Beide Befehle können für diese Aufgabe verwendet werden. Der Befehl `OPEN CURSOR` ist insbesondere zusammen mit der Option `WITH HOLD` interessant. Er kann die Datenbankverbindung aufrechterhalten, auch wenn nach der Verarbeitung eines Blocks ein DB-Commit abgesetzt wird.

Beispiel

Das Beispiel zeigt die Funktion der beiden Befehle an einem einfachen Fall, in dem nur gelesen wird. Es soll nur die Funktion gezeigt werden, Laufzeitmessungen sind nicht so wichtig.

Der Zusatzbefehl `PACKAGE SIZE` ist sinnvoll, um die Datenmenge in Teilen abzuarbeiten. Der Befehl ist insbesondere dann sinnvoll, wenn viele Änderungen erfolgen sollen.

4.4 Minimale Zahl von Ausführungen

Die Zusammenfassung mehrerer Einzelausführungen wird als Array-Verarbeitung bezeichnet. Diese hat den Vorteil, dass der Overhead pro Ausführung eingespart werden kann.

4.4.1 Vermeiden von unnötigen Ausführungen

Vor der Diskussion der technischen Einschränkungen sollte immer die logische Einschränkung stehen. Es ist nicht sinnvoll, sich genau zu überlegen, wie sich Ausführungen zusammenfassen lassen, wenn sich schließlich herausstellt, dass sie gar nicht notwendig sind. Dabei sind folgende Punkte zu beachten:

- Bei jedem `SELECT`, den Sie programmieren, sollte Ihnen klar sein, wofür die Daten benötigt werden. Daten sollten nicht zu früh auf Vorrat gelesen werden, sondern erst dann, wenn sicher ist, dass sie gebraucht werden.
- Bei wiederholt benötigten Ausführungen sollte über ein Pufferkonzept nachgedacht werden. Bei Tabellenpufferung müssen die Daten so gelesen werden, dass der Puffer genutzt wird. Dazu finden Sie im fünften Kapitel die notwendigen Informationen.
- Unnötige identische Ausführungen, ausgelöst zum Beispiel durch einen `SELECT` in einer Schleife ohne Abhängigkeit von der Schleife, sind zu vermeiden.

Die verbleibenden Ausführungen sind notwendig und unterschiedlich. Sie lassen sich eventuell durch Zusammenfassen noch weiter reduzieren.

4.4.2 SELECT/ENDSELECT oder SELECT INTO TABLE

Es existiert das weit verbreitete Missverständnis, dass diese beiden Befehle die Standardbeispiele für Einzelsatzzugriffe und Array-Zugriff seien. Dabei nutzen beide Befehle die Array-Schnittstelle zur Datenbank.

```
SELECT * INTO ls_dbtab
       FROM dbtab
       WHERE field1 = 'xyz'.
  APPEND ls_dbtab TO lt_dbtab.
ENDSELECT.

SELECT * INTO TABLE lt_dbtab
       FROM dbtab
       WHERE field1 = 'xyz'.
```

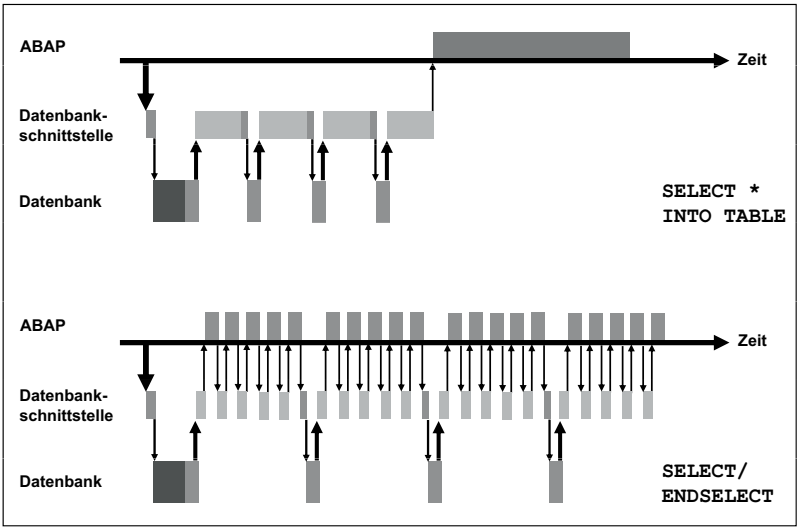

Abb. 4–5 *SELECT/ENDSELECT oder SELECT INTO TABLE*

Der SELECT/ENDSELECT-Befehl ermöglicht es, nach jedem Satz auf dem Applikationsserver eine Verarbeitungslogik einzuschieben. Das Verhalten wird noch klarer, wenn Sie das Beispiel ausführen.

Beispiel

Das vorbereitete Beispiel hat zwei Testfälle. Der erste enthält SELECT/ENDSELECT und einen SELECT INTO TABLE ohne weitere Verarbeitung.

Im zweiten Testfall enthält der SELECT/ENDSELECT etwas Verarbeitungslogik, deshalb folgt dem SELECT INTO TABLE noch eine Verarbeitungsschleife. Schauen Sie sich auf jeden Fall die SQL-Trace an: Dort sehen Sie, dass beide Befehle die FETCH-Operation als Array ausführen.

Das Beispiel zeigt, dass der Unterschied in der Laufzeit zwischen den beiden Befehlen recht gering ist.

Meist ist es empfehlenswert, den Befehl SELECT INTO TABLE zu nutzen. Der Unterschied zum SELECT/ENDSELECT ist jedoch weit geringer als vielfach angenommen, da beide Befehle die Array-Schnittstelle nutzen.

4.4.3 Einzelzugriffe und Array-Zugriffe

Das eigentliche Beispiel für einen häufig ausgeführten Einzelsatzzugriff ist der SELECT in einem LOOP:

```
LOOP AT itab INTO wa.
  SELECT *
         APPENDING TABLE lt_dbtab
         FROM dbtab
         WHERE field1 = wa-field1.
ENDLOOP.
```

Für jeden Satz der Tabelle itab wird eine Verbindung zur Datenbank gemacht und die zugehörenden Sätze von der Datenbank geholt. Es wäre sicherlich besser, in einem Zugriff die Daten für mehrere Sätze von itab zu holen. Dafür gibt es zwei Lösungen: die RANGES-Tabelle und den Befehl FOR ALL ENTRIES.

Auch bei den Änderungsbefehlen, siehe Abschnitt 4.9, spielt die Array-Verarbeitung eine wichtige Rolle.

4.4.4 Die RANGES-Tabelle

Die RANGES-Tabelle ist das programmtechnische Analogon zur Selektionstabelle. Im Abschnitt 4.2.3 wurden die Selektionsoptionen vorgestellt und ihre Möglichkeiten erläutert.

Die Benutzung der RANGES-Tabelle im Programm ist eher ein Nebenprodukt. Aus einer internen Tabelle itab lässt sich sehr einfach eine RANGES-Tabelle aufbauen, die sich in einer SELECT-Anweisung nutzen lässt.

```
DATA: rangetab     TYPE RANGE OF dbtab-field1,
      wa_rangetab  LIKE LINE OF rangetab.

LOOP AT itab INTO wa.
  wa_rangetab-sign   = 'I'.
  wa_rangetab-option = 'EQ'.
  wa_rangetab-low    = wa-field1.
  APPEND wa_rangetab TO rangetab.
ENDLOOP.

SELECT *
       INTO TABLE lt_dbtab
       FROM dbtab
       WHERE field1 IN rangetab.
```

Bei der Ausführung der SELECT-Anweisung werden alle Bedingungen auf einmal an die Datenbank gereicht. Der resultierende Zugriff ist deshalb um einiges schneller als viele Einzelzugriffe. Andererseits wird die Anweisung immer komplexer und kann schließlich so groß werden, dass sie zum Abbruch führt.

Beispiel

Das Beispiel vergleicht die Ausführung einzelner SELECTs in einem LOOP mit einem SELECT und einer RANGES-Tabelle. Es sind zwei Testfälle vorbereitet. Der eine enthält relativ wenige Sätze und sollte auf jedem System laufen. Beim zweiten Testfall werden die internen Tabellen so groß, dass sie irgendwann zu einem Abbruch beim RANGES-Zugriff führen.

Es zeigt sich, dass die RANGES-Tabelle um einiges schneller als die Einzelsatzverarbeitung ist. Der zweite Testfall soll jedoch verdeutlichen, dass es bei großen RANGES-Tabellen zu einem Abbruch kommen kann. Sie sollten eine RANGES-Tabelle deshalb nur nutzen, wenn sicher ist, dass die Tabelle nicht zu groß wird, oder wenn Sie alternativ die Tabelle in kleineren Blöcken verarbeiten. Um die Erzeugung der Blöcke müssen Sie sich jedoch selbst kümmern, der nächste Befehl übernimmt die blockweise Verarbeitung selbst.

Die RANGES-Tabelle wurde zur Verarbeitung der Selektionsoptionen auf den Auswahlschirmen konstruiert. Wenn die Einträge Benutzereingaben sind, ist ihre Zahl meist gering und der Befehl funktioniert gut.

Bei der programmtechnischen Benutzung der RANGES-Anweisung müssen Sie vorsichtig sein. Die RANGES-Tabelle darf nicht zu viele Einträge enthalten, sonst führt sie zu einem Programmabbruch.

4.4.5 SELECT FOR ALL ENTRIES

Eine andere Methode, um mehrere SELECT-Befehle zusammenzufassen, ist der SELECT FOR ALL ENTRIES. Anstelle eines SELECT in einem LOOP können alle Sätze einer Tabelle in einem einzigen SELECT an die Datenbank gereicht werden.

```
SELECT *
       INTO TABLE lt_dbtab
       FROM dbtab
       FOR ALL ENTRIES IN itab
       WHERE field1 = itab-field1
       AND   field2 = itab-field2
       AND   field3 > '100'.
```

Dabei bezeichnet man die Tabelle itab als **Treibertabelle**, um sie von der **Ergebnistabelle** lt_dbtab zu unterscheiden. Es ist sinnvoll, diesen Befehl mit der RANGES-Tabelle zu vergleichen.

Beispiel

Das erste Beispiel vergleicht den Befehl FOR ALL ENTRIES mit der RANGES-Tabelle. Sie sollten sich hier auch die SQL-Trace anschauen.

Offensichtlich greift der FOR ALL ENTRIES-Befehl nicht auf einen Schlag zu, sondern zerlegt die Treibertabelle in Blöcke mit einer bestimmten Anzahl von Sätzen. Die Blockgröße wird durch einen Profilparameter festgelegt und später genauer erklärt. Sie sehen die Größe mit Detailbild der Anweisung. Meist enthalten die Blöcke nur fünf oder zehn Sätze, deshalb ist der FOR ALL ENTRIES auch langsamer als die RANGES-Tabelle. Dafür funktioniert die FOR ALL ENTRIES-Verarbeitung jedoch auch für sehr große Tabellen, es kommt zu keinem Abbruch.

Bedingungen

Bei der Nutzung des FOR ALL ENTRIES sind die drei Bedingungen zu beachten, die alle Folgen der blockweisen Verarbeitung sind:

- Als Sortierung ist nur der ORDER BY PRIMARY KEY erlaubt,
- Aggregatfunktionen sind nicht erlaubt,
- es ist nur eine interne Tabelle möglich.

Empfehlungen

Außer den Bedingungen gibt es noch Empfehlungen:

- Die WHERE-Bedingungen mit den Feldern der Treibertabelle sollten nach Möglichkeit Gleichheitsbedingungen sein. Zwar sind auch andere Operatoren erlaubt, Sie sollten sich jedoch vor der Benutzung genau überlegen, was die OR-Verknüpfung von solchen Bedin-

gungen bedeutet. Insbesondere bei negativen Bedingungen sollten Sie vorsichtig sein.

- Die Kombination von FOR ALL ENTRIES und einer RANGES-Tabelle sollte möglichst vermieden werden. Es ist damit möglich, zwei interne Tabellen in einer SELECT-Anweisung zu nutzen. Durch die Kombination wird die Anweisung aber noch komplizierter und die Gefahr eines Abbruchs steigt. Bei einer Benutzung sollten Sie sich dessen bewusst sein.

Wichtiges zum FOR ALL ENTRIES-Befehl:

Der Befehl ist sehr wichtig, deshalb sollten Sie mit folgenden vier Punkten vertraut sein:

1. Den FOR ALL ENTRIES-Befehl gibt es im SQL-Standard nicht. Er wird immer in eine äquivalente Anweisung auf der Datenbank umgesetzt. Profilparameter steuern die Umsetzung und auch die Blockgröße.
2. Wenn die Treibertabelle keinen Satz enthält, schickt die Datenbankschnittstelle eine SELECT-Anweisung ohne WHERE-Klausel an die Tabelle. Diese liest alle Sätze der Tabelle. Dabei spielt es keine Rolle, ob die WHERE-Klausel noch andere, nicht zum FOR ALL ENTRIES gehörende Bedingungen enthält.
3. Die Blockverarbeitung erfordert, dass das Ergebnis des FOR ALL ENTRIES keine doppelten Ergebnissätze enthalten darf. Dies kann erst am Ergebnis auf dem Applikationsserver sichergestellt werden. Natürlich ist es günstiger, doppelte Einträge gar nicht zu lesen und zu übertragen. Deshalb sollten Sie sicherstellen, dass die Treibertabelle keine doppelten Zeilen enthält.
4. Oft soll das Gesamtergebnis die Felder aus Treibertabelle und Ergebnistabelle in einer Tabelle zusammenfassen. Wie ist dies am effizientesten zu bewerkstelligen?

1. Profilparameter bestimmen die Umsetzung

Der FOR ALL ENTRIES-Befehl wurde zunächst als OR-Verknüpfung von mehreren Bedingungen umgesetzt. Es gibt Datenbankplattformen, die schon bei wenigen mit OR verknüpften Bedingungen in einer Anweisung diese als wenig selektiv betrachten und einen Scan der gesamten Tabelle anstelle eines Indexzugriffs wählen.

Deshalb gibt es mittlerweile eine ganze Reihe von Profilparametern, welche die Umsetzungsart und die Blockgröße steuern. Die genaue und aktuelle Beschreibung der Parameter finden Sie im Hinweis 48230. Die Parameter lauten:

Nr	Parameter	Bedeutung
1	rsdb/prefer_union_all	Umsetzung in UNION-Befehl
2	rsdb/prefer_join	Umsetzung in JOIN-Anweisung (seit SAP NetWeaver Release 7.0 EhP1)
3	rsdb/max_blocking_factor	Standardblockgröße
4	rsdb/prefer_in_itab_opt	Umsetzung in IN-Liste bei einer Bedingung
5	rsdb/max_in_blocking_factor	Blockgröße der IN-Liste
6	rsdb/prefer_fix_blocking	Da die unterschiedlichen Reste unterschiedliche Anweisungen erzeugen, reduziert man die Anweisungen durch das Auffüllen der Rest
7	rsdb/min_blocking_factor	Aufgefüllter Restblock normal
8	rsdb/min_in_blocking_factor	Aufgefüllter Restblock IN-Liste

Tab. 4–8
Profilparameter für den FOR ALL ENTRIES

Die ersten beiden Parameter steuern die Art der Umsetzung. Der erste nutzt den UNION-Befehl anstelle der OR-Bedingungen. Der zweite legt für die interne Tabelle eine temporäre Datenbanktabelle an und führt dann einen Join aus. Sind beide gesetzt, gewinnt der Join-Parameter.

Der dritte Parameter steuert die Blockgröße. Ein großer Wert verbessert natürlich den Effekt der Array-Verarbeitung. Der Wert ist jedoch meist nicht so groß, da viele mit OR verknüpfte Bedingungen bei kleinen und sogar mittelgroßen Tabellen, d.h. bei mehreren tausend Sätzen, als wenig selektiv angesehen werden und zu einem Scan der Tabelle führen können.

Die nächsten beiden Parameter (4, 5) kommen bei Anweisungen zum Tragen, bei denen nur eine Bedingung mit der FOR ALL ENTRIES-Tabelle verknüpft ist. Ist der Parameter rsdb/prefer_in_itab_opt gesetzt, dann wird diese Bedingung in eine IN-Liste umgesetzt. Manche Datenbanken kommen bei einer IN-Liste mit weit größeren Blockgrößen zurecht. Diese Blockgröße bestimmt der fünfte Parameter.

Die letzten drei Parameter (6, 7, 8) sorgen dafür, dass die Blockbildung nicht zu viele verschiedene Anweisungen im Cursor-Cache erzeugt, siehe Abschnitt 3.3. Jeder unterschiedliche Rest bei der Blockbildung ergibt nämlich eine neue Anweisung. Zu einem Block der Größe n gibt es (n-1) verschiedene Reste. Ist der fix_blocking-Parameter gesetzt, dann werden die Reste durch Wiederholung auf die von den anderen beiden Parametern festgelegte Blockgröße aufgefüllt. Die dabei entstehenden mehrfachen Bedingungen ignoriert die Datenbank, siehe Abschnitt 4.3.2.

Mit der Transaktion »Pflege der Profil Parameter (RZ11)« können Sie sich die aktuellen Werte in Ihrem System anschauen. Im Hinweis 48230 finden Sie auch weitere Hinweise, die Ihnen die optimalen Parameterwerte für Ihre Datenbankplattform nennen. Die Parameter sind sehr wichtig, weil sie erklären, warum die gleiche Anweisung auf den verschiedenen Plattformen sehr unterschiedlich umgesetzt wird, was Sie im Datenbank-Explain sehen können.

Beispiel

Das Beispiel enthält zwei `FOR ALL ENTRIES`-Anweisungen, die die gleichen Daten lesen. Einmal werden mehrere Felder von der Treibertabelle übergeben. Das andere Mal wird nur ein Feld übergeben, sodass es in eine `IN`-Liste umgesetzt werden kann. Sehen Sie sich auch die Profilparameter für Ihr System mit der Transaktion RZ11 an und vergleichen Sie die Effekte im Explain.

Wenn Sie die Möglichkeit haben, schauen Sie sich das Beispiel auf verschiedenen Systemen an.

Auf einer Datenbank, die größere Blöcke bei der `IN`-Liste nutzt, sind merkliche Verbesserungen in der Laufzeit möglich.

2. Problem: Leere Treibertabelle

Die problematischste Eigenschaft des `FOR ALL ENTRIES`-Befehls ist sein Verhalten bei leeren Treibertabellen `itab`. Wenn die Treibertabelle leer ist, wird die gesamte `WHERE`-Klausel nicht ausgewertet. Es wird ohne `WHERE`-Klausel selektiert und der komplette Inhalt der Datenbanktabelle zurückgeliefert. Dies ist etwas überraschend. Deshalb sollten Sie sich das folgende Beispiel anschauen:

```
SELECT *
       INTO TABLE lt_dbtab
       FROM dbtab
       FOR ALL ENTRIES IN itab
       WHERE feld1 = itab-feld1
       AND   feld2 = 'a'.
```

Beispiel

Schauen Sie sich das Beispiel zum `FOR ALL ENTRIES`-Befehl mit der leeren Treibertabelle `itab` an.

Das Verhalten rührt daher, dass der `FOR ALL ENTRIES`-Befehl in Analogie zu der `RANGES`-Tabelle konstruiert wurde. In der `RANGES`-Tabelle bedeuten Sätze Einschränkungen, also ist kein Satz auch keine Einschrän-

kung. Aufgrund der Abwärtskompatibilität kann dieses Verhalten des FOR ALL ENTRIES-Befehls nicht mehr verändert werden.

Es ist deshalb sehr wichtig, dass Sie vor jeder Nutzung des FOR ALL ENTRIES-Befehls überprüfen, ob die Treibertabelle nicht leer ist.

3. Problem: Doppelte Einträge

Es wurde bereits erklärt, wie es zu doppelten Einträgen in der Ergebnismenge kommen kann, siehe Abschnitt 4.3.2. Dies hat Auswirkungen für den FOR ALL ENTRIES-Befehl. Angenommen es gibt genau zwei identische Bedingungen in einer Treibertabelle itab:

- Fallen die beiden identischen Bedingungen zufällig in den gleichen Block, dann liefert die Datenbank das zugehörige Ergebnis nur einmal zurück.
- Verteilen sich die beiden identischen Bedingungen jedoch auf unterschiedliche Blöcke, dann kommt mit jedem Block ein Ergebnissatz zurück.

Das Ergebnis des FOR ALL ENTRIES würde im zweiten Fall einen Satz mehr enthalten und damit von der Größe der Blöcke bzw. der zufälligen Sortierung abhängen. Da dies nicht sein darf, muss die Datenbankschnittstelle die doppelten Ergebnissätze beseitigen. Der FOR ALL ENTRIES garantiert, dass das Endergebnis keine doppelten Sätze enthält, der Befehl ist implizit DISTINCT.

Die doppelten Sätze werden jedoch zunächst gesucht und auf den Applikationsserver übertragen, es wird kein DISTINCT auf der Datenbank ausgeführt.

Ursache doppelter Einträge

Doppelte Ergebnissätze kommen fast alle durch doppelte Bedingungen zustande, die sich durch die Reduktion der Treibertabelle vermeiden lassen. Eine stark reduzierte Feldliste, die nicht mehr alle Schlüsselfelder der Tabelle enthält, kann ebenfalls zu doppelten Sätzen führen. Diese Möglichkeit ist aber bei dem FOR ALL ENTRIES selten, da meist ein klarer Bezug zwischen Ergebnissätzen und den Sätzen der Treibertabelle benötigt wird.

Beachten Sie bitte, dass Sie die identischen Bedingungen nur dann in der SQL-Trace sehen, wenn ganze Blöcke komplett identisch sind. Um zu sehen, ob identische Sätze übertragen wurden, müssen Sie die Anzahl der Sätze in der Ergebnistabelle mit der Anzahl der übertragenen Sätze, d. h. dem Wert der Spalte Rec für die Anweisung, vergleichen.

Vermeiden der doppelten Bedingungen

Da doppelte Einträge in der Treibertabelle itab zu identischen Ergebnissätzen führen, sollten Sie diese auf jeden Fall vermeiden, um das zeitaufwendige Suchen und Übertragen von der Datenbank einzu-

sparen. Beachten Sie, dass dabei nicht die ganzen Sätze der Treibertabelle entscheidend sind, sondern nur die Spalten, die in der WHERE-Bedingung der FOR ALL ENTRIES-Anweisung genutzt werden.

Deshalb wird empfohlen, immer wenn Sie vermuten, dass die Treibertabelle doppelte Einträge hat, diese durch folgende Befehlskombination zu beseitigen:

```
IF NOT ( itab IS INITIAL ).
  SORT itab by field3 field5.
  DELETE ADJACENT DUPLICATES itab
         COMPARING field3 field5.
  SELECT …
```

Falls die Treibertabelle itab später noch gebraucht wird, sollte nur eine Kopie der Tabelle verändert werden. Die Kopie sollte möglichst nur die benötigten Felder enthalten. Wenn es nur charakterartige Felder sind, kann anstelle eines Kopierens und Löschens auch der Aufbau mit dem COLLECT-Befehl sinnvoll sein, siehe Abschnitt 6.7.5.

Beispiel

Das Beispiel vergleicht mehrere Ausführungen mit verschiedenen Anteilen von doppelten Einträgen.

Das Beispiel verdeutlicht, dass es sich fast immer lohnt, die doppelten Einträge zu löschen. Die Vorbereitung ist eine Operation auf internen Tabellen und viel schneller als die Datenbankoperation, sodass sich der Zusatzaufwand eigentlich immer rechtfertigen lässt. Auch wenn einmal unnötigerweise sortiert wird, ist dies kein Problem, da der Mehraufwand auf dem Applikationsserver und nicht auf der Datenbank anfällt.

4. Problem: Effizientes Zusammenführen der Ergebnisse

Meist sollen die Daten aus der Treibertabelle und der Ergebnistabelle des FOR ALL ENTRIES am Ende zusammen in einer Tabelle stehen. Dazu müssen die Daten der beiden Tabellen miteinander abgemischt werden, ein geschachtelter LOOP ist unvermeidlich. Geschachtelte LOOPs sind bekannt für ihre Performanceprobleme, jedoch nur, wenn sie nachlässig programmiert sind. Im sechsten Kapitel wird dieser Punkt ausführlich diskutiert. Hier gibt das Beispiel einen kurzen Ausblick.

Beispiel

Das Beispiel vergleicht vier Methoden, um den geschachtelten LOOP über die Ergebnis- und die Treibertabelle des FOR ALL ENTRIES zu bewältigen. Die Operation auf die innere Tabelle kann eine Hash-Tabelle, eine Sortierte Tabelle oder eine Standardtabelle nutzen. Bei der Standardtabelle gibt es noch die Optimierung mit BINARY SEARCH. Vorbereitungskosten wie zum Beispiel Sortieren werden auch berücksichtigt. Es werden zwei Fälle betrachtet:

- mit gleich bleibender Treffermenge (1 : 1)
- mit größer werdender Treffermenge (1 : c)

Es zeigt sich, dass die Lösung mit der Standardtabelle bei großen Tabellen länger dauern kann als das Lesen von der Datenbank. Die anderen drei Lösungen sind um einiges schneller. Vergleichen Sie die Lösung mit der Sortierten Tabelle und dem BINARY SEARCH insbesondere im zweiten Fall. Die Sortierte Tabelle ist weit einfacher zu programmieren. Zur Problematik der effizienten Bearbeitung von internen Tabellen erfahren Sie im sechsten Kapitel mehr.

Ab SAP Web AS Release 6.10 können Sie das Ergebnis in die Treibertabelle schreiben. Beachten Sie jedoch, dass die Tabellen nicht zusammengeführt werden, sondern es wird einfach die Treibertabelle überschrieben.

Der FOR ALL ENTRIES ist hilfreich, wenn Folgendes beachtet wird:

- Wenn die Umsetzung in eine äquivalente Anweisung mit nur einer Bedingung für ein FOR ALL ENTRIES-Feld möglich ist, dann nutzen Sie diese, da sie auf manchen Plattformen als IN-Liste viel schneller verarbeitet werden kann.
- Überprüfen Sie immer vor dem Aufruf des Befehls, ob die Treibertabelle nicht leer ist.
- Falls es wahrscheinlich ist, dass es bei den verwendeten Feldern der Treibertabelle doppelte Zeilen gibt, dann löschen Sie diese oder bilden eine eindeutige Projektionstabelle vor dem Aufruf des FOR ALL ENTRIES.
- Falls Sie die Felder der Treibertabelle und der Ergebnistabelle in einer Gesamttabelle benötigen, dann nutzen Sie bitte eine effiziente Verarbeitung der internen Tabellen, am besten mit einer Sortierten Tabelle oder Hash-Tabelle.

4.5 Minimale Satzbreite

Beim Übertragen des Ergebnisses von der Datenbank zum Applikationsserver spielt die übertragene Datenmenge eine Rolle. Die Datenmenge ist das Produkt aus der Anzahl der Sätze und der Breite der Sätze. Der Einfluss der beiden Faktoren ist nicht ganz gleich, deshalb