme abrufen und diese Daten auf wirklich interessante Weise weiterverarbeiten. In diesem Kapitel erhalten Sie die Grundlagen zu SELECT, die Sie für die nachfolgenden weiter fortgeschrittenen Kapitel benötigen.

Was ist eine SELECT-Anweisung?

Die SELECT-Anweisung ist das *Wer, Was, Wann* und *Wo* für den Abruf von Daten aus Ihrer Datenbank. Beachten Sie jedoch, dass es nicht um das Wie des Abrufs geht. Sie als Programmierer geben an, welche Daten (Spalten) Sie wünschen, wo die Daten herstammen (aus welchen Tabellen) und wann Datensätze gewählt werden sollten, aber Sie sagen dem PC nicht, wie er die Datensätze abrufen sollte. Das ist das Schöne an SQL – der Computer entscheidet automatisch, wie die Abfrage am besten implementiert wird, um Ihnen die maximale Leistung zu ermöglichen. Tabelle 2.1 listet die SQL-SELECT-Klauseln auf.

Klausel	**Beschreibung**
FROM	Gibt an, welche Tabellen und Views gelesen werden sollen, um die Daten aufzufinden.
WHERE	Spezifiziert Bedingungen, die die Daten erfüllen müssen, um als Ergebnismenge (Result Set) ausgegeben zu werden.
ORDER BY	Schreibt die Reihenfolge der zurückgegebenen Datensätze vor.
GROUP BY	Erlaubt Ihnen, Ergebnisse nach Spalten zu aggregieren, die in der SELECT Anweisung aufgelistet werden.
HAVING	Erlaubt Ihnen, nach der Aggregation Datensatzauswahlbedingungen zu spezifizieren, wenn Sie eine GROUP BY-Klausel verwenden. Damit wird die Ergebnismenge weiter reduziert.

Tab. 2.1: SQL-SELECT-Klauseln

Die beiden wichtigsten Wörter

Die beiden wichtigsten Wörter in einer SELECT-Anweisung sind SELECT und FROM.

Diese beiden Wörter spezifizieren das „was erhalte ich“ und „wo stammt es her“ des SQL-Datenzugriffs. Eine SELECT-Anweisung muss immer mit dem SELECT-Schlüsselwort beginnen und immer eine FROM-Klausel enthalten. Das FROM kann eine Tabelle, View oder einen Index angeben, der von der DDS erstellt wurde. Wenn der Index mit einer SQL CREATE-Anweisung erstellt wird, kann auf den Index in der FROM-Klausel verwiesen werden.

Der Schlüssel für das Verständnis einer SELECT-Anweisung ist wiederum die Tatsache, dass Sie angeben müssen, was Sie sehen möchten und von wo die AS/400 die Daten abrufen muss. Denken Sie nur an die Datenbanktabelle WEBTEMP, die auf die AS/400 heraufgeladen wurde. (Das Heraufladen wird in Anhang C behandelt.) Hier die DDL für die Tabelle WEBTEMP:

```
CREATE TABLE WEBTEMP
             (REQGEO CHAR(20),
             REQTYPE CHAR(10),
             REQFILE CHAR(80),
             BROWSER CHAR(10),
             REQTS TIMESTAMP,
             REQSIZE INTEGER,
             REQUSER INTEGER))
```

Diese fiktionale Tabelle repräsentiert die Datensätze von zwei Monaten mit den Besuchen auf einer Webseite. Jede Zeile gibt eine Anforderung für ein Dokument vom Webserver an. Die erste Spalte, WEBGEO, ist das Land, aus dem die Anforderung stammt.

Sie enthält Werte wie USA und CANADA. Die zweite Spalte, REQTYPE, gibt an, ob die Anforderung vom Typ HEAD war, vom Typ GET oder POST. Die Spalte REQFILE enthält den Namen des Dokuments, das vom Web-Client angefordert wurde. Die Spalte REQTS enthält den Zeitstempel für den Zeitpunkt der Anforderung. Die Spalte BROWSER gibt Auskunft über die Client-Browser-Plattform und die Version, von der die Anforderung stammt, und REQSIZE ist die Größe des Datentransfers in Kilobyte. REQUSER ist schließlich ein Integer, der den Computer, der das Webdokument anfordert, eindeutig kennzeichnet.

Die Tabelle WEBTEMP wird als Grundlage für alle SELECT-Anweisungen in diesem Kapitel dienen. Es folgen Erläuterungen zu den SELECT-Anweisungen.

Ihre erste Abfrage

Folgende SQL-Anweisung wird eine Liste aller REQFILE-Werte zurückgeben, die in der Tabelle WEBTEMP in der AS/400-Bibliothek SQLBOOK gespeichert sind:

```
SELECT REQFILE FROM SQLBOOK.WEBTEMP
```

Die Ausgabe der Abfrage wird in Abbildung 2.1 gezeigt.

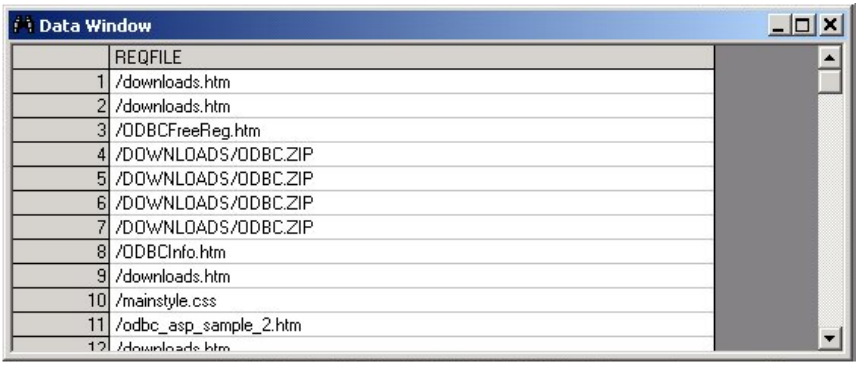


Abb. 2.1: HIer die Ergebnisse für Ihre erste SQL-Abfrage. Wie Sie sehen, wird mit der Abfrage nur die REQFILE-Spalte aus der Tabelle WEBTEMP abgefragt.

Bevor Sie genauer untersuchen, wie diese Abfrage funktioniert, müssen Sie noch eine Aufgabe erledigen. Die nächste Anweisung wird die Tabelle WEBTEMP öffnen und lesen und die Werte zurückgeben, die in den Feldern REQTS, REQGEO und REQFILE gespeichert sind.

```
SELECT REQTS, REQGEO, REQFILE FROM
  SQLBOOK.WEBTEMP
```

Sehen wir uns nun die Anweisung im Einzelnen an. Sofort nach dem SELECT-Schlüsselwort geben Sie an, dass Sie möchten, dass die AS/400 die Spalten REQTS, REQGEO und REQFILE zurückgibt. Nach der Liste von Feldern gibt die FROM-Klausel an, dass diese Felder aus der Tabelle WEBTEMP in der AS/400-Bibliothek SQLBOOK gelesen werden sollen. So einfach ist das – die Ausgabe aus der Abfrage wird in Abbildung 2.2 gezeigt.

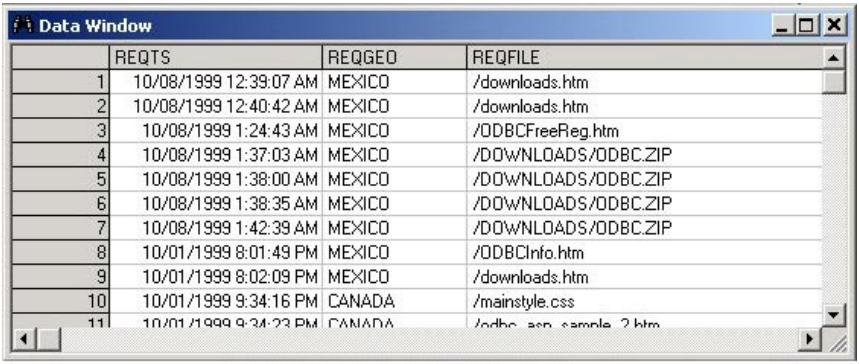


Abb. 2.2: Dieser Screenshot repräsentiert die Ausgabe der zweiten Abfrage. Beachten Sie, dass nun drei Spalten anstatt einer angezeigt werden.

Eine Liste mit Spalten

Das Schöne dabei ist, dass diese Spaltenliste keine echten Datenbankfelder enthalten muss, sondern auch *Spaltenfunktionen* oder *Skalarfunktionen* enthalten kann. Eine Skalarfunktion ist eine Funktion, die einen Wert für jede Zeile ausgibt, die von der Abfrage zurückgegeben wird. Sie kann eine Spalte oder einen Ausdruck als Argument erhalten. Eine Spaltenfunktion kann für die Gruppierung von Werten verwendet werden. Die Eingaben erfolgen in Form von mehreren Reihen. Später erfahren Sie mehr über Spaltenfunktionen. Zunächst finden Sie hier eine SQL-Anweisung, die eine Spaltenfunktion verwendet, um die Datensätze in der Tabelle WEBTEMP zu zählen:

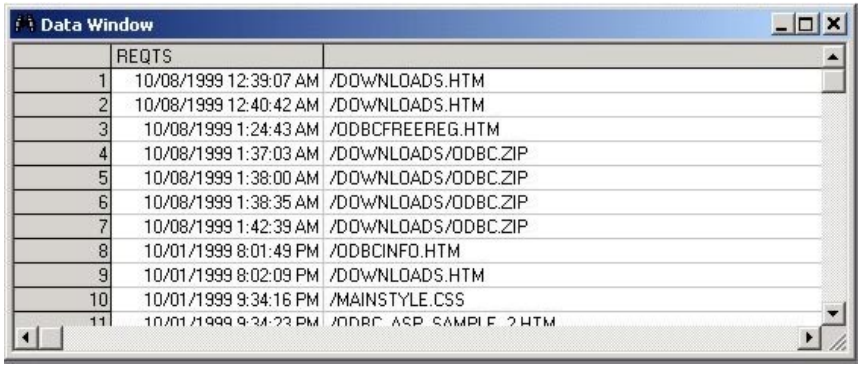
```
SELECT COUNT(*) FROM SQLBOOK.WEBTEMP
```

Diese Anweisung liest alle Datensätze in der Tabelle WEBTEMP ein und übergibt sie der Funktion COUNT. Die Funktion COUNT gibt die Anzahl der Datensätze in der Tabelle zurück. Daher gibt die Anweisung als Ergebnis eine einzelne Zeile zurück. Diese Zeile enthält die Anzahl der Zeilen in der Tabelle WEBTEMP, 3.035.

Hier ist eine weitere Anweisung, die eine Funktion einsetzt, aber in diesem Fall wird die Skalarfunktion UPPER verwendet, um das Feld REQFILE in Großbuchstaben zu schreiben:

```
SELECT REQTS, UPPER(REQFILE) FROM
  SQLBOOK.WEBTEMP
```

Diese Anweisung gibt die Spalten REQTS und REQFILE aus, aber bevor die Spalte REQFILE zurückgegeben wird, erfolgt eine Bearbeitung durch die Funktion UPPER, um sicherzustellen, dass der Dokumentnamen in Großbuchstaben dargestellt wird. Abbildung 2.3 zeigt die Ausgabe der Abfrage.



	REQTS	
1	10/08/1999 12:39:07 AM	/DOWNLOADS.HTM
2	10/08/1999 12:40:42 AM	/DOWNLOADS.HTM
3	10/08/1999 1:24:43 AM	/ODBCFREEREG.HTM
4	10/08/1999 1:37:03 AM	/DOWNLOADS/ODBC.ZIP
5	10/08/1999 1:38:00 AM	/DOWNLOADS/ODBC.ZIP
6	10/08/1999 1:38:35 AM	/DOWNLOADS/ODBC.ZIP
7	10/08/1999 1:42:39 AM	/DOWNLOADS/ODBC.ZIP
8	10/01/1999 8:01:49 PM	/ODBCINFO.HTM
9	10/01/1999 8:02:09 PM	/DOWNLOADS.HTM
10	10/01/1999 9:34:16 PM	/MAINSTYLE.CSS
11	10/01/1999 9:34:23 PM	/ODBC_ASP_SAMPLE_2.HTM

Abb. 2.3: Diese Ausgabe reflektiert, wie Daten unter Verwendung von Skalarfunktionen eingesetzt werden und in Großbuchstaben zurückgegeben werden können.

Umbenennung von Spalten in der Spaltenliste

Ein wichtiger, aber häufig übersehener Punkt bei der Spaltenliste ist, dass Sie in Ihrem Anwendungsprogramm normalerweise auf Spalten mit den Spaltennamen verweisen. SQL bietet einen Mechanismus für die Umbenennung von Spalten in der zurückgegebenen Spaltenliste, und diese Funktionalität ist besonders wichtig aus der Perspektive eines Anwendungsentwicklers. Sehen Sie sich folgende Anweisung an:

```
SELECT REQTS FROM SQLBOOK.WEBTEMP
```

Die Anweisung SELECT gibt eine Liste von Zeilen zurück, die die Spalte REQTS enthält. Ein Anwendungsprogramm wird voraussichtlich den Spaltennamen REQTS verwenden, um auf den aktuellen Wert zu verweisen. In VB sieht das eventuell so aus:

```
X = MyRecordSet.Fields(„REQTS“).Value
```

Was wird jedoch geschehen, wenn Sie eine Spalte erstellen, bei der es sich um einen Ausdruck handelt? In diesem Fall erstellt SQL einen Spaltennamen für die virtuelle Spalte, weil SQL den Spaltennamen nicht anhand des Systemkatalogs feststellen kann. Wenn SQL einen virtuellen Spaltennamen generiert, wie nennen Sie diese Spalte dann in Ihrem Code? Hier ist die Antwort:

```
SELECT COUNT(REQTS) AS MYCOUNT FROM
  SQLBOOK.WEBTEMP
```

Das Schlüsselwort AS erlaubt Ihnen die Erstellung eines Spaltennamens für eine Spalte in der Liste SELECT zum Zeitpunkt der Ausführung der Anweisung. Wenn Sie für die oben genannte Anweisung keinen neuen Spaltennamen mit dem Schlüsselwort AS angeben, erstellt SQL einen virtuellen Spaltennamen, um auf das

Feld zu verweisen. Indem Sie Ihren eigenen Spaltennamen definieren, wissen Sie genau, wie Sie auf die Spalte in Ihrem Code verweisen können. Zusätzlich erhalten Sie die Fähigkeit, Datenbankänderungen vor Ihren Anwendungen zu verbergen (maskieren).

In diesem Abschnitt wird erklärt, wie die SELECT-Anweisung mit einer oder mehreren Spalten oder Funktionen in der Spaltenliste ausgestattet werden kann, und wie der generierte oder tatsächliche Namen einer Spalte mit dem Schlüsselwort AS überschrieben werden kann. Ausgerüstet mit diesem neuen Konzept können Sie anschließend beginnen, die zurückgegebenen Daten zu steuern.

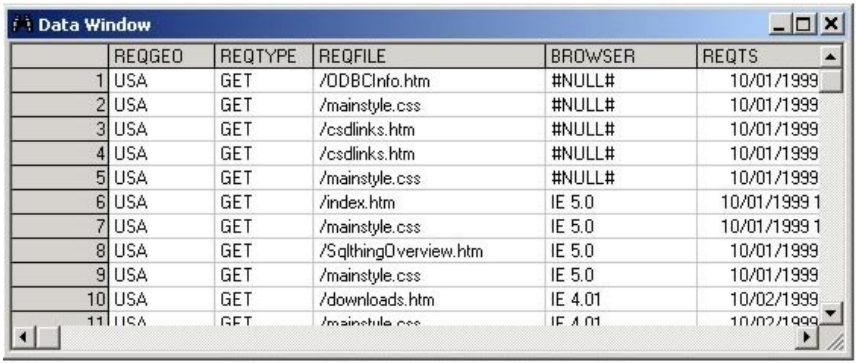
Die WHERE-Klausel

Alle Beispiele haben bisher alle Daten in der Tabelle WEBTEMP zurückgegeben oder eingelesen. Das ist cool, aber wie oft möchten Sie wirklich in einem Programm alle Daten in einer Tabelle einlesen? Häufig ist eine Untergruppe der Informationen wie diese empfehlenswert: „Zeige mir alle Mitarbeiter, die in Florida leben," oder diese: „Zeige mir eine Liste der WEBTEMP-Einträge, die zwischen Januar und Februar auftreten." Sie verwenden die Klausel WHERE der SQL-Anweisung, um die von Ihrer Abfrage zurückgegebenen Informationen zu filtern.

Die WHERE-Klausel folgt in einer SQL SELECT-Anweisung immer auf die FROM-Klausel. Dem Schlüsselwort WHERE folgt immer eine Liste der logischen Ausdrücke, die ein Datensatz erfüllen muss, damit dieser Datensatz zurückgegeben wird. Hier ein einfaches Beispiel, das alle WEBTEMP-Zeilen für die Besuche aus den USA auswählt:

```
SELECT * FROM SQLBOOK.WEBTEMP WHERE
  REQGEO='USA'
```

Dieses Beispiel demonstriert auch die Verwendung des Operators * (Sternchen). Das * zeigt an, dass SQL alle Spalten aller Tabellen zurückgeben soll, auf die in der Anweisung SELECT verwiesen wird. Dies ist eine sinnvolle Abkürzung, wenn Sie alle Spalten zurückgeben möchten. Damit müssen Sie nicht mehr die Spaltennamen eintippen. Die Ausgabe der Abfrage wird in Abbildung 2.4 gezeigt.



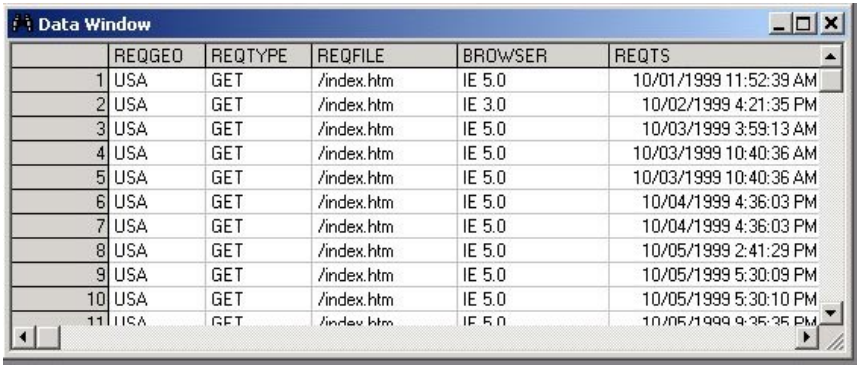
	REQGEO	REQTYPE	REQFILE	BROWSER	REQTS
1	USA	GET	/ODBCInfo.htm	#NULL#	10/01/1999
2	USA	GET	/mainstyle.css	#NULL#	10/01/1999
3	USA	GET	/csdlinks.htm	#NULL#	10/01/1999
4	USA	GET	/csdlinks.htm	#NULL#	10/01/1999
5	USA	GET	/mainstyle.css	#NULL#	10/01/1999
6	USA	GET	/index.htm	IE 5.0	10/01/1999 1
7	USA	GET	/mainstyle.css	IE 5.0	10/01/1999 1
8	USA	GET	/SqlthingOverview.htm	IE 5.0	10/01/1999
9	USA	GET	/mainstyle.css	IE 5.0	10/01/1999
10	USA	GET	/downloads.htm	IE 4.01	10/02/1999
11	USA	GET	/mainstyle.css	IE 4.01	10/02/1999

Abb. 2.4: Diese Ergebnisse sind eine Untergruppe der gesamten WEBTEMP-Tabelle. Durch Verwendung der WHERE-Klausel können Sie z.B. ausschließlich die Besuche aus den USA abrufen.

Die WHERE-Klausel kann eine Liste von Ausdrücken erhalten, die mit den logischen Schlüsselwörtern AND und OR miteinander verknüpft sind. Diese Ausdruckarten werden als Suchbedingungen bezeichnet. Eine Suchbedingung ist ein Ausdruck, der

„Wahr“ oder „Falsch“ zurückgeben muss. Folgende Beispielabfrage verwendet das logische AND, um zwei Suchausdrücke zu verbinden. Die entsprechende Ausgabe wird in Abbildung 2.5 gezeigt.

```
SELECT * FROM SQLBOOK.WEBTEMP WHERE
  REQGEO='USA' AND REQFILE='/index.htm'
```



	REQGEO	REQTYPE	REQFILE	BROWSER	REQTS
1	USA	GET	/index.htm	IE 5.0	10/01/1999 11:52:39 AM
2	USA	GET	/index.htm	IE 3.0	10/02/1999 4:21:35 PM
3	USA	GET	/index.htm	IE 5.0	10/03/1999 3:59:13 AM
4	USA	GET	/index.htm	IE 5.0	10/03/1999 10:40:36 AM
5	USA	GET	/index.htm	IE 5.0	10/03/1999 10:40:36 AM
6	USA	GET	/index.htm	IE 5.0	10/04/1999 4:36:03 PM
7	USA	GET	/index.htm	IE 5.0	10/04/1999 4:36:03 PM
8	USA	GET	/index.htm	IE 5.0	10/05/1999 2:41:29 PM
9	USA	GET	/index.htm	IE 5.0	10/05/1999 5:30:09 PM
10	USA	GET	/index.htm	IE 5.0	10/05/1999 5:30:10 PM
11	USA	GET	/index.htm	IE 5.0	10/05/1999 9:35:35 PM

Abb. 2.5: Hier die Ergebnisse einer Abfrage von Datensätzen, wobei REQGEO='USA' und REQFILE='/index.htm'

Da AND ein logischer Operator ist, müssen beide Ausdrücke, 1 und 2, „Wahr“ sein. Wenn einer der Ausdrücke „Falsch“ ist, wird diese Zeile von SQL nicht zurückgegeben. Damit ein Datensatz von der oben gezeigten Abfrage zurückgegeben werden kann, muss der Datensatz in der Spalte REQGEO den Wert USA aufweisen und in der Spalte REQFILE den Wert /index.htm. Natürlich können Sie auch eine dumme Anweisung wie diese programmieren:

```
SELECT * FROM SQLBOOK.WEBTEMP WHERE
  REQGEO='USA' AND REQGEO='AFRICA'
```

Diese Anweisung kann niemals Zeilen zurückgeben, da die Spalte WEBGEO in einem einzelnen Datensatz nicht sowohl USA als auch AFRICA enthalten kann.

Damit kommen wir zu einem wichtigen Punkt. Bei jedem Debuggen einer Abfrage sollten Sie immer die Logik in Ihrer WHERE-Klausel untersuchen. Denken Sie daran, die WHERE-Klausel ist ein logischer Filter der Datensätze, die Sie erhalten möchten, und logische Filter sind meist der Grund dafür, dass eine Abfrage zu falschen Ergebnissen führt.

Ausdrücke und Prädikate in WHERE-Klauseln

Die WHERE-Klausel erhält eine Liste mit logischen Ausdrükken, aber um welche Ausdrücke kann es sich dabei handeln? Sehen wir uns eine Anzahl von Beispielen an, die Ihnen die Grundlagen von Ausdrücke vermitteln sollen.

Ausdrücken liegen einfache mathematische Operationen zugrunde. Die meisten Einfügeoperatoren sollten Ihnen bekannt sein. Diese Operatoren, wie z.B. +, -, *, **, und /, repräsentieren Addition, Subtraktion, Multiplikation, Energie und Division. Diese Einfügeoperatoren können für numerische Datentypen eingesetzt werden, um ein unterschiedliches Ergebnis zu erzielen, das anschließend mit Hilfe derVergleichsprädikate verglichen werden kann.

Sie sollten sich auch dessen bewusst sein, dass mathematische Ausdrücke den Datentyp des Ergebnisses verändern können. Wenn Sie z.B. den Präfixoperator hinzufügen – vor einer Spalte SMALLINT, um den Wert in einen negativen Wert umzuwandeln, wird die Spalte zu einem INTEGER-Wert umgewandelt. Wenn Sie eine INTEGER-Spalte mit einer DECIMAL-Spalte multiplizieren, wird die Ganzzahl in eine Dezimalzahl umgewandelt und erzielt ein Dezimalergebnis. Wenn Sie den Operator ** verwenden, erhalten Sie eine Gleitkommazahl. Die meisten dieser Dinge machen zwar keine Furore, es ist aber wichtig, dass Sie diese Datentypen kennen und wissen, dass sie mit Ausdrücken geändert werden können. Das ist besonders wichtig, wenn Sie sich mit Optimierungsproblemen befassen, da Vergleiche zwischen den Datentypen unterschiedlicher Präzision mehr Prozessorzeit benötigen (da sie temporär zum Vergleich in den gleichen Datentyp umgewandelt werden müssen).

Vergleichsoperatoren, wie z.B. >, < , und = können verwendet werden, um einen logischen Wert zu ergeben. Beachten Sie, dass diese Prädikate nicht nur für numerische Datentypen vorgesehen sind, sie können auch verwendet werden, um beliebige SQL-Datentypen zu vergleichen, solange die Werte auf der linken und rechten Seite des Prädikatoperators typkompatibel sind (d. h., Strings werden mit Strings verglichen, Zahlen mit Zahlen etc.). Selbst wenn sie nicht typ-kompatibel sind, ist es manchmal möglich, einen Wert in einen kompatiblen Datentyp umzuwandeln. Tabelle 2.2 listet einen Vergleich von Prädikaten auf, die logische Werte ergeben.

Prädikat	**Beschreibung**
X<Y	Wahr wenn X kleiner als Y.
X>Y	Wahr wenn X größer als Y.
X<=Y	Wahr wenn X gleich oder kleiner als Y.
X>=Y	Wahr wenn X größer als oder gleich Y.
X<>Y	Wahr wenn X ungleich Y.
X=Y	Wahr wenn X gleich Y.
X¬<Y	Wahr wenn X nicht kleiner als Y. Kann auch als X!<Y ausgedrückt werden.
X¬>Y	Wahr wenn X nicht größer als Y. Kann auch als X!>Y ausgedrückt werden.
X BETWEEN Y AND Z	Wahr wenn X größer oder gleich Y und kleiner als oder gleich Z.
X NOT BETWEEN Y AND Z	Wahr wenn X nicht größer als oder gleich Y und kleiner als oder gleich Z.
EXISTS (SELECT statement)	Wahr wenn die SELECT-Anweisung Zeilen ausgibt.
NOT EXISTS (SELECT statement)	Wahr wenn die SELECT-Anweisung keine Zeilen ausgibt.
X LIKE Y	Wahr wenn X ähnlich wie Y. X und Y müssen Strings sein. Y kann Wildcards enthalten.
X NOT LIKE Y	Wahr wenn X ungleich Y. X und Y müssen Strings sein. Y kann Wildcards enthalten.

Prädikat	Beschreibung
X IN Y	Wahr wenn X in der Liste Y enthalten ist. Y kann eine parameterisierte Liste von Konstanten oder eine Liste von Konstanten oder eine SELECT-Anweisung sein, die eine einzelne Spalte zurückgibt.
X NOT IN Y	Wahr wenn X nicht in Liste Y enthalten ist. Y kann eine parameterisierte Liste von Konstanten oder eine SELECT-Anweisung sein, die eine einzelne Spalte zurückgibt.
X IS NULL	Wahr nur, wenn X der Nullwert ist.
X IS NOT NULL	Wahr nur, wenn X NICHT der Nullwert ist.

Tab. 2.2: SQL-Prädikate

Temporale Ausdrücke

Neben den numerischen Operationen können Sie temporale Ausdrücke verwenden. Bei temporalen Ausdrücken handelt es sich um Mathematik, die auf Datum-, Zeit- und Zeitstempel-Spalten angewendet wird. Temporale Operatoren sind HOURS, MINUTES, SECONDS, MICROSECONDS, DAYS, MONTHS, und YEARS. Sie können mit diesen temporalen Schlüsselwörtern nur Additionen und Substraktionen ausführen. Führen wir also einige Beispielabfragen mit einfachen mathematischen und temporalen Ausdrücken aus.

Sehen Sie sich diese Abfrage an, die alle WEBTEMP-Besuche auflistet, die mehr als 100 Tage alt sind:

```
SELECT * FROM SQLBOOK.WEBTEMP WHERE REQTS
  < CURRENT_TIMESTAMP -100 DAYS
```

Diese Abfrage betont zwei Dinge: Manipulation der Zeitstempelwerte und die Funktion CURRENT_TIMESTAMP. CURRENT_TIMESTAMP gibt das aktuelle Datum und Zeitpunkt auf der AS/400 zurück. Wenn Sie 100 Tage vom aktuellen Zeitstempel abziehen, gibt die AS/400 alle Datensätze zurück, die mehr als 100 Tage alt sind.

Die AS/400 verfügt über integrierte Funktionen für die Rückgabe der Stunde, Minute und Sekunde der Zeitwerte. Folgende Abfrage ruft Webbesuche ab, die auf Besuche zwischen 10.00 Uhr und 13.00 Uhr zurückzuführen sind. Die Abfrage basiert auf der Skalarfunktion HOUR, die eine Zahl zwischen 0 und 23 für die genaue Tageszeit zurückgibt:

```
SELECT * FROM SQLBOOK.WEBTEMP WHERE
  HOUR(REQTS)>= 10 AND HOUR(REQTS)<13
```

Um dieses Beispiel noch weiter auszubauen, sehen wir uns die Webbesuche an, die auf Besuche an Montagen und Mittwochen zwischen 10.00 und 1.00 Uhr zurückzuführen sind:

```
SELECT * FROM SQLBOOK.WEBTEMP
  WHERE HOUR(REQTS)>= 10 AND
  HOUR(REQTS)<13 AND DAYOFWEEK(REQTS) = 2
  OR DAYOFWEEK(REQTS) = 4
```

Diese Abfrage nutzt den Vorteil der Skalarfunktion von DAYOFWEEK, die eine Zahl zwischen 1 und 7 zurückgibt, wobei 1 Sonntag repräsentiert und 7 Samstag repräsentiert. Beachten Sie, dass der OR-Operator verwendet wird, so dass eine Un-

terscheidung zwischen DAYOFWEEK 2 und DAYOFWEEK 4 möglich wird. Wenn Sie den AND-Operator verwenden, werden keine Datensätze zurückgegeben, da ein einzelner Datensatz kein Datum aufweisen kann, das sowohl Montag als auch Mittwoch reflektiert.

Wenn Sie OR verwenden, sollten Sie sich überlegen, ob Sie nicht Klammern einsetzen sollten, um Ihre Logik besser darstellen zu können. Die Abfrage wäre besser lesbar und leichter verständlich, wenn sie so gegliedert wäre:

```
SELECT * FROM SQLBOOK.WEBTEMP
  WHERE HOUR(REQTS)>= 10 AND
  HOUR(REQTS)<13 AND (DAYOFWEEK(REQTS) = 2
  OR DAYOFWEEK(REQTS) = 4)
```

Die Klammern erlauben Ihnen die Gruppierung von logischen Elementen, und wie bereits festgestellt wurde, stammen die meisten Probleme mit SQL-Anweisungsergebnissen von Logikfehlern.

Eine dritte Methode schließlich für diese Abfrage besteht darin, den IN-Operator zu verwenden, der eine Liste von Werten erhält und den Wert links vom IN-Operator mit der Liste der Werte rechts vergleicht. Hier ist die gleiche Abfrage, die IN für beide Ausdrücke verwendet:

```
SELECT * FROM SQLBOOK.WEBTEMP
  WHERE HOUR(REQTS) IN (10,11,12)
  AND DAYOFWEEK(REQTS) IN (2,4)
```

Wie Sie sehen, geben alle diese Abfragen die gleichen Ergebnisse zurück. In SQL gibt es immer mehrere Methoden, um die gleiche Abfrage zu stellen, und Sie sollten stets die eleganteste und lesbarste Form verwenden.

Ein weiterer nützlicher Operator ist BETWEEN, der Ihnen erlaubt, zwei Werte für ein Feldelement anzugeben und inklusive ist. Hier ist die gleiche Abfrage, die für die Rückgabe der Ergebnisse unter Verwendung von BETWEEN modifiziert wurde:

```
SELECT * FROM SQLBOOK.WEBTEMP
  WHERE HOUR(REQTS) BETWEEN 10 AND 12
  AND DAYOFWEEK(REQTS) in (2,4)
```

BETWEEN ist zwar recht nett, kann aber die SQL-Syntax gelegentlich verwirrend gestalten. Ich bin dafür, die Syntax in einfachen Abfragen zu verwenden, aber in sehr komplexen WHERE-Klauseln darauf zu verzichten.

Verwendung des Prädikats LIKE für das Durchsuchen von Strings

SQL besitzt einen leistungsfähigen Operator für das Durchsuchen von zeichenbasierten Spalten, das LIKE-Prädikat. Wie mit IN erfordert LIKE einen Wert auf der linken und einen Wert auf der rechten Seite. Der Wert auf der linken Seite ist normalerweise eine Spalte, während der Wert auf der rechten Seite ein String-Ausdruck ist. LIKE hat auch spezielle Operatoren, die auf der rechten Seite angezeigt werden. Diese Operatoren sind die Symbole % und ?. Das %-Zeichen ist ein Platzhalterzeichen (Wildcard), das ein oder mehrere Zeichen repräsentieren kann. Das ? ist ein Platzhalterzeichen, mit dem ein einzelnes Zeichen ersetzt wird.

Hier ist eine Abfrage, die das LIKE-Prädikat und dessen verknüpfte Wildcards verwendet:

```
SELECT * FROM SQLBOOK.WEBHITS WHERE
  REQFILE LIKE ,/I%‘
```

Diese Abfrage gibt alle Datensätze zurück, bei denen die REQFILE-Spalte mit den Zeichen /I beginnt. Diese Abfrage würde Werten wie */Indexes.htm*, /I oder */Ivan.HTM* entsprechen. Das %-Zeichen sagt grundsätzlich aus, dass es keine Rolle spielt, was nach dem initialen String steht.

Was geschieht, wenn Sie innerhalb einer zeichenbasierten Spalte nach einem Substring suchen? Folgende Abfrage findet alle Datensätze, bei denen die Buchstaben *jpg* in der Spalte REQFILE auftreten:

```
SELECT * FROM SQLBOOK.WEBHITS WHERE
  REQFILE LIKE ,%jpg%‘
```

Achten Sie auf die Verwendung der Wildcards. Positionieren Sie den Platzhalter vor und nach der Zeichenkette, um anzugeben, dass Sie alle Datensätze mit *jpg* erhalten möchten, gleichgültig, ob die Zeichenkette mit *jpg* beginnt, ob sie mit *jpg* endet oder ob irgendwo in der Mitte *jpg* steht.

Das ?-Zeichen wird als Wildcard für ein einzelnes Zeichen verwendet. Wenn Sie z.B. alle Datensätze finden möchten, bei denen das dritte Zeichen der Buchstabe *n* ist, finden Sie diese mit folgender Abfrage:

```
SELECT * FROM SQLBOOK.WEBHITS WHERE
  REQFILE LIKE ,??n%‘
```

Dieser Typ von Musterabgleich kann bei Legacy-Feldern nützlich sein, die für die Darstellung von mehreren Daten verwendet werden. Nehmen wir z.B. an, dass Sie eine Arbeitsauftrag-Anwendung mit einem 10-Zeichen-String-Feld mit der Bezeichnung LOCATION erstellt haben. Bei der Eingabe von Arbeit-

saufträgen in die Datenbank gibt der Operator eine zweistellige Gebäudenummer ein, die zweistellige Etagennummer und die Zimmernummer, getrennt mit Bindestrichen. Wenn Sie die Musterzeichenkette ‚??-13-%' verwenden, werden alle Arbeiten aufgelistet, die in der 13. Etage eines beliebigen Gebäudes ausgeführt wurden.

Wildcard-Zeichen sind nützlich, aber sie können bei großen Tabellen zu stark beeinträchtigter Abfrageleistung führen. SQL kann für die Beschleunigung keinen Index verwenden, wenn Sie nicht die ersten Zeichen in der Subzeichenkette abgleichen. Wenn Sie ‚%ILE%' als Ziel der LIKE-Klausel verwenden, wird kein Index verwendet, ‚ILE%' wird einen Index verwenden, wenn dieser zur Verfügung steht. In großen Tabellen muss der Einsatz von LIKE sorgfältig geprüft werden.

Nun sehen wir uns an, wie LIKE effizient verwendet werden kann. Nehmen wir an, dass Sie eine Anwendung erstellt haben, mit der das Datenerfassungspersonal Werbeanzeigen in den Computer eingibt. Die Inserententabelle besteht aus einem Integer-Primärschlüssel und einem 50-Zeichen-Namensfeld für den Inserenten. Ein Index mit der Bezeichnung ADVERNAME wurde über dem Feld mit dem Inserenten erstellt. Diese Tabelle umfasst über 1 Million Zeilen mit Informationen. Folgender Abfragetyp kann täglich mehr als 100.000 Mal ausgeführt werden und pro Sekunde mehr als 500 Datensätze zurückgeben:

```
SELECT ADVERNO, ADVERNAME FROM ADVERTISER
  WHERE ADVERNAME LIKE ‚ABC%'
```

Diese Abfrage ermittelt alle Inserenten, deren Namen mit ABC beginnen. Indem Sie den Typ „beginnt mit" von LIKE verwenden, wird der Index ADVERNAME für die Auflösung der Abfrage verwendet. Dem gegenüber benötigen die Ergebnisse mehr

als 15 Sekunden für die Anzeige, wenn ein Operator die folgende Abfrage einsetzt, da SQL eine vollständige Tabellenprüfung durchführen muss:

```
SELECT ADVERNO, ADVERNAME FROM ADVERTISER
  WHERE ADVERNAME LIKE ,%ABC%‘
```

Wenn Sie verstehen, wie die AS/400 Ihre Abfrage implementiert, fällt es Ihnen leichter, Abfragen zu erstellen, die nicht Ihre gesamten Systemressourcen benötigen. Dieses Thema wird ausführlich in Kapitel 6 behandelt, das sich mit dem Debugging und der Optimierung Ihrer Abfragen befasst.

Schließlich sollten Sie auch wissen, dass auf beiden Seiten einer LIKE-Abfrage ein Ausdruck vorhanden sein muss. Sehen Sie sich dieses Beispiel an:

```
SELECT * FROM SQLBOOK.WEBTEMP WHERE
  UCASE(REQFILE) LIKE ,%.JPG‘
```

Diese Abfrage verwendet UCASE, um den Inhalt der Spalte REQFILE in Großbuchstaben umzuwandeln, bevor dieser mit dem Ausdruck LIKE verglichen wird. So finden Sie alle Vorkommnisse von Dateien mit der Erweiterung .JPG.

Ergebnisse von SELECT-Anweisungen anordnen

Jetzt wissen Sie, wie Daten aus SQL-Tabellen mit Hilfe der SELECT-Anweisung abgerufen werden können. Die Reihenfolge der abgerufenen Daten wird dabei jedoch nicht beachtet. Es gibt keine Garantie, dass die Daten aus einer SQL-Anweisung in einer bestimmten festen Reihenfolge aus einer Tabelle abgerufen werden. Dies kann sehr verwirrend sein. Einige Menschen nehmen an, dass die Daten einer Tabelle, die mit einem Schlüssel definiert wird, genau in dieser Reihenfolge abgerufen werden. Das ist aber nicht der Fall. Sehen Sie sich folgende Anweisung an:

```
SELECT PARTID FROM SQLBOOK.PARTS
```

PARTID ist der Primärschlüssel der Tabelle PARTS. Ein Primärschlüssel stellt sicher, dass sich über der Tabelle ein Index befindet, mit dem sichergestellt wird, dass keine zwei PARTID-Werte gleich sind. Da dieser Index erforderlich ist, kann angenommen werden, dass die obige Anweisung den Index verwendet, um die PARTID-Werte einzulesen, und daher die Daten in geordneter Reihenfolge zurückzugeben. Das ist jedoch nicht der Fall. Die einzige Methode, eine Reihenfolge sicherzustellen, ist die Spezifizierung der Reihenfolge mit Hilfe der Klausel ORDER BY der SQL-Anweisung. Nachfolgende Anweisung gibt eine Liste aller PARTID-Codes in der jeweiligen Reihenfolge aus:

```
SELECT PARTID FROM SQLBOOK.PARTS
  ORDER BY PARTID
```

Die Klausel ORDER BY kann eine Liste von Feldern oder numerischen Feldkennzeichnern erhalten, die die Sortierreihenfolge der Ergebnisse sicherstellen. Hier ist ein weiteres Beispiel:

```
SELECT REQTS, REQPAGE FROM SQLBOOK.WEBTEMP
  ORDER BY 2
```

Beachten Sie die ORDER BY 2-Klausel. Damit wird angezeigt, dass SQL die Ergebnisse nach dem zweiten Feld in der Auswahlliste sortieren sollte. Das zweite Feld ist REQPAGE, daher werden die Ergebnisse nach dem angeforderten Seitendateinamen sortiert. Diese Technik ist praktisch, wenn Sie in Ihrer Liste Ausdrücke verwenden. Sehen Sie sich dieses Beispiel an:

```
SELECT YEAR(REQTS),MONTH(REQTS),REQFILE
  FROM SQLBOOK.WEBTEMP ORDER BY 1,2
```

Mit dieser Anweisung wird das Jahr des Webbesuches, der Monat des Webbesuches und die Zielseite ausgewählt. Hier zeigt die Klausel ORDER BY an, dass die Daten nach den Spalten 1 und 2 sortiert werden sollten (entsprechend den Skalarspalten Jahr und Monat). Die Anweisung, die davor steht, kann auch so geschrieben werden:

```
SELECT YEAR(REQTS) as YR, MONTH(REQTS) AS
  MO, REQFILE
  FROM SQLBOOK.WEBTEMP ORDER BY YR, MO
```

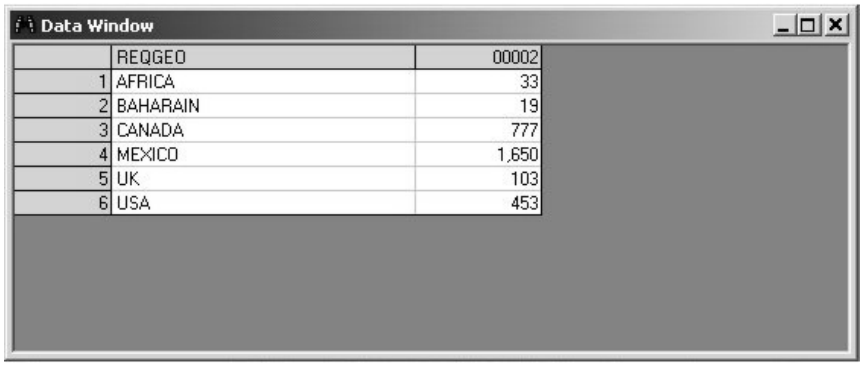
Dieses Beispiel zeigt, dass die Klausel ORDER BY auch Feldaliase beinhalten kann (durch den AS-Operator erstellt).

Gruppieren von Daten

Was bei SQL leicht vergessen wird ist die Tatsache, dass es sich um eine mengenorientierte Sprache, nicht um eine auf einer physischen Datei basierten Sprache handelt. In einer auf Datensatzebenenzugriff basierten Programmiersprache müssen Sie, wenn Sie einen Bericht von Webbesuchen nach Land erstellen möchten, ein Array für die Aufnahme der Daten deklarieren und jeden Datensatz in die Datei einlesen. Der Ländercode des Datensatzes wird verwendet, um festzulegen, welchen Offset Ihres Arrays Sie für die Inkrementierung benötigen. SQL als eine mengenorientierte Sprache macht Operationen dieser Art viel einfacher, weil es die Gruppierung über die Funktion GROUP BY unterstützt. Die Funktion GROUP BY illustriert die mengenorientierten Funktionen von SQL und zeigt, wie SQL ein wirklich mächtiger Verbündeter für Berichtssysteme sein kann.

Das Schlüsselwort GROUP BY folgt immer auf die WHERE-Klausel in einer SQL-Anweisung, aber die Anweisung muss keine WHERE-Klausel besitzen, um GROUP BY verwenden zu können. GROUP BY wird normalerweise in Verbindung mit den COLUMN-Gruppierfunktionen verwendet. Hier ist eine Beispiel-SQL-Anweisung, die die Anzahl der Besuche abhängig von der Geographie zurückgibt. Die Ausgabe der Anweisung wird in Abbildung 2.6 gezeigt:

```
SELECT REQGEO, COUNT(*)
  FROM SQLBOOK.WEBTEMP
  GROUP BY REQGEO
```



	REQGEO	00002
1	AFRICA	33
2	BAHARAIN	19
3	CANADA	777
4	MEXICO	1,650
5	UK	103
6	USA	453

Abb. 2.6: Dies ist die Ausgabe, die von der Anweisung GROUP BY zuückgegeben wird. Beachten Sie, dass SQL die Besuche entsprechend den verschiedenen Werten des Felds REQGEO aggregiert.

Was hier geschieht, ist recht interessant. Zuerst liest SQL alle REQGEO-Feldwerte aus der Tabelle WEBTEMP. Anschließend wird jeder einzelne Wert gezählt und die Ausgabe erzeugt, die in Abbildung 2.6 gezeigt wird. Diese Funktionalität ist sehr leistungsfähig, da Sie einen Server die ganze Arbeit des Addierens der numerischen Spalten erledigen lassen können und es nicht erforderlich ist, komplexe Kodierarbeiten auf Clientseite auszuführen.

Wenn Sie eine GROUP BY-Anweisung verwenden, müssen Sie stets auch alle Spalten angeben, die in Ihrer SQL-Anweisung nicht unter die GROUP BY-Klausel fallen. Anderenfalls wird die Anweisung fehlschlagen. Hier ist ein Beispiel für eine schlechte Gruppierung:

```
SELECT REQGEO, BROWSER, COUNT(*)
  FROM SQLBOOK.WEBTEMP
  GROUP BY REQGEO
```

Da die Spalte BROWSER nicht in die Gruppierfunktion aufgenommen wurde, wird diese Anweisung fehlschlagen. Hier die korrekte Syntax für die Anweisung:

```
SELECT REQGEO, BROWSER, COUNT(*)
  FROM SQLBOOK.WEBTEMP
  GROUP BY REQGEO, BROWSER
```

Die Ausgabe für die Anweisung wird in Abbildung 2.7 gezeigt. Beachten Sie, dass diese Ausgabe nicht sortiert wird. Auch hier wird SQL keine sortierten Ergebnisse zurückgeben, wenn Sie nicht spezifisch danach fragen.

Data Window

	REQGEO	BROWSER	00003
1	AFRICA	IE 5.0	33
2	BAHARAIN	IE 3.02	2
3	BAHARAIN	IE 5.0	17
4	CANADA	IE 3.02	9
5	CANADA	IE 4.0	2
6	CANADA	IE 4.01	76
7	CANADA	IE 5.0	670
8	CANADA	IE 5.01	15
9	CANADA	#NULL#	5
10	MEXICO	IE 3.0	98
11	MEXICO	IE 3.02	49
12	MEXICO	IE 4.0	15

Abb. 2.7: Sie können in SQL nach mehr als einer Spalte gruppieren. Diese Ausgabe repräsentiert Besuche aus Ländern, die durch den Client-Browser aggregiert wurden.

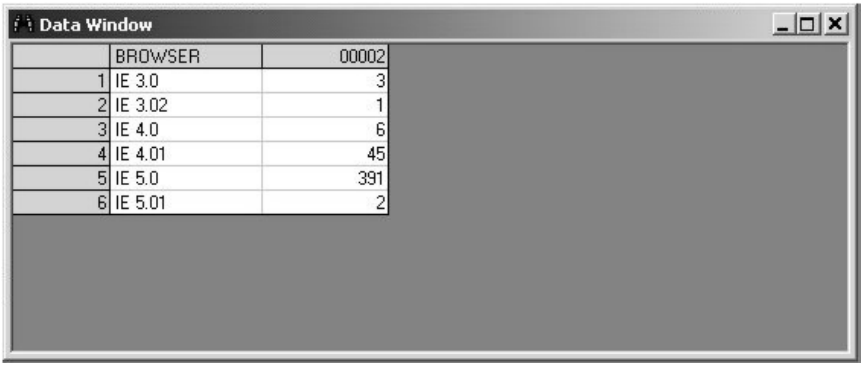
Wenn eine Anweisung GROUP BY verwendet, muss die Klausel ORDER BY nach der Klausel GROUP BY stehen. Hier ist die gleiche Anweisung, jedoch mit sortierter Ausgabe:

```
SELECT REQGEO, BROWSER, COUNT(*)
  FROM SQLBOOK.WEBTEMP
  GROUP BY REQGEO, BROWSER
  ORDER BY 1,2
```

Eine GROUP BY-Anweisung kann eine WHERE-Klausel enthalten, die die Daten beschränkt, die in die Anweisung aufgenommen werden. Die WHERE-Klausel wird vor der Aggregation verarbeitet. Hier ein Beispiel, das die Besuche aus den USA mit dem Browser Internet Explorer auflistet:

```
SELECT BROWSER, COUNT(*) FROM
  SQLBOOK.WEBTEMP
    WHERE BROWSER LIKE ‚IE%‘ AND
    REQGEO = ‚USA‘ GROUP BY
    REQGEO, BROWSER ORDER BY 1
```

Die BROWSER-Spalte beginnt mit IE und enthält alle Besuche mit Internet Explorer. Abbildung 2.8 zeigt die Ausgabe dieser Anweisung.



	BROWSER	00002
1	IE 3.0	3
2	IE 3.02	1
3	IE 4.0	6
4	IE 4.01	45
5	IE 5.0	391
6	IE 5.01	2

Abb. 2.8: Diese Ausgabe zeigt nur die USA-Besuche, die von IE-Browsern stammen.

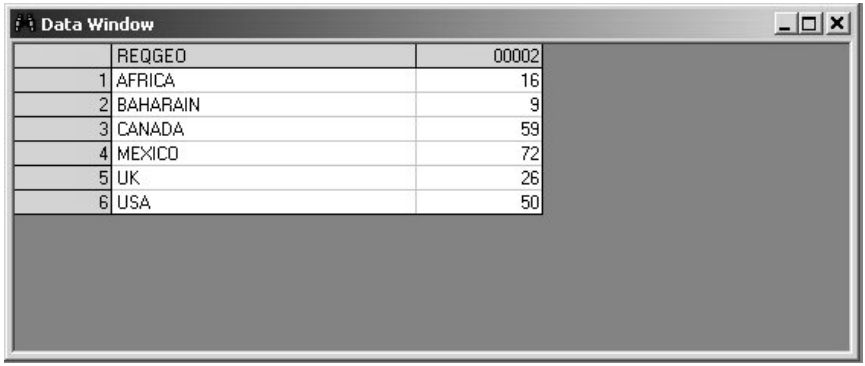
Wenn Sie GROUP BY verwenden, müssen Sie nicht nur einzelne Aggregatfelder wie in den obigen Beispielen abrufen. Die folgenden Funktionen sind in den GROUP BY-Anweisungen vorhanden:

COUNT(x)

Die Funktion COUNT gibt eine Summe des Felds oder des Ausdrucks zurück. COUNT verwendet optional das Argument DISTINCT. DISTINCT sorgt dafür, dass COUNT nur Werte zählt, die sich voneinander unterscheiden. Hier ist ein Beispiel:

```
SELECT REQGEO, COUNT(DISTINCT REQFILE)
  FROM SQLBOOK.WEBTEMP
  GROUP BY REQGEO
```

Diese Anweisung, deren Ausgabe in Abbildung 2.9 gezeigt wird, unterscheidet sich von der Ausgabe, die Sie in Abbildung 2.6 sehen. Diese Ausgabe reflektiert die Anzahl der unterschiedlichen Seiten, die pro Land betrachtet werden (während Abbildung 2.6 die Gesamtzahl der Besuche nach Land zeigt). Wenn das Schlüsselwort DISTINCT angegeben wird, wird jede Seite nur einmal gezählt, selbst wenn die Seite Millionen mal vorkommt.



	REQGEO	00002
1	AFRICA	16
2	BAHARAIN	9
3	CANADA	59
4	MEXICO	72
5	UK	26
6	USA	50

Abb. 2.9: Diese Ausgabe wurde mit COUNT(DISTINCT REQFILE) generiert, im Gegensatz zu Abbildung 2.6, in der alle Seiten gezählt werden.

MIN(x)

MIN berechnet den Mindestwert von X, bei dem es sich um eine Spalte eines beliebigen Datentyps oder eines Ausdrucks handelt. Hier ist ein Beispiel für eine Abfrage mit MIN:

```
SELECT MIN(REQTS) FROM SQLBOOK.WEBTEMP
```

Diese Anweisung gibt den niedrigeren Datums-/Zeitwert zurück, der in der Tabelle WEBTEMP angegeben wird.

MAX(x)

MAX findet den maximalen Wert von X, der eine Spalte eines beliebigen Datentyps oder ein Ausdruck sein kann. Hier ein Beispiel für eine Abfrage, die MAX verwendet:

```
SELECT MAX(REQTS) FROM SQLBOOK.WEBTEMP
```

Diese Anweisung gibt den höheren Wert für Datum/Zeit zurück, der in der Tabelle WEBTEMP vorkommt.

SUM(x)

SUM addiert alle numerischen Felder oder Ausdrücke; dies kann extrem nützlich sein. Hier ist zum Beispiel eine Abfrage, die die Anzahl der Kilobyte nach Geographie auffindet, die ein Webserver während des Monats Oktober an Clients gesendet hat:

```
SELECT REQGEO,SUM(REQSIZE) FROM
  SQLBOOK.WEBTEMP
  WHERE MONTH(REQTS)=10
  GROUP BY REQGEO;
```

AVG(x)

AVG berechnet den durchschnittlichen Wert für einen beliebigen numerischen Ausdruck. Folgende SELECT-Anweisung ergibt die durchschnittliche Anzahl an Kilobyte je Geographie:

```
SELECT REQGEO,AVG(REQSIZE) FROM
  SQLBOOK.WEBTEMP
  GROUP BY REQGEO
```

VAR(x) oder VARIANCE

VAR berechnet die Varianz in einer Reihe von Zahlen, wobei X eine numerische Spalte oder ein numerischer Ausdruck sein kann.

STDDEV(x)

STDDEV berechnet die Standardabweichung bei einer Reihe von Zahlen, wobei X eine Spalte oder ein numerischer Ausdruck sein kann.

Alles wieder zusammenfügen

Sie können in eine GROUP BY-Abfrage mehr als eine Gruppierungsfunktion aufnehmen. Hier ist eine Beispielabfrage, die einige Funktionen verwendet:

```
SELECT REQGEO, MIN(REQSIZE) as MINIMUM,
  MAX(REQSIZE) AS MAXIMUM, AVG(REQSIZE) AS
  AVERAGE
  FROM SQLBOOK.WEBTEMP
  GROUP BY REQGEO
```

Die Ausgabe dieser Abfrage wird in Abbildung 2.10 gezeigt. Beachten Sie die Verwendung des Schlüsselworts AS in dieser Abfrage bei der Umbenennung der zurückgegebenen Spalten. Wegen AS verfügen die Spalten in Abbildung 2.10 über von Menschen lesbare Namen anstelle von automatisch generierten Funktionsnummern.

Data Window

	REQGEO	MINIMUM	MAXIMUM	AVERAGE
1	AFRICA	1	8,564	704
2	BAHARAIN	1	162	32
3	CANADA	1	8,564	163
4	MEXICO	1	8,564	409
5	UK	1	4,501	69
6	USA	1	8,564	101

Abb. 2.10: Dieses Beispiel zeigt, dass Sie in einer GROUP BY-Abfrage mehr als eine Aggregatfunktion verwenden.

Gruppierung nach Skalarfunktionen

Eine Begrenzung der AS/400 ist die Tatsache, dass sie in Versionen vor V4R3 nicht nach Skalarfunktionen oder abgeleiteten Spalten gruppieren kann. Dies ist in anderen Versionen der DB2 UDB-Datenbank nicht der Fall. Diese Begrenzung ist ärgerlich, und ich habe IBM bereits mehrere Male gebeten, sich darum zu kümmern. Sehen Sie sich die folgende Abfrage an:

```
SELECT MONTH(REQTS),REQGEO,COUNT(*)
  ,SUM(REQSIZE) FROM SQLBOOK.WEBTEMP
  GROUP BY MONTH(REQTS),REQGEO
```

Auf AS/400-Systemen vor V4R3 würde diese Abfrage fehlschlagen oder fehlerhafte Ergebnisse zurückgeben. Um diese Begrenzung zu überwinden müssen Sie eine View erstellen und anschließend aus der View für die Gruppierung auswählen. Hier ist die Lösung:

```
CREATE VIEWSQLBOOK.WEBTEST (M,G,C,S) AS
  SELECT MONTH(REQTS),REQGEO, 1, REQSIZE
  FROM SQLBOOK.WEBTEMP
```

Diese Anweisung erstellt eine View mit der Bezeichnung WEBTEST in der Bibliothek SQLBOOK. Die View WEBTEST hat vier Felder mit der Bezeichnung M, G, C und S. Die Felder enthalten den Monat der Seitenanforderung, die Geographie der Anforderung, die Nummer 1, um einen Seitenaufruf anzuzeigen und die Größe des zurückgegebenen Werts. Sehen wir uns nun folgende Abfrage an:

```
SELECT M,G,SUM(C),SUM(S) FROM
  SQLBOOK.WEBTEST
  GROUP BY M,G
```

Diese Abfrage entspricht semantisch der letzten Abfrage, jedoch ohne die genannte Einschränkung, nach skalaren Funktionen nicht ohne Weiteres gruppieren zu können. Wenn Sie Spalte C summieren, hat die Spalte, die immer den Wert 1 enthält, den gleichen Effekt wie die Auswahl von COUNT(*).

Da die Fähigkeit, nach Skalarfunktionen zu gruppieren so neu ist, schlage ich vor, dass Sie einige Zeit lang die Fehlerumgehungsmethode für die View einsetzen. Auch können einige Ausdrücke in Ihrer SELECT-Klausel so komplex sein, dass Ihre GROUP BY-Anweisungen nicht mehr lesbar sind. Die View-Fehlerumgehung kann Ihre Abfragegrammatik vereinfachen und Ihr SQL weniger einschüchternd wirken lassen.

Mit HAVING Gruppen filtern

Die WHERE-Klausel erlaubt Ihnen das Filtern von Daten, bevor die Daten mit der GROUP BY aggregiert werden. HAVING erlaubt Ihnen das Filtern von Daten, die an den Client zurückgegeben werden, nachdem die Daten aggregiert wurden. Sehen Sie sich eine Anforderung an, mit der eine Liste von Webseiten erstellt wird, auf die zwischen 100 und 200 Mal zugegriffen wird. Folgende Abfrage erreicht dieses Ziel, wie in der Ausgabe in Abbildung 2.11 gezeigt:

```
SELECT REQFILE, COUNT(*) FROM SQLBOOK.WEBTEMP
  GROUP BY REQFILE
  HAVING COUNT(*) BETWEEN 100 and 200
```

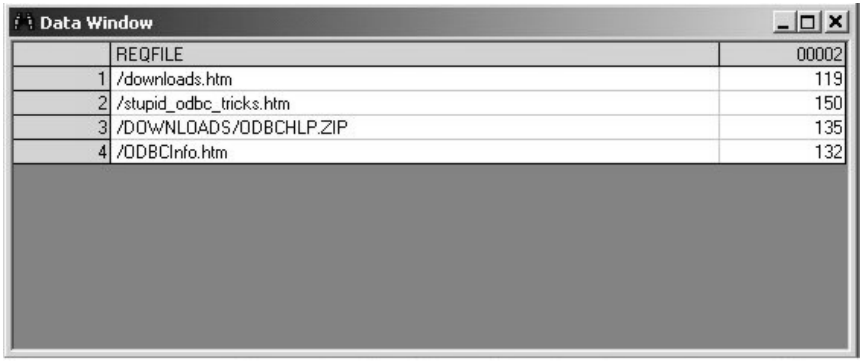


Abb. 2.11: Die Ergebnisse der HAVING-Abfrage. Beachten Sie, dass nur Seiten, die zwischen 100 und 200 Besuchen erhalten, gezeigt werden.

Es ist wichtig, zu verstehen, dass die HAVING-Klausel nicht ausgeführt wird, nachdem die Daten aggregiert werden. Hier sehen Sie, was mit der obenstehenden Abfrage geschieht:

1. SQL ruft alle Datensätze ab und sortiert sie nach REQFILE.
2. SQL stellt die Ergebnismenge zusammen, die die Spalte REQFILE enthält, sowie die damit verknüpften Counts durch Iteration durch die sortierten Daten.
3. SQL entfernt alle Datensätze aus den Ergebnismengen, die der HAVING-Klausel nicht entsprechen.

Nun sehen Sie sich folgende Abfrage an, die sowohl HAVING als auch WHERE enthalten:

```
SELECT REQGEO, REQFILE, COUNT(*)
  FROM SQLBOOK.WEBTEMP
```

```
WHERE BROWSER LIKE ‚IE%‘ AND
DAYOFWEEK(REQTS)=1 GROUP BY
REQGEO, REQFILE HAVING
COUNT(*)<20 ORDER BY 3 DESC
```

Das Ergebnis dieser Abfrage wird in Abbildung 2.12 gezeigt. Die AS/400 ruft zuerst die Ergebnismenge aus allen Datensätzen ab, die den WHERE-Kriterien entsprechen. In diesem Fall wäre das der Geographie- und Dateinamen der Dateien, die an Montagen abgerufen werden (DAYOFWEEK(REQTS)=1), und die nur von Internet-Explorer-Browsern stammen (BROWSER LIKE ‚IE%‘). Sobald alle Datensätze in der temporären Tabelle enthalten sind, aggregiert SQL die Gruppe und entfernt anschließend Datensätze mit Seitenaufrufen größer als oder gleich 20 Anfragen (HAVING COUNT(*)<20). Schließlich sortiert SQL die Ergebnismenge nach der dritten Spalte in den Ergebnisdaten in absteigender Reihenfolge (in diesem Fall die Anzahl der Seitenanforderungen, die durch ORDER BY 3 DESC angegeben wird).

Data Window

	REQGEO	REQFILE	00003
1	MEXICO	/mainstyle.css	13
2	MEXICO	/stupid_odbc_tricks.htm	6
3	USA	/mainstyle.css	6
4	MEXICO	/csdlinks.htm	5
5	MEXICO	/ODBCInfo.htm	5
6	MEXICO	/SQLThing.htm	5
7	MEXICO	/DOWNLOADS/SQLThingProductDocumentation.PDF	5
8	UK	/mainstyle.css	4
9	MEXICO	/favicon.ico	4
10	UK	/csdlinks.htm	4
11	CANADA	/ODBCFreeReg.htm	4
12	MEXICO	/ODBCASPdynamic_query.htm	3

Abb. 2.12: Dieses Beispiel verwendet WHERE, HAVING und ORDER BY, um die Dateien abzurufen, die an Montagen weniger als 20-mal von Internet Explorer-Browsern abgerufen werden.

Übersicht

Dieses Kapitel hat Sie mit Beispielen zur Leistungsfähigkeit der SELECT-Anweisung förmlich überschüttet. Nun sollten Sie den Datenabruf von einer einzelnen Tabelle in eine SQL-Datenbank sowie die wichtigsten Bereiche von SELECT-Anweisungen beherrschen. Insbesondere kennen Sie nun die folgenden Konzepte:

– SELECT ruft Daten von der Datenbank ab.

– WHERE-Klauseln filtern die Daten, die Sie abrufen.

– Die WHERE-Klausel besteht aus logischen Prädikatoperationen.

– Die ORDER BY-Klausel erlaubt Ihnen, Ihre Datensätze zu sortieren.

– Die GROUP BY-Klausel ermöglicht Ihnen die Gruppierung von Datensätzen.

– Die HAVING-Klausel erlaubt Ihnen das Filtern von aggregierten Datensätzen.