IBM solidDB
IBM solidDB Universal Cache

Version 6.5

Speicherinterne Datenbank - Benutzerhandbuch

Information Management software

SC12-4375-00

IBM solidDB
IBM solidDB Universal Cache

Version 6.5

Speicherinterne Datenbank - Benutzerhandbuch

Information Management software

SC12-4375-00

Hinweis

Vor Verwendung dieser Informationen und des darin beschriebenen Produkts sollten die Informationen unter „Bemerkungen" auf Seite 53 gelesen werden.

Diese Ausgabe gilt für Version 6 Release 5 von IBM solidDB (Produktnummer 5724-V17) und IBM solidDB Universal Cache (Produktnummer 5724-W91) und alle nachfolgenden Releases und Modifikationen, bis in einer neuen Ausgabe eine andere andere Aussage gemacht wird.

Diese Veröffentlichung ist eine Übersetzung des Handbuchs
IBM solidDB IBM soldDB Universal Cache Version 6.5, In-Memory Database User Guide
IBM Form SC23-9875-00,
herausgegeben von International Business Machines Corporation, USA

Informationen, die nur für bestimmte Länder Gültigkeit haben und für Deutschland, Österreich und die Schweiz nicht zutreffen, wurden in dieser Veröffentlichung im Originaltext übernommen.

Möglicherweise sind nicht alle in dieser Übersetzung aufgeführten Produkte in Deutschland angekündigt und verfügbar; vor Entscheidungen empfiehlt sich der Kontakt mit der zuständigen IBM Geschäftsstelle.

Änderung des Textes bleibt vorbehalten.

Herausgegeben von:
SW TSC Germany
Kst. 2877
Oktober 2009

Inhaltsverzeichnis

Tabellen

Informationen zu diesem Handbuch

Die speicherinterne IBM® solidDB-Datenbank ermöglicht Ihnen die Auswahl des optimalen Verhältnisses zwischen maximaler Leistung und der Möglichkeit zur Bearbeitung großer Datenvolumen durch die Bereitstellung einer einzigartigen Architektur des Datenbankverwaltungssystems (DBMS) mit zwei Engines. Im Datenbankserver gibt es zwei Engines: eine Hauptspeicher-Engine (Main Memory Engine, MME) für den schnellstmöglichen Zugriff auf leistungskritische Daten und eine herkömmliche, auf einer Festplatte befindliche Engine für die effiziente Bearbeitung praktisch aller Datenvolumen.

Die Hauptspeicher-Engine von solidDB basiert auf der plattenbasierten Engine von solidDB sowie auf Leistungsmerkmalen von solidDB. Daher umfasst die Hauptspeicher-Engine von solidDB auch die gesamte Funktionalität dieser Produkte. Die Hauptspeicher-Engine von solidDB kann in eingebetteten Systemen verwendet werden und erfordert praktisch keine Verwaltung oder Pflege. Sie können die Hauptspeicher-Engine von solidDB für Hochverfügbarkeitssysteme optimieren, indem Sie solidDB als eine Konfiguration mit hoher Verfügbarkeit (High Availability, HA) implementieren. Sie können auch die Komponente 'Advanced Replication' implementieren, mit der mehrere Server mit Hauptspeicher-Engines und plattenbasierten Engines von solidDB Daten gemeinsam nutzen und synchronisieren können.

Dieses Handbuch enthält eine Einführung in die Funktionen, mit denen Sie die Leistung Ihres Datenbankservers mithilfe der speicherinternen Datenbanktechnologie optimieren können.

In diesem Handbuch wird vorausgesetzt, dass der Leser über allgemeine Kenntnisse von Verwaltungssystemen für relationale Datenbanken verfügt und mit SQL vertraut ist. Darüber hinaus wird in dieser Veröffentlichung davon ausgegangen, dass der Leser über grundlegende Kenntnisse der solidDB-Produktfamilie verfügt. Vor dem Lesen des vorliegenden Handbuchs sollten Sie die Veröffentlichung *IBM solidDB Administrator Guide* lesen. Wenn Sie noch nicht mit relationalen Datenbanken vertraut sind, sollten Sie ebenfalls zuerst die Veröffentlichungen *IBM solidDB Einführung* und *IBM solidDB SQL Guide* lesen.

Typografische Konventionen

In der solidDB-Dokumentation werden die folgenden typografischen Konventionen verwendet:

Tabelle 1. Typografische Konventionen

Format	Verwendungszweck
Datenbanktabelle	Diese Schriftart wird für normalen Text verwendet.
NOT NULL	Großbuchstaben in dieser Schriftart geben SQL-Schlüsselwörter und Makronamen an.
`solid.ini`	Diese Schriftart gibt Dateinamen und Pfadausdrücke an.
`SET SYNC MASTER YES;` `COMMIT WORK;`	Diese Schriftart wird für Programmcode und die Programmausgabe verwendet. Außerdem wird diese Schriftart für SQL-Beispielanweisungen verwendet.
run.sh	Diese Schriftart wird für Beispielbefehlszeilen verwendet.
TRIG_COUNT()	Diese Schriftart wird für Funktionsnamen verwendet.
java.sql.Connection	Diese Schriftart wird für Schnittstellennamen verwendet.
LockHashSize	Diese Schriftart wird für Parameternamen, Funktionsargumente und Einträge in der Windows®-Registrierungsdatenbank verwendet.
Argument	Wörter, die auf diese Weise hervorgehoben sind, stehen für Informationen, die vom Benutzer oder der Anwendung angeben werden müssen.
Administrator Guide	Diese Darstellung wird für Verweise auf andere Dokumente oder auf Kapitel im vorliegenden Dokument verwendet. Außerdem werden auch neue Begriffe und hervorgehobene Aspekte auf diese Weise geschrieben.
Darstellung von Dateipfaden	Sofern nicht anders angegeben, werden Dateipfade im UNIX®-Format dargestellt. Der Schrägstrich (/) stellt das Installationsstammverzeichnis dar.
Betriebssysteme	Wenn die Dokumentation Unterschiede zwischen den Betriebssystemen enthält, wird das UNIX-Format zuerst genannt. Das Microsoft® Windows-Format wird in runden Klammern nach dem UNIX- Format genannt. Weitere Betriebssysteme werden separat erwähnt. Es kann auch verschiedene Kapitel für verschiedene Betriebssysteme geben.

Konventionen für Syntaxdiagramme

In der solidDB-Dokumentation werden für Syntaxdiagramme die folgenden Konventionen verwendet:

Tabelle 2. Konventionen für Syntaxdiagramme

Format	Verwendungszweck
`INSERT INTO` *`Tabellenname`*	Für Syntaxbeschreibungen wird diese Schriftart verwendet. Für austauschbare Abschnitte wird *diese* Schriftart verwendet.
`solid.ini`	Diese Schriftart gibt Dateinamen und Pfadausdrücke an.
[ ]	Eckige Klammern geben optionale Elemente an. Werden die eckigen Klammern in Fettdruck dargestellt, müssen sie in der Syntax angegeben werden.
|	Ein vertikaler Balken trennt zwei sich gegenseitig ausschließende Auswahlmöglichkeiten in einer Syntaxzeile.
{ }	Geschweifte Klammern begrenzen eine Gruppe sich gegenseitig ausschließender Auswahlmöglichkeiten in einer Syntaxzeile. Werden die geschweiften Klammern in Fettdruck dargestellt, müssen sie in der Syntax angegeben werden.
...	Eine Auslassung gibt an, dass Argumente mehrmals wiederholt werden können.
. . .	Eine Spalte mit drei Punkten gibt an, dass die vorherigen Codezeilen fortgesetzt werden.

1 Grundlegende Funktionen

Die Hauptspeicher-Engine von IBM solidDB verbindet die hohe Leistung von speicherinternen Tabellen mit der nahezu unbegrenzten Kapazität von plattenbasierten Tabellen. Diesen Vorteil können andere Lösungen auf dem Markt nicht bieten. Reine speicherinterne Datenbanken sind zwar schnell, aber durch die Speichergröße streng begrenzt. Reine plattenbasierte Datenbanken bieten nahezu unbegrenzte Speichermengen, aber ihre Leistung wird durch den Plattenzugriff bestimmt. Auch wenn der Speicher des Computers für die Speicherung der gesamten Datenbank in Hauptspeicherpuffern ausreicht, sind für plattenbasierte Tabellen konzipierte Datenbankserver langsam, da die für plattenbasierte Tabellen optimal ausgelegten Datenstrukturen für speicherinterne Tabellen bei weitem nicht optimal sind.

Die Lösung von solidDB besteht aus einem einzigen Datenbankserver, der zwei optimierte Server enthält: ein Server ist für den plattenbasierten Zugriff optimiert und der andere Server ist für den speicherinternen Zugriff optimiert. Beide Server koexistieren in demselben Prozess und eine einzige SQL-Anweisung kann auf die Daten von beiden Engines zugreifen.

Speicherinterne Tabellen

Speicherinterne und plattenbasierte Tabellen

Wenn eine Tabelle als speicherinterne Tabelle definiert ist, wird der gesamte Inhalt dieser Tabelle im Speicher gespeichert, sodass der Datenzugriff so schnell wie möglich erfolgen kann. Bei einer plattenbasierten Tabelle werden die Daten dagegen vorwiegend auf der Platte gespeichert und der Server kopiert jeweils nur einen kleinen Teil der Daten in den Speicher.

Speicherinterne Tabellen sind plattenbasierten Tabellen in vielerlei Hinsicht ähnlich. Der wichtigste Punkt hierbei ist, dass beide Tabellentypen vollständige Datenpersistenz bieten, sofern nicht anders angegeben. Sie können für beide Tabellentypen dieselben Abfragetypen durchführen. Sie können plattenbasierte und speicherinterne Tabellen auch in derselben SQL-Abfrage oder -Transaktion kombinieren. Unter Verwendung der SA API können Sie Abfragen für beide Tabellentypen durchführen. Darüber hinaus können speicherinterne Tabellen u. a. mit Indizes, Triggern, gespeicherten Prozeduren usw. verwendet werden. Speicherinterne Tabellen lassen auch Integritätsbedingungen zu, einschließlich Integritätsbedingungen über Primärschlüssel und Integritätsbedingungen über Fremdschlüssel. Bei nicht persistenten Tabellen gibt es jedoch einige Einschränkungen für Integritätsbedingungen über Fremdschlüssel.

Mit solidDB können Sie festlegen, welche Tabellen speicherinterne Tabellen und welche Tabellen plattenbasierte Tabellen sein sollen. Beispielsweise können Sie häufig verwendete Tabellen in den Hauptspeicher stellen, sodass schneller auf sie zugegriffen werden kann. Wenn Sie über ausreichend Speicher verfügen, können Sie auch Ihre gesamten Tabellen in den Hauptspeicher stellen.

Typen speicherinterner Tabellen

solidDB bietet zwei verschiedene Kategorien von speicherinternen Tabellen: persistente und nicht persistente Tabellen. Diese werden im Folgenden beschrieben.

Persistente speicherinterne Tabellen

Persistente speicherinterne Tabellen bleiben auf unbestimmte Zeit bestehen. Obwohl Clientabfragen auf die Kopie der Daten im Speicher zugreifen, speichert der Server die speicherinternen Tabellen, wenn er heruntergefahren wird, auf der Platte. Daher sind diese Daten nach jedem Serverstart verfügbar. Speicherinterne Tabellen verwenden auch die Transaktionsprotokollierung, sodass der Server, falls er aus einem unerwarteten Grund (beispielsweise ein Stromausfall) beendet wird, über einen Datensatz der stattgefundenen Transaktionen verfügt und die Tabellen aktualisieren kann, um sicherzustellen, dass sie alle Daten aller festgeschriebenen Transaktionen enthalten. Die Daten speicherinterner Tabellen werden wie auch bei plattenbasierten Tabellen während der Prüfpunkte auf das Festplattenlaufwerk kopiert. (Eine Beschreibung der Prüfpunkte finden Sie in der Veröffentlichung *IBM solidDB Administrator Guide*).

Speicherinterne Tabellen können auch mit der Komponente 'HotStandby' von solidDB verwendet werden. Die Daten in speicherinternen Tabellen werden auf den sekundären Server kopiert, wo sie bei einem Ausfall des primären Servers verfügbar sind.

Zwischen speicherinternen und auf einer Festplatte befindlichen Tabellen gibt es einige Unterschiede, die beim Entwurf einer Anwendung, die speicherinterne Tabellen verwendet, berücksichtigt werden müssen. Der wichtigste Unterschied besteht darin, dass speicherinterne Tabellen immer eine pessimistische Steuerung des gemeinsamen Zugriffs (Sperren) auf Zeilenebene verwenden und plattenbasierte Tabellen standardmäßig eine optimistische Steuerung des gemeinsamen Zugriffs (Versionssteuerung) verwenden. Aus diesem Grund blockieren Leseoperationen in speicherinternen Tabellen Schreiboperationen für die Dauer der Lesetransaktion. Darüber hinaus sind bei speicherinternen Tabellen Deadlocks möglich. Diese können jedoch nicht bei der Versionssteuerung von plattenbasierten Tabelle auftreten. Andererseits können Konflikte beim gemeinsamen Zugriff auftreten, wenn die Überprüfung auf gleichzeitige optimistische Schreib- und Lesezugriffe verwendet wird.

Beachten Sie diese Punkte unbedingt beim Entwurf der Fehlerbehandlung ihrer Anwendung. Abhängig vom verwendeten Tabellentyp muss die Fehlerbehandlung unterschiedliche Fehlercodes berücksichtigen.

Ein weiterer wichtiger Unterschied ist der verwendete Algorithmus für das Prüfpunktverfahren. Das Prüfpunktverfahren für speicherinterne Tabellen unterscheidet sich grundlegend von dem für plattenbasierte Tabellen verwendeten Algorithmus. Der wichtigste Vorteil besteht darin, dass das Prüfpunktverfahren für speicherinterne Tabellen den Zugriff der Transaktionen auf die Tabellen während des Prüfpunkts in keiner Weise blockiert. Daher können die Antwortzeiten bei speicherinternen Tabellen besser vorhergesagt werden als bei plattenbasierten Tabellen.

Der dritte wesentliche Unterschied zwischen speicherinternen und plattenbasieren Tabellen liegt darin, dass die Sekundärindizes bei speicherinternen Tabellen nie auf die Platte geschrieben werden. Stattdessen werden sie speicherintern gehalten und beim Serverstart erneut erstellt. Daher beeinträchtigen sie die Schreibleistung wesentlich weniger als bei plattenbasierten Tabellen. Zudem weisen alle Indizes von speicherinternen Tabellen dieselbe Geschwindigkeit auf; bei plattenbasierten Tabellen ist der Primärschlüssel jedoch wesentlich schneller als die anderen Indizes.

Bezüglich aller anderen Aspekte unterscheiden sich speicherinterne Tabellen praktisch nicht von plattenbasierten Tabellen, außer, dass speicherinterne Tabellen generell wesentlich schneller sind.

Nicht persistente speicherinterne Tabellen

Nicht persistente speicherinterne Tabellen werden beim Herunterfahren des Servers nicht auf die Platte geschrieben. Daher gehen die Daten in nicht persistenten Tabellen bei jedem Herunterfahren des Servers verloren, unabhängig davon ob der Server normal oder abnormal beendet wurde. Ihre Daten werden nicht protokolliert und auch das Prüfpunktverfahren wird nicht angewendet. Hierdurch können sie zwar nicht wiederhergestellt werden, sind aber wesentlich schneller als persistente Tabellen.

Nicht persistente Tabellen sind vorwiegend als Arbeitspuffertabellen hilfreich. Sie können beispielsweise Daten aus einer persistenten Tabelle kopieren, dann eine Reihe von Analysen ausführen und anschließend die nicht persistente Kopie löschen. Sie können auch zahlreiche Umsetzungen durchführen und dann die Endergebnisse zurück in persistente Tabellen kopieren.

Es gibt zwei verschiedene Typen nicht persistenter speicherinterner Tabellen: transiente Tabellen und temporäre Tabellen. Die Unterschiede zwischen den beiden Typen werden ausführlich in 2, „Temporäre Tabellen und transiente Tabellen", auf Seite 15 beschrieben. An dieser Stelle wird lediglich ein kurzer Überblick gegeben.

Transiente Tabellen

Transiente Tabellen bleiben bis zum Herunterfahren des Servers bestehen. Mehrere Benutzer können dieselbe transiente Tabelle verwenden und jeder Benutzer kann die Daten aller anderen Benutzer anzeigen.

Für transiente Tabellen gelten im Gegensatz zu persistenten speicherinternen Tabellen jedoch einige Einschränkungen. So gibt es bei transienten Tabellen beispielsweise einige Einschränkungen bei der Verwendung von Fremdschlüsseln (referenzielle Integritätsbedingungen). Darüber hinaus können transiente Tabellen nicht auf einen sekundären HotStandby-Server kopiert werden.

Temporäre Tabellen

Die Daten in temporären Tabellen werden nur der Verbindung angezeigt, die die Daten eingefügt hat. Darüber hinaus werden die Daten lediglich für die Dauer der Verbindung beibehalten. Temporäre Tabellen sind wie persönliche Arbeitspuffer, die kein anderer Benutzer sehen kann.

Sie führen keine Protokollierung durch und verwenden keinerlei Mechanismus für die Steuerung des gemeinsamen Zugriffs (wie Eintragssperren), wodurch sie sogar noch schneller als transiente Tabellen sind.

Leistungsverbesserung mit speicherinternen Tabellen

Wenn Daten in einer plattenbasierten Tabelle gespeichert werden, müssen sie in den Speicher gelesen werden, damit sie verwendet werden können. Nachdem sie verwendet wurden, müssen die Daten dann wieder zurück auf die Platte geschrieben werden. Speicherinterne Tabellen bieten einen höhere Leistung, da alle Daten immer im Hauptspeicher gehalten werden und der Server daher effizientere Verfahren nutzen kann, um die maximale Leistung für den Datenzugriff und die Datenbearbeitung bereitzustellen.

Fast alle Datenbankserver bieten eine schnellere Ausführung, wenn sie über mehr Speicher verfügen und einen höheren Prozentsatz ihrer Daten im Cache des Servers speichern können. Die leistungsfähige speicherinterne Technologie der Hauptspeicher-Engine von solidDB tut jedoch wesentlich mehr als nur Daten in den Speicher zu kopieren. Die Hauptspeicher-Engine von solidDB verwendet auch Indexstrukturen, die für die Arbeit mit Daten optimiert sind, die vollständig im Speicher gespeichert werden. Die Hauptspeicher-Engine von solidDB berücksichtigt auch Probleme, die bei speicherinternen Tabellen auftreten können, wie Speicherfragmentierung bei der Vergrößerung oder Verkleinerung von Tabellen.

Vorgehensweise beim Definieren von Tabellen als speicherinterne Tabellenn

Idealerweise verfügt Ihr Computer über ausreichend Speicher, damit all Ihre Tabellen im Speicher gespeichert werden können, und ermöglicht so die bestmögliche Leistung für Datenbanktransaktionen. Unter realen Bedingungen müssen die meisten Benutzer jedoch eine Untergruppe der Tabellen auswählen, die im Speicher gespeichert werden sollen, während die restlichen Tabellen plattenbasiert sind.

Wenn nicht alle Tabellen in den Speicher passen, wollen Sie sicher die am häufigsten genutzten Daten in den Speicher stellen. Aus verständlichen Gründen sollten kleine, häufig verwendete Tabellen in den Speicher gestellt werden, während große, selten verwendete Tabellen auf der Platte bleiben können. Aber was empfiehlt sich bei den anderen möglichen Kombinationen, beispielsweise bei großen, häufig verwendeten Tabellen oder kleinen, selten verwendeten Tabellen?

Am besten sollte hier die Dichte des Zugriffs auf eine Tabelle berücksichtigt werden. Je höher die Anzahl der Zugriffe pro Megabyte pro Sekunde, desto besser ist es, diese Tabelle in den Speicher zu stellen. Einen ausführlicheren Algorithmus finden Sie in „Algorithmus für Auswahl der Tabellen, die im Speicher gespeichert werden sollen” auf Seite 21.

Wenn eine Tabelle im Speicher gespeichert werden soll, müssen Sie auswählen, ob die Daten in einer persistenten Tabelle, in einer transienten Tabelle oder in einer temporären Tabelle gespeichert werden sollen. Die grundlegenden Regeln hierzu finden Sie weiter unten. Diese Regeln sind lediglich Richtlinien, jedoch keine strengen Regeln. Vor einer endgültigen Entscheidung sollten Sie die ausführlichen Beschreibungen der transienten Tabellen und der temporären Tabellen lesen.

Bitte beachten Sie, dass der Begriff ″Serversitzung″ eine einzelne ″Ausführung″ des Servers bedeutet, vom Zeitpunkt seines Starts bis zu seinem beabsichtigten Herunterfahren bzw. bis zu seiner Beendigung aus einem unerwarteten Grund (beispielsweise ein Stromausfall). Eine ″Verbindung″ besteht ab dem Zeitpunkt, zu dem ein einzelner Benutzer die Verbindung zum Server herstellt, bis zu dem Zeitpunkt, zu dem dieser Benutzer diese Verbindung trennt. (Ein Benutzer kann mehrere Verbindungen herstellen, die aber jeweils unabhängig sind.)

Sie können den geeignetsten Tabellentyp bestimmen, indem Sie sich die folgenden Fragen stellen, bis Sie eine Frage mit ″Ja″ beantworten können.

1. Müssen die Daten beim nächsten Serverstart wieder verfügbar sein? Ist dies der Fall, verwenden Sie eine persistente Tabelle.
2. Müssen die Daten auf den sekundären HotStandby-Server kopiert werden? Ist dies der Fall, verwenden Sie eine persistente Tabelle.

3. Brauchen Sie die Daten nur während der aktuellen Serversitzung, aber die Daten müssen mehreren Benutzern zur Verfügung stehen (oder mehreren Verbindungen desselben Benutzers)? Ist dies der Fall, verwenden Sie eine transiente Tabelle.
4. Wenn keine der obigen Regeln gilt, verwenden Sie eine temporäre Tabelle.

Vor einer endgültigen Entscheidung sollten Sie unbedingt die ausführlichen Informationen zu temporären und transienten Tabellen lesen. Für temporäre und transiente Tabellen gelten einige Einschränkungen wie Grenzwerte für Fremdschlüssel (referenzielle Integritätsbedingungen), die sich auf Ihre Entscheidung auswirken können.

Vorgehensweise beim Angeben, dass eine Tabelle im Speicher gespeichert werden soll

Sie können auf zwei Arten explizit angeben, ob Tabellen speicherintern oder auf der Platte gespeichert werden sollen. Das System erstellt standardmäßig plattenbasierte Tabellen.

1. Verwenden Sie die Klausel STORE des Befehls CREATE TABLE oder ALTER TABLE. Beispiel:

   ```
   CREATE TABLE Mitarbeiter (name CHAR(20)) STORE MEMORY;
   CREATE TABLE ... STORE DISK;
   ALTER TABLE Netzwerkadressen SET STORE MEMORY;
   ```

   (Weitere Informationen zur Syntax der Anweisungen CREATE TABLE und ALTER TABLE finden Sie in der Veröffentlichung *IBM solidDB SQL Guide*.)
2. Geben Sie in der Datei `solid.ini` den Standardwert an. Wenn die meisten oder alle von Ihnen erstellten neuen Tabellen den gleichen Typ (z. B. speicherintern) haben sollen, können Sie der Einfachheit halber den Server anweisen, diesen Speichertyp automatisch so lange zu verwenden, bis eine andere Angabe gemacht wird. Wenn Sie den Standardtyp für neue Tabellen angeben wollen, setzen Sie den folgenden Parameter in der Konfigurationsdatei `solid.ini`:

   ```
   [General]
   DefaultStoreIsMemory=yes
   ```

   Wenn dieser Parameter auf 'yes' gesetzt ist, werden neue Tabellen so lange als speicherinterne Tabellen erstellt, bis in der Anweisung CREATE TABLE eine andere Angabe gemacht wird. Wenn dieser Parameter auf 'no' gesetzt ist, werden neue Tabellen so lange als plattenbasierte Tabellen erstellt, bis in der Anweisung CREATE TABLE eine andere Angabe gemacht wird. Wenn dieser Parameter geändert wird, wird - wie auch bei anderen Parametern in der Datei `solid.ini` - der neue Wert erst beim nächsten Start des Servers wirksam. Weitere Informationen zum Parameter **DefaultStoreIsMemory** finden Sie in der Veröffentlichung *IBM solidDB Administrator Guide*.

Anmerkung: Diese Anweisungen gelten ausschließlich für persistente Tabellen. Als temporäre Tabellen oder transiente Tabellen definierte Tabellen werden selbst dann automatisch im Speicher gespeichert, wenn Sie die Klausel STORE MEMORY nicht verwenden.

Speicherbelegung

Das Verstehen und Steuern der Speicherbelegung ist wichtig, da Sie keine Daten mehr hinzufügen oder aktualisieren können, wenn die speicherinterne Datenbank oder der Serverprozess den gesamten verfügbaren virtuellen Speicher im System belegt. Wenn der Server den gesamten physischen Hauptspeicher belegt und beginnt, den virtuellen Speicher zu verwenden, ist der Server zwar weiterhin aktiv, aber die Leistung wird erheblich verringert.

Die Hauptspeicherbelegung der speicherinternen Datenbank unterscheidet sich erheblich vom solidDB-Standardprodukt. Die speicherinterne Datenbank befindet sich in ihrem eigenen Hauptspeicherpool. Informationen zur Speicherbelegung von solidDB finden Sie in der Veröffentlichung *IBM solidDB Administrator Guide*.

Die Hauptspeicher-Engine von solidDB stellt Befehle und Konfigurationsparameter bereit, die Sie bei der Überwachung und Steuerung der Speicherbelegung der speicherinternen Datenbank und des Serverprozesses unterstützen. Diese Befehle und Parameter sind auf die Funktion der speicherinternen Datenbank des Servers ausgerichtet und nicht auf den Server als Ganzes.

Speicherbelegung überwachen

Für die Überwachung der Speicherbelegung stehen mehrere ADMIN COMMAND-Befehle zur Verfügung. Diese sind:

- ADMIN COMMAND 'info imdbsize';
- ADMIN COMMAND 'info processsize';
- ADMIN COMMAND 'pmon mme';
- ADMIN COMMAND 'memory';

Diese Befehle werden im Folgenden erläutert.

Der Befehl

ADMIN COMMAND 'info imdbsize';

gibt die aktuelle Speichermenge zurück, die zur Verwendung durch speicherinterne Datenbanktabellen und Indizes zugeordnet wurde. Der zurückgegebene Wert ist ein VARCHAR-Wert und gibt die Anzahl der vom Server verwendeten Kilobyte an. Beachten Sie, dass dadurch die Menge des belegten virtuellen Speichers zurückgegeben wird, nicht die Menge des belegten physischen Speichers.

Im Laufe der Zeit kann der durch imdbsize zurückgegebene Wert größer werden, da die Rückgabe von Daten an das Betriebssystem nur in Zuordnungseinheiten erfolgen kann, die vollständig leer sein müssen, damit sie Daten für die Rückgabe an das Betriebssystem aufnehmen können.

Transiente Speicherzuordnungen (wie SQL-Ausführungsdiagramme) sind in dem Bericht von ADMIN COMMAND 'info imdbsize'; nicht enthalten.

Der Befehl

ADMIN COMMAND 'info processsize';

gibt die Prozessgröße des virtuellen Speichers zurück, also die Größe des gesamten Adressraums des Datenbankservers, der von dem speicherinternen Datenbankprozess verwendet wird. Der zurückgegebene Wert ist ein VARCHAR-Wert und gibt die Anzahl der vom Prozess verwendeten Kilobyte an. Beachten Sie, dass dadurch die Menge des belegten virtuellen Speichers zurückgegeben wird, nicht die Menge des belegten physischen Speichers.

Es stehen auch mehrere Leistungsdaten zur Verfügung, z. B. die Laufzeitinformationen für den speicherinternen Datenbankserver. Durch die Eingabe des Befehls

ADMIN COMMAND 'pmon mme';

wird die folgende Liste mit aktuellen Werten der Leistungsdaten erstellt.

```
RC TEXT
-- ----
 0 Performance statistics:
 0 Time (sec)                              30    21    Total
 0 MME current number of locks       :     0     0        0
 0 MME maximum number of locks       :     0     0        0
 0 MME current number of lock chains :     0     0        0
 0 MME maximum number of lock chains :     0     0        0
 0 MME longest lock chain path       :     0     0        0
 0 MME memory used by tuples         :     0     0        0
 0 MME memory used by indexes        :     0     0        0
 0 MME memory used by page structures:     0     0        0
10 rows fetched.
```

In der Liste mit der Leistungsstatistik wird die von Tupeln, Indizes und Seitenstrukturen verwendete Speichermenge in Kilobyte angegeben.

Der Befehl

ADMIN COMMAND 'memory';

listet lediglich die Menge des dynamisch zugeordneten Zwischenspeichers auf. Bei der zwischenspeicherbasierten Speicherzuordnung wird der Speicher aus einem großen Pool ungenutzten Hauptspeicherbereichs, Zwischenspeicher genannt, zugeordnet. Die Größe der Zwischenspeicherzuordnung kann während der Ausführung ermittelt werden. Transiente Speicherzuordnungen (wie SQL-Ausführungsdiagramme) sind im Bericht von ADMIN COMMAND 'mem'; enthalten.

Speicherbelegung steuern

Die Speicherbelegung der speicherinternen Datenbank wird durch die folgenden drei Konfigurationsparameter im Abschnitt `[MME]` der Datei `solid.ini` gesteuert:

- **ImdbMemoryLimit**
- **ImdbMemoryLowPercentage**
- **ImdbMemoryWarningPercentage**

Zusätzlich wird die *Prozess*speicherbelegung durch die folgenden vier Konfigurationsparameter im Abschnitt `[SRV]` der Datei `solid.ini` gesteuert:

- **ProcessMemoryLimit**
- **ProcessMemoryLowPercentage**
- **ProcessMemoryWarningPercentage**
- **ProcessMemoryCheckInterval**

Die Nichteinhaltung der speicherinternen Datenbankspeicher- und der Verarbeitungsgrenzwerte wird in der Protokolldatei `solmsg.out` protokolliert. Bei jeder Überschreitung des Speichergrenzwerts, der durch die Parameter **ImdbMemoryLimit** und **ProcessMemoryLimit** definiert ist, wird ein Systemereignis gesendet. Diese Systemereignisse werden in der Veröffentlichung *IBM solidDB SQL Guide* beschrieben.

ImdbMemoryLimit

Der Parameter **MME.ImdbMemoryLimit** gibt den Maximalwert des virtuellen Speichers an, der speicherinternen Tabellen (einschließlich temporären Tabellen, transienten Tabellen und ″normalen″ speicherinternen Tabellen) sowie den Indizes für diese speicherinternen Tabellen zugeordnet werden kann.

Der Standardwert für **ImdbMemoryLimit** ist 0, das bedeutet ″kein Grenzwert″. Es wird jedoch dringend von der Verwendung des Standardwerts abgeraten. Sie sollten den Parameter auf einen Wert setzen, der sicherstellt, dass die speicherinternen Daten vollständig in den physischen Hauptspeicher passen. In der Regel müssen Sie die folgenden Faktoren berücksichtigen:

- Größe des physischen Hauptspeichers im Computer
- Vom Betriebssystem belegter Speicher
- Von solidDB (dem eigentlichen Programm) belegter Speicher
- Für den Cache des solidDB-Servers vorgesehene Speichermenge (Konfigurationsparameter **IndexFile.CacheSize** in der Datei `solid.ini`)
- Für gleichzeitig auf dem Server ablaufende Verbindungen, Transaktionen und Anweisungen erforderliche Speichermenge. Je mehr gleichzeitig bestehende Verbindungen und aktiven Anweisungen auf dem Server vorhanden sind, desto mehr Arbeitsspeicher ist für den Server erforderlich. In der Regel müssen Sie für jede Clientverbindung im Server 0,5 MB zuordnen.
- Von anderen Prozessen (Programmen und Daten), die auf dem Computer ausgeführt werden, verwendeter Speicher

Wenn der Grenzwert erreicht ist (d. h., wenn die speicherinternen Tabellen den von **ImdbMemoryLimit** angegebenen Speicher vollständig belegen, verhindert der Server Aktualisierungen (UPDATE-Operationen) für speicherinterne Tabellen. Bevor der Grenzwert erreicht wird, verhindert der Server die Erstellung neuer speicherinterner Tabellen und Einfügungen (INSERT-Operationen) für diese Tabellen. In der Beschreibung des Parameters **ImdbMemoryLowPercentage** finden Sie ausführlichere Informationen hierzu.

Beispiel:

```
[MME]
ImdbMemoryLimit=1000MB
```

ImdbMemoryLowPercentage

Die Variable **ImdbMemoryLowPercentage** setzt einen unteren Grenzwert, der als Prozentsatz des Speichers ausgedrückt wird (d. h. als Prozentsatz der Variablen **ImdbMemoryLimit**). Wenn der Server den angegebenen Prozentsatz des Speichers belegt hat, beginnt er, Aktivitäten einzuschränken, damit kein weiterer Speicher mehr belegt wird. Wenn beispielsweise **ImdbMemoryLimit** 1000 MB ist, **ImdbMemoryLowPercentage** 90 % ist und wenn der den speicherinternen Tabellen zugeordnete Speicher 900 MB übersteigt, beginnt der Server, Aktivitäten einzuschränken. Der Server wird insbesondere Folgendes tun:

- Verhindert die weitere Erstellung von speicherinternen Tabellen (einschließlich temporärer und transienter Tabellen) sowie Indizes für speicherinterne Tabellen.
- Verhindert Einfügungen (INSERT-Operationen) in speicherinterne Tabellen.

Wenn die Obergrenze (d. h. **ImdbMemoryLimit**) erreicht ist, verhindert der Server auch Aktualisierungen (UPDATE-Operationen) für Datensätze in speicherinternen Tabellen.

Die gültigen Werte für **ImdbMemoryLowPercentage** liegen im Bereich von 60 bis 99 (Prozent).

ImdbMemoryWarningPercentage

Der Parameter ″ImdbMemoryWarningPercentage″ setzt einen Warngrenzwert für die IMDB-Speicherkapazität. Der Warngrenzwert wird als Prozentsatz des

Parameterwerts von ″ImdbMemoryLimit″ ausgedrückt. Wenn der Grenzwert von ″ImdbMemoryWarningPercentage″ überschritten wird, wird ein Systemereignis ausgegeben.

Maßnahmen beim Erreichen des Werts von ″ImdbMemoryLimit″

Wenn eine Fehlernachricht angezeigt wird, die angibt, dass dieser Grenzwert erreicht wurde, müssen Sie sofort wirksame Maßnahmen ergreifen. Sie müssen sowohl die dringenden Probleme als auch das langfristige Problem lösen. Die dringenden Probleme bestehen darin, dass das Auftreten schwerwiegender Fehler bei Benutzern verhindert und Speicher freigegeben werden muss, bevor Sie den Server herunterfahren, damit beim Neustart des Servers nicht dieselbe Situation (kein ausreichender Speicher) auftritt. Das langfristige Problem besteht darin, dass sichergestellt werden muss, dass diese Situation künftig bei der Erweiterung von Tabellen nicht mehr auftritt.

In der Regel sollten Sie wie folgt vorgehen, um das dringende Problem zu lösen:

1. Benachrichtigen Sie die Benutzer, dass sie die Verbindung zum Server trennen müssen. Dadurch werden zwei Dinge erreicht: Weniger Benutzer sind von negativen Auswirkungen betroffen und, falls Benutzer, die die Verbindung trennen, temporäre Tabellen verwenden, würde durch das Trennen der Verbindung Speicher freigegeben. Eventuell möchten Sie eine Richtlinie oder einen Fehlerprüfcode verwenden, um sicherzustellen, dass Benutzer und/oder Programme die Verbindung kontrolliert trennen, wenn sie diesen Fehler feststellen.
2. Wenn zu wenige temporäre Tabellen verwendet werden, sodass diese keinen wesentlichen Einfluss auf die Speicherbelegung haben, löschen Sie ggf. einige Indizes von transienten Tabellen oder transiente Tabellen selbst.

Wenn zu wenige temporäre oder transiente Tabellen verwendet werden, sodass diese keinen wesentlichen Einfluss auf die Speicherbelegung haben, müssen Sie drastischere Maßnahmen ergreifen.

1. Löschen Sie einen oder mehrere Indizes für speicherinterne Tabellen.
2. Fahren Sie den Server herunter.
3. Wenn Sie wirklich nichts im Speicher löschen können (z. B. wenn Sie ausschließlich ″normale″ speicherinterne Tabellen haben, die alle keinen Index haben und alle wertvolle Daten enthalten), dann erhöhen Sie den Wert von **ImdbMemoryLimit** leicht, bevor Sie den Server erneut starten. Hierdurch zwingen Sie den Server möglicherweise zum Paging des virtuellen Speichers. Dies verringert zwar die Leistung wesentlich, aber Sie können so den Server weiterhin verwenden und sich den langfristigen Problemen widmen. Wenn Sie zuvor den Wert von **ImdbMemoryLimit** etwas niedriger als den Maximalwert gesetzt haben, können Sie ihn nun wieder etwas erhöhen, ohne dass das System zum Paging des virtuellen Speichers gezwungen wird.
4. Starten Sie den Server erneut.
5. Verringern Sie die Anzahl der Benutzer, die das System verwenden, soweit wie möglich, bis Sie das langfristige Problem gelöst haben. Stellen Sie sicher, dass die Benutzer keine temporären oder transienten Tabellen erstellen, bis das langfristige Problem gelöst ist.

Wenn Sie das dringende Problem behoben und sichergestellt haben, dass der Server zumindest über etwas freien Speicher verfügt, können Sie sich dem langfristigen Problem widmen.

Um das langfristige Problem zu lösen, müssen Sie das in den speicherinternen Tabellen gespeicherte Datenvolumen verringern. Hierzu können Sie die Anzahl

oder die Größe von speicherinternen Tabellen (einschließlich temporärer und transienter Tabellen) verringern oder die Anzahl der Indizes für speicherinterne Tabellen verringern.

Wenn das Problem lediglich aufgrund intensiver Nutzung temporärer Tabellen (oder transienter Tabellen) verursacht wurde, müssen Sie möglicherweise lediglich sicherstellen, dass nicht zu viele Sitzungen zu viele umfangreiche temporäre Tabellen (oder transiente Tabellen) gleichzeitig erstellen.

Wenn das Problem verursacht wurde, weil zu viel Speicher für "normale" speicherinterne Tabellen verwendet wurde und wenn Sie die für den Server verfügbare Speichermenge nicht erhöhen können, müssen Sie eine oder mehrere Tabellen aus dem Hauptspeicher auf die Platte verschieben. Dies ist erfreulicherweise nicht sehr schwer. Gehen Sie wie folgt vor, um eine Tabelle aus dem Speicher auf die Platte zu verschieben:

1. Erstellen Sie eine leere plattenbasierte Tabelle mit derselben Struktur (aber einem anderen Namen) wie eine der Tabellen im Speicher.
2. Kopieren Sie die Informationen von der speicherinternen Tabelle in die plattenbasierte Tabelle.[1]
3. Löschen Sie die speicherinterne Tabelle.
4. Benennen Sie die plattenbasierte Tabelle um, sodass sie den ursprünglichen Namen der nun gelöschten speicherinternen Tabelle hat.

Als bewährte Vorsichtsmaßnahme können Sie den Wert von **ImdbMemoryLimit** bewusst auf einen etwas niedrigeren Wert als Ihren tatsächlich zur Verfügung stehenden Maximalwert setzen. So können Sie, wenn Ihnen der Speicher ausgeht und Sie keine entbehrlichen speicherinternen Tabellen oder Indizes haben, die gelöscht werden können, den Wert von **ImdbMemoryLimit** etwas erhöhen und anschließend den Server mit ausreichend freiem Speicher erneut starten, sodass Sie dem langfristigen Bedarf gerecht werden können.

Denken Sie außerdem daran, dass Sie sich auch mithilfe des Parameters **ImdbMemoryWarningPercentage** eine Warnung über die steigende Speicherbelegung ausgeben lassen können.

Sie müssen nicht in allen Situationen die Anzahl der speicherinternen Tabellen verringern. In einigen Fällen kann es praktischer sein, einfach mehr Speicher im Computer zu installieren.

Beachten Sie bitte auch, dass es besser ist, das Problem zu vermeiden, als es zu beheben. Es wird daher dringend empfohlen, den Parameter **ImdbMemoryWarningPercentage** auf einen geeigneten Wert zu setzen, damit eine zuverlässige Warnung angezeigt wird, bevor Sie den gesamten, Ihren speicherinternen Tabellen zur Verfügung stehenden Speicher belegen.

---

1. Wenn Sie versuchen, die Datensätze einer großen Tabelle mithilfe einer einzigen SQL-Anweisung (INSERT INTO ...VALUES SELECT FROM) in eine andere Tabelle zu kopieren, beachten Sie bitte, dass die gesamte Operation in *einer* Transaktion stattfindet. Eine solche Operation ist nur dann effektiv, wenn das gesamte Datenvolumen in den Cache des Servers passt. Wenn die Transaktionsgröße die Cachegröße übersteigt, wird die Leistung erheblich verringert. Daher wird dringend empfohlen, die Daten einer großen Tabelle mithilfe einer einfachen gespeicherten Prozedur oder Anwendung in kleineren Transaktionen (z. B. einige Tausend Zeilen pro Transaktion) in eine andere Tabelle zu kopieren.

   Für die Zwischentabelle sind keine Indizes erforderlich. Die Indizes sollten in der neuen "tatsächlichen Tabelle" erneut erstellt werden, nachdem die Daten erfolgreich kopiert wurden.

ProcessMemoryLimit

Der Parameter **ProcessMemoryLimit** gibt den Maximalwert des virtuellen Speichers an, der dem speicherinternen Datenbankprozess zugeordnet werden kann. Der Parameter **ProcessMemoryLimit** wird durch den Parameter **ProcessMemoryCheckInterval** gesteuert.

Wenn der Wert des Parameters **ProcessMemoryCheckInterval** 0 (werkseitige Einstellung) ist, ist der Parameter **ProcessMemoryLimit** nicht wirksam, d. h. es gibt keinen Grenzwert für den Prozessspeicher.

Die werkseitige Einstellung für den Parameter **ProcessMemoryLimit** ist ″1G″ (1 GB). Setzen Sie den Parameter auf einen Wert, der sicherstellt, dass der speicherinterne Datenbankprozess vollständig in den physischen Hauptspeicher passt. Folgende Faktoren beeinflussen die erforderliche Speichermenge:

- Größe des physischen Hauptspeichers im Computer
- Vom Betriebssystem belegter Speicher
- Von den speicherinternen Tabellen (einschließlich temporären Tabellen, transienten Tabellen, und ″normalen″ speicherinternen Tabellen) sowie den Indizes für diese speicherinternen Tabellen verwendeter Speicher
- Für den Cache des solidDB-Servers vorgesehene Speichermenge (Konfigurationsparameter **CacheSize** in der Datei `solid.ini`)
- Für gleichzeitig auf dem Server ablaufende Verbindungen, Transaktionen und Anweisungen erforderliche Speichermenge. Je mehr gleichzeitig bestehende Verbindungen und aktive Anweisungen auf dem Server vorhanden sind, desto mehr Arbeitsspeicher ist für den Server erforderlich. In der Regel müssen Sie für jede Clientverbindung im Server 0,5 MB zuordnen.
- Von anderen Prozessen (Programmen und Daten), die auf dem Computer ausgeführt werden, verwendeter Speicher

Wenn der Grenzwert erreicht ist, d. h., wenn der speicherinterne Datenbankprozess den von **ProcessMemoryLimit** angegebenen Speicher vollständig verwendet, akzeptiert der Server nur noch ADMIN COMMAND-Befehle). Mithilfe der Parameter **ProcessMemoryWarningPercentage** und **ProcessMemoryLowPercentage** können Sie sich eine Warnung über die steigende Speicherbelegung ausgeben lassen.

ProcessMemoryLowPercentage

Der Parameter **ProcessMemoryLowPercentage** setzt einen Warngrenzwert für die Gesamtprozessgröße. Der Grenzwert wird als Prozentsatz des Parameterwerts von **ProcessMemoryLimit** ausgedrückt. Bevor dieser Grenzwert überschritten wird, wurde eine Warnung ausgegeben, da Sie bereits den mithilfe des Parameters **ProcessMemoryWarningPercentage** definierten Warngrenzwert überschritten haben. Wenn der Grenzwert von **ProcessMemoryLowPercentage** überschritten wird, wird ein Systemereignis ausgegeben.

Der durch den Parameter **ProcessMemoryLowPercentage** angegebene Grenzwert muss höher sein als der durch den Parameter **ProcessMemoryWarningPercentage** angegebene Grenzwert. Wenn für den Parameter **ProcessMemoryWarningPercentage** beispielsweise ein Wert von 82 festgelegt wurde, muss der Wert des Parameters **ProcessMemoryLowPercentage** mindestens 83 betragen.

ProcessMemoryWarningPercentage

Der Parameter **ProcessMemoryWarningPercentage** setzt den ersten Warngrenzwert für die Gesamtprozessgröße. Der Warngrenzwert wird als Prozentsatz des

Parameterwerts von **ProcessMemoryLimit** ausgedrückt. Wenn der Grenzwert von **ProcessMemoryWarningPercentage** überschritten wird, wird ein Systemereignis ausgegeben.

Der durch den Parameter **ProcessMemoryWarningPercentage** angegebene Grenzwert muss niedriger sein als der durch den Parameter **ProcessMemoryLowPercentage** angegebene Grenzwert.

ProcessMemoryCheckInterval

Die Grenzwerte für die Prozessgröße werden in bestimmten Abständen überprüft. Das Prüfintervall wird mithilfe des Parameters **ProcessMemoryCheckInterval** gesetzt. Das Intervall wird in Millisekunden angegeben.

Der Mindestwert ungleich null ist ″1000″ (ms). Es sind nur die Werte ″0″, ″1000″ oder größer ″1000″ (1 Sekunde) zulässig. Wenn der bestimmte Wert größer ″0″, aber kleiner ″1000″ ist, wird eine Fehlernachricht ausgegeben.

Die werkseitige Einstellung ist ″0″, d. h., die Überprüfung der Prozessgröße ist inaktiviert.

Der Parameter **ProcessMemoryCheckInterval** steuert auch den Parameter **ProcessMemoryLimit**; wenn der Wert des Parameters **ProcessMemoryCheckInterval** 0 ist, ist der Parameter **ProcessMemoryLimit** nicht wirksam, d. h. es gibt keinen Grenzwert für den Prozessspeicher.

Berechnung des erforderlichen Plattenspeicherplatzes

Beim Berechnen des für das Speichern Ihrer Datenbank erforderlichen Plattenspeicherplatzes müssen Sie den Speicherplatz berücksichtigen, der für das Speichern der speicherinternen Tabellen sowie der plattenbasierten Tabellen erforderlich ist. Der Grund hierfür liegt darin, dass beim Herunterfahren des Servers die speicherinternen Daten auf dem Plattenlaufwerk gespeichert werden. (Wenn der Server erneut gestartet wird, werden diese Informationen zurück in den Speicher gelesen.)

Da der Server die speicherinternen Daten auf das Plattenlaufwerk schreiben muss, wenn er heruntergefahren wird, braucht ein Server mit speicherinternen Tabellen normalerweise länger zum Herunterfahren als ein Server mit nur plattenbasierten Tabellen. Gleichermaßen braucht der Server, wenn er speicherinterne Tabellen enthält, normalerweise länger zum Starten, da er bei seinem Start die Daten wieder vom Plattenlaufwerk in den Speicher laden muss.

Einhaltung von Standards

Die Funktion für speicherinterne Tabellen unterliegt nicht dem ANSI-Standard SQL-99.

Einschränkungen von speicherinternen Tabellen

Physischer Hauptspeicher und virtueller Speicher

Die offensichtlichste Einschränkung liegt darin, dass die Gesamtgröße der speicherinternen Datenbanktabellen die verfügbare Menge des virtuellen Speichers nicht übersteigen kann.

Wichtig:

Da der virtuelle Speicher häufig auf die Platte ausgelagert wird, wird bei der Verwendung des virtuellen Speichers der Vorteil von speicherinternen Tabellen teilweise wieder zunichte gemacht. Es wird daher dringend empfohlen, dass Sie nicht die Größe des verfügbaren virtuellen Speichers begrenzen, sondern Ihre speicherinternen Tabellen begrenzen, sodass sie die Größe des verfügbaren physischen Hauptspeichers nicht überschreiten.

Berücksichtigen Sie beim Berechnen des für Tabellen erforderlichen Speicherplatzes unbedingt die BLOB-Daten. Normalerweise sollten die BLOB-Daten in plattenbasierten Tabellen gespeichert werden, da die maximale Größe einer BLOB-Spalte in Hauptspeichertabellen wesentlich geringer ist.

Beachten Sie, dass der zum Speichern einer Tabelle erforderliche Speicherplatz nicht nur den Platz für die Daten in der Tabelle umfasst, sondern auch für alle Indizes für diese Tabelle, einschließlich aller Indizes, die zur Unterstützung von Integritätsbedingungen über Primärschlüssel und Integritätsbedingungen über Fremdschlüssel erstellt wurden. Darüber hinaus belegen Tabellen wesentlich mehr Platz im Speicher als auf der Platte.

Wenn dem Server beim Versuch, Speicher zuzuordnen (z. B. beim Erweitern einer Tabelle während einer INSERT- oder ALTER TABLE-Operation), der virtuelle Speicher ausgeht, wird eine Fehlernachricht angezeigt.

Tabelle von speicherintern in plattenbasiert und umgekehrt ändern

Wenn die Tabelle leer ist, kann der Tabellentyp von speicherintern in plattenbasiert und umgekehrt geändert werden. Verwenden Sie hierzu den folgenden Befehl:

```
ALTER TABLE Tabellenname SET STORE MEMORY | DISK
```

Wenn die Tabelle Daten enthält, müssen Sie eine neue Tabelle (mit einem anderen Namen) erstellen, in die Sie die Daten kopieren. Nachdem Sie die Daten in die neue Tabelle kopiert haben, können Sie die alte Tabelle löschen und die neue Tabelle mit dem folgenden Befehl umbenennen:

```
ALTER TABLE aktueller_Tabellenname SET TABLE NAME neuer_Tabellenname
```

Transaktionsisolation

Die Isolationsstufe SERIALIZABLE wird nicht unterstützt.

Speicherinterne Tabellen können nicht in Transaktionen verwendet werden, deren Transaktionsisolationsstufe SERIALIZABLE ist. Für speicherinterne Tabellen werden folgende Tansaktionsisolationsstufen unterstützt: REPEATABLE READ und READ COMMITTED.

Für den sekundären HotStandby-Server wird immer die Transaktionsisolationsstufe READ COMMITTED verwendet.

Wenn Sie HotStandby verwenden und mit dem sekundären HotStandby-Server verbunden sind und dann Daten aus speicherinternen Tabellen lesen, wird die Transaktionsisolationsstufe automatisch auf READ COMMITTED gesetzt, auch wenn Sie REPEATABLE READ angegeben haben. (Speicherinterne Tabellen unterstützen weder auf dem primären noch auf dem sekundären Server die Transaktionsisolationsstufe SERIALIZABLE.)

Weitere Erweiterungen der speicherinternen Engine

Neben speicherinternen Tabellen bietet die Hauptspeicher-Engine von solidDB zwei weitere Merkmale, die die Leistung verbessern. Erstens warten Leseoperationen nicht auf den Plattenzugriff, selbst während das System Aktivitäten wie das Prüfpunktverfahren und die Transaktionsprotokollierung durchführt. Zweitens warten Schreiboperationen bei der gelockerten Transaktionsprotokollierung nicht auf den Plattenzugriff. Eine Beschreibung der gelockerten und der strikten Transaktionsprotokollierung finden Sie in der Veröffentlichung *IBM solidDB Administrator Guide*.

Gemeinsamen Speicherzugriff, LLA und HotStandby mit der speicherinternen Engine von solidDB verwenden

Gemeinsamer Speicherzugriff und Zugriff auf verlinkte Bibliothek

Der gemeinsame Speicherzugriff (SMA - Shared Memory Access) und der Zugriff auf die verlinkte Bibliothek (LLA) stellen den solidDB-Server in Form einer verlinkbaren Bibliothek bereit. Sie können Ihre Anwendung direkt mit der SMA- oder LLA-Bibliothek verlinken und über Funktionsaufrufe auf sie zugreifen, ohne ein Netzkommunikationsprotokoll zu verwenden.

SMA und LLA sind mit der Funktion für speicherinterne Tabellen der Hauptspeicher-Engine von solidDB kompatibel. Sie können speicherinterne Tabellen in einem SMA- oder LLA-Server erstellen.

Weitere Informationen zu SMA und LLA finden Sie in der Veröffentlichung *IBM solidDB Shared Memory Access and Linked Library Access User Guide*.

HotStandby

Die Komponente 'solidDB High Availability' bietet einen Bereitschaftsmodus (HotStandby). Das bedeutet, dass Ihr Datenbankserver paarweise mit einem zweiten Server verbunden ist und die Daten beider Server automatisch synchronisiert werden. Wenn ein Server ausfällt, kann der andere weiterhin verwendet werden.

Die speicherinterne Funktion der Hauptspeicher-Engine von solidDB ist mit solidDB HotStandby kompatibel.

Persistente speicherinterne Tabellen (d. h. speicherinterne Tabellen, die nicht explizit als temporär oder transient (TEMPORARY bzw. TRANSIENT) definiert wurden) werden vom primären HotStandby-Server auf den sekundären Server repliziert.

Temporäre Tabellen und transiente Tabellen werden NICHT auf den sekundären Server repliziert.

Inkompatibilitäten mit früheren solidDB-Produkten

Die Hauptspeicher-Engine von solidDB ist fast vollständig mit den früheren Versionen von solidDB-Produkten kompatibel. Sämtliche Ausnahmen werden in den Releaseinformationen beschrieben.

2 Temporäre Tabellen und transiente Tabellen

Temporäre und transiente Tabellen bieten eine höhere Leistung als speicherinterne Standardtabellen. Die Daten in temporären und transienten Tabellen sind jedoch nicht permanent und daher müssen Sie die Daten möglicherweise aus der temporären oder transienten Tabelle in eine andere Tabelle kopieren, wenn Sie sie permanent speichern wollen.

Wenn Sie eine speicherinterne Tabelle erstellen, ist diese Tabelle standardmäßig "persistent". Die Tabelle wird beim Herunterfahren des Servers auf die Platte geschrieben und beim erneuten Start des Servers wieder von der Platte gelesen. Die Hauptspeicher-Engine von solidDB bietet jedoch zwei Typen von speicherinternen Tabellen, die nicht persistent sind: temporäre Tabellen und transiente Tabellen. Diese beiden Tabellentypen sind für die Verwendung mit "temporären" Daten konzipiert.

Temporäre Tabellen und transiente Tabellen bieten aus folgenden Gründen eine höhere Leistung:

- Die Daten in temporären Tabellen und transienten Tabellen werden ausschließlich im Speicher gespeichert. Sie werden nie auf die Platte geschrieben. (Wenn Sie den Server herunterfahren und neu starten oder wenn der Server abnormal beendet wird, gehen die Daten verloren. Bei temporären Tabellen werden die Daten am Ende der Benutzersitzung gelöscht, sie bleiben nicht bis zum Herunterfahren des Servers erhalten.)
- Temporäre und transiente Tabellen protokollieren Transaktionsdaten nicht auf der Platte. (Die Daten können nach einer abnormalen Beendigung des Servers nicht wiederhergestellt werden.)
- Wenn der Server seine regelmäßigen Prüfpunktoperationen durchführt, bei denen die Datenbankdaten auf das Plattenlaufwerk geschrieben werden, werden die Daten in den temporären und transienten Tabellen nicht auf die Platte geschrieben. (Eine ausführlichere Erläuterung der Prüfpunkte finden Sie in der Veröffentlichung *IBM solidDB Administrator Guide*.)
- Temporäre Tabellen und transiente Tabellen verwenden nicht nur die leistungsfähige Technologie von solidDB für speicherinterne Tabellen, sondern auch eine effizientere Datenspeicherstruktur als herkömmliche speicherinterne Tabellen.
- Temporäre Tabellen bieten einen weiteren Leistungsvorteil. Da in Sitzungen (Verbindungen) die Datensätze in einer temporären Tabelle der jeweils anderen Sitzungen nicht angezeigt werden, ist für temporäre Tabellen keine hoch entwickelte Steuerung des gemeinsamen Zugriffs erforderlich (d. h., es ist keine Überprüfung auf Sperrenkonflikte für Datensätze innerhalb der Tabelle erforderlich).

Temporäre Tabellen und transiente Tabelle sind besonders als Arbeitspuffer hilfreich. Sie können beispielsweise Daten aus einer persistenten Tabelle kopieren, eine Reihe intensiver Operationen für diese Daten in einer temporären Tabelle ausführen und die Ergebnisse anschließend wieder in einer persistenten Tabelle speichern. Hierdurch können Sie die Leistung maximieren und dennoch einen Teil oder alle Daten nach Abschluss Ihrer Aktionen beibehalten. Wenn Ihre Arbeit aus irgendeinem Grund unterbrochen wird, sind die Originaldaten weiterhin in der persistenten Tabelle sicher und Sie können die Verarbeitung wieder aufnehmen.

Der wesentliche Unterschied zwischen temporären Tabellen und transienten Tabellen ist der, dass die Daten einer temporären Tabelle nur in einer einzelnen Verbindung angezeigt werden, während die Daten einer transienten Tabelle allen Benutzern angezeigt werden.

Im Folgenden finden Sie ausführlichere Informationen zu jedem der beiden Tabellentypen. Temporäre Tabellen und transiente Tabelle weisen viele Gemeinsamkeiten auf. Aus diesem Grund enthalten die beiden folgenden Abschnitte einige Wiederholungen. Jeder dieser beiden Tabellentypen wird separat und vollständig beschrieben und anschließend werden die Unterschiede zwischen den beiden Typen hervorgehoben. Weitere Informationen zu den Unterschieden finden Sie in „Informationen zur Leistungsverbesserung für temporäre und transiente Tabellen" auf Seite 23.

Temporäre Tabellen

Die Daten in temporären Tabellen haben eine sehr eingeschränkte Transparenz und eine sehr begrenzte Bestandsdauer.

Eingeschränkte Transparenz

Die Daten bieten eine eingeschränkte Transparenz, da sie nur der Sitzung (Verbindung) angezeigt werden, die die Daten eingefügt hat. Wenn Ihre Sitzung eine temporäre Tabelle erstellt und Daten in sie einfügt, werden Ihre Daten keiner anderen Benutzersitzung angezeigt, selbst wenn Sie Zugriffsrechte für diese Tabelle erteilt haben. Zwar können mehrere Sitzungen dieselbe Tabelle gleichzeitig verwenden, aber jeder Sitzung werden nur ihre eigenen Daten angezeigt. (Da jeder Sitzung nur ihre eigenen Daten angezeigt werden, ist selbst bei einer Tabelle mit eindeutiger Integritätsbedingung keine Koordination mit anderen Sitzungen erforderlich, um sicherzustellen, dass Sie eindeutige Werte in die Tabelle einfügen. Wenn Sie beispielsweise eine temporäre Tabelle erstellen, die über eine eindeutige Integritätsbedingung für die Spalte "ID" verfügt, können sowohl Sie als auch eine andere Sitzung Datensätze einfügen, deren ID auf den Wert "1" gesetzt wurde.) Da jeder Sitzung ausschließlich ihre eigenen Daten angezeigt werden, können Operationen wie UPDATE und DELETE auch lediglich Auswirkungen auf die Daten der entsprechenden Sitzung haben.

Begrenzte Bestandsdauer

Die Daten haben eine begrenzte Bestandsdauer und werden gelöscht, wenn Sie Ihre aktuelle Sitzung beenden (d. h., wenn Sie die Verbindung zum Server trennen). Wenn Sie die Verbindung wieder herstellen, sind Ihre Daten nicht mehr vorhanden.

Es ist wichtig zu verstehen, dass sich das Wort "temporär" in der Bezeichnung "temporäre Tabellen" auf die Daten bezieht, nicht auf die Tabelle selbst. Der Server speichert die Definition der temporären Tabelle (jedoch nicht die Daten) in den Systemtabellen des Servers und behält diese Definition bei, auch nachdem Sie die Verbindung getrennt haben. Wenn Sie die Verbindung zum Server zu einem späteren Zeitpunkt wiederherstellen, ist diese Tabelle daher noch vorhanden, sie ist jedoch leer. Aus diesem Grund müssen Sie die Tabelle, nachdem Sie sie erstellt haben, in künftigen Sitzungen nicht nochmals erstellen. Wenn Sie oder andere Benutzer versuchen, eine temporäre Tabelle mit demselben Namen wie eine vorhandene temporäre Tabelle zu erstellen, wird eine Fehlernachricht angezeigt. Dieses Verhalten kann bei der Vorstellung, dass "temporäre Tabelle" bedeutet, dass die Tabelle (und nicht nur die Daten) beim Trennen der Verbindung gelöscht wird, unerwartet sein.

Da die Tabellen bestehen bleiben (auch wenn die Daten nicht gespeichert werden), müssen Sie die Tabellendefinition, wenn Sie sie nicht mehr brauchen, mit dem Befehl DROP TABLE löschen.

Weil die Tabelle bestehen bleibt, wenn Sie eine Datenbankschemadefinition exportieren, enthält die Ausgabe die Befehle zur Neuerstellung der temporären Tabellen.

Da der Inhalt der temporären Tabellen einer Sitzung gelöscht wird, wenn der Benutzer die Verbindung trennt, kann der Server eine Zeit lang eine hohe CPU-Belastung aufweisen, nachdem die Verbindung einer Sitzung mit einem hohen Datenvolumen in den temporären Tabellen getrennt wurde.

Weitere Einschränkungen

Im Folgenden werden einige weitere Einschränkungen für temporäre Tabellen aufgeführt.

- Daten der temporären Tabellen werden bei Verwendung der Komponente Hot-Standby nicht auf den sekundären Server repliziert. Beachten Sie, dass die Definitionen der temporären Tabellen jedoch auf den sekundären Server (HotStandby-Server) repliziert werden. Bei einer Funktionsübernahme durch den sekundären Server müssen Sie die bereits erstellten temporären Tabellen also nicht erneut erstellen. Die Daten in diesen Tabellen müssen jedoch erneut erstellt werden.
- Temporäre Tabellen können nicht als 'Mastertabellen' in einem Advanced Replication-System verwendet werden. (Sie können in einem Advanced Replication-System jedoch als Replikattabellen verwendet werden.)
- Für temporäre Tabellen gibt es Einschränkungen hinsichtlich ihrer Verwendung mit referenziellen Integritätsbedingungen. Eine temporäre Tabelle kann auf eine andere temporäre Tabelle verweisen, jedoch nicht auf einen anderen Tabellentyp (d. h. transient oder persistent). Kein anderer Tabellentyp kann auf eine temporäre Tabelle verweisen. Siehe Tabelle zu den referenzielle Integritätsbedingungen in diesem Kapitel.

Mit Ausnahme der in diesem Abschnitt aufgelisteten Einschränkungen verhalten sich temporäre Tabellen wie normale (persistente) speicherinterne Tabellen. Beispiele:

- Für temporäre Tabellen können Indizes existieren.
- Temporäre Tabellen können in Sichten verwendet werden.
- Für temporäre Tabellen können Trigger existieren.
- Temporäre Tabellen können BLOB-Spalten enthalten (die Länge dieser Spalten ist jedoch auf ein paar KB begrenzt.)
- Temporäre Tabellen befinden sich in einem bestimmten Katalog und in einem bestimmten Schema.
- Für temporäre Tabellen gelten Berechtigungen. D. h., der Ersteller der temporären Tabelle kann Berechtigungen für die Tabelle erteilen und entziehen. Der Datenbankadministrator kann ebenfalls Berechtigungen für die Tabelle erteilen und entziehen. Wenn jedoch in einer Sitzung Daten in eine temporäre Tabelle gestellt werden, können diese Daten in einer anderen Sitzung selbst dann nicht angezeigt werden, wenn der Eigentümer dieser Sitzung der Datenbankadministrator oder ein Benutzer ist, dem das Zugriffsrecht SELECT für die temporäre Tabelle erteilt wurde. Beim Erteilen von Zugriffsrechten für eine Tabelle wird dem anderen Benutzer also lediglich das Recht zur Verwendung Ihrer Tabelle gewährt, jedoch nicht das Recht zur Verwendung Ihrer Daten. Bitte

beachten Sie, dass die Standardzugriffsrechte für temporäre Tabellen mit den Standardzugriffsrechten für persistente Tabellen identisch sind.

Verwenden Sie zum Erstellen einer temporären Tabelle die Syntax unten, wobei "<...>" die Syntax kennzeichnet, die mit der für einen beliebigen anderen Tabellentyp identisch ist.

```
CREATE [GLOBAL] TEMPORARY TABLE <...>;
```

Die vollständige Syntax des Befehls CREATE TABLE finden Sie in der Veröffentlichung *IBM solidDB SQL Guide*.

Eine temporäre Tabelle ist immer eine speicherinterne Tabelle. Wenn Sie die Klausel STORE DISK verwenden, gibt der Server einen Fehler aus. Wenn Sie die Klausel STORE MEMORY verwenden oder wenn Sie die Klausel STORE völlig übergehen, erstellt der Server die temporäre Tabelle als eine speicherinterne Tabelle.

Die Implementierung temporärer Tabellen durch solidDB entspricht vollständig dem ANSI-Standard SQL:1999 für globale temporäre Tabellen. Alle temporären Tabellen von solidDB sind globale Tabellen, unabhängig davon, ob das Schlüsselwort GLOBAL angegeben ist. solidDB unterstützt keine lokalen temporären Tabellen, wie vom ANSI-Standard definiert.

Transiente Tabellen

Die Daten in transienten Tabellen haben eine begrenzte Bestandsdauer. Der Server speichert die Daten nur bis zum Herunterfahren des Servers.

Daten in transienten Tabellen haben denselben "Geltungsbereich" oder dieselbe Transparenz wie Daten in persistenten Tabellen. Die Daten, die Sie in eine transiente Tabelle einfügen, können in Sitzungen anderer Benutzer angezeigt werden, wenn diese Benutzer über die entsprechenden Berechtigungen verfügen.

Für transiente Tabellen gelten einige Einschränkungen:

- Die Daten der transienten Tabellen werden bei Verwendung der Komponente 'HotStandby' nicht auf den sekundären Server repliziert. Die transienten Tabellen selbst (nicht jedoch ihre Daten) werden auf den sekundären HotStandby-Server repliziert. Bei einer Funktionsübernahme durch den sekundären Server müssen Sie die bereits erstellten transienten Tabellen also nicht erneut erstellen. Die Daten in diesen Tabellen müssen jedoch erneut erstellt werden.
- Für transiente Tabellen gibt es Einschränkungen hinsichtlich ihrer Verwendung mit referenziellen Integritätsbedingungen. Transiente Tabellen können auf andere transiente Tabellen und auf persistente Tabellen verweisen. Sie können jedoch nicht auf temporäre Tabellen verweisen. Weder temporäre Tabellen noch persistente Tabellen können auf eine transiente Tabelle verweisen. Siehe Tabelle zu den referenzielle Integritätsbedingungen in diesem Kapitel.
- Transiente Tabellen können nicht als "Mastertabellen" in einem Advanced Replication-System verwendet werden. (Sie können in einem Advanced Replication-System jedoch als Replikattabellen verwendet werden.)

Mit Ausnahme der in diesem Abschnitt aufgelisteten Einschränkungen verhalten sich transiente Tabellen wie "normale" (persistente) speicherinterne Tabellen. Beispiel:

- Transiente Tabellen können in Sichten verwendet werden.
- Für transiente Tabellen können Indizes existieren.

- Für transiente Tabellen können Trigger existieren.
- Transiente Tabellen können BLOB-Spalten enthalten (die Länge dieser Spalten ist jedoch wie bei allen speicherinternen Tabellen auf ein paar KB begrenzt.)
- Für transiente Tabellen gelten Zugriffsrechte.
- Transiente Tabellen befinden sich in einem speziellen Katalog und in einem speziellen Schema.

Wenn Sie eine Datenbank mit transienten Tabellen exportieren, werden die Daten in den transienten Tabellen exportiert (sowie die Struktur der Tabellen).

Verwenden Sie zum Erstellen einer transienten Tabelle die im Folgenden angegebene Syntax, wobei "<...>" die Syntax kennzeichnet, die mit der für einen beliebigen anderen Tabellentyp identisch ist.

```
CREATE TRANSIENT TABLE <...>;
```

Die vollständige Syntax des Befehls CREATE TABLE finden Sie in der Veröffentlichung *IBM solidDB SQL Guide*.

Transiente Tabellen sind immer speicherinterne Tabellen. Wenn Sie die Klausel STORE DISK verwenden, gibt der Server einen Fehler aus. Wenn Sie die Klausel STORE MEMORY verwenden oder wenn Sie die Klausel STORE völlig übergehen, erstellt der Server die transiente Tabelle als eine speicherinterne Tabelle.

Der Server speichert die Definition der transienten Tabelle (jedoch nicht die Daten) in den Systemtabellen des Servers und behält diese Definition auch noch bei, nachdem der Server heruntergefahren wurde. Wenn Sie den Server zu einem späteren Zeitpunkt erneut starten, ist zwar die Tabelle noch vorhanden, aber die Daten sind nicht mehr vorhanden. Daher müssen Sie die Tabelle, nachdem Sie sie erstellt haben, nicht nochmals erstellen. Wenn Sie oder andere Benutzer versuchen, eine transiente Tabelle mit demselben Namen wie eine vorhandene transiente Tabelle zu erstellen, wird selbst dann eine Fehlernachricht angezeigt, wenn der Server, seitdem die Tabelle mit diesem Namen ursprünglich erstellt wurde, heruntergefahren und erneut gestartet wurde. Dieses Verhalten kann bei der Vorstellung, dass "transiente Tabelle" bedeutet, dass die Tabelle beim Herunterfahren des Servers gelöscht wird, unerwartet sein.

Da die transiente Tabelle bestehen bleibt (auch wenn die Daten nicht gespeichert werden), können Sie die Tabelle, wenn Sie sie nicht mehr brauchen, mit dem Befehl DROP TABLE löschen.

Mit dem Dienstprogramm "solload" können Sie Daten in transiente Tabellen importieren.

Transiente Tabellen werden nicht vom ANSI-Standard für SQL definiert. Transiente Tabellen sind eine Erweiterung von solidDB zum SQL-Standard.

Unterschiede zwischen temporären Tabellen und transienten Tabellen

Die wichtigsten Unterschiede zwischen transienten Tabellen und temporären Tabellen:

- Transiente Tabellen ermöglichen es allen Sitzungen (Verbindungen) im System, dieselben Daten zu sehen. Temporäre Tabellen ermöglichen es nur dem Benutzer, der einen Teil der Daten erstellt hat, diese Daten zu sehen.

Da Benutzer möglicherweise auf dieselben Daten zugreifen, verwenden transiente Tabellen die Steuerung des gemeinsamen Zugriffs. Derzeit unterstützen transiente Tabellen lediglich die pessimistische Steuerung des gemeinsamen Zugriffs (″Sperre″).

Da temporäre Tabellen keine Steuerung des gemeinsamen Zugriffs verwenden, sind sie schneller als transiente Tabellen.

- Die Daten in transienten Tabellen bleiben bis zum Herunterfahren des Servers erhalten. Die Daten in temporären Tabellen bleiben lediglich erhalten, bis sich der Benutzer von der Sitzung abmeldet. Wenn in einer Sitzung also Daten in eine transiente Tabelle eingefügt werden, können diese Daten in anderen Sitzungen selbst dann angezeigt werden, nachdem der Ersteller der Daten die Verbindung getrennt hat.
- Daten in transienten Tabellen können mit dem Tool ″solexp″ exportiert werden. Bei Daten in temporären Tabellen ist dies nicht möglich.
- Die referenziellen Integritätsbedingungen für die beiden Tabellentypen sind unterschiedlich.

  In der folgenden Tabelle wird gezeigt, welche Tabellentypen auf andere Typen verweisen können. Wenn beispielsweise eine transiente Tabelle einen Fremdschlüssel haben kann, der auf eine persistente Tabelle verweist, wird in der Zelle beim Schnittpunkt der Zeile ″Transiente Tabelle″ und der Spalte ″Persistente Tabelle″ die Angabe ″JA″ angezeigt. Wenn die Integritätsbedingung über Fremdschlüssel nicht zulässig ist, wird ein Strich (″-″) angezeigt.

Tabelle 3. Referenzielle Integritätsbedingungen

TABELLE, AUF DIE VERWIESEN WIRD **VERWEISENDE TABELLE**	**Persistente plattenbasierte Tabelle**	**Persistente speicherinterne Tabelle**	**Transiente Tabelle**	**Temporäre Tabelle**
Persistente plattenbasierte Tabelle	JA	JA	-	-
Persistente speicherinterne Tabelle	JA	JA	-	-
Transiente Tabelle	JA	JA	JA	-
Temporäre Tabelle	-	-	-	JA

Jeder Tabellentyp kann auf sich selbst verweisen. Darüber hinaus können transiente Tabellen auf persistente Tabellen verweisen (aber nicht umgekehrt). Alle anderen Kombinationen sind ungültig.

3 Server optimieren

Dieses Kapitel enthält ausführliche Informationen zur Optimierung des Servers mithilfe der in der Hauptspeicher-Engine von solidDB bereitgestellten Funktionen.

Algorithmus für Auswahl der Tabellen, die im Speicher gespeichert werden sollen

Wenn Sie über ausreichend Speicher verfügen, um einige, jedoch nicht alle Tabellen in den Speicher zu stellen, können Sie anhand der folgenden Strategie auswählen, welche Tabellen in den Speicher gestellt werden sollen.

Grundsätzlich wird die ″Dichte″ des Zugriffs berücksichtigt. Je höher die Zugriffshäufigkeit, desto höher ist auch die ″Zugriffsdichte″. Ebenso gilt: Je größer die Tabelle ist, desto geringer ist die Zugriffsdichte für eine bestimmte Anzahl von Zugriffen pro Sekunde.

Die Zugriffsdichte wird in Einheiten von Zugriffen pro Megabyte pro Sekunde gemessen und als Zeilen/MB/s dargestellt. (Der Einfachheit halber wird ein Zugriff pro Zeile angenommen.) Wenn Sie beispielsweise eine 1-MB-Tabelle haben und in einem Zeitraum von 10 Sekunden auf 300 Zeilen zugreifen, ergibt sich folgende Dichte:

30 Zeilen/MB/s = 300 Zeilen / 1 MB / 10 Sekunden

In einem zweiten Beispiel wird angenommen, dass Sie eine 500-KB-Tabelle haben und pro Sekunde auf 300 Zeilen zugreifen. Daraus ergibt sich folgende Zugriffsdichte:

600 Zeilen/MB/s = 300 Zeilen / 0,5 MB / Sekunden

Die zweite Tabelle hat eine deutlich höhere Zugriffsdichte als die erste. Falls nur eine dieser Tabellen in den Speicher passt, sollten Sie daher die zweite Tabelle in den Speicher stellen.

Diese Formel ist etwas vereinfacht dargestellt.

1. Sie können auch die Anzahl der Byte berücksichtigen, auf die jedes Mal zugegriffen wird. Normalerweise ist dies die durchschnittliche Zeilengröße. Diese kann jedoch abweichen, wenn Sie große Binärobjekte verwenden, oder wenn der Server alle erforderlichen Informationen findet, indem nur ein Index und nicht die gesamte Tabelle gelesen wird.

   Da der Server normalerweise die Daten von der Platte in einem Vielfachen von einem ″Block″ liest (hierbei hat ein Block normalerweise 8 KB), stellt die Anzahl Byte pro Zugriff oder die Anzahl Byte pro Zeile Ihnen daher nur geringfügig genauere Zahlen bereit als die Formel ohne diese Angaben. Unabhängig davon, ob Sie eine 10-Byte-Zeile oder eine 2000-Byte-Zeile lesen, ist der Arbeitsaufwand des Servers etwa gleich.

2. Bei der Tabellengröße müssen Sie auch die Größe aller Indizes für diese Tabelle berücksichtigen. Bei jedem Hinzufügen eines Index werden auch mehr Daten hinzugefügt, die zu dieser Tabelle gespeichert werden. Wenn Sie einer Tabelle eine Integritätsbedingung über Fremdschlüssel hinzufügen, erstellt der Server darüber hinaus einen entsprechenden Index (sofern es nicht schon einen gibt), um bestimmte Typen von Suchoperationen für diese Tabelle zu beschleunigen. Wenn Sie die Größe Ihrer Tabelle im Speicher berechnen, müssen Sie die Tabelle, all ihre Indizes sowie all ihre BLOBs einbeziehen.

Nachdem Sie die Zugriffsdichte all Ihrer Tabellen berechnet haben, ordnen Sie diese Tabellen von der höchsten bis zur niedrigsten Dichte an. Anschließend beginnen Sie mit der Tabelle mit der höchsten Dichte und gehen dann in der Liste nach unten und definieren Tabellen als speicherinterne Tabellen, bis Sie den gesamten verfügbaren physischen Hauptspeicher belegt haben.

Leider ist dieser Prozess nicht ganz einfach. In dieser Beschreibung wird davon ausgegangen, dass Sie über optimale Informationen verfügen und eine Tabelle jederzeit von plattenbasiert in speicherintern (oder umgekehrt) ändern können. Möglicherweise kennen Sie jedoch die Gesamtgröße des freien Speichers in Ihrem Computer nicht und definieren unbeabsichtigt mehr speicherinterne Tabellen, als der Computer im physischen Hauptspeicher aufnehmen kann. Als Folge können Tabellen auf die Platte ausgelagert werden, wodurch sich die Leistung erheblich verringern kann. Darüber hinaus wissen Sie möglicherweise nicht genau, wie oft auf jede einzelne Tabelle zugegriffen wird, bis die jeweilige Tabelle ein großes Datenvolumen enthält. In der aktuellen Version dieses solidDB-Servers müssen Sie jedoch schon bei der Erstellung einer Tabelle, noch bevor Sie Daten in diese stellen, die Tabelle als speicherintern oder plattenbasiert definieren. Daher basieren Ihre Berechnungen zwangsläufig auf Schätzungen, wie häufig jede Tabelle verwendet wird, Schätzungen der Größe der jeweiligen Tabelle sowie Schätzungen des freien Speichers. Darüber hinaus wird davon ausgegangen, dass sich die durchschnittliche Zugriffsdichte im Laufe der Zeit nicht ändert.

Dieser Ansatz geht auch davon aus, dass Sie später keine weiteren Tabellen hinzufügen möchten und dass Ihre Tabellen nicht umfangreicher werden. Unter normalen Umständen sollten Sie nicht den gesamten zur Verfügung stehenden Speicher belegen, sondern genügend Platz lassen, da Ihre Tabellen wahrscheinlich umfangreicher werden. Zudem sollten Sie eine kleine Platzreserve für Fehler vorsehen, sodass Ihnen der Speicher nicht ausgeht.

Dieser Algorithmus bietet Ihnen dennoch eine grundlegende Orientierung bei der Auswahl der Tabellen, die in den Speicher gestellt werden sollen.

Wichtig: Da der virtuelle Speicher häufig auf die Platte ausgelagert werden kann, wird bei der Verwendung des virtuellen Speichers der Vorteil von speicherinternen Tabellen wieder zunichte gemacht. Stellen Sie daher immer sicher, dass der gesamte Datenbankverwaltungssystemsprozess in den physischen Hauptspeicher des Computers passt.

Informationen zur Leistungsverbesserung für temporäre und transiente Tabellen

Vorsicht:
Beachten Sie bei der Entscheidung, ob temporäre Tabellen oder transiente Tabellen verwendet werden sollen, dass die Daten in temporären und transienten Tabellen temporäre Daten und keine persistenten Daten sind. Die Daten werden nicht auf der Platte gespeichert. Wenn Sie HotStandby verwenden, werden die Daten nicht auf den sekundären HotStandby-Server kopiert. Die Daten werden auch nicht in das Transaktionsprotokoll geschrieben und können daher nach einer abnormalen Beendigung des Servers nicht wiederhergestellt werden. Verwenden Sie aus diesem Grund temporäre Tabellen oder transiente Tabellen nur dann, wenn Sie es sich leisten können, die darin befindlichen Daten zu verlieren und Ihre Arbeit erneut zu beginnen.

Die wichtigsten Aspekte für die Entscheidung zwischen temporären Tabellen und transienten Tabellen sind:

- Daten in temporären Tabellen bleiben nur bis zum Ende der Sitzung bestehen und Daten in transienten Tabellen bleiben bestehen, bis der Server heruntergefahren wird.
- Daten in temporären Tabellen werden anderen Sitzungen/Verbindungen nicht angezeigt.
- Temporäre Tabellen sind schneller als transiente Tabellen, da sie nur in sehr geringem Umfang Prüfungen auf Konflikte beim gemeinsamen Zugriff durchführen.

Wenn Sie Ihre Daten nicht "gemeinsam nutzen" wollen und Ihre Daten nicht mehr benötigen, nachdem Sie die Verbindung zur aktuellen Sitzung getrennt haben, wählen Sie temporäre Tabellen, da sie weniger Systemaufwand erfordern und eine höhere Leistung bieten.

Indizes

Wenn eine Tabelle im Speicher gespeichert wird, werden alle Indizes für diese Tabelle ebenfalls im Speicher gespeichert. Hierdurch wird natürlich die Leistung erhöht, aber auch Hauptspeicherkapazität verbraucht.

Normalerweise können speicherinterne Indizes extrem schnell sein. Daher wird ihre Verwendung empfohlen, um den schnellen Zugriff auf die Tabellendaten sicherzustellen.

Wenn Ihr Speicher nicht zur Speicherung all Ihrer Tabellen und Indizes im Speicher ausreicht, ist in einigen Fällen das Hinzufügen eines bestimmten Index nicht empfehlenswert, da dies zwar einige Abfragen beschleunigt, andere Abfragen jedoch aufgrund der Verwendung von Speicher verlangsamt, der andernfalls verwendet werden könnte, um andere Tabellen in den Speicher zu stellen.

4 Speicherinterne Datenbank konfigurieren

In diesem Kapitel wird die Konfiguration der speicherinternen solidDB-Datenbank beschrieben, sodass sie Ihre Umgebungs-, Leistungs- und Betriebsanforderungen erfüllt. Diese Beschreibung ist ein Teil der im Abschnitt *Configuring solidDB* der Veröffentlichung *IBM solidDB Administrator Guide* enthaltenen Informationen.

Einen Überblick mit der vollständigen Liste der verfügbaren Parameter, die für die Hauptspeicher-Engine von solidDB relevant sind, finden Sie in Anhang C, „Konfigurationsparameter", auf Seite 43.

Konfigurationsdateien und Parametereinstellungen

solidDB ruft die meisten seiner Konfigurationsdaten aus der Datei `solid.ini` ab. Genauer gesagt gibt es zwei verschiedene Konfigurationsdateien `solid.ini`, eine auf dem Server und eine auf dem Client. Keine der Konfigurationsdateien ist jedoch zwingend erforderlich. Wenn es keine Konfigurationsdatei gibt, werden die werkseitigen Einstellungen verwendet. Die Konfigurationsdateien `solid.ini` enthalten Konfigurationsparameter für den Client bzw. für den Server. Die clientseitige Konfigurationsdatei wird verwendet, wenn der ODBC-Treiber verwendet wird, und die Datei muss sich im Arbeitsverzeichnis der Anwendung befinden.

Anmerkung: In der Dokumentation zu solidDB beziehen sich Verweise auf die Datei `solid.ini` normalerweise auf die serverseitige Datei `solid.ini`.

Beim Starten versucht solidDB die Datei `solid.ini` zuerst von dem von der Umgebungsvariablen SOLIDDIR festgelegten Verzeichnis aus zu öffnen. Wenn die Datei nicht in dem von dieser Variablen angegebenen Pfad gefunden wird oder wenn die Variable nicht gesetzt ist, versucht der Server oder der Client, die Datei vom aktuellen Arbeitsverzeichnis aus zu öffnen. Das aktuelle Arbeitsverzeichnis ist normalerweise das Verzeichnis, von dem aus Sie den solidDB-Server oder eine Clientanwendung gestartet haben. Mit der Serverbefehlszeilenoption "-c" können Sie ein anderes Arbeitsverzeichnis angeben. Weitere Informationen zu Befehlszeilenoptionen finden Sie in *Appendix B, solidDB Command Line Options* in der Veröffentlichung *IBM solidDB Administrator Guide*.

Die Konfigurationsdateien enthalten Einstellungen für die solidDB-Parameter. Wenn in der Datei `solid.ini` für einen bestimmten Parameter kein Wert gesetzt ist, verwendet solidDB die werkseitige Einstellung für diesen Parameter. Die werkseitigen Einstellungen können von dem von Ihnen verwendeten Betriebssystem abhängig sein.

Im Allgemeinen bieten die werkseitigen Einstellungen sowohl eine gute Leistung als auch eine gute Betriebsbereitschaft. In einigen Fällen kann die Änderung einiger Parameterwerte jedoch die Leistung verbessern.

Sie können die Konfiguration ändern, indem Sie Paare aus Parametername und -wert in der Datei `solid.ini` festlegen. Wenn Sie beispielsweise die Netzwerkadresse des Servers angeben wollen, verwenden Sie den Parameternamen "Listen" und einen entsprechenden Wert wie zum Beispiel:

```
Listen=tcp 192.168.255.1 1315
```

Wenn der Server für Clientanforderungen empfangsbereit ist, wird hierdurch angegeben, dass das die Empfangsbereitschaft über das TCP/IP-Protokoll erfolgt, die Netzwerkadresse 192.168.255.1 und die Portnummer 1315 ist.

Parameter werden entsprechend den Abschnittskategorien in der Konfigurationsdatei gruppiert. Einen Überblick über die Abschnittskategorien sowie alle verfügbaren Parameter finden Sie in *Appendix A, Server-Side Configuration Parameters* und *Appendix B, Client-Side Configuration Parameters* in der Veröffentlichung *IBM solidDB Administrator Guide*.

Jede Abschnittskategorie beginnt mit einem Abschnittsnamen in eckigen Klammern wie zum Beispiel:

```
[com]
```

Im Abschnitt ″[com]″ werden Kommunikationsinformationen aufgelistet. Bei Abschnittsnamen muss die Groß-/Kleinschreibung nicht beachtet werden. Die Abschnittsnamen ″[COM]″, ″[Com]″ und ″[com]″ sind somit identisch.

Es folgt ein Beispiel für einen Abschnitt aus einer serverseitigen Konfigurationsdatei `solid.ini`:

```
[IndexFile]
FileSpec_1=C:\soldb\solid1.db 1000M
CacheSize=64M
```

Serverseitige Parameter verwalten

Sie können solidDB-Parameter und ihre Werte auf folgende Arten anzeigen und ändern:

- Durch Eingabe der Befehle
  ADMIN COMMAND 'parameter'
  und
  ADMIN COMMAND 'describe parameter'
  im solidDB SQL Editor (TTY-Modus - seriell).
- Direkt, durch das Bearbeiten der Datei `solid.ini` im Verzeichnis `solidDB`.

Die Abschnitte unten enthalten Anweisungen für die Verwaltung der Parameter mit ADMIN COMMAND und der Datei `solid.ini`.

Anmerkung: Ausführliche Informationen nur zum Anzeigen und Setzen der Parameter für das Serverkommunikationsprotokoll finden Sie im Abschnitt *Managing Network Connections* in der Veröffentlichung *IBM solidDB Administrator Guide*.

Parameter mit ADMIN COMMAND anzeigen und setzen

Mit ADMIN COMMAND können Sie die Parameter über Fernzugriff über einen solidDB-Server ändern, ohne diesen erneut zu starten. Sie können auf alle Parameter zugreifen, selbst dann, wenn sie nicht in der Konfigurationsdatei `solid.ini` vorhanden sind. Wenn der Parameter nicht vorhanden ist, wird die werkseitige Einstellung verwendet.

Parameter anzeigen

Mit folgendem Befehl kann eine Zusammenfassungssicht vieler Parameter oder eines Parameters angezeigt werden:

```
ADMIN COMMAND 'parameter [-r] [Abschnittsname[.Parametername]]';
```

Dabei gilt Folgendes:

- Die Option "-r" gibt an, dass nur der aktuelle Wert erforderlich ist.
- *Abschnittsname* ist der Name der Kategorie, in der sich der Parameter in der Datei `solid.ini` befindet.

Geben Sie zum Anzeigen aller Parameter den folgenden Befehl im solidDB SQL Editor (TTY-Modus - seriell) ein:

ADMIN COMMAND 'parameter';

Daraufhin wird eine Liste aller Parameter mit dem *aktuellen Wert*, dem *Startwert* und den *werkseitigen Einstellungen* zurückgegeben. Sie können die angezeigten Parameter auf einen bestimmten Abschnitt beschränken, indem Sie einen Abschnittsnamen eingeben. Beispiel:

```
ADMIN COMMAND 'parameter logging';
```

Sie können die zu einem einzelnen Parameter gehörenden Werte anzeigen, indem Sie den vollständigen Parameternamen eingeben. Beispiel:

```
ADMIN COMMAND 'parameter logging.durabilitylevel';
    RC TEXT
    -- ----
    0 Logging DurabilityLevel 3 2 2
1 rows fetched.
```

Die folgenden drei Werte werden (in der angegebenen Reihenfolge) angezeigt:

- *Aktueller Wert*
- *Startwert*, der beim Serverstart verwendet wurde
- *Werkseitige Einstellung*, die im Produkt voreingestellt ist

Wenn Sie möchten, können Sie diesen Befehl mit der Option "-r" qualifizieren, damit nur die aktuellen Werte angezeigt werden. Beispiel:

ADMIN COMMAND 'parameter -r';

Beschreibung eines bestimmten Parameters anzeigen

Sie können auch eine ausführlichere Beschreibung eines bestimmten Parameters anzeigen, die Informationen zu den gültigen Parametertypen und Zugriffsmodi enthält. Diese Informationen sind hilfreich, vor allem, weil Parameter möglicherweise dynamisch bearbeitet werden müssen. Die Unterstützung von Parametern kann abhängig von den Produkten, Plattformen und Releases variieren.

Zum Anzeigen der Beschreibung eines Parameters geben Sie den folgenden Befehl mit dem solidDB SQL Editor (TTY-Modus - seriell) ein:

```
ADMIN COMMAND 'describe parameter [Abschnittsname[.Parametername]] ';
```

Eine Ergebnismenge für einen einzelnen Parameter sieht wie folgt aus:

```
ADMIN COMMAND 'describe parameter logging.durabilitylevel';
    RC TEXT
    -- ----
    0 DurabilityLevel
    0 Default transaction durability level
    0 LONG
    0 RW
```

```
    0 2
    0 3
    0 2
7 rows fetched.
```

Die Zeilen der Ergebnismenge lauten:

- *Parametername* ist der Name des Parameters, beispielsweise "CacheSize".
- *Beschreibung* des Parameters
- *Datentyp*
- *Zugriffsmodus*, dieser kann einer der folgenden sein:
  - RO: Schreibgeschützt - Der Wert kann nicht dynamisch geändert werden.
  - RW: Schreib-/Lesezugriff - Der Wert kann dynamisch geändert werden und die Änderung wird sofort wirksam.
  - RW/STARTUP: Schreib-/Lesezugriff/Start - Der Wert kann dynamisch geändert werden, aber die Änderung wird erst beim nächsten Start des Servers wirksam.
  - RW/CREATE: Schreib-/Lesezugriff/Erstellen - Der Wert kann dynamisch geändert werden, aber die Änderung wird erst beim Erstellen einer neuen Datenbank wirksam.
- *Startwert* zeigt den Startwert des Parameters an.
- *Aktueller Wert* zeigt den aktuellen Wert des Parameters an.
- *Werkseitige Einstellung* zeigt den im Produkt voreingestellten Wert an.

Parameterwert festlegen

Zum Festlegen eines Werts für einen bestimmten Parameter geben Sie den folgenden Befehl mithilfe des solidDB SQL Editors (TTY-Modus - seriell) ein:

```
ADMIN COMMAND 'parameter Abschnittsname.Parametername=Wert [temporary]';
```

Dabei gilt Folgendes:

Wert ist ein gültiger Parameterwert.

Anmerkung:

Wenn kein Wert angegeben ist, wird hierdurch der Parameter auf die werkseitige Einstellung gesetzt (der Wert wird zurückgesetzt). Wenn Sie für den Parameterwert einen Stern (*) eingeben, wird der Parameter auf seine werkseitige Einstellung gesetzt.

Ist 'temporary' festgelegt, wird der geänderte Wert nicht in der Datei `solid.ini` gespeichert.

Optional können Sie um das Gleichheitszeichen auch Leerzeichen setzen.

Beispiel:

```
--set communication trace on
ADMIN COMMAND 'parameter com.trace = yes';
```

Anmerkung:

Operationen zur Parameterverwaltung sind nicht Bestandteil einer Transaktion und können daher nicht rückgängig gemacht werden.

Die Befehle geben den neuen Wert als Ergebnismenge zurück. Wenn der Zugriffsmodus des Parameters schreibgeschützt (RO) ist oder der eingegebene Wert ungültig ist, gibt die ADMIN COMMAND-Anweisung einen Fehler zurück.

Persistenz von Parameteränderungen

Alle Änderungen an Parametern mit dem Zugriffsmodus RW* (Schreib-/Lesezugriff) werden beim nächsten Prüfpunkt in der Datei `solid.ini` gespeichert. Dies gilt nicht für Werte, die mit der Option 'temporary' gesetzt wurden.

Geänderte Werte können mit dem folgenden Befehl auch sofort gespeichert werden:

```
ADMIN COMMAND 'save parameters [Name_der_INI-Datei]';
```

Wenn kein Name für die INI-Datei (*Name_der_INI-Datei*) angegeben wird, wird die aktuelle Datei `solid.ini` neu geschrieben. Andernfalls wird eine vollständige Konfigurationsdatei an eine neue Position geschrieben. So können die Prüfpunkte der Konfigurationsdatei für eine spätere Verwendung auf einfache Art gespeichert werden.

Parameter in `solid.ini` anzeigen und setzen

1. Öffnen Sie die Datei `solid.ini` im Arbeitsverzeichnis Ihres solidDB-Prozesses.
2. Zeigen Sie den Wert des Parameters an.

   Die angezeigten Parameter sind die derzeit im Server aktiven Parameter. Wenn Sie keinen Parameterwert gesetzt haben, wird die werkseitige Einstellung beim Start verwendet. Die werkseitige Einstellung kann von dem Betriebssystem abhängig sein, unter dem solidDB ausgeführt wird.
3. Fügen Sie falls erforderlich den Abschnitt, den Parameter und den Parameterwert hinzu.
4. Speichern Sie die Änderungen.

Sie müssen den Server erneut starten, um die Änderungen zu aktivieren.

Konstante Parameterwerte

Der Parameterzugriffsmodus für den Parameter "Blocksize" im Abschnitt "[IndexFile]" der Konfigurationsdatei ist RO (schreibgeschützt). Der Parameter wird beim Erstellen der Datenbank gesetzt und kann danach nicht geändert werden.

Wenn Sie einen anderen konstanten Wert verwenden wollen, müssen Sie eine neue Datenbank erstellen. Bevor Sie eine neue Datenbank erstellen, legen Sie den neuen konstanten Parameterwert fest, indem Sie die Datei `solid.ini` im Verzeichnis `solidDB` bearbeiten.

Im folgenden Beispiel wird einen neue Blockgröße für die Indexdatei festgelegt, indem der Datei `solid.ini` die folgenden Zeilen hinzugefügt werden:

```
[IndexFile]
Blocksize = 4096
```

Nachdem Sie die Datei `solid.ini` bearbeitet und gespeichert haben, verschieben Sie die alte Datenbank und die alten Protokolldateien oder löschen Sie diese und starten Sie solidDB.

Anmerkung: Die Protokollblockgröße kann zwischen Startvorgängen des Servers geändert werden.

Anhang A. Maximale BLOB-Größe berechnen

Zweck

Ein wichtiger Unterschied zwischen speicherinternen Tabellen und plattenbasierten Tabellen liegt darin, dass Spaltenwerte in speicherinternen Tabellen in eine einzige "Seite" passen müssen (die Seitengröße wird in der Konfigurationsdatei `solid.ini` angegeben und ihre Größe beträgt maximal 32 KB). Daher können speicherinterne Tabellen keine zeichenbasierten Dateien oder Binärdateien speichern, die größer als die Seitengröße sind. Kleinere Binärdateien werden jedoch unterstützt.

In diesem Anhang wird die Berechnung der maximalen Größe eines Zeichen- oder Binärspaltenwerts beschrieben, der in Ihre speicherinternen Tabellen passt.

Hintergrund

In vielen Anwendungen werden heute Daten verwendet, die nicht mehr einfach in den Standarddatentypen INT, CHAR usw. gespeichert werden können. Möglicherweise ist stattdessen ein Format, das große Zeichen- oder Binärobjekte unterstützt, besser geeignet. In diesen Fällen können die Daten als *CLOBs* (*Character Long Objects, große Zeichenobjekte*) oder *BLOBs* (*Binary Large Objects, große Binärobjekte*) gespeichert werden. Ein CLOB umfasst bis zu 2 Milliarden interpretierbare Zeichen. Ein BLOB-Datentyp kann praktisch alle Daten umfassen, die als Serie von Binärzahlen (8-Bit-Byte) gespeichert werden können. Normalerweise werden BLOBs zur Speicherung umfangreicher Daten mit variabler Länge verwendet, die nicht einfach als Zahlen oder Zeichen interpretiert werden können. BLOBs können beispielsweise digitalisierten Ton (z. B. Musik auf einer CD), Multimediadateien oder von Sensoren gelesene Zeitreihedaten enthalten.

In solidDB werden BLOBs weitgehend unterstützt und es stehen mehrere unterschiedliche Datentypen zur Auswahl: BINARY, VARBINARY und LONG VARBINARY, von denen der letzte dem Standarddatentyp BLOB entspricht.

CLOB wird mit sechs Datentypen implementiert: CHAR, WCHAR, VARCHAR, WVARCHAR, LONG VARCHAR und LONG WVARCHAR. Die letzten beiden Datentypen entsprechen den Standarddatentypen CLOB und NCLOB. Ausführliche Informationen zu den Datentypen CLOB und BLOB finden Sie in den Abschnitten *Character Data Types* und *Binary Data Types* im *Appendix A* in der Veröffentlichung *IBM solidDB SQL Guide*.

Bei plattenbasierten Tabellen stimmt die Implementierung des BLOB-Speichers von solidDB die Zugriffsgeschwindigkeit mit der erforderlichen Möglichkeit zur Speicherung großer Datenmengen ab. Unabhängig vom Datentyp (VARCHAR, VARBINARY usw.) werden kurze Werte generell in der Tabelle gespeichert, während die Daten von längeren Werten teilweise oder vollständig in einem separaten Bereich in der Baumstruktur des Datenbankspeichers gespeichert werden. Dies ist für den Benutzer vollkommen transparent. Der Benutzer legt einfach den Datentyp fest und solidDB erledigt alles Weitere. Der Zugriff auf Ihre Daten erfolgt immer auf die gleiche Art und es scheint, als würden die Daten in der Tabelle gespeichert, unabhängig von der tatsächlichen physischen Position der Daten. In plattenbasierten Tabellen beträgt die maximale Länge eines VARCHAR- oder eines VARBINARY-Felds 2 GB.

Bei speicherinternen Tabellen werden BLOB-Daten vollständig in der Tabelle selbst gespeichert und die maximale Länge eines BLOB wird durch die ″Blockgröße″ begrenzt (keine Zeile einer speicherinternen Tabelle darf die Länge einer Seite oder eines ″Blocks″ überschreiten). Die Informationen in diesem Anhang sollen Sie bei der Schätzung der Maximalgröße von VARCHAR- oder VARBINARY-Daten unterstützen, die Sie in einer speicherinternen Tabelle speichern können.

Berechnung

Bitte beachten Sie, dass der Algorithmus für die Berechnung des für BLOBs verfügbaren Speicherplatzes nur grob ist. Machen Sie sich eine Kopie der unten stehenden Tabelle und tragen Sie in diese die entsprechenden Werte für Ihre Tabelle ein. Führen Sie die Schritte zur Berechnung des für BLOB-Daten verbleibenden Speicherplatzes aus.

Tabelle 4. Verfügbaren Speicherplatz für BLOB-Daten berechnen

	WERT	EINGABE FÜR WERT	BEDEUTUNG DES WERTS
1		Geben Sie links entweder Ihre Blockgröße oder 32767 ein (je nachdem, welcher Wert kleiner ist). Die Blockgröße ist entweder der Wert, den Sie im Abschnitt ″[IndexFile] BlockSize″ der Konfigurationsdatei `solid.ini` eingegeben haben, oder der in der Veröffentlichung *IBM solidDB Administrator Guide* dokumentierte Standardwert.	Die Blockgröße (Seitengröße) ist die Anzahl Byte in einem ″Block″, analog zu einem Plattenblock. Da jede Zeile in einen Block passen muss, wird hierdurch die maximale Größe einer Zeile angegeben.
2	17	Verwenden Sie den links angegebenen fest codierten Wert.	Dies ist der Systemaufwand pro Seite in Byte.
3	10	Verwenden Sie den links angegebenen fest codierten Wert.	Dies ist der Systemaufwand pro Zeile in Byte. Es wird davon ausgegangen, dass Sie bei großen BLOBs nur 1 Zeile pro Seite haben.
4		Wenn Sie einen expliziten Primärschlüssel für Ihre Tabelle angegeben haben, geben Sie den Wert 10 ein. Geben Sie andernfalls 20 ein.	Dieser Wert stellt die Byte dar, die für Spalten verwendet werden, die der Server automatisch jeder Tabelle hinzufügt.
5		Geben Sie die Anzahl Spalten in Ihrer Tabelle multipliziert mit 2 ein.	Dies ist der Systemaufwand für die Spalten in Byte.
6		Geben Sie die Summe der Größen der Datenspalten mit fester Größe in Ihrer Tabelle ein. (Die Größe jedes Datentyps mit fester Größe können Sie der folgenden Tabelle entnehmen.)	Dieser Wert stellt den von Spalten mit fester Größe belegten Speicherbereich dar.
7		Geben Sie die Anzahl BLOB-Spalten ein.	Dieser Wert stellt die Anzahl der Byte dar, die für die Beendigung von BLOB-Werten verwendet werden (1 Byte pro Wert).
8		Addieren Sie die Werte in den Zeilen 2 bis 7.	Dies ist der gesamte, von allen Werten, außer den BLOB-Werten, belegte Speicherplatz.

Tabelle 4. Verfügbaren Speicherplatz für BLOB-Daten berechnen (Forts.)

	WERT	EINGABE FÜR WERT	BEDEUTUNG DES WERTS
9		Subtrahieren Sie Zeile 8 von Zeile 1.	Dieser Wert ist die ungefähre Anzahl der Byte, die für BLOB-Daten zur Verfügung stehen. Wenn Ihre Tabelle eine einzelne BLOB-Spalte enthält, ist dies die ungefähre maximale Größe dieses BLOB-Werts.

Anmerkung: Die maximale Blockgröße beträgt 64 K. Die maximale Zeilengröße (und somit auch die maximale BLOB-Größe) beträgt jedoch lediglich 32 K (32 K - 1 oder 32767). Wenn Ihre Blockgröße 64 K oder 32 K beträgt, geben Sie bitte 32767 anstelle der Blockgröße in Zeile 1 der Tabelle ein.

Die Tabelle unten gibt die Anzahl der Byte an, die zur Speicherung eines Werts jeden Datentyps mit fester Größe erforderlich sind. Zur Speicherung eines Werts des Typs SQL FLOAT sind beispielsweise 8 Byte erforderlich.

Tabelle 5. Zur Speicherung von Werten erforderliche Anzahl Byte

Datentyp	Speichergröße (in Byte)
TINYINT	1
SMALLINT	2
INT	4
BIGINT	8
DATE/TIME/TIMESTAMP	11
FLOAT / DOUBLE PRECISION	8
REAL	4
NUMERIC / DECIMAL	11
CHAR / VARCHAR / LONG VARCHAR	Zeichenlänge(Spaltenwert) + 1
WCHAR / WVARCHAR / LONG WVARCHAR	Zeichenlänge(Spaltenwert) * 2 + 1
BINARY / VARBINARY / LONG VARBINARY	Oktettlänge(Spaltenwert) + 1

Anhang B. Speicherbedarf berechnen

Mithilfe der Informationen in diesem Anhang können Sie die Größe des Speichers oder des Plattenspeichers schätzen, der zum Speichern einer Tabelle und ihrer Indizes im Speicher oder auf der Platte erforderlich ist.

Bitte beachten Sie, dass die hier angegebenen Formeln u. a. aus folgenden Gründen nicht präzise sind:

- solidDB komprimiert einige Daten.
- Der für Daten mit variabler Länge (z. B. VARCHAR) erforderliche Speicherplatz variiert abhängig von den Längen der gespeicherten Werte.
- In den speicherinternen Datenstrukturen wird nicht zwangsläufig dieselbe Anzahl Zeiger für jeden Datensatz gespeichert.

In dieser Beschreibung wird in den meisten Fällen davon ausgegangen, dass die Daten nicht komprimiert sind und es wird zudem die maximale Anzahl von Zeigern angenommen. Daher sind die mithilfe dieser Formeln erhaltenen Ergebnisse normalerweise eher konservativ, d. h., mit diesen Formeln wird der erforderliche Speicherplatz normalerweise höher geschätzt.

Für die in den weiteren Abschnitten aufgeführten Formeln gilt bezüglich der Notation Folgendes:

```
Summe_von(x)
```

Dies bedeutet, dass die Summe der Größen von jedem ″x″ berechnet wird. Beispiele:

```
Summe_von(Spaltengröße)
```

Dies bedeutet, dass die Summe der Größen aller Spalten in der Tabelle oder im Index berechnet wird.

```
Summe_von(Indexgrößen)
```

Dies bedeutet, dass die Summe der Größen aller Indizes für die Tabelle berechnet wird.

Plattenbasierte Tabellen

Die allgemeine Formel für den für eine plattenbasierte Tabelle erforderlichen Platz lautet:

```
Prüfpunktfaktor x (Tabellengröße + Summe_von(Indexgrößen))
Dabei gilt Folgendes:
        Der Prüfpunktfaktor liegt zwischen 1,0 und 3,0 (siehe Erläuterung unten) und
        Tabellengröße =
            1,4 x Zeilen x (Summe_von(Spaltengröße + 1) + 12)
Dabei gilt Folgendes:
        Zeilen ist die Anzahl Zeilen und
        Summe_von(Spaltengröße + 1) ist die Summe der Größen der Spalten
            plus ein Byte pro Spalte.
Die Spaltengrößen werden später in einer Tabelle dargestellt.
```

Für jeden plattenbasierten Index ist die Indexgröße:

`1,4 x Zeilen x (`*`Primärschlüsselgröße + Indexgröße`*`)`

Dabei gilt Folgendes: Primärschlüsselgröße ist die Summe der Größen der Spalten im Primärschlüssel und Indexgröße ist die Summe der Größen der Spalten im Index.

Der Prüfpunktfaktor ist erforderlich, um zu berücksichtigen, dass Prüfpunktoperationen kurzzeitig bis zur dreifachen Größe der Datenbank erfordern können. Während einer Prüfpunktoperation wird eine Kopie jeder der geänderten Seiten in der Datenbank aus dem Speicher auf die Platte gestellt. Wenn alle Seiten in der Datenbank aktualisiert wurden, können so viele Seiten aus dem Speicher kopiert werden wie sich bereits auf der Platte befinden. Darüber hinaus wird der letzte erfolgreiche Prüfpunkt erst gelöscht, nachdem der aktuelle Prüfpunkt erfolgreich abgeschlossen wurde. Daher können sich während eines Prüfpunkts gleichzeitig bis zu 3 Kopien jeder Seite auf der Platte befinden (1 Kopie für die Seite in der Datenbank, 1 Kopie des letzten erfolgreichen Prüfpunkts und 1 Kopie für den aktuellen Prüfpunkt, während dieser ausgeführt wird). Der Prüfpunktfaktor kann daher zwischen 1,0 und 3,0 liegen. Werte, die sich 3,0 nähern, sind in den meisten Datenbanken jedoch selten. Normalerweise reicht ein Wert von 1,5 auch für kleine Datenbanken mit hohen Aktivitätsebenen gut aus. Bitte beachten Sie Folgendes: Je seltener der Prüfpunkt ausgeführt wird, desto höher muss möglicherweise der Prüfpunktfaktor sein.

Anmerkung: Wenn Sie in einem plattenbasierten Index nicht explizit einen Primärschlüssel definieren, verwendet der Server eine von ihm generierte "Zeilennummer" als Primärschlüssel. Dadurch muss der Primärschlüsselindex Datensätze in derselben Reihenfolge speichern, in der sie eingefügt wurden.

Speicherinterne Tabellen

Die allgemeine Formel für eine speicherinterne Tabelle lautet:

Tabellengröße + Summe_von(*Indexgrößen*)

Tabellengröße =

1,3 x Zeilen x (Summe_von(*Spaltengrößen*) + (3 x *Wortgröße*) + (2 * *Spaltenanzahl*) + 2)

Dabei gilt Folgendes: Zeilen ist die Anzahl der Zeilen.

Wortgröße ist die Wortgröße des Systems (z. B. 4 Byte unter 32-Bit-Betriebssystemen und 8 Byte unter 64-Bit-Betriebssystemen).

Spaltenanzahl ist die Anzahl der Spalten.

Summe_von(*Spaltengrößen*) ist die Summe der Größen der Spalten.

Für jeden speicherinternen Index ist die Indexgröße:

1,3 x Zeilen x ((*Verteilungsfaktor* x Summe_von(*Spaltengrößen* + 1)) + (8 x *Wortgröße*) + 4)

Dabei gilt Folgendes: Verteilungsfaktor ist ein Wert zwischen 1,0 und 2,0, der von der Verteilung der Schlüsselwerte abhängt. Wenn sich die Schlüsselwerte stark unterscheiden, verwenden Sie einen Wert, der näher an 2,0 liegt. Wenn sich die Schlüsselwerte stark ähneln, verwenden Sie einen Wert, der näher an 1,0 liegt.

Tabelle mit Spaltengrößen

TINYINT: 2 Byte

SMALLINT: 2 Byte

INT: 4 Byte

BIGINT: 8 Byte

DATE/TIME/TIMESTAMP: 11 Byte

FLOAT / DOUBLE PRECISION: 8 Byte

REAL: 4 Byte

NUMERIC / DECIMAL: 12 Byte

CHAR / VARCHAR / LONG VARCHAR: Zeichenlänge(Spaltenwert) + 5

WCHAR / WVARCHAR / LONG WVARCHAR: Zeichenlänge(Spaltenwert) * 2 + 5

BINARY / VARBINARY / LONG VARBINARY: Oktettlänge(Spaltenwert) + 5

Speicherbelegung ermitteln

Nachdem Sie Ihre Tabellen und Indizes erstellt haben, können Sie die belegte Speichermenge mithilfe des folgenden Befehls ermitteln:

ADMIN COMMAND 'info imdbsize';

Dieser Befehl gibt die gesamte Speicherbelegung der speicherinternen Tabellen und Indizes aus. Die Einheit ist KB.

Ausführliche Informationen

Dieses Kapitel enthält ausführliche Informationen zur Speicherung der Daten in den verschiedenen Baumstrukturen des Speichers. Möglicherweise können Sie mithilfe dieser Informationen die Grundlagen der zuvor aufgeführten Formeln besser verstehen.

Plattenbasierte Tabellen

In plattenbasierten Tabellen werden Daten und Indizes in einem B-Tree (B-Baum) gespeichert. Jeder Eintrag im Baum belegt Platz für den Header und die Daten.

Der von den tatsächlichen Daten belegte Platz kann mithilfe der in diesem Dokument zuvor aufgeführten Tabelle mit den Spaltengrößen berechnet werden. Die Werte in dieser Tabelle stellen die maximalen Längen dar. Daten mit variabler Länge (beispielsweise VARCHAR) oder komprimierbare Daten können weniger Byte erfordern.

Darüber hinaus ist in plattenbasierten Tabellen für den Server 1 zusätzliches Byte pro Spalte erforderlich. Dieses Byte wird als Teil des Längenanzeigers (der auch als Nullanzeiger dient) verwendet.

Der Header für jede Zeile belegt 12 Byte:

Tabelle 6. Byte für Header

Anzahl Byte	**Verwendung**
3 Byte	Zeilenheader
3 Byte	Tabellen-ID
6 Byte	Zeilenversion

Wenn eine plattenbasierte Tabelle Indizes (außer dem Primärschlüssel) enthält, muss die Größe der Einträge in diesen Indizes mithilfe derselben Richtlinie separat geschätzt werden. Ein Indexeintrag enthält die folgenden Komponenten:

- im Index definierte Spalten
- Spalten des Primärschlüssels der Tabelle
- einen Zeilenheader (12 Byte)

Zusätzlich enthalten die Datenbankseiten in der Regel einen leeren Bereich (z. B. 20 - 40 %). Aus diesem Grund enthalten die Formeln einen Multiplikator von 1,4 sowohl für Tabellen als auch für Indizes.

Beispiel: Wir haben eine plattenbasierte Tabelle:

```
CREATE TABLE Subskribent (
             id INTEGER NOT NULL PRIMARY KEY,
             name VARCHAR(50),
             salary FLOAT) ;
```

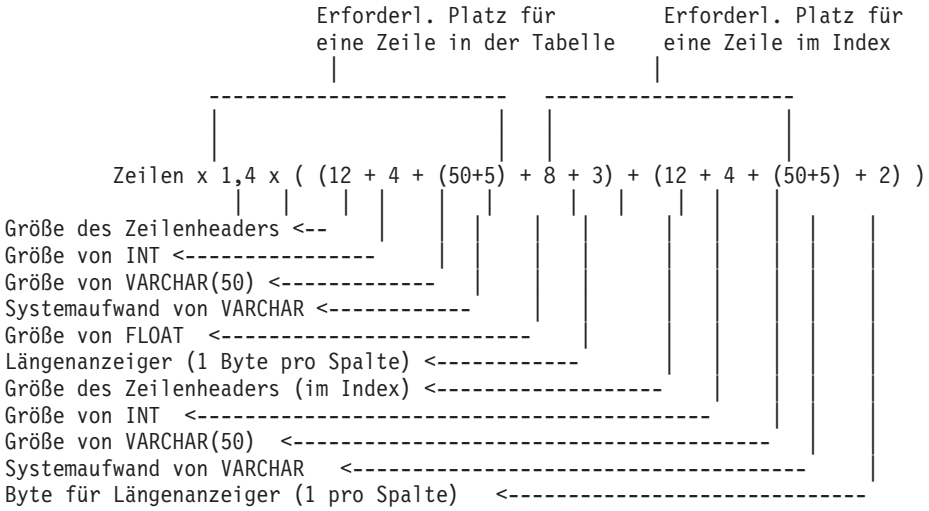
Darüber hinaus haben wir einen Sekundärindex erstellt:

```
CREATE INDEX Subskribent_Indexname ON Subskribent (Name);
```

Der Indexeintrag enthält die Spalte "NAME". Darüber hinaus enthält er die Primärschlüsselspalte, die in diesem Fall "ID" ist. Der für diesen Index erforderliche Platz muss separat geschätzt werden. Vorausgesetzt, der Faktor für den leeren Bereich ist 1,4, wird die Gesamtgröße der plattenbasierten Tabelle wie folgt berechnet:

```
Zeilen x 1,4                  // 1,4 = die Schätzung für den leeren Bereich
   x ( (12 + 4 + (50+5) + 8 + 3)   // Größe des Tabelleneintrags
        + (12 + 4 + (50+5) + 2) )   // Größe des Sekundärindexeintrags
```

Alternative Darstellung:



Speicherinterne Tabellen

Der für speicherinterne Tabellen erforderliche Platz wird unterschiedlich geschätzt.

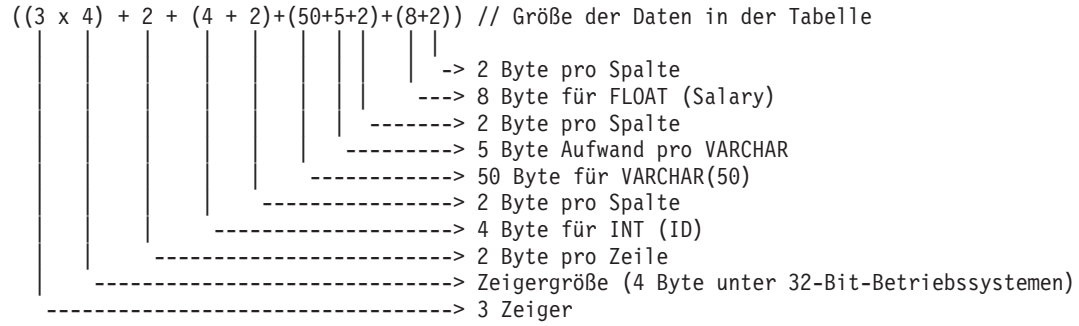
Die Größe jedes Eintrags ist die kombinierte Größe der Daten der Tabelle plus drei Speicherzeiger (jeweils 4 Byte unter 32-Bit-Betriebssystemen bzw. jeweils 8 Byte unter 64-Bit-Betriebssystemen) Aufwand pro Zeile. Darüber hinaus fällt ein Aufwand von zwei Byte pro Zeile und zwei Byte pro Spalte der Zeile an. Beachten Sie, dass es nicht notwendig ist, 1 Byte pro Spalte hinzuzufügen, um den Längenanzeiger zu berücksichtigen, da dieser bereits in den 2 Byte pro Zeile enthalten ist. Die Hauptspeichertabellen können zudem Indizes enthalten, die beim Serverstart gefüllt werden. Jeder Indexeintrag enthält die Daten der im Index definierten Spalten. Jeder Indexeintrag enthält zusätzlich bis zu acht Speicherzeiger. (Für einen speicherinternen Index ist KEINE Kopie des Primärschlüssels erforderlich.)

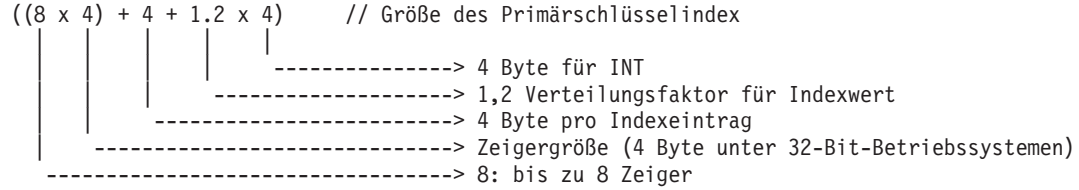
Darüber hinaus fällt abhängig von den tatsächlichen Datenwerten des Index weiterer Aufwand an. Dies ist ein Prozentsatz der Datenmenge des Index. Es kann kein genauer Wert angegeben werden, da der Wert von der Verteilung der Schlüsselwerte abhängt. Der Multiplikator liegt jedoch zwischen 1,0 und 2,0.

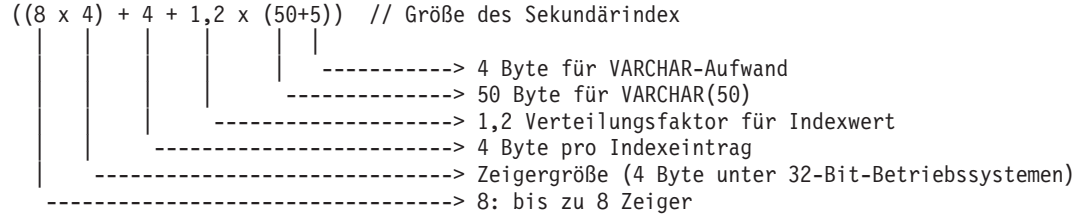
Zusätzlich benötigt die Indexstruktur selbst durchschnittlich 4 Byte pro Indexeintrag (d. h. pro Zeile).

Für die Beispieltabelle und den Beispielindex oben kann die Speicherbelegung in einem 32-Bit-Betriebssystem wie folgt geschätzt werden:

```
Zeilen * 1,3 x (
  ((3 x 4) + 2 + (4 + 2) + (50+5+2) + (8+2)) // Größe der Daten in der Tabelle
 + ((8 x 4) + 4 + 1.2 x 4)       // Größe des Primärschlüsselindex
 + ((8 x 4) + 4 + 1.2 x (50+5))  // Größe des Sekundärindex
)
```







Verwenden Sie unter einem 64-Bit-Betriebssystem eine Speicherzeigergröße von 8 Byte statt von 4 Byte.

Der Faktor 1,2 in der obigen Schätzung ist der ″TRIE-Verteilungsfaktor für den Indexwert″, dessen genauer Wert von den tatsächlichen Werten der indexierten Spalte abhängt. Der Wert liegt in der Regel zwischen 1 und 2. Bei der wahlfreien Werteverteilung liegt der Wert näher an 2,0 und bei der sequenziellen Werteverteilung liegt er näher an 1,0. 4 Byte ist der von einem Indexeintrag durchschnittlich benötigte Datenaufwand.

Mit dem Faktor 1,3 wird der interne Aufwand der Speicherzuordnungsfunktion berücksichtigt.

Anmerkung:

Indizes von Hauptspeichertabellen werden bei jedem Serverstart dynamisch erstellt. Sie werden nie auf die Platte geschrieben und belegen daher keinen Plattenspeicherplatz. Die Tabellen selbst werden jedoch beim Herunterfahren des Servers (sowie bei Prüfpunkten) auf die Platte geschrieben. Aus diesem Grund muss der gesamte Ihnen zur Verfügung stehende Plattenspeicherplatz groß genug sein, um sowohl die plattenbasierten Tabellen als auch die speicherinternen Tabellen zu speichern.

Anhang C. Konfigurationsparameter

Parameter werden entsprechend den Abschnittskategorien in der Konfigurationsdatei `solid.ini` gruppiert. Die zur speicherinternen solidDB-Datenbank gehörenden Parameter werden im Abschnitt ″[MME]″ der Konfigurationsdatei gespeichert. Darüber hinaus wird ein Parameter, DefaultStoreIsMemory, im Abschnitt [General] gespeichert.

Sie können Konfigurationsparameter ändern, indem Sie entweder die Konfigurationsdatei `solid.ini` manuell bearbeiten oder den folgenden Befehl im solidDB SQL Editor eingeben:

```
ADMIN COMMAND 'parameter Abschnittsname.Parametername=Wert'
```

Beispiel:

ADMIN COMMAND ’parameter mme.imdbmemorylimit=1gb’;

Anmerkung:

Da der Server die Konfigurationsdatei nur beim Start liest, werden Änderungen an der Konfigurationsdatei erst beim nächsten Start des Servers wirksam.

Weiter unten finden Sie die vollständige Liste der IMDB-bezogenen Konfigurationsparameter.

Abschnitt 'General'

Tabelle 7. MME-bezogene Parameter im Abschnitt [General]

[General]	Beschreibung	Werkseitige Einstellung	Zugriffsmodus
DefaultStoreIsMemory	Wenn dieser Parameter auf "Yes" (JA) gesetzt ist, werden neue Tabellen als speicherinterne Tabellen erstellt, wenn sie mit der Anweisung CREATE TABLE ohne explizite Angabe der Klausel STORE erstellt werden. Wenn der Parameter auf "No" (Nein) gesetzt ist, werden neue Tabellen standardmäßig auf der Platte gespeichert. Sie können den Standardwert überschreiben, indem Sie die Klausel STORE in der Anweisung CREATE TABLE angeben. Dieser Parameter kann nur für Produkte angewendet werden, die die speicherinterne solidDB-Datenbank unterstützen. Systemtabellen werden auf der Platte gespeichert, auch wenn dieser Parameter auf "Yes" gesetzt ist.	JA	RW (Schreib-/Lesezugriff)
MultipropcessingLevel	Definiert die Anzahl der physischen Verarbeitungseinheiten (Prozessor, Kerne), die im Computersystem zur Verfügung stehen. Sie können den gemeinsamen Zugriff von Schreiboperationen in der Datenbank verbessern, indem Sie diesen Wert so festlegen, dass er mit der Anzahl der in Ihrem System vorhandenen Prozessoren (Kerne) übereinstimmt.	4	RW/Startup (Schreib-/Lesezugriff/Start)

Abschnitt 'MME'

Anmerkung:

Der Parameter 'DefaultStoreIsMemory' (im Abschnitt '[General]' der Datei `solid.ini`) gilt auch für die speicherinterne solidDB-Datenbank.

Tabelle 8. MME-Parameter

[MME]	Beschreibung	Werkseitige Einstellung	Zugriffsmodus

ImdbMemoryLimit	Dieser Parameter setzt eine Obergrenze für die Speichermenge (virtueller Speicher), die der Server für speicherinterne Tabellen und Indizes für speicherinterne Tabellen zuordnet. ″Speicherinterne Tabellen″ umfassen temporäre Tabellen, transiente Tabellen sowie ″normale″ (persistente) speicherinterne Tabellen.
	
	Der Grenzwert kann in Byte, Kilobyte (kb), Megabyte (mb) oder Gigabyte (gb) angegeben werden. Beispiel:
	`ImdbMemoryLimit=1073741824`
	`ImdbMemoryLimit=1048576kb`
	`ImdbMemoryLimit=1024mb`
	`ImdbMemoryLimit=1gb`
	
	Wenn Sie den Wert ″0″ verwenden, bedeutet dies ″kein Grenzwert″.
	
	Als allgemeine Regel sollten Sie bei Servern mit bis zu 1 GB Speicher für speicherinterne Tabellen normalerweise maximal 30 bis 70 Prozent des physischen Hauptspeichers des Systems zuordnen. Je mehr Speicher das System hat, desto größer ist der Prozentsatz, den Sie für speicherinterne Tabellen verwenden können.
	Anmerkung: Dieser Parameter gilt nur für solidDB-Hauptspeicher-Engine-Tabellen. Er gilt nicht für andere Versionen von solidDB oder für festplattenbasierte Tabellen.
	
	Sie können diesen Parameter mit dem folgenden Befehl ändern:
	`ADMIN COMMAND 'parameter MME.ImdbMemoryLimit=n[kb|mb|gb]';`
	
	Dabei ist 'n' eine positive Ganzzahl. Während der Server ausgeführt wird, können Sie diesen Wert nur vergrößern, aber nicht verkleinern. Der Befehl wird sofort wirksam. Der neue Wert wird beim Herunterfahren zurück in die Datei `solid.ini` geschrieben.
	
	Vorsicht:
	Stellen Sie sicher, dass Ihre speicherinternen Tabellen in den verfügbaren physischen Hauptspeicher passen. Wenn Sie den verfügbaren physischen Hauptspeicher überschreiten, wird die Leistung erheblich verringert. Wenn Sie den gesamten verfügbaren virtuellen Speicher belegen, schränkt der Server Einfügungen und Aktualisierungen usw. sofort ein und gibt Fehlercodes zurück.
		0
		
		Einheit: 1 Byte kb=KB mb=MB gb=GB
			RW (Schreib-/ Lesezugriff)

Tabelle 8. MME-Parameter (Forts.)

[MME]	Beschreibung	Werkseitige Einstellung	Zugriffsmodus
ImdbMemoryLowPercentage	Wenn Sie den Parameter **ImdbMemoryLimit** gesetzt haben, können Sie diesen zusätzlichen Parameter setzen, damit Sie gewarnt werden, bevor Sie den gesamten Speicher belegen. Mit dem Parameter **ImdbMemoryLowPercentage** können Sie den Prozentsatz des Speichers angeben, den Sie belegen können, bevor der Server das Einfügen von Zeilen in speicherinterne Tabellen usw. einschränkt. Wenn beispielsweise **ImdbMemoryLimit** 1000 MB und **ImdbMemoryLowPercentage** 90 (Prozent) ist, akzeptiert der Server keine Einfügungen mehr, wenn Sie 900 MB Speicher für Ihre speicherinternen Tabellen belegt haben. Gültige Werte liegen zwischen 60 und 99 (Prozent). **Anmerkung:** Dieser Parameter gilt nur für solidDB-Hauptspeicher-Engine-Tabellen. Er gilt nicht für andere Versionen von solidDB oder für festplattenbasierte Tabellen.	90	RW/Startup (Schreib-/ Lesezugriff/Start)
ImdbMemoryWarningPercentage	Dieser Parameter setzt einen Warngrenzwert für die IMDB-Hauptspeichergröße. Der Warngrenzwert wird als Prozentsatz des Parameterwerts von **ImdbMemoryLimit** ausgedrückt. Wenn der Grenzwert von **ImdbMemoryWarningPercentage** überschritten wird, wird ein Systemereignis ausgegeben. Der Parameterwert von **ImdbMemoryWarningPercentage** wird automatisch auf Konsistenz geprüft. Er muss niedriger als der Parameterwert von **ImdbMemoryLimit** sein. **Anmerkung:** Dieser Parameter gilt nur für solidDB-Hauptspeicher-Engine-Tabellen. Er gilt nicht für andere Versionen von solidDB oder für festplattenbasierte Tabellen.	80	RW/Startup (Schreib-/ Lesezugriff/Start)

Tabelle 8. MME-Parameter (Forts.)

[MME]	Beschreibung	Werkseitige Einstellung	Zugriffsmodus
LockEscalationEnabled	Wenn der Server Sperren verwenden muss, um Konflikte beim gemeinsamen Zugriff zu verhindern, sperrt er normalerweise einzelne Zeilen. Das bedeutet, dass jeder Benutzer nur Auswirkungen auf diejenigen anderen Benutzer hat, die dieselbe(n) Zeile(n) verwenden wollen. Je mehr Zeilen jedoch gesperrt sind, desto mehr Zeit braucht der Server für die Überprüfung auf Konflikte verursachende Sperren. In einigen Fällen ist es sinnvoll, eine ganze Tabelle zu sperren statt zahlreiche Zeilen in dieser Tabelle. Wenn **LockEscalationEnabled** auf ″yes″ gesetzt ist, wird die Sperrstufe von Zeilenebene auf Tabellenebene eskaliert, nachdem eine bestimmte Anzahl von Zeilen (in derselben Tabelle) innerhalb der aktuellen Transaktion gesperrt wurde. Die Sperreneskalation verbessert die Leistung, verringert jedoch den gemeinsamen Zugriff, da dies bedeutet, dass andere Benutzer dieselbe Tabelle temporär selbst dann nicht verwenden können, wenn sie andere Zeilen in dieser Tabelle verwenden wollen. In der Beschreibung des Parameters **LockEscalationLimit** finden Sie auch weitere Informationen hierzu. Der Wert kann ″yes″ oder ″no″ sein. **Anmerkung:** Dieser Parameter gilt nur für speicherinterne Tabellen.	no	RW/Startup (Schreib-/ Lesezugriff/Start)
LockEscalationLimit	Wenn **LockEscalationEnabled** auf ″yes″ gesetzt ist, zeigt dieser Parameter an, wie viele Zeilen (in einer einzelnen Tabelle) gesperrt werden müssen, bevor der Server die Sperrstufe von Zeilenebene auf Tabellenebene eskaliert. In der Beschreibung des Parameters **LockEscalationEnabled** finden Sie weitere Informationen. Der Wert kann eine beliebige Zahl von 1 bis 2.147.483.647 (2^32-1) sein. **Anmerkung:** Dieser Parameter gilt nur für speicherinterne Tabellen.	1000	RW/Startup (Schreib-/ Lesezugriff/Start)

Tabelle 8. MME-Parameter (Forts.)

[MME]	Beschreibung	Werkseitige Einstellung	Zugriffsmodus
LockHashSize	Der Server verwendet eine Hashtabelle (Array), um Information zu Sperren zu speichern. Wenn die Größe des Arrays wesentlich unterschätzt wurde, wird die Leistung verringert. Eine zu große Hashtabelle hat keine direkten Auswirkungen auf die Leistung, sie verursacht jedoch einen Speicheraufwand. Der Parameter **LockHashSize** ermittelt die Anzahl Elemente in einer Hashtabelle. Diese Informationen sind erforderlich, wenn der Server eine pessimistische Steuerung des gemeinsamen Zugriffs (Sperren) verwendet. Der Server verwendet separate Arrays für speicherinterne Tabellen und plattenbasierte Tabellen. Dieser Parameter gilt für speicherinterne Tabellen. Im Allgemeinen gilt Folgendes: Je mehr Sperren Sie benötigen, desto größer muss dieses Array sein. Die Berechnung der Anzahl der von Ihnen benötigten Sperren ist jedoch schwierig. Daher müssen Sie wahrscheinlich den besten Wert für Ihre Anwendungen durch Ausprobieren herausfinden. Sie geben als Wert die Anzahl der Hashtabelleneinträge ein. Jeder Tabelleneintrag hat eine Größe von einem Zeiger (4 Byte in 32-Bit-Architekturen). Wenn Sie daher beispielsweise eine Hashtabellengröße von 1.000.000 wählen, beträgt die erforderliche Speichermenge 4.000.000 Byte (bei 32-Bit-Zeigern).	1000000	RW/Startup (Schreib-/ Lesezugriff/Start)
MaxCacheUsage	Der Wert des Parameters **MaxCacheUsage** begrenzt den Umfang des D-Tabellencache, der während des Prüfpunktverfahrens für M-Tabellen verwendet wird. Es wird erwartet, dass der Wert in Byte angegeben wird. Unabhängig vom Wert des Parameters **MaxCacheUsage** wird maximal die Hälfte des D-Tabellencache (**IndexFile.CacheSize**) für das Prüfpunktverfahren für M-Tabellen verwendet. Der Wert **MaxCacheUsage=0** setzt den Wert auf unbegrenzt. Das bedeutet eine Cachebelegung von **IndexFile.CacheSize**/2.	8 MB	RW/Startup (Schreib-/ Lesezugriff/Start)
NumberOfMemoryPools	Dieser Parameter definiert die Anzahl der globalen Hauptspeicherpools. Größere Werte bieten auf Systemen mit mehreren Kernen bei bestimmten Ladeszenarien eine höhere Leistung. Sie vergrößern jedoch auch den Speicherpuffer und damit die Größe des Serverprozesses. Der Mindestwert ist 1. Es gibt keinen Maximalwert; die Anzahl der Kerne im System sollte jedoch nicht überschritten werden.	1	RW/Startup (Schreib-/ Lesezugriff/Start)

Tabelle 8. MME-Parameter (Forts.)

[MME]	Beschreibung	Werkseitige Einstellung	Zugriffsmodus
ReleaseMemoryAtShutdown	Wenn dieser Parameter auf ″yes″ gesetzt ist, wird der Server angewiesen, beim Herunterfahren explizit den von speicherinternen Tabellen verwendeten Speicherbereich freizugeben und sich nicht darauf zu verlassen, dass das Betriebssystem den gesamten, diesem Prozess zugeordneten Speicher bereinigt. Bei einigen Betriebssystemen (wie VxWorks) müssen Sie diesen Parameter auf möglicherweise ″yes″ setzen, um sicherzustellen, dass der gesamte Speicherbereich freigegeben wird. Die möglichen Werte sind ″yes″ und ″no″. Die werkseitige Einstellung ist ″no″, da der Server so schneller heruntergefahren wird.	No	RW/Startup (Schreib-/ Lesezugriff/Start)
RestoreThreads	Definiert die maximale Anzahl Threads, die beim Durchführen eines Restores für eine speicherinterne Datenbank während des Datenbankstarts verwendet wird. Ein Wert von ’1’ bedeutet, dass das Laden mit Einzelthread ausgeführt wird. Der Restore der speicherinternen Datenbank ordnet pro Tabelle einen Thread zu, wenn die Anzahl der Tabellen kleiner oder gleich der Anzahl des Parameterwerts ist. Der maximale gemeinsame Zugriff wird erreicht, wenn der Parameterwert kleiner als die folgenden beiden Werte ist: Anzahl der Kerne/Verarbeitungseinheiten und die Anzahl der Tabellen in der Datenbank. Mögliche Werte sind 1..2^31 -1.	4	RW/Startup (Schreib-/ Lesezugriff/Start)

Index

Sonderzeichen

A

B

C

D

E

G

H

I

K

L

M

N

P

R

S

T

Z

Bemerkungen

Dieses Produkt ist durch die folgenden US-Patente geschützt: 6144941, 7136912, 6970876, 7139775, 6978396, 7266702, 7406489 und 7502796.

Die US-amerikanische Export Control Classification Number (ECCN) für dieses Produkt lautet 5D992b.

Die vorliegenden Informationen wurden für Produkte und Services entwickelt, die auf dem deutschen Markt angeboten werden.

Möglicherweise bietet IBM die in dieser Dokumentation beschriebenen Produkte, Services oder Funktionen in anderen Ländern nicht an. Informationen über die gegenwärtig im jeweiligen Land verfügbaren Produkte und Services sind beim zuständigen IBM Ansprechpartner erhältlich. Hinweise auf IBM Lizenzprogramme oder andere IBM Produkte bedeuten nicht, dass nur Programme, Produkte oder Services von IBM verwendet werden können. An Stelle der IBM Produkte, Programme oder Services können auch andere, ihnen äquivalente Produkte, Programme oder Services verwendet werden, solange diese keine gewerblichen oder anderen Schutzrechte der IBM verletzen. Die Verantwortung für den Betrieb von Produkten, Programmen und Services anderer Anbieter liegt beim Kunden.

Für in diesem Handbuch beschriebene Erzeugnisse und Verfahren kann es IBM Patente oder Patentanmeldungen geben. Mit der Auslieferung dieses Handbuchs ist keine Lizenzierung dieser Patente verbunden. Lizenzanforderungen sind schriftlich an folgende Adresse zu richten (Anfragen an diese Adresse müssen auf Englisch formuliert werden):

IBM Director of Licensing
IBM Europe, Middle East & Africa
Tour Descartes
2, avenue Gambetta
92066 Paris La Defense
France

Trotz sorgfältiger Bearbeitung können technische Ungenauigkeiten oder Druckfehler in dieser Veröffentlichung nicht ausgeschlossen werden. Die hier enthaltenen Informationen werden in regelmäßigen Zeitabständen aktualisiert und als Neuausgabe veröffentlicht. IBM kann ohne weitere Mitteilung jederzeit Verbesserungen und/oder Änderungen an den in dieser Veröffentlichung beschriebenen Produkten und/oder Programmen vornehmen.

Verweise in diesen Informationen auf Websites anderer Anbieter werden lediglich als Service für den Kunden bereitgestellt und stellen keinerlei Billigung des Inhalts

dieser Websites dar. Das über diese Websites verfügbare Material ist nicht Bestandteil des Materials für dieses IBM Produkt. Die Verwendung dieser Websites geschieht auf eigene Verantwortung.

Werden an IBM Informationen eingesandt, können diese beliebig verwendet werden, ohne dass eine Verpflichtung gegenüber dem Einsender entsteht.

Lizenznehmer des Programms, die Informationen zu diesem Produkt wünschen mit der Zielsetzung: (i) den Austausch von Informationen zwischen unabhängig voneinander erstellten Programmen und anderen Programmen (einschließlich des vorliegenden Programms) sowie (ii) die gemeinsame Nutzung der ausgetauschten Informationen zu ermöglichen, wenden sich an folgende Adresse:

IBM Canada Limited
Office of the Lab Director
8200 Warden Avenue
Markham, Ontario
L6G 1C7
CANADA

Die Bereitstellung dieser Informationen kann unter Umständen von bestimmten Bedingungen - in einigen Fällen auch von der Zahlung einer Gebühr - abhängig sein.

Die Lieferung des im Dokument aufgeführten Lizenzprogramms sowie des zugehörigen Lizenzmaterials erfolgt auf der Basis der IBM Rahmenvereinbarung bzw. der Allgemeinen Geschäftsbedingungen von IBM, der IBM Internationalen Nutzungsbedingungen für Programmpakete oder einer äquivalenten Vereinbarung.

Alle in diesem Dokument enthaltenen Leistungsdaten stammen aus einer kontrollierten Umgebung. Die Ergebnisse, die in anderen Betriebsumgebungen erzielt werden, können daher erheblich von den hier erzielten Ergebnissen abweichen. Einige Daten stammen möglicherweise von Systemen, deren Entwicklung noch nicht abgeschlossen ist. Eine Gewährleistung, dass diese Daten auch in allgemein verfügbaren Systemen erzielt werden, kann nicht gegeben werden. Darüber hinaus wurden einige Daten unter Umständen durch Extrapolation berechnet. Die tatsächlichen Ergebnisse können davon abweichen. Benutzer dieses Dokuments sollten die entsprechenden Daten in ihrer spezifischen Umgebung prüfen.

Alle Informationen zu Produkten anderer Anbieter stammen von den Anbietern der aufgeführten Produkte, deren veröffentlichten Ankündigungen oder anderen allgemein verfügbaren Quellen. IBM hat diese Produkte nicht getestet und kann daher keine Aussagen zu Leistung, Kompatibilität oder anderen Merkmalen machen. Fragen zu den Leistungsmerkmalen von Produkten anderer Anbieter sind an den jeweiligen Anbieter zu richten.

Die oben genannten Erklärungen bezüglich der Produktstrategien und Absichtserklärungen von IBM stellen die gegenwärtige Absicht von IBM dar, unterliegen Änderungen oder können zurückgenommen werden und repräsentieren nur die Ziele von IBM.

Diese Veröffentlichung enthält Beispiele für Daten und Berichte des alltäglichen Geschäftsablaufes. Sie sollen nur die Funktionen des Lizenzprogramms illustrieren; sie können Namen von Personen, Firmen, Marken oder Produkten enthalten. Alle diese Namen sind frei erfunden; Ähnlichkeiten mit tatsächlichen Namen und Adressen sind rein zufällig.

COPYRIGHTLIZENZ:

Diese Veröffentlichung enthält Musteranwendungsprogramme, die in Quellensprache geschrieben sind und Programmiertechniken in verschiedenen Betriebsumgebungen veranschaulichen. Sie dürfen diese Musterprogramme kostenlos kopieren, ändern und verteilen, wenn dies zu dem Zweck geschieht, Anwendungsprogramme zu entwickeln, zu verwenden, zu vermarkten oder zu verteilen, die mit der Anwendungsprogrammierschnittstelle für die Betriebsumgebung konform sind, für die diese Musterprogramme geschrieben werden. Diese Beispiele wurden nicht unter allen denkbaren Bedingungen getestet. Daher kann IBM die Zuverlässigkeit, Wartungsfreundlichkeit oder Funktion dieser Programme weder zusagen noch gewährleisten.

Kopien oder Teile der Musterprogramme bzw. daraus abgeleiteter Code müssen folgenden Copyrightvermerk beinhalten:

© (Name Ihrer Firma) (Jahr). Teile des vorliegenden Codes wurden aus Musterprogrammen der IBM Corporation abgeleitet.

Marken

IBM, das IBM Logo, ibm.com, Solid, solidDB, InfoSphere, DB2, Informix und WebSphere sind Marken oder eingetragene Marken der International Business Machines Corporation in den USA und/oder anderen Ländern. Sind diese und weitere Markennamen von IBM bei ihrem ersten Vorkommen in diesen Informationen mit einem Markensymbol (® oder ™) gekennzeichnet, bedeutet dies, dass IBM zum Zeitpunkt der Veröffentlichung dieser Informationen Inhaber der eingetragenen Marken oder der Common-Law-Marken (common law trademarks) in den USA war. Diese Marken können auch eingetragene Marken oder Common-Law-Marken in anderen Ländern sein. Eine aktuelle Liste der IBM Marken finden Sie auf der Webseite Copyright and trademark information (www.ibm.com/legal/copytrade.shtml).

Java™ und alle auf Java basierenden Marken und Logos sind Marken von Sun Microsystems, Inc. in den USA und/oder anderen Ländern.

Linux® ist eine eingetragene Marke von Linus Torvalds in den USA und/oder anderen Ländern.

Microsoft und Windows sind Marken der Microsoft Corporation in den USA und/oder anderen Ländern.

UNIX ist eine eingetragene Marke von The Open Group in den USA und anderen Ländern.

Weitere Unternehmens-, Produkt- oder Servicenamen können Marken anderer Hersteller sein.

SC12-4375-00