Daten verändern: insert, update und delete

Beinahe alles, was Sie bisher über Oracle, SQL und SQLPLUS kennen gelernt haben, bezog sich auf die Auswahl von Daten aus den Tabellen in der Datenbank. Dieses Kapitel zeigt Ihnen, wie man die Daten in einer Tabelle *verändert*: wie neue Zeilen eingefügt, die Werte aktualisiert und Zeilen vollständig gelöscht werden. Obwohl diese Themen nicht explizit besprochen wurden, können Sie Ihr gesamtes Wissen über SQL, d.h. die Datentypen, Berechnungen, Zeichenkettenformate, **where**-Klauseln usw. einsetzen, sodass es im Prinzip nicht allzu viel Neues gibt. Oracle bietet Ihnen eine transparente Datenbankumgebung, in der Sie auch in Ferndatenbanken Daten einfügen, aktualisieren und löschen können (siehe Kapitel 22).

15.1 insert

Mit dem SQL-Befehl **insert** stellen Sie eine Zeile mit Informationen direkt in einer Tabelle ab (oder indirekt, über eine View). Die COMFORT-Tabelle protokolliert für verschiedene Städte die Temperaturen um die Mittagsstunde, um Mitternacht und die täglichen Niederschläge. In unserer Tabelle sind die Daten für vier Beispieltage enthalten:

```
describe COMFORT

Name                            Null?    Type
------------------------------- -------- ----
CITY                                     VARCHAR2(13)
SAMPLEDATE                               DATE
NOON                                     NUMBER
MIDNIGHT                                 NUMBER
PRECIPITATION                            NUMBER
```

Um ein neues Ziel einzufügen, geben Sie Folgendes ein:

```
insert into COMFORT
values ('WALPOLE', TO_DATE('21-MAR-1993','DD-MON-YYYY'),
         56.7, 43.8, 0);

1 row created.
```

Das Wort **values** ist der Liste mit Daten voranzustellen, die eingefügt werden sollen. Zahlen können alleine stehen. Eine Zeichenkette ist in einfachen Anführungszeichen abzustellen. Die einzelnen Felder werden durch Kommas getrennt. Die Felder müssen in der gleichen Reihenfolge stehen wie die Spalten in der Tabellenbeschreibung.

Ein Datum muss in einfache Anführungszeichen gesetzt werden und im Standard-Oracle-Format vorliegen. Zum Einfügen eines Datums, das nicht diesem Standardformat entspricht, verwenden Sie die **TO_DATE**-Funktion zusammen mit einer Formatierungsmaske:

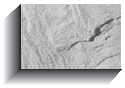
```
insert into COMFORT
values ('WALPOLE', TO_DATE('06/22/1993','MM/DD/YYYY'),
        56.7, 43.8, 0);

1 row created.
```

15.2 Eine Zeitangabe einfügen

Beide Methoden für das Einfügen einer Zeitangabe erzeugen eine Standardzeit um Mitternacht, also am Tagesbeginn. Wenn Sie eine abweichende Zeitangabe einfügen möchten, greifen Sie auf die **TO_DATE**-Funktion zurück:

```
insert into COMFORT
values ('WALPOLE', TO_DATE('06/22/1993 1:35',
        'MM/DD/YYYY HH24:MI'), 56.7, 43.8, 0);

1 row created.
```

Wenn Sie als Erstes (vor dem Wort **values**) die Reihenfolge der Spalten aufführen, können Sie auch Daten einfügen, die nicht der Standardanordnung einer Tabelle folgen. Damit wird die Reihenfolge der Spalten innerhalb der Tabelle nicht verändert. Sie haben lediglich die Möglichkeit, die Daten in einer abweichenden Reihenfolge aufzuführen.



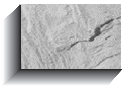
Hinweis:
Grundsätzlich können Sie auch einen **Null***-Wert einfügen. Das bedeutet allerdings nur, dass die Spalte für diese Zeile leer gelassen wird. Ein Beispiel:*

```
insert into COMFORT
       (SampleDate,  Precipitation, City, Noon, Midnight)
values (TO_DATE('23-SEP-1993','DD-MON-YYYY'), NULL,
        'WALPOLE', 86.3, 72.1);

1 row created.
```

15.3 insert mit select

Weiterhin können Sie auch Daten einfügen, die aus einer Tabelle ausgewählt wurden. In unserem Beispiel werden aus der COMFORT-Tabelle verschiedene Spalten ausgewählt und übliche Werte für SampleDate (22-DEC-93) und City (WALPOLE) eingefügt. Achten Sie in der **select**-Anweisung auf die **where**-Klausel, über die nur eine Zeile abgeholt wird. Hätte die **select**-Anweisung fünf Zeilen enthalten, wären fünf Zeilen eingefügt worden, bei zehn wären es zehn etc. gewesen.

```
insert into COMFORT
       (SampleDate, Precipitation, City, Noon, Midnight)
select TO_DATE('22-DEC-1993','DD-MON-YYYY'), Precipitation,
       'WALPOLE', Noon, Midnight
  from COMFORT
 where City = 'KEENE' and SampleDate = '22-DEC-93';

1 row created.
```



Hinweis:
*Beim Datentyp LONG kann die **insert into...select from**-Syntax nicht verwendet werden.*

Natürlich müssen Sie den Wert nicht nur einfach in einer ausgewählten Spalte einfügen. Sie können die Spalte innerhalb der **select**-Anweisung mit den entsprechenden String-, Datums- oder Zahlenfunktionen verändern. Die Ergebnisse dieser Funktionen sind die Inhalte, die eingefügt werden. Sie können beispielsweise versuchen, einen Wert einzufügen, der die zulässige Breite der Spalte (bei Datentypen für Zeichen) überschreitet oder dessen Genauigkeit (bei Zahlen) nicht eingehalten wird. Sie müssen die Vorgaben für die jeweiligen Spalten einhalten. Bei einem solchen Versuch erhalten Sie lediglich eine Fehlermeldung. Wenn Sie jetzt die COMFORT-Tabelle nach der Stadt Walpole abfragen, werden Ihnen als Ergebnis die eingefügten Datensätze angezeigt:

```
select * from COMFORT
 where City = 'WALPOLE';

CITY           SAMPLEDAT       NOON   MIDNIGHT PRECIPITATION
-------------- --------- ---------- ---------- -------------
WALPOLE        21-MAR-93        56.7       43.8             0
WALPOLE        22-JUN-93        56.7       43.8             0
WALPOLE        23-SEP-93        86.3       72.1
WALPOLE        22-DEC-93        -7.2       -1.2           3.9

4 rows selected.
```

Ein Anmerkung zur Performance

Wie in Kapitel 36 noch ausgeführt wird, benutzt Oracle zur Ermittlung des effizientesten Weges zur Ausführung der SQL-Befehle einen Optimizer. Bei **insert**-Anweisungen versucht Oracle, die neuen Datensätze in Blöcke einzufügen, die von der Tabelle bereits reserviert sind. Der Ausführungsplan optimiert die Ausnutzung des Speicherplatzes, der zum Abspeichern der Daten benötigt wird. Dennoch kann die Performance bei einem **insert** mit einem **select**-Befehl, wo viele Zeilen eingefügt werden, nicht befriedigend sein. Bei großen **insert**s können Sie mit dem APPEND-Hint den Ausführungsplan des Optimizers überschreiben und damit die Performance von großen **insert**s verbessern. Der APPEND-Hint weist die Datenbank an, den letzten Datenblock zu finden, in den bereits Daten eingefügt wurden. Der neue Datensatz wird dann am Anfang des unmittelbar darauf folgenden Datenblocks hinterlegt. Da die Daten in neue Blöcke geschrieben werden, fallen für die Datenbank während der Einfügevorgänge sehr viel weniger Verwaltungsarbeiten an. Deshalb kann ein **insert** zusammen mit APPEND schneller ausgeführt werden.

Den APPEND-Hint definieren Sie im **insert**-Befehl. Ein Hint sieht wie ein Kommentar aus – er beginnt mit /* und endet mit */. Der einzige Unterschied ist, dass vor dem Namen des Hints ein Pluszeichen steht. Das folgende Beispiel zeigt einen **insert**-Befehl, bei dem die Daten an die Tabelle angehängt werden:

```
insert /*+ APPEND */ into WORKER (Name)
select Name from PROSPECT;
```

Die Datensätze aus der PROSPECT-Tabelle werden in die WORKER-Tabelle eingefügt. Anstatt zu versuchen, den vorher genutzten Platz innerhalb der WORKER-Tabelle zu belegen, werden die neuen Datensätze am Ende des physischen Speicherbereichs der Tabelle angefügt.

Da die neuen Datensätze nicht versuchen, den verfügbaren Platz innerhalb der Tabelle zu belegen, kann die WORKER-Tabelle durchaus größer werden. Ganz allgemein sollten Sie den APPEND-Hint nur einsetzen, wenn große Datenmengen in Tabellen einzufügen sind, bei denen nur sich wenig Platz wiederverwenden lässt. Den Punkt, an dem die Daten eingefügt werden, bezeichnet man als die *Hochwassermarke* der Tabelle. Diese Hochwassermarke lässt sich nur zurücksetzen, indem man die Inhalte der Tabelle mit **truncate** löscht. Da **truncate** alle Datensätze löscht und kein Rollback mölich ist, sollten Sie vor der Ausführung des Befehls unbedingt prüfen, ob Sie eine aktuelle Datensicherung besitzen. Weitere Einzelheiten zu **truncate** entnehmen Sie bitte der Alphabetischen Referenz.

15.4 rollback, commit und autocommit

Wenn Sie Daten in einer Datenbank eingefügt, aktualisiert oder gelöscht haben, können Sie die Aktionen über ein *Rollback* nachträglich wieder rückgängig machen. Das kann insbesondere bei Fehlern äußerst wichtig sein. Der Prozess für die Speicherung oder das Rollback wird über zwei SQLPLUS-Befehle gesteuert: **commit** und **rollback**. SQLPLUS besitzt die Möglichkeit, Ihre Daten automatisch abzuspeichern, ohne Ihnen darüber explizit eine Meldung zukommen zu lassen. Diese Verfahrensweise wird über das **autocommit**-Merkmal von **set** gesteuert. Alle über **set** gesetzten Merkmale können Sie sich mit **show** anzeigen lassen:

```
show autocommit

autocommit OFF
```

OFF ist die Standardeinstellung. Für den Wert von **autocommit** können Sie eine Zahl vorgeben: Diese Einstellung legt die Anzahl der Befehle fest, nach denen Oracle ein **commit** ausführt. Das bedeutet wiederum, dass Aktionen wie **insert**, **update** und **delete** solange nicht endgültig ausgeführt sind, bis Sie ein **commit** abgesetzt haben:

```
commit;

commit complete
```

Bis zum Absetzen des **commit**s können Sie nachvollziehen, wie sich Ihre Arbeit auf die Tabellen auswirkt. Alle Benutzer, die auf diese Tabelle zugreifen, erhalten weiterhin die alten Informationen. Sie sehen die neuen Informationen, sobald Sie eine **select**-Anweisung ausführen. Ihre Arbeit befindet sich in einer Art „Zwischenbereich", mit dem Sie bis zur Ausführung des **commit**s interagieren. Selbst wenn Sie viele **insert**s, **update**s oder **delete**s abgesetzt haben, können Sie die Aktionen über den folgenden Befehl wieder rückgängig machen:

```
rollback;

rollback beendet
```

Die Meldung „Rollback beendet“ kann dennoch ein wenig irreführend sein. Sie bedeutet lediglich, dass die Aktionen, für die bisher noch kein **commit** abgesetzt wurde, wieder rückgängig gemacht wurden. Wenn Sie eine Reihe von Aktionen mit **commit** oder einem impliziten Befehl bestätigen, bedeutet diese Meldung rein gar nichts. Unter den Überschriften „Die Funktionsweise von commit und rollback“ und „Die Funktionsweise von commit und rollback in SQL“ erfahren Sie anhand eines Beispiels, wie sich diese Befehle auf eine Tabelle auswirken.

15.4.1 Ein implizites commit

Zu den Aktionen, die ohne besondere Anweisung ein implizites **commit** auslösen, gehören **quit**, **exit** (das Äquivalent von **quit**), **create table** oder **create view**, **drop table** oder **drop view**, **grant** oder **revoke**, **connect** oder **disconnect**, **alter**, **audit** und **noaudit**.

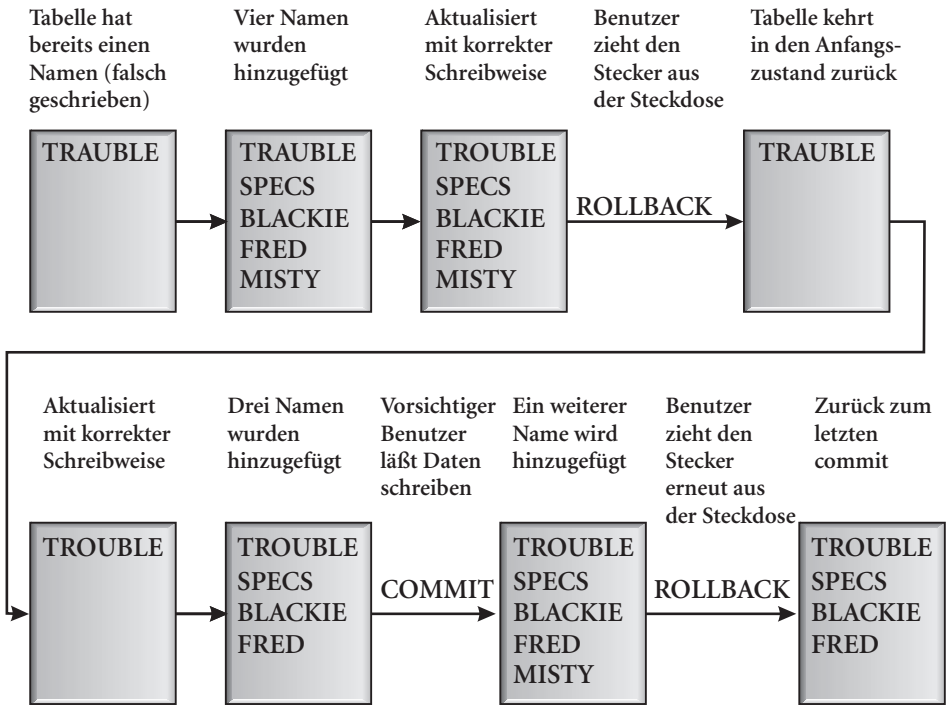
Setzen Sie einen dieser Befehle ab, kommt das einem **commit** gleich.

15.4.2 Auto-Rollback

Wenn es nach der Ausführung verschiedener **insert**s, **update**s oder **delete**s, die weder explizit noch implizit über **commit** bestätigt wurden, größere Probleme gibt (z.B. beim Ausfall eines Rechners), führt Oracle für alle offenen Aktionen ein automatisches Rollback durch. Bricht der Rechner oder die Datenbank zusammen, werden nach dem Neustart der Datenbank alle notwendigen Aufräumarbeiten durchgeführt.

Die Funktionsweise von commit und rollback

Eine Datenbanktabelle besitzt nur eine Spalte und eine Zeile. Die Tabelle heißt CANINE und die Spalte ist Name. Die Tabelle enthält bereits einen Hund namens TROUBLE, der allerdings falsch geschrieben wurde – TRAUBLE. Das Diagramm zeigt die Bemühungen des Hundehalters, andere Namen in die Tabelle einzufügen und den falsch geschriebenen Namen zu ändern. Dabei wird angenommen, dass die **autocommit**-Funktion von SQLPLUS ausgeschaltet wurde. Gleich unter der nächsten Überschrift finden Sie SQL-Anweisungen, die den Aktionen im Diagramm entsprechen:



MISTY muss später, nachdem der Anwender den Rechner wieder ans Stromnetz angeschlossen hat, eingefügt werden.

Die Funktionsweise von commit und rollback in SQL

In SQL sehen die gleichen Sequenzen wie folgt aus:

```
select * from CANINE:

NAME
-------
TRAUBLE

insert into CANINE values ('SPECS');
insert into CANINE values ('BLACKIE');
insert into CANINE values ('FRED');
insert into CANINE values ('MISTY');

update CANINE set Name = 'TROUBLE' where Name = 'TRAUBLE';

rollback;
```

```
select * from CANINE:

NAME
--------
TRAUBLE

update CANINE set Name = 'TROUBLE' where Name = 'TRAUBLE';

insert into CANINE values ('SPECS');
insert into CANINE values ('BLACKIE');
insert into CANINE values ('FRED');

commit;

insert into CANINE values ('MISTY');

rollback;

select * from CANINE:

NAME
-------
TROUBLE
SPECS
BLACKIE
FRED
```

15.5 delete

Um eine oder mehrere Zeilen aus einer Tabelle zu löschen, benötigen Sie den **delete**-Befehl. In der **where**-Klausel definieren Sie zu löschende Zeilen. Ein **delete** ohne **where**-Klausel löscht den gesamten Inhalt einer Tabelle.

```
delete from COMFORT where City = 'WALPOLE';

4 rows deleted.
```

Natürlich kann die **where**-Klausel in einer **delete**-Anweisung genau so viel Logik wie eine **select**-Anweisung, und daneben auch Unterabfragen, Schnitt- und Vereinigungsmengen und einige Dinge mehr enthalten. Über **rollback** können Sie eine falsche Anweisung zwar immer wieder rückgängig machen, doch sollten Sie mit der **select**-Anweisung unbedingt ein wenig experimentieren, bevor Sie damit die Datenbank verändern.

Wenn Sie die Zeilen gelöscht haben, bei denen City = 'WALPOLE' ist, können Sie die Auswirkung von **delete** mit einer einfachen Abfrage testen:

```
select * from COMFORT
 where City = 'WALPOLE';

no rows selected
```

Führen Sie jetzt ein Rollback der **delete**-Anweisung aus und lassen Sie die gleiche Abfrage erneut laufen:

```
rollback;
rollback complete

select * from COMFORT
 where City = 'WALPOLE';

CITY          SAMPLEDAT       NOON   MIDNIGHT PRECIPITATION
------------- --------- ---------- ---------- -------------
WALPOLE       21-MAR-93       56.7       43.8             0
WALPOLE       22-JUN-93       56.7       43.8             0
WALPOLE       23-SEP-93       86.3       72.1
WALPOLE       22-DEC-93       -7.2       -1.2           3.9

4 rows selected.
```

Damit ist klar, dass ein Rollback möglich ist, solange kein **commit** ausgeführt wurde.

Ein weiterer Befehl für das Löschen von Datensätzen ist **truncate**. Diese Anweisung verhält sich allerdings nicht wie **delete**: Während bei **delete** ein Rollback möglich ist, werden mit **truncate** automatisch alle Datensätze aus der Tabelle gelöscht. Weitere Erklärungen zu diesem Befehl finden Sie in der Alphabetischen Referenz.

15.6 update

Bei **update** müssen für jede zu ändernde Tabelle bestimmte Werte angegeben und über eine **where**-Klausel muss definiert werden, welche Zeilen von der Änderung betroffen sind:

```
update COMFORT set Precipitation = .5, Midnight = 73.1
 where City = 'WALPOLE'
   and SampleDate = TO_DATE('22-DEC-1993','DD-MON-YYYY');

1 row updated.
```

Und so sieht das Ergebnis aus:

```
select * from COMFORT
 where City = 'WALPOLE';

CITY          SAMPLEDAT       NOON   MIDNIGHT PRECIPITATION
------------- --------- ---------- ---------- -------------
WALPOLE       21-MAR-93       56.7       43.8             0
WALPOLE       22-JUN-93       56.7       43.8             0
WALPOLE       23-SEP-93       86.3       72.1
WALPOLE       22-DEC-93       -7.2       73.1            .5

4 rows selected.
```

Was wäre, wenn Sie später feststellen, dass das in Walpole verwendete Thermometer die Temperatur immer um ein Grad zu niedrig anzeigt? In einer **update**-Anweisung können Sie Berechnungen, String-Funktionen und viele andere Dinge ausführen und den Wert für die Aktualisierung entsprechend manipulieren (genau wie bei **insert** oder der **where**-Klausel einer **delete**-Anweisung). Hier wird die Temperatur für WALPOLE jeweils um ein Grad erhöht.

```
update COMFORT set Midnight = Midnight + 1, Noon = Noon + 1
 where City = 'WALPOLE';

4 rows updated.
```

Und so sieht das Ergebnis aus:

```
select * from COMFORT
 where City = 'WALPOLE';

CITY          SAMPLEDAT       NOON   MIDNIGHT PRECIPITATION
------------- --------- ---------- ---------- -------------
WALPOLE       21-MAR-93       57.7       44.8             0
WALPOLE       22-JUN-93       57.7       44.8             0
WALPOLE       23-SEP-93       87.3       73.1
WALPOLE       22-DEC-93       -6.2       74.1            .5

4 rows selected.
```

Genau wie bei der **delete**-Anweisung ist auch in diesem Fall die **where**-Klausel das kritische Element. Mit einer falsch aufgebauten **where**-Klausel werden nicht die richtigen Zeilen aktualisiert, was sich manchmal nur sehr schwer feststellen lässt. Das gilt insbesondere dann, wenn die Daten bereits festgeschrieben wurden. Bei einer Aktualisierung sollten Sie deshalb immer mit **set feedback on** arbeiten und das Ergebnis der Transaktion überprüfen. Nach der Ausführung von **update** sollten Sie die Zeilen nochmals abfragen und feststellen, ob die gewünschten Änderungen tatsächlich ausgeführt wurden.

15.6.1 update mit einem eingebetteten select

Um in einer **update**-Anweisung einen Wert zu setzen, binden Sie genau in der Mitte ein **select** ein. Beachten Sie, dass **select** eine eigene **where**-Klausel besitzt, mit der die Temperatur von MANCHESTER aus der WEATHER-Tabelle herausgesucht wird. Auch **update** hat seine **where**-Klausel, mit der nur die Zeile der Stadt WALPOLE für einen bestimmten Tag bearbeitet wird:

```
update COMFORT set Midnight =
       (select Temperature
          from WEATHER
         where City = 'MANCHESTER')
 where City = 'WALPOLE'
   AND SampleDate = TO_DATE('22-DEC-1993','DD-MON-YYYY');

1 row updated.
```

Das Ergebnis dieser **update**-Anweisung stellt sich wie folgt dar:

```
select * from COMFORT
 where City = 'WALPOLE';

CITY          SAMPLEDAT       NOON   MIDNIGHT PRECIPITATION
------------- --------- ---------- ---------- -------------
WALPOLE       21-MAR-93        57.7       44.8             0
WALPOLE       22-JUN-93        57.7       44.8             0
WALPOLE       23-SEP-93        87.3       73.1
WALPOLE       22-DEC-93        -6.2         66            .5

4 rows selected.
```

Wenn innerhalb einer **update**-Anweisung Unterabfragen eingesetzt werden, müssen Sie sicherstellen, dass die Unterabfrage für jeden der aktualisierten Datensätze nicht mehr als einen Datensatz zurückliefert: andernfalls schlägt **update** fehl. In Kapitel 12 finden Sie einige Anmerkungen zu den korrelierenden Abfragen.

Um mehrere Zeilen auf einmal zu aktualisieren, können Sie in **update** auch eine **select**-Anweisung hinterlegen. Die Spalten müssen in Klammern gesetzt und durch Kommas voneinander getrennt werden:

```
update COMFORT set (Noon, Midnight) =
       (select Humidity, Temperature
          from WEATHER
         where City = 'MANCHESTER')
 where City = 'WALPOLE'
   AND SampleDate = TO_DATE('22-DEC-1993','DD-MON-YYYY');

1 row updated.
```

Das Ergebnis:

```
select * from COMFORT
 where City = 'WALPOLE';

CITY          SAMPLEDAT       NOON   MIDNIGHT PRECIPITATION
------------- --------- ---------- ---------- -------------
WALPOLE       21-MAR-93        57.7       44.8             0
WALPOLE       22-JUN-93        57.7       44.8             0
WALPOLE       23-SEP-93        87.3       73.1
WALPOLE       22-DEC-93          98         66            .5

4 rows selected.
```

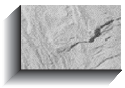
Um eine Reihe von Aktualisierungen zu erzeugen, können Sie SQLPLUS als Codegenerator einsetzen (siehe Kapitel 20).

15.6.2 update mit NULL

Es besteht auch die Möglichkeit, eine Tabelle zu aktualisieren und dabei eine Reihe von Spalten gleich **NULL** zu setzen. Das ist die einzige Instanz, bei der im Zusammenhang mit **NULL** an Stelle des Wortes „ist“ ein Gleichheitszeichen verwendet wird. Ein Beispiel:

```
update COMFORT set Noon = NULL
 where City = 'WALPOLE'
   AND SampleDate = TO_DATE('22-DEC-1993','DD-MON-YYYY');

1 row updated.
```

Damit wird die Mitternachtstemperatur für WALPOLE am 22. Dezember 1993 auf **NULL** gesetzt.



Hinweis:
Bei **insert**, **update** *und* **delete** *spielt der Aufbau der* **where***-Klausel eine wesentliche Rolle. Es ist äußert wichtig, dass Sie vor der Festschreibung der Ergebnisse die notwendige Sorgfalt walten lassen. Diese drei Befehle sind sehr viel leistungsfähiger als eine einfache Abfrage und erlauben die direkte Manipulation der Daten.*