SQL auf iSeries

Structured Query Language (SQL) ist ein plattformunabhängiger Standard, der unabhängig von Hard- und Software für den Zugriff auf Datenbanken entwickelt wurde. Die Basis für diesen Standard bilden die Regeln des „American National Standards Institute – ANSI). Dabei existieren für einige Datenbank-Systeme gewisse Abwandlungen und Eigenarten des jeweils etwas spezifischen SQL-Standards. Diese Abweichungen können sowohl die einfache Syntax einiger Anweisungen betreffen, oder sich auf die Unterstützung ganzer SQL-Funktionen beziehen.

In diesem Kapitel werden wir uns die iSeries spezifischen SQL Eigenschaften anschauen. Gleichzeitig wird mit diesem Kapitel die Grundlage für das Verständnis der SQL-Funktionen in Verbindung mit den iSeries-Anforderungen gelegt. Wenn Sie sich bereits vorher ein wenig mit SQL auf iSeries beschäftigt haben, dann können Sie dieses Einführungskapitel auch problemlos überspringen und Ihre Arbeit mit dem Kapitel 2 fortsetzen.

Software Voraussetzungen

Für die praktische Ausführung der in diesem Buch gezeigten Beispiele benötigen Sie auf Ihrer iSeries das Lizenzprogramm 5722-ST1 „DB2 Query Manager and SQL Development Toolkit“.

Dieses Lizenzprogramm beinhaltet unter anderem einige wirklich hilfreiche Werkzeuge, mit denen Sie zum Beispiel Query Reports erstellen und Precompiler anwenden können, welche für die Unterstützung von Embedded SQL in anderen Programmiersprachen (z.B SQL RPG) benötigt werden.

Ebenfalls ein Bestandteil dieses Lizenzprogramms: der interaktiv ausführbare OS/400 Befehl „STRSQL“, mit dem Sie SQL-Anweisungen im Dialog ausführen können. Diese Möglichkeit der Ausführung von SQL-Anweisungen bietet sich besonders dann an, wenn Sie ad hoc SQL-Abfragen (z. B. für den schnellen Zugriff auf Inhalte einer Datei oder das das Verändern von Datensätzen) ausführen wollen oder müssen.

Wir werden uns in vielen Beispielen dieses Buches den Einsatz von STRSQL zu Nutze machen. In einem späteren Kapitel werde ich Ihnen dann auch zeigen, wie die Funktionen, die mit STRSQL zur Verfügung stehen, auch aus Client-basierten Anwendungen heraus – wie zum Beispiel Microsoft Access, Visual Basic und Active Server Pages – realisiert werden können.

SQL Tools

Sie werden in den folgenden Kapiteln dieses Buches eine Vielzahl möglicher Tools kennen lernen (sowohl solche, die über eine grafische Oberfläche verfügen, als auch Greenscreen basierte Werkzeuge), die Ihnen dabei helfen, SQL Anweisungen auf der iSeries schnell und effektiv eingeben zu können.

Der bereits erwähnte Befehl STRSQL kann von jeder beliebigen OS/400 Befehlseingabezeile aus eingegeben werden. Nach der Eingabe von STRSQL wird der interaktive SQL-Utility Bereich gestartet. In diesem Bereich steht mit dem Einsatz der Befehlstaste „F4“ eine erweiterte Bedienerführung bereit. Nach der Verwendung der F4-Taste erscheint eine Auflistung aller verfügbaren SQL-Befehle, die in der interaktiven SQL-Sitzung ausgeführt werden können.

SQL bietet verschiedene Arten der Ergebnisausgabe. So kann nach der Eingabe eines SQL-Befehls das Ergebnis wahlweise angezeigt oder in eine Druckausgabe umgeleitet werden. Die

Art der Ausgabe können Sie mit der F13-Taste auswählen. Selektieren Sie in dem nach dem Drücken der F13-Taste erscheinenden Bild zunächst die Option 1 und geben Sie danach in dem, Feld „SELECT-Ausgabe“ den Wert „1“ für die Ausgabe in einer Dialog-Anzeige, die „2“ für die Druckausgabe oder die „3“ für die Ausgabe in eine Datei an. Die Abildung 1.1. zeigt dazu ein Beispiel.

```
                          Change Session Attributes

Type choices, press Enter.

  Statement processing . . . . .   *RUN          *RUN, *VLD, *SYN
  SELECT output  . . . . . . . .   1             1=Display, 2=Printer
                                                 3=File
  Commitment control . . . . . .   *NONE         *NONE, *CHG, *CS, *ALL, *RR
                                                 *NC, *UR, *RS
  Date format  . . . . . . . . .   *MDY          *JOB, *USA, *ISO, *EUR, *JIS
                                                 *MDY, *DMY, *YMD, *JUL
  Date separator . . . . . . . .   '/'           *JOB, '/', '.', ',', '-'
                                                 ' ', *BLANK
  Time format  . . . . . . . . .   *HMS          *HMS, *USA, *ISO
                                                 *EUR, *JIS
  Time separator . . . . . . . .   ':'           *JOB, ':', '.', ','
                                                 ' ', *BLANK
  Data refresh . . . . . . . . .   *ALWAYS       *ALWAYS, *FORWARD
  Allow copy data  . . . . . . .   *YES          *YES, *OPTIMIZE, *NO
  Naming convention  . . . . . .   *SQL          *SYS, *SQL

                                                                   More...
F3=Exit   F4=Prompt   F5=Refresh   F12=Cancel

MA    a                                                          05/037
```

Abb. 1.1 SQL Select Ausgabesteuerung

In unserem Beispiel wurde die Option „1“ selektiert. Wenn Sie in dem Feld „SELECT output“ eine „2“ eingeben, dann wird das Ergebnis der SQL Anweisung in eine Druckausgabe umgeleitet. Option „3“ wird für die Ausgabe des Ergebnisses in eine Datei verwendet. Bei der Angabe dieser Option müssen Sie zusätzlich den Namen der Datei und die Bibliothek angeben, in der sich diese Datei befindet bzw. in der sie erstellt werden soll.

Beachten Sie, dass Sie in dieser Anzeige auch noch weitere Spezifikationen vornehmen können, wie zum Beispiel die Formatierung des Datums oder der Zeitangaben.

IBM hat auch in dem grafischen Administrationstool für die iSeries – dem iSeries Navigator – die Unterstützung von SQL integriert. Diese SQL Funktionen können wir verwenden, indem zunächst der Bereich „Datenbanken“ erweitert wird. Anschließend markieren wir den Eintrag und klicken mit der rechten Maustaste. In der angezeigten Optionsauswahl selektieren wir den Auswahlpunkt „SQL Prozedur ausführen“.

Das Ergebnis einer solchen Selektion sehen Sie in der Abbildung 1.2.

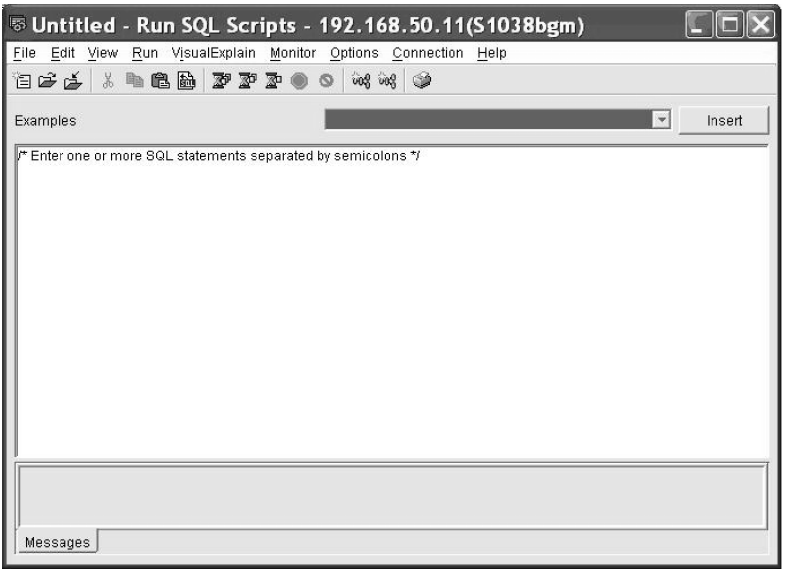

Abb. 1.2 SQL mit iSeries Navigator

Sie können diese Art der SQL-Angaben in ähnlicher Weise einsetzen, wie den Befehl STRSQL. Die Auswahlliste der Beispiele enthält Vorlagen für die Verwendung einiger SQL-Anweisungen. Ich werde Ihnen später noch zeigen, wie Sie auch diesen Bereich für die Erstellung von Funktionen und Prozeduren nutzen können. Es bleibt Ihnen überlassen, welche

der beiden Möglichkeiten Sie für die Ausführung der Beispiele in diesem Buch verwenden: Den Befehl „STRSQL“ in einer 5250-Sitzung oder die SQL-Tools des iSeries Navigators.

Einige SQL Anweisungen im Detail...

In diesem Abschnitt möchte ich Ihnen anhand einiger ausgewählter SQL-Anweisungen den Aufbau und die Arbeitsweise dieser Anweisungen erläutern.

Die Grundlagen

Das wohl am häufigsten verwendete SQL-Statement ist die Anweisung „SELECT, mit der wir in der Lage sind, auf bestimmte Datensätze innerhalb einer Datei zugreifen zu können. Die Abbildung 1.3 enthält ein Beispiel für die Verwendung der SELECT Anweisung.

```
SELECT field_list
FROM table_definition(s)
WHERE condition_expression
GROUP BY group_field_list
ORDER BY sort_field_list
HAVING group_level_conditions
```

Abb. 1.3 Die Basis-Struktur der SELECT-Anweisung

In diesem Beispiel ist nur die Angabe der Bestimmung „FROM“ in jedem Fall erforderlich. Alle übrigen Angaben sind optional – also bei Bedarf zu tätigen.

Anstelle der Angaben „Field_list“ könnte wahlweise auch eine Auflistung der auszuwertenden einzelnen Felder verwendet werden. Die einzelnen Felder sind dann jeweils mit einem Komma voneinander zu trennen. Sollen alle Felder einer Datei oder Tabelle verarbeitet werden, dann können Sie dafür an Stelle der Aufzählung aller einzelnen Felder den Platzhalter in Form eines Asterix-Zeichens „*“ verwenden.

In dem Parameter „table_definitions(s)“ können wir entweder eine Join-Angabe verschiedener Tabellen verwenden, oder den Namen einer physischen oder logischen Datei angeben (ich werde Ihnen die Besonderheiten der Join-Dateien später näher erläutern).

Mit der Angabe der WHERE-Bestimmung können wir gezielt auf die gewünschten Datensätze innerhalb der angegebenen Tabelle oder Datei zugreifen – also eine Selektion durchführen.

Die Bestimmung „GROUP BY“ stellt eine Ausgabegruppierung des Ergebnisses anhand der vorgegebenen Spalten dar. Mit dieser Bestimmung werden Zwischenergebnistabellen gefüllt, deren Inhalt ebenfalls ausgegeben werden kann.

Mit der Angabe von „sort_field_list“ für die Bestimmung „ORDER BY“ legen wir die Reihenfolge der Ausgabe des Ergebnisses der SELECT Anweisung fest.

Die Gruppen-Ebenen-Bedingung (group_level_conditions), welche mit der Angabe von „HAVING“ definiert werden kann, stellt eine mit der WHERE-Angabe vergleichbare Funktion dar. Der Unterschied zu WHERE liegt darin, dass „HAVING“ nach der Gruppierung (GROUP BY) ausgeführt wird. Damit können wir zum Beispiel weitere Bedingungs-Prüfungen durchführen.

Wie Sie bereits anhand dieses einfachen Beispiels erkennen können, bietet die SELECT Anweisung einige vielfältige Eingabemöglichkeiten, die durchaus auch komplexere Formen annehmen können.

Die nachfolgende Abbildung zeigt dagegen eine sehr einfache Form von „SELECT“:

```
SELECT *
FROM QSYS2.SYSTABLES
```

In diesem Beispiel verwenden wir an Stelle der sonst für den Zugriff auf eine Datei üblichen Angabe in Form von „Bibliothek/Datei“ eine leicht abgewandelte Variante, bei der die Trennung zwischen Bibliothek und Datei nicht mit einem Schrägstrich (/) erfolgt, sondern mittels eines Punktes (.) angegeben wird: „Bibliothek.Datei“.

In dem Beispiel zuvor haben wir demzufolge definiert, dass wir alle Feldinhalte der Datei „SYSTABLES“ aus der Bibliothek „QSYS2“ ermitteln wollen.

Es ist nicht immer sinnvoll, alle Feldinhalte einer Datei oder Tabelle mit einer SELECT-Anweisung abzurufen. Vielmehr besteht häufig die Anforderung, gezielt auf die Feldinhalte nur weniger Felder innerhalb einer Datei zugreifen zu müssen. Schauen Sie sich dazu dasselbe Beispiel, diesmal in etwas abgewandelter Form an. Jetzt werden nur die wirklich erforderlichen Feldinhalte ermittelt.

```
SELECT TABLE_NAME, TABLE_TYPE
FROM QSYS2.SYSTABLES
WHERE TABLE_SCHEMA = ‘QSYS’
ORDER BY TABLE_NAME
```

Mit diesem Beispiel wird lediglich der Inhalt der Felder (oder Spalten) „TABLE_NAME“ und „TABLE-TYPE“ ermittelt. Dabei werden nur die Inhalte berücksichtigt, welche in der Spalte „TABLE_SCHEMA“ den Wert „QSYS“ besitzen.

Auch dieses Beispiel lässt sich weiter verfeinern oder auf unsere speziellen Anforderungen anpassen – zum Beispiel mit der Verwendung einer Summenangabe und der Angabe einer Gruppierung des Ergebnisses:

```
SELECT TABLE_SCHEMA, COUNT(*) AS TABLE_COUNT,
 SUM(ROW_LENGTH) AS TOTAL_ROW_LEN
FROM QSYS2.SYSTABLES
WHERE TABLE_TYPE = 'P'
GROUP BY TABLE_SCHEMA
HAVING COUNT(*)>10
```

Hier werden die Bibliotheksnamen (TABLE_SCHEMA) und die Anzahl der in jeder Bibliothek befindlichen physischen Dateien samt der darin befindlichen Zeilen ermittelt. Mit der Angabe von „HAVING“ legen wir fest, dass nur solche Bibliotheken berücksichtigt werden sollen, in denen mehr als 10 physische Dateien existieren.

Eine leichte Abwandlung dieser SELECT Anweisung stellt die Angabe von „SELECT INTO“ dar. Mit dieser Angabe können wir erreichen, dass das Ergebnis einer SELECT-Ausführung als Variablen einer Stored Procedure oder Embedded SQL in einem RPG Programm übertragen werden kann. Bei dem Einsatz von “SELECT INTO“ kann nicht mehr als eine Zeile pro Ausführung zurückgegeben werden. Das bedeutet, dass mit einer Anweisung jeweils nur eine Ergebniszeile ermittelt werden kann.

Die SELECT-Anweisung dient lediglich dem Abrufen bzw. Einlesen von Daten aus einer Datenbank, bietet jedoch keine Möglichkeit, diese Daten zu manipulieren. Mit der Anweisung „DELETE“ können wir beispielsweise einen Datensatz innerhalb einer Datei oder einer Tabelle löschen. Die Abbildung 1-4 zeigt die Syntax einer solchen DELETE-Anweisung.

```
DELETE
FROM table_definition(s)
WHERE condition_expression
```

Abb. 1.4 Die Anweisung „DELETE“ zum Löschen eines Datensatzes

Der Aufbau dieser Anweisung ist weniger komplex, als der der SELECT-Anweisung. Hier sind lediglich die Angaben von „FROM“ und „WHERE“ Klauseln möglich.

Nachfolgende finden Sie ein einfaches Beispiel für die Angabe einer DELETE-Anweisung.

```
DELETE FROM MYLIB.MYFILE WHERE RCSTAT = ‘D’
```

Mit diesem Beispielcode werden alle in der Datei „MYLIB“ befindlichen Datensätze gelöscht, deren Feldinhalt „RCSTAT“ den Wert „D“ aufweisen.

Zum Manipulieren bzw. Verändern von Datensätzen verwenden wir die Anweisung „UPDATE“, deren Struktur durchaus mit „DELETE“ vergleichbar ist. Die Abbildung 1.5 zeigt dazu ein Beispiel.

```
UPDATE table_definition(s)
SET field = value
WHERE condition_expression
```

Abb. 1.5 Die Anweisung „UPDATE“ wird zum Verändern von Datensätzen verwendet

Beachten Sie bitte, dass wir in diesem Fall auf die Angabe der „table_definition“ innerhalb der Option „FROM“ verzichten. Mit der Angabe von „SET“ legen wir sowohl das zu verändernde Feld, als auch dessen neuen Inhalt fest. Und wie wir es bereits von den Anweisungen „SELECT“ und „DELETE“ kennen: mit der Angabe von „WHERE“ filtern wir die zu verarbeitenden Datensätze.

Die Anweisung „INSERT INTO“ ermöglicht uns, neue Datensätze in eine Tabelle oder Datei zu schreiben. Die dabei zu verwendende Syntax finden Sie exemplarisch in der nachfolgenden Abbildung 1.6 dargestellt:

```
INSERT INTO table_definition(s) (field list)
VALUES (value list)

Or

INSERT INTO table_definition(s) (field list)
Select statement
```

Abb. 1.6 Die Syntax von „INSERT INTO“ akzeptiert zwei unterschiedliche Eingabeversionen

Mit der Option „table_definition“ geben wir die Tabelle an, in welche der neue Datensatz geschrieben werden soll. Optional können wir nach dieser Angabe eine Feldliste verwenden, die mit dieser Anweisung verarbeitet wird. Wenn Sie auf die Angabe einer solchen Feldliste verzichten, dann werden alle in der Tabelle enthaltenen Felder fortgeschrieben bzw. es wird davon ausgegangen, dass für alle in der Tabelle existierenden Felder Werte zugewiesen werden.

Für die Wertzuweisung können wir zwei unterschiedliche Methoden verwenden:

1. Eine Werte-Liste mit den von Ihnen angegebenen Werten, welche in eine Zeile eingetragen werden sollen.
2. Eine SELECT-Anweisung mit der eine oder mehrere Zeilen in eine Tabelle geschrieben werden.

Die Anzahl der angegebenen Werte muss dabei mit der Anzahl der Felder übereinstimmen, die eingefügt werden sollen. Dies trifft für beide Verfahren Werte-Liste und SELECT-Anweisung gleichermaßen zu.

Es geht auch komplexer

Jede der bisher genannten Anweisung wird umso komplexer, wenn sie im Zusammenhang mit der Verarbeitung mehrerer Dateien gemeinsam verwendet eingesetzt wird. Dabei ist es gerade die Methode des Datenzugriffs, die sich zum Teil deut-

lich unterscheidet, wenn es darum geht, die Werte nur einer Datei, oder die Werte aus mehreren Dateien verarbeiten zu müssen.

Die SELECT-Anweisung arbeitet in diesem Zusammenhang mit der JOIN-Anweisung, um auf den Inhalt mehrerer Dateien zugreifen zu können. Die JOIN-Anweisung verfügt dabei über mehrere Optionen, die einen flexiblen und individuellen Zugriff auf die verschiedenen Dateien oder Tabellen ermöglichen. In der Tabelle 1.1 finden Sie eine Übersicht dieser möglichen Optionen und eine Beschreibung zu jeder einzelnen Variante.

Join Typ	Beschreibung
INNER JOIN/JOIN	Mit dieser Join-Anweisung werden nur solche Zeilen verarbeitet, die in beiden Tabellen enthalten sind und mit den Auswahlangaben übereinstimmen.
LEFT JOIN/ LEFT OUTER JOIN	Mit dieser Join-Anweisung werden solche Zeilen der links von der Join-Anweisung definierten Tabelle für die Verarbeitung ausgewählt. Dabei werden nur die Zeilen selektiert, welche mit den Zeilen der rechts neben der Join-Anweisung angegebenen Tabelle übereinstimmen.
RIGHTER JOIN/ RIGHT OUTER JOIN	Mit dieser Join-Anweisung werden solche Zeilen der rechtes von der Join-Anweisung definierten Tabelle für die Verarbeitung ausgewählt. Dabei werden nur die Zeilen selektiert, welche mit den Zeilen der links neben der Join-Anweisung angegebenen Tabelle übereinstimmen.

Join Typ	Beschreibung
LEFT EXCEPTION JOIN/ EXCEPTION JOIN	Bei Verwendung dieser Join-Anweisung werden nur die Inhalte der Datensätze einer auf der linken Seite der Join-Anweisung angegebenen Tabelle verarbeitet. Dabei werden nur solche Datensätze verarbeitet, die keine Übereinstimmung „passender" Datensätze innerhalb der rechts neben der Join-Anweisung definierten Tabelle besitzen. Feldinhalte der Tabelle, die auf der rechten Seite angegeben wurden, werden im Falle einer Nichtübereinstimmung mit „null" zurückgegeben.
RIGHT EXCEPTION JOIN	Bei der Verwendung dieser Join-Anweisung werden nur die Inhalte der Datensätze einer auf der rechten Seite der Join-Anweisung angegebenen Tabelle verarbeitet. Dabei werden nur solche Datensätze verarbeitet, die keine Übereinstimmung „passender" Datensätze innerhalb der links neben der Join-Anweisung definierten Tabelle besitzen. Feldinhalte der Tabelle, die auf der linken Seite angegeben wurden, werden im Falle einer Nichtübereinstimmung mit „null" zurückgegeben.
CROSS JOIN	Bei dieser Form der Join-Anweisung wird jede einzelne Zeile der links neben der Join-Anweisung definierten Tabelle mit jeder Zeile der Tabelle verglichen, die sich in der rechts neben der Join-Anweisung angegebenen Tabelle befindet.

Tabelle 1.1 JOIN-Bestimmungen und deren Beschreibungen

Es kommt auf die jeweilige Anforderung an, welche von diesen verschiedenen Join-Anweisungen in einem konkreten Fall eingesetzt werden sollte. Ein „INNER JOIN" wird sicher dann zum Einsatz kommen, in dem Sie bereits wissen oder davon ausgehen, dass es in zwei unterschiedlichen Tabellen übereinstimmende Datensätze gibt, und in dem Sie nur diese Daten-

sätze sehen bzw. ermitteln wollen, welche über eine Übereinstimmung in den angegebenen Dateien verfügen. Das bedeutet auch, dass für den Fall einer Nicht-Übereinstimmung der Werte beider Tabellen, die links und rechts neben der Join-Anweisung angegeben wurden, die Datensätze ohne entsprechende Übereinstimmung nicht ausgewählt werden und damit auch für eine weitere Verarbeitung nicht zur Verfügung stehen. Dabei ist es unabhängig in welcher der beiden Dateien oder Tabellen ein entsprechend übereinstimmender Datensatz fehlt. Schlägt ein Vergleich fehl, dann werden die Datensätze nicht verarbeitet.

Ein gutes Beispiel für eine solche Join-Anweisung ist die Verarbeitung von Datensätzen innerhalb einer Auftragskopf-Datei und einer Auftragspositions-Datei. Hier ist es sicher interessant, nur die Datensätze zu verarbeiten, die auch „wirkliche“ Aufträge darstellen – das sind solche, die sowohl in der Auftragskopf-Datei als auch in der Auftragspositions-Datei enthalten sind. Die Abbildung 1.7 zeigt ein Beispiel für solch eine Join-Anweisung.

```
SELECT *
FROM MYLIB.ORDHEADR INNER JOIN  MYLIB.ORDDETL
     ON ORDHEADR.ORDNO = ORDDETL.ORDNO
```

Abb. 1.7 Beispiel für einen INNER JOIN

Mit diesem Beispiel werden alle Felder der Datensätze der Dateien (oder Tabellen) „ORDHEADR“ und „ORDDETL“ zurückgegeben, für die eine Übereinstimmung der Felder „ORDNO“ in beiden Dateien gefunden wurde.

Die Tabelle 1.2 zeigt ein mögliches Ergebnis dieser INNER-JOIN-Anweisung.

ORDHEADR			
ORDNO	CUSNO	ORDAT	ORVAL
12345	ABC	20050101	500.00
12346	DEF	20050115	350.00
12347	GHI	20050110	160.00
12348	AAA	20041231	0.00

ORDDETL			
ORDNO	ORDLN	ORITM	ORQTY
12345	1	11111	5
12345	2	22222	10
12345	3	33333	16
12346	1	344A	5
12346	2	224B	10
12347	1	122DF	25
12347	2	11111	12

SELECT Statement Results							
ORDNO	CUSNO	ORDAT	ORVAL	ORDNO	ORDLN	ORITM	ORQTY
12345	ABC	20050101	500.00	12345	1	11111	5
12345	ABC	20050101	500.00	12345	2	22222	10
12345	ABC	20050101	500.00	12345	3	33333	16
12346	DEF	20050115	350.00	12346	1	344A	5
12346	DEF	20050115	350.00	12346	2	224B	10
12347	GHI	20050110	160.00	12347	1	122DF	25
12347	GHI	20050110	160.00	12347	2	11111	12

Tabelle 1.2 Beispiele für einen INNER JOIN

Wie Sie anhand dieses Beispiels sehen können, werden bei der Verwendung einer solchen INNER-JOIN-Anweisung alle übereinstimmenden Dateiinhalte zurückgegeben. „Alle Dateiinhalte“ bedeutet im Speziellen auch, dass mehrfach vorkommende Dateiinhalte verarbeitet werden. So werden zum Beispiel für einen in der Auftragskopf-Datei „ORDHEADR“ gefundenen Auftrag die entsprechenden Auftragspositionen in der Datei „ORDDETL“ über den Vergleichswert „Auftragsnummer“ (ORDNO) ermittelt. Dabei werden auch mehrere Auftragspositionen (wenn vorhanden) verarbeitet.

Beachten Sie in diesem Beispiel auch die Auftragsnummer „12348“, die zwar in der Datei „ORDHEADR“ enthalten ist, für die es aber keinen passenden Eintrag in der Datei „ORDDETL“ gibt. Diese Zeile wird in dem Ergebnis der JOIN-Anweisung nicht berücksichtigt.

Mit einer Left-Join-Anweisung werden alle Datensätze einer links neben der Join-Anweisung definierten Tabelle verarbeitet, für die ein passender Datensatz in der rechts neben der Join-Anweisung definierten Tabelle enthalten ist. Jetzt wird selbst in dem Fall, dass in der rechts neben der Join-Anweisung definierten Tabelle kein passender Datensatz zu der links neben der Anweisung angegebenen Tabelle gefunden wurde, der Inhalt der links neben der Anweisung definierten Tabelle trotzdem für die weitere Verarbeitung zur Verfügung stehen.

Ein Beispiel für die Verwendung einer solchen Left-Join-Anweisung kann der Zugriff auf eine Kommentar-Datei sein. Nehmen wir wieder unsere Auftragskopf-Datei. Jetzt möchten wir mit einer Join-Anweisung auch alle Auftrags-Kommentare ermitteln, die sich eventuell in der Kommentar-Datei befinden könnten. Grundlage für die Verwendung ist: In der Kommentar-Datei können passende Datensätze enthalten sein, müssen es aber nicht.

Die Abbildung 1.8 enthält dazu ein passendes Beispiel.

```
SELECT *
FROM MYLIB.ORDHEADR LEFT JOIN MYLIB.COMMENTS
     ON ORDHEADR.ORDNO = COMMENTS.ORDNO
```

Abb. 1.8 Verwendung eines LEFT-JOIN

In diesem Beispiel werden alle Zeilen der Auftragskopf-Datei „ORDHEADR“ verarbeitet. Zusätzlich wollen wir alle eventuell für einen Auftrag vorhandenen Kommentare einlesen, die

sich in der Tabelle „COMMENTS“ befinden können. Die Tabelle 1.3 zeigt das Ergebnis einer solchen Abfrage.

ORDHEADR			
ORDNO	CUSNO	ORDAT	ORVAL
12345	ABC	20050101	500.00
12346	DEF	20050115	350.00
12347	GHI	20050110	160.00
12348	AAA	20041231	0.00

COMMENTS	
ORDNO	TEXT
12345	Ship to Bob
12346	Next-day air
12347	No foam peanuts
12347	Use recycled packing

SELECT Statement Results					
ORDNO	CUSNO	ORDAT	ORVAL	ORDNO	TEXT
12345	ABC	20050101	500.00	12345	Ship to Bob
12346	DEF	20050115	350.00	12346	Next-day air
12347	GHI	20050110	160.00	12347	No foam peanuts
12347	GHI	20050110	160.00	12347	Use recycled packing
12348	AAA	20041231	0.00		

Tabelle 1.3 Beispiel eines LEFT-JOIN

Wie Sie mit Hilfe dieses Beispiels sehen können, werden auch die Aufträge verarbeitet bzw. ermittelt, für die kein Kommentar in der Datei „COMMENTS“ enthalten ist. Das bedeutet, dass alle Datensätze, der links neben der Join-Anweisung definierten Tabelle verarbeitet werden – und das unabhängig davon, ob ein entsprechend passender Eintrag in der rechts neben der Join-Anweisung befindlichen Tabelle enthalten ist. Wenn keine Übereinstimmung gefunden wurde, dann werden die „Feldinhalte“ der rechts neben der Anweisung befindlichen Tabelle als „null“ zurück geliefert.

Der RIGHT-JOIN arbeitet umgekehrt, wie der LEFT-JOIN. In der nachfolgenden Abbildung 1.9 finden Sie ein Beispiel für die Verwendung einer RIGHT-JOIN.

```
SELECT *
FROM MYLIB.COMMENTS RIGHT JOIN MYLIB.ORDHEDR
     ON ORDHEDR.ORDNO = COMMENTS.ORDNO
```

Abb. 1.9 Die RIGHT-JOIN Anweisung

Das Ergebnis, welches dieses Beispiel zurückliefert, ist das Gleiche, wie wir es auch bei dem Einsatz des LEFT-JOIN gesehen haben. Der einzige Unterschied liegt in der Sortierung des Ergebnisses, denn die für die Sortierung der Ausgabe der Datensätze „verantwortliche“ Tabelle befindet sich jetzt rechts neben der Join-Anweisung und besitzt offensichtlich eine andere Sortierung, als die links neben der Anweisung definierte Tabelle.

Wir fassen also zusammen:

Die LEFT-EXCEPTION-JOIN und RIGHT-EXCEPTION-JOIN Anweisungen werden für alle möglichen Fälle eingesetzt – auch für solche, für die es eventuell keine „passenden“ Einträge in einer anderen Tabelle oder Datei gibt.

Die Abb. 1.10 zeigt ein Beispiel einer SELECT-Anweisung, mit einem LEFT-EXCEPTION-JOIN.

```
SELECT *
FROM CUSTOMERS C LEFT EXCEPTION JOIN SALES S
ON C.CUSNO = S.CUSNO
```

Abb. 1.10 LEFT-EXCEPTION-JOIN zum Auffinden von fehlenden Datensätzen

Mit diesem Beispiel werden alle Datensätze einer Kundenstammdatei „CUSTOMERS“ ermittelt, für die es keine passenden Einträge in der Umsatzdatei „SALES“ gibt. Beachten Sie in diesem Beispiel auch die einzelstehenden Buchstaben „C“ und „S“. Diese stellen so genannte Alias für jede Tabelle dar, mit denen wir die Dateinamen quasi abkürzen können. Damit verringert sich der spätere Codieraufwand zum Teil erheblich. Die Verwendung von solchen Alias-Abkürzungen bietet sich zum Beispiel auch für die Verarbeitung von vielen einzelnen Feldern innerhalb eines SQLs an.

Die CROSS-JOIN Anweisung kombiniert alle Datensätze einer Tabelle mit allen Datensätzen einer weiteren Tabelle. Bei dieser Anweisung können wir auf die Angabe von Join-Feldern verzichten. Mit jedem Zugriff auf einen Datensatz in der Tabelle links neben der Join-Anweisung werden alle Datensätze der rechts neben der Join-Anweisung enthaltenen Datei zurückgegeben.

In der Abbildung 1.11 finden Sie ein Beispiel für die Angabe eins CROSS-JOIN.

```
SELECT *
FROM TABLE1 CROSS JOIN TABLE2
```

Abb. 1.11 Die Anweisung CROSS-JOIN ermittelt alle Datensätze der beiden angegebenen Tabellen

Bei der Ausführung dieses CROSS-JOIN kann unter Umständen eine große Anzahl an Ergebnis-Einträgen zurückgegeben werden. Die Tabelle 1.4 zeigt ein Beispiel für ein mögliches Ergebnis einer solchen Anweisung. In diesem Beispiel sehen

Sie unter anderem auch, wie die Daten miteinander kombiniert werden.

TABLE 1	
RECN1	VALUE1
1	100.75
2	200.25
3	195.55

TABLE 2	
RECN2	VALUE2
1	12.76
2	115.95

SELECT Statement Results			
RECN1	VALUE1	RECN2	VALUE2
1	100.75	1	12.76
1	100.75	2	115.95
2	200.25	1	12.76
2	200.25	2	115.95
3	195.55	1	12.76
3	195.55	2	115.95

Tabelle 1.4 Beispiel für ein CROSS-JOIN

Wie Sie an diesem einfachen Beispiel sehen können, liefert CROSS-JOIN alle Datensätze der links neben der Anweisung angegebenen Tabelle, multipliziert mit den Einträgen der rechts neben der JOIN-Anweisung definierten Tabelle zurück.

Mit jeder, der bisher betrachteten JOIN-Varianten können wir spezielle Anforderungen realisieren. Bei der Verarbeitung mehrerer Dateien in einem SQL können wir natürlich neben dem Lesen der Inhalte der Datensätze auch Veränderungen (UPDATE) dieser Datensätze durchführen. Allerdings lässt sich diese nicht mit Hilfe von einfachen JOIN-Anweisungen realisieren. In einem solchen Fall müssen wir eine zweite „Sub“-SELECT-Anweisung verwenden, mit der wir auf den Inhalt der zweiten Tabelle zugreifen können.

Es gibt zwei unterschiedliche Szenarien, in denen wir solch eine Sub-SELECT-Anweisung verwenden können:

- Innerhalb einer Vergleichsoperation (WHERE-Anweisung)
- Als Teil des zu verändernden Wertes

In den Fällen, in denen die Kriterien eines Datensatzes einer Tabelle mit den Kriterien eines Datensatzes in einer zweiten Tabelle verglichen werden sollen, werden wir diesen Vergleich am besten mit einem SQL SELECT realisieren können. Ein gutes Beispiel ist das Verändern eines Datensatzes in der Auftragspositions-Datei in Abhängigkeit des in der Auftragskopf-Datei befindlichen Datensatzes. Die Abbildung 1.12 zeigt dazu ein entsprechendes Beispiel:

```
UPDATE MYLIB.ORDLINS
         SET ODPURG = 'YES'
         WHERE ODORDR  IN (SELECT OHORDR FROM MYLIB.ORDHEAD
              WHERE OHODAT<20030101)
```

Abb. 1.12 Beispiel für embedded SQL mit Kriterienvergleich

Mit diesem Beispiel werden nur solche Datensätze in dem Sub-Select (also aus der Datei ORDHEAD) ermittelt, deren Auftragsdatum „OHODAT“ älter als „20030101“ ist.

Die WHERE-Anweisung des UPDATE-Statements wird deshalb nur solche Datensätze verarbeiten, für die es eine in der Sub-Select-Anweisung definierte Übereinstimmung gibt.

In der WHERE-Anweisung können wir auch Werte der zu verändernden Datei einbinden. In unserem Beispiel beziehen wir uns auf das Feld „ORDODR“, welches aus der anzupassenden Datei „MYLIB.ORDLINS“ stammt.

Bei der Ausführung eines SQL UPDATE müssen wir häufig auch solche Fälle verarbeiten, in denen nicht nur ein Ver-

gleichskriterium abgefragt werden soll, sondern Mehrfachvorkommen innerhalb einer weiteren Datei verarbeitet werden müssen. Ein Beispiel dazu ist die Verarbeitung aller zu einem Kopf-Satz gehörenden Positions-Sätze und die Summierung der Positions-Einzelwerte. Eine solche Anforderung lässt sich mit Hilfe eines Sub-SELECT realisieren, wie ich sie Ihnen bereits in einem der vorhergehenden Beispiele vorgestellt habe. Die Abbildung 1.13 zeigt die Verwendung eines SQL UPDATEs.

```
UPDATE MYLIB.ORDHEAD
        SET OHOTOT = (SELECT SUM(ODOQTY) FROM MYLIB.ORDLINS
                      WHERE ODORDR=OHORDR)
        WHERE OHODAT<20030101
```

Abb. 1.13 Beispiel eines Sub-SELECT zur Ermittlung von Feldwerten und Summenbildung

In diesem Beispiel verwenden wir das Sub-SELECT als Ziel für die Set-Anweisung. Während der Ausführung dieses Scripts werden alle Werte des Feldes „ODOQTY“ aus der Datei „ORDLINS“ summiert. Diese Summierung wird nur für die Datensätze durchgeführt, deren Wert „ORORDR“ gleich dem Wert des Feldes „OHORDR“ aus der Datei „ORDHEAD“ ist. Anschließend soll die Datei „ORDHEAD“ fortgeschrieben werden. Auch in diesem Beispiel wird der Wert der „primary“ Datei „ORDHEAD“ innerhalb der WHERE-Anweisung des Sub-SELECT verwendet.

In diesem speziellen Fall ist es wichtig zu berücksichtigen, dass mit dem Sub-SELECT lediglich ein Wert für jeden Datensatz verarbeitet werden darf. Dies bedeutet, dass wir eine 1:1 Beziehung eines jeden Datensatzes der fortzuschreibenden Datei und des Ergebnisses des Sub-SELECT abfragen müssen. Wenn eine solche Beziehung nicht gegeben ist, dann wird der Fehler

SQL0811 ausgegeben, der aussagt, dass es für eine Abfrage mehrere Zeilen gibt. Dasselbe trifft auch für den Fall zu, in welchem die Sub-SELECT Anweisung als Teil der „WHERE"-Angabe codiert wird. In unserem Beispiel stellt das kein Problem dar, da mit der Summenfunktion „SUM()" alle Datensätze innerhalb der Datei „ODORDR" verarbeitet werden, welche den abgefragten Kriterien entsprechen.

Diese beiden Methoden können für die Sub-SELECT Verwendung als Teil einer SET und einer WHERE Abfrage angegeben werden. Ein praktisches Beispiel stellt die Neuberechnung des Auftragswertes innerhalb einer Auftragskopf-Datei dar, der mit Hilfe der Einzelwerte der Positions-Sätze aus der Auftragspositions-Datei ermittelt wird.

In der Abbildung 1.14 sehen Sie ein Beispiel für das Fortschreiben der Inhalte einer Datei, die wir bereits in einem vorhergehenden Beispiel verarbeitet haben.

```
UPDATE MYLIB.ORDHEAD
        SET OHOTOT = (SELECT SUM(ODOQTY) FROM MYLIB.ORDLINS
                        WHERE ODORDR=OHORDR)
        WHERE OHCUSN IN(SELECT CMCUSN FROM MYLIB.CUSTMAST
                        WHERE CMREGN='NE' AND CMCUSN=OHCUSN)
```

Abb. 1.14 Verwendung von Sub-SELECT für die Kriterienabfrage und für Wertangaben

Auf den ersten Blick erscheint die Verwendung der WHERE-Anweisung ein wenig verwirrend. Mit der Sub-SELECT-Angabe werden nur die Datensätze aus der Kundenstammdatei „CUSTMAST" ermittelt, deren Feldwert „CMREGN" gleich „NE" ist. Die Angabe dieses Sub-SELECTs innerhalb der „IN" Abfrage erzeugt eine Werteliste, die mit dem Inhalt von „OHCUSN" verglichen wird.

Die Sub-SELECT Anweisung innerhalb der „SET“-Angabe fasst die Feldwerte „ODOQTY“ für all die Datensätze zusammen, deren Feldinhalt „ODORDR“ mit dem Wert des Feldes „OHORDR“ aus der Datei „ORDHEAD“ übereinstimmt.

Bei der Ausführung dieser Anweisung werden alle Auftragswerte für die Kunden neu berechnet, die sich in der Region „NE“ befinden. In diesem Beispiel schreiben wir den Inhalt einer Datei in Abhängigkeit des Vergleiches von zwei separaten Tabellen fort.

Eine vergleichbare Technik können wir auch zum Löschen von Datensätzen innerhalb einer Datei bzw. Tabelle anwenden. Dabei können wir wieder gewisse Abhängigkeiten von Datensätzen einer Datei zu einer weiteren Datei abfragen und die gewünschten Datensätze anschließend löschen. So lässt sich eine Löschanforderung zum Beispiel mit Hilfe einer Sub-SELECT-Anweisung innerhalb der WHERE-Abfrage realisieren, wie in der Abb. 1.15 dargestellt. In dieser Abbildung finden Sie ein leicht abgewandeltes Beispiel, das wir uns bereits vorher angeschaut haben.

```
DELETE
FROM MYLIB.ORDLINS
        WHERE EXISTS (SELECT OHORDR FROM MYLIB.ORDHEAD
                            WHERE OHORDR=ODORDR AND
                              OHODAT<20030101)
```

Abb. 1.15 Verwendung eines Sub-SELECT zum Löschen von ausgewählten Datensätzen

In diesem Fall greifen wir auf das Prädikat „EXISTS“ zurück. Mit Hilfe dieser Anweisung können wir feststellen, ob ein Datensatz mit Hilfe einer Sub-SELECT Anweisung zurückgegeben wurde. Das bedeutet konkret, dass mit Hilfe von „EXISTS“ nach einer Sub-SELECT-Angabe ermittelt werden kann, ob ein entsprechender Zeileninhalt existiert. Werden

zutreffende Datensätze gefunden und damit auch Feldinhalte zurückgegeben, dann ergibt dies als Rückgabewert für das Prädikat „EXISTS“ „TRUE“. Ist diese nicht der Fall – es konnte also kein zutreffender Datensatz ermittelt werden – dann liefert EXISTS „FALSE“ zurück.

Von der Programmstruktur zur Funktion

In diesem Kapitel haben wir einen Kurzüberblick über die Struktur der SQL-Sprache erhalten und einige spezielle SQL-Anweisungen exemplarisch angewendet. Nach dieser Kurzeinweisung in die elementaren Strukturen von SQL wollen wir uns jetzt die Welt der SQL-Funktionen näher anschauen.