IBM solidDB
Version 7.0

Speicherinterne Datenbank Benutzerhandbuch

SC12-4630-04

Hinweis

Vor Verwendung dieser Informationen und des darin beschriebenen Produkts sollten die Informationen unter „Bemerkungen" auf Seite 43 gelesen werden.

Erste Ausgabe, vierte Überarbeitung

Diese Ausgabe bezieht sich auf Version 7.0 Fixpack 5 von IBM solidDB (Produktnummer 5724-V17) und alle nachfolgenden Releases und Modifikationen, bis dieser Hinweis in einer Neuausgabe geändert wird.

Diese Veröffentlichung ist eine Übersetzung des Handbuchs
IBM solidDB, Version 7.0, InMemory Database User's Guide,
IBM Form SC27-3845-04,
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:
TSC Germany
Kst. 2877
März 2013

Inhaltsverzeichnis

Tabellen

Zusammenfassung der Änderungen

Änderungen für Überarbeitung 04

- Redaktionelle Korrekturen.

Änderungen für Überarbeitung 03

- Redaktionelle Korrekturen.

Änderungen für Überarbeitung 02

- Informationen zur Steuerung des gemeinsamen Zugriffs im Abschnitt Persistente speicherinterne Tabellen wurden aktualisiert.

Änderungen für Überarbeitung 01

- Redaktionelle Korrekturen.

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 Hochverfügbarkeitskonfiguration (High Availability, HA) implementieren. Sie können auch die Komponente für die erweiterte Replikation (Advanced Replication) implementieren, mit der mehrere solidDB-Server mit Hauptspeicher-Engines und plattenbasierten Engines 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 zudem 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.

Tabelle 1. Typografische Konventionen (Forts.)

Format	Verwendungszweck
`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 angegeben werden müssen.
Administratorhandbuch	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 Strich 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 Übersicht über die speicherinternen Features von solidDB

Die Hauptspeicher-Engine von solidDB verbindet die hohe Leistung von speicherinternen Tabellen mit der nahezu unbegrenzten Kapazität von plattenbasierten Tabellen. 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.

1.1 Speicherinterne und plattenbasierte Tabellen

Wenn es sich bei einer Tabelle um eine speicherinterne Tabelle (M-Tabelle) handelt, wird der gesamte Inhalt der Tabelle im Speicher gespeichert, sodass der Datenzugriff so schnell wie möglich erfolgen kann. Bei einer plattenbasierten Tabelle (D-Tabelle) werden die Daten dagegen vorwiegend auf der Platte gespeichert und der Server kopiert jeweils nur einen kleinen Teil der Daten in den Speicher.

Hinsichtlich des Anwendungsdesigns gibt es zwischen speicherinternen und plattenbasierten Tabellen kaum Unterschiede.

- Beide Tabellentypen bieten vollständige Datenpersistenz, sofern nicht anders angegeben.
- Sie können für beide Tabellentypen dieselben Abfragetypen ausführen.
- Sie können plattenbasierte und speicherinterne Tabellen in derselben SQL-Abfrage oder -Transaktion kombinieren.
- Beide Tabellentypen können mit Indizes, Triggern, gespeicherten Prozeduren und anderen allgemeinen Datenbankobjekten verwendet werden.
- Beide Tabellentypen lassen auch Integritätsbedingungen zu, einschließlich Integritätsbedingungen über Primärschlüssel und Integritätsbedingungen über Fremdschlüssel. Bei nicht persistenten speicherinternen Tabellen gibt es jedoch einige Einschränkungen für Integritätsbedingungen über Fremdschlüssel.

Der Hauptunterschied zwischen M-Tabellen und D-Tabellen liegt in der Leistung. M-Tabellen bieten eine höhere Leistung. Sie können jedoch dieselbe Dauerhaftigkeit und Wiederherstellbarkeit wie D-Tabellen bieten. Beispielsweise warten Leseoperationen für M-Tabellen nicht auf den Plattenzugriff, selbst während das System Aktivitäten wie das Prüfpunktverfahren und die Transaktionsprotokollierung durchführt.

Mit solidDB können Sie festlegen, welche Tabellen speicherinterne Tabellen und welche Tabellen plattenbasierte Tabellen sind. 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.

1.2 Typen speicherinterner Tabellen

Es gibt zwei Basistypen von speicherinternen Tabellen: persistente Tabellen und nicht persistente Tabellen. Persistente Tabellen bieten die Wiederherstellbarkeit von Daten, nicht persistente Tabellen bieten einen schnellen Zugriff.

Plattenbasierte Tabellen sind immer persistente Tabellen.

1.2.1 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 persistenten speicherinternen Tabellen, wenn er heruntergefahren wird, auf der Platte. Daher sind diese Daten nach jedem Serverstart verfügbar. Persistente speicherinterne Tabellen verwenden auch die Transaktionsprotokollierung. Falls der Server aus einem unerwarteten Grund (beispielsweise bei einem Stromausfall) beendet wird, verfügt er noch über einen Datensatz der stattgefundenen Transaktionen und kann die Tabellen aktualisieren, um sicherzustellen, dass sie alle Daten aller festgeschriebenen Transaktionen enthalten. Wie auch bei plattenbasierten Tabellen werden die Daten in persistenten speicherinternen Tabellen bei Prüfpunkten auf die Festplatte kopiert.

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

Unterschiede zwischen persistenten speicherinternen Tabellen und plattenbasierten Tabellen

Unter den meisten Aspekten lassen sich speicherinterne Tabellen praktisch nicht von plattenbasierten Tabellen unterscheiden, außer, dass speicherinterne Tabellen generell wesentlich schneller sind. In den folgenden Abschnitte wird besonders auf die Unterschiede zwischen speicherinternen Tabellen und plattenbasierten Tabellen eingegangen.

Steuerung des gemeinsamen Zugriffs
: Speicherinterne Tabellen verwenden immer eine pessimistische Steuerung des gemeinsamen Zugriffs (Sperren) auf Zeilenebene. Plattenbasierte Tabellen verwenden standardmäßig eine optimistische Steuerung des gemeinsamen Zugriffs (Versionssteuerung).

  Abhängig vom verwendeten Tabellentyp muss die Fehlerbehandlung unterschiedliche Fehlercodes berücksichtigen.

Algorithmus für Prüfpunktverfahren
: Das Prüfpunktverfahren für speicherinterne Tabellen unterscheidet sich grundlegend von dem für plattenbasierte Tabellen verwendeten Algorithmus. Beim Prüfpunktverfahren für speicherinterne Tabellen wird der 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.

Sekundärindizes
: Bei speicherinternen Tabellen werden Sekundärindizes nie auf die Platte geschrieben. Stattdessen werden sie im Speicher gehalten und beim Serverstart erneut erstellt. Die Auswirkung der sekundären Indizes auf die Schreibleistung von speicherinternen Tabellen ist wesentlich geringer 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.

1.2.2 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. Dies bewirkt zwar, dass sie nicht wiederhergestellt werden können, macht sie aber wesentlich schneller als persistente Tabellen.

Es gibt zwei verschiedene Typen nicht persistenter speicherinterner Tabellen: *transiente Tabellen* und *temporäre Tabellen*. 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.

Nicht persistente Tabellen sind vorwiegend 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.

Da Transaktionen für nicht persistente Tabellen nicht protokolliert werden, können Sie nicht mit HotStandby oder solidDB Universal Cache verwendet werden.

Temporäre Tabellen

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. Temporäre Tabellen sind noch schneller als transiente Tabellen, da sie keine Protokollierung und keinerlei Mechanismus für die Steuerung des gemeinsamen Zugriffs (wie Eintragssperren) verwenden.

Eingeschränkte Transparenz

Daten haben 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 jeweils die eigenen Daten angezeigt.

Da jeder Sitzung nur die 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

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

Das Wort *temporär* im Begriff *temporäre Tabelle* bezieht sich auf die Daten und nicht auf die Tabelle selbst. Der Server speichert die Definition der temporären Tabelle (jedoch nicht die Daten) in den Systemtabellen und behält diese Definition auch noch bei, nachdem die Verbindung getrennt wurde. Wenn Sie die Verbindung zum Server zu einem späteren Zeitpunkt wiederherstellen, ist diese Tabelle daher noch vorhanden, sie ist jedoch leer. Sobald Sie die Tabelle erstellt haben, müssen Sie sie 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 wird bei der Vorstellung, dass "temporäre Tabelle" bedeutet, dass die Tabelle (und nicht nur die Daten) beim Trennen der Verbindung gelöscht wird, möglicherweise nicht erwartet.

Da die Tabellen bestehen bleiben (auch wenn die Daten nicht gespeichert werden), müssen Sie die Tabellendefinition, wenn Sie sie nicht mehr benötigen, mit dem Befehl DROP TABLE löschen. Da 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 gelöscht wird, wenn der Benutzer die Verbindung trennt, kann die Prozessorbelegung noch einige Zeit nach dem Schließen einer Sitzung bei einer temporären Tabelle mit hohem Datenvolumen relativ hoch sein.

Weitere Merkmale

- Bei Verwendung der HotStandby-Komponente werden Daten in temporären Tabellen nicht auf den sekundären Server repliziert. Allerdings werden die Definitionen der temporären Tabellen auf den sekundären Server repliziert. 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.
- In Universal Cache werden temporäre Tabellen nicht unterstützt und können nicht Teil einer Subskription sein, wenn solidDB als Quellendatenspeicher verwendet wird. Temporäre Tabellen können in einer Subskription verwendet werden, in der solidDB der Zieldatenspeicher ist.
- Temporäre Tabellen können in Systemen mit erweiterter Replikation nur als Replikattabellen, nicht als Originaltabellen verwendet werden.
- Für temporäre Tabellen gelten Einschränkungen hinsichtlich ihrer Verwendung mit referenziellen Integritätsbedingungen. Eine temporäre Tabelle kann auf eine andere temporäre Tabelle verweisen, nicht jedoch auf transiente oder persistente Tabellen. Es gibt keinen anderen Tabellentyp, der auf eine temporäre Tabelle verweisen kann.

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 Kilobyte 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. Die Standardzugriffsrechte für temporäre Tabellen sind mit den Standardzugriffsrechten für persistente Tabellen identisch.

Einhaltung von Standards

Die Implementierung temporärer Tabellen durch solidDB entspricht in vollem Umfang dem ANSI-Standard SQL:1999 für *globale temporäre Tabellen*. Alle temporären Tabellen von solidDB sind globale Tabellen. In der Syntax der Anweisung CREATE TABLE wird das Schlüsselwort GLOBAL aus Kompatibilitätsgründen unterstützt. Allerdings sind alle temporären Tabellen global, auch wenn das Schlüsselwort GLOBAL nicht angegeben wird.

Der solidDB-Server unterstützt keine *lokalen temporären Tabellen*, wie vom ANSI-Standard definiert.

Transiente Tabellen

Transiente Tabellen bleiben bis zum Herunterfahren des Servers bestehen. Mehrere Benutzer können dieselbe transiente Tabelle verwenden und jedem Benutzer werden die Daten aller anderen Benutzer angezeigt.

In der Regel verhalten sich transiente Tabellen wie standardmäßige speicherinterne (persistente) Tabellen. Beispiel:

- 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.
- 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. Allerdings ist die Länge der BLOB-Spalten auf wenige Kilobyte in allen speicherinternen Tabellen begrenzt.
- Für transiente Tabellen gelten Zugriffsrechte.

- Transiente Tabellen befinden sich in einem speziellen Katalog und in einem speziellen Schema.
- Mit dem Dienstprogramm Speed Loader (**solload**) von solidDB können Sie Daten in transiente Tabellen importieren.

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

Der Server speichert die Definition der transienten Tabelle (jedoch nicht die Daten) in den Systemtabellen und behält diese Definition auch noch bei, nachdem der Server heruntergefahren wird. 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 nur ein Mal erstellen. Wenn Sie oder andere Benutzer versuchen, eine transiente Tabelle mit demselben Namen wie eine vorhandene transiente Tabelle zu erstellen, wird eine Fehlernachricht angezeigt, und zwar auch dann, wenn der Server zwischenzeitlich 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 benötigen, mit dem Befehl DROP TABLE löschen.

Einschränkungen

Für transiente Tabellen gelten im Gegensatz zu persistenten speicherinternen Tabellen jedoch einige Einschränkungen.

- Die Daten der transienten Tabellen werden bei Verwendung der HotStandby-Komponente nicht auf den sekundären Server repliziert. Die transienten Tabellen selbst (nicht jedoch ihre Daten) werden auf den sekundären HotStandby-Server repliziert. Daher müssen Sie transiente Tabellen für eine Funktionsübernahme durch den sekundären Server nicht erneut erstellen. Die Daten in diesen Tabellen müssen jedoch erneut erstellt werden.
- In Universal Cache werden transiente Tabellen nicht unterstützt und können nicht Teil einer Subskription sein, wenn solidDB als Quellendatenspeicher verwendet wird. Transiente Tabellen können in einer Subskription verwendet werden, in der solidDB der Zieldatenspeicher ist.
- Für transiente Tabellen gelten 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. Temporäre Tabellen und persistente Tabellen können nicht auf eine transiente Tabelle verweisen.
- Transiente Tabellen können in Systemen mit erweiterter Replikation nur als Replikattabellen, nicht als Originaltabellen verwendet werden.

Einhaltung von Standards

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

Unterschiede zwischen temporären 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. Nur pessimistische Steuerung des gemeinsamen Zugriffs (Sperren) wird unterstützt.
- Temporäre Tabellen sind schneller als transiente Tabellen, da sie nicht die Steuerung des gemeinsamen Zugriffs verwenden.
- 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.

1.2.3 Tabellentypen und referenzielle Integrität

Persistente und nicht persistente Tabellen unterscheiden sich in Bezug auf die referenzielle Integrität.

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.

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

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	-

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

1.3 Aspekte der Anwendungsentwicklung mit speicherinternen Tabellen

Prüfen Sie vor der Entwicklung von Anwendungen mit speicherinternen Tabellen die folgenden Aspekte zur Leistung, Speicherbelegung, Transaktionsisolation sowie zur Verwendung von M-Tabellen mit den Zugriffsmethoden HotStandby oder SMA (Shared Memory Access) und LLA (Linked Library Access).

1.3.1 Leistung und speicherinterne 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 wieder zurück auf die Platte geschrieben werden. Speicherinterne Tabellen bieten eine höhere Leistung, da alle Daten immer im Hauptspeicher gehalten werden. Der Server kann so effizientere Verfahren nutzen, 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 geht jedoch weit über das reine Kopieren von Daten in den Speicher hinaus. 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.

Leistung bei temporären und transienten Tabellen

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

- 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 einmal 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.
- Temporäre Tabellen und transiente Tabellen verwenden eine effizientere Datenspeicherstruktur als herkömmliche speicherinterne Tabelle.
- Temporäre Tabellen bieten einen weiteren Leistungsvorteil gegenüber transienten Tabellen. 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).

Indizes

Wenn eine Tabelle im Speicher gespeichert wird, werden alle Indizes für diese Tabelle ebenfalls im Speicher gespeichert. Dies bedeutet eine Leistungsverbesserung, verbraucht allerdings auch mehr Hauptspeicherkapazität. Normalerweise sind speicherinterne Indizes extrem schnell und sollten daher verwendet werden, um schnellen Zugriff auf die Tabellendaten sicherzustellen. Wenn die Speicherkapazität nicht für all Ihre Tabellen und Indizes ausreicht, ist das Hinzufügen eines bestimmten Index allerdings nicht in allen Fällen die richtige Lösung. Dies beschleunigt zwar einige Abfragen, andere Abfragen werden jedoch verlangsamt, da sie Speicher belegen, der andernfalls verwendet werden könnte, um andere Tabellen in den Speicher zu stellen.

1.3.2 Physischer Hauptspeicher und virtueller Speicher

Die Gesamtgröße der speicherinternen Datenbanktabellen kann nicht die verfügbare Menge des virtuellen Speichers übersteigen.

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. Daher sollten Sie nicht die Größe des verfügbaren virtuellen Speichers begrenzen, sondern die Größe Ihre speicherinternen Tabellen, sodass sie den verfügbaren physischen Hauptspeicher 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.

Der zum Speichern einer Tabelle erforderliche Speicherplatz umfasst nicht nur den Platz für die Daten in der Tabelle, 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.

1.3.3 Einschränkungen der Transaktionsisolation bei speicherinternen Tabellen

Die Isolationsstufe SERIALIZABLE wird in M-Tabellen nicht unterstützt.
Sie können speicherinterne Tabellen nicht in Transaktionen verwenden, deren Transaktionsisolationsstufe SERIALIZABLE ist. Für speicherinterne Tabellen werden die folgenden Tansaktionsisolationsstufen unterstützt: REPEATABLE READ und READ COMMITTED.

Für speicherinterne Tabellen im sekundären HotStandby-Server wird immer die Transaktionsisolationsstufe READ COMMITTED verwendet.
Wenn Sie HotStandby verwenden und mit dem sekundären HotStandby-Server verbunden sind, wird beim Lesen von Daten aus speicherinternen Tabellen die Transaktionsisolationsstufe automatisch auf READ COMMITTED gesetzt, auch wenn Sie REPEATABLE READ angegeben haben.

REPEATABLE READ-bezogene Unterschiede zwischen M-Tabellen und D-Tabellen.
Wenn Sie speicherinterne Tabellen verwenden und die Transaktionsisolationsstufe auf REPEATABLE READ gesetzt haben (der Standard ist READ COMMITTED), werden Schreiboperationen durch Leseoperationen für die Dauer der Lesetransaktion blockiert. Darüber hinaus sind bei der Isolationsstufe REPEATABLE READ und speicherinternen Tabellen Deadlocks möglich. Diese können jedoch nicht bei der Versionssteuerung von plattenbasierten Tabellen auftreten. Andererseits können Konflikte beim gemeinsamen Zugriff auftreten, wenn die Überprüfung auf gleichzeitige optimistische Schreib- und Lesezugriffe verwendet wird.

1.3.4 Gemeinsamer Speicherzugriff und Zugriff auf verlinkte Bibliotheken

Der gemeinsame Speicherzugriff (SMA - Shared Memory Access) und der Zugriff auf die verlinkten Bibliotheken (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 speicherinternen Tabellen kompatibel.

Weitere Informationen finden Sie in Übersicht über den gemeinsamen Speicherzugriff und Zugriff auf verlinkte Bibliotheken.

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

1.3.5 HotStandby- und speicherinterne Tabellen

Speicherinterne Tabellen können mit solidDB High Availability (Hochverfügbarkeit) unter Berücksichtigung folgender Aspekte verwendet werden:

- Persistente speicherinterne Tabellen werden vom primären HotStandby-Server auf den sekundären Server repliziert.
- Temporäre Tabellen und transiente Tabellen werden nicht auf den sekundären Server repliziert.

2 Mit speicherinternen Tabellen arbeiten

2.1 Vorgehensweise beim Definieren von Tabellen als speicherinterne Tabellen

Idealerweise würde Ihr Computer über ausreichend Speicher verfügen, damit all Ihre Tabellen im Speicher gespeichert werden können, und so die bestmögliche Leistung für Datenbanktransaktionen ermöglichen. 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, empfiehlt es sich, die am häufigsten genutzten Daten in den Speicher zu stellen. Grundsätzlich sollten kleine, häufig verwendete Tabellen in den Speicher gestellt werden, während große, selten verwendete Tabellen auf der Platte bleiben können. Bei den anderen möglichen Kombinationen, beispielsweise bei großen, häufig verwendeten Tabellen oder kleinen, selten verwendeten Tabellen, sollte der Tabellentyp von der "Dichte" des Zugriffs auf eine Tabelle abhängen. Für speicherinterne Tabellen gilt: je höher die Anzahl der Zugriffe pro Megabyte pro Sekunde, desto besser.

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 Richtlinien hierzu finden Sie weiter unten.

Sie können den am besten geeigneten 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.

   Der Begriff *Serversitzung* bezieht sich auf eine einzelne Ausführung eines Servers, vom Zeitpunkt seines Starts bis zu seinem beabsichtigten Herunterfahren bzw. bis zu seiner Beendigung aus einem unerwarteten Grund (beispielsweise durch einen 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.
4. Wenn keine der obigen Regeln zutrifft, verwenden Sie eine temporäre Tabelle.

Anmerkung: Für temporäre und transiente Tabellen gelten Einschränkungen, die Ihre Entscheidungen beeinflussen können. So kann eine temporäre Tabelle beispielsweise auf eine andere temporäre Tabelle verweisen, nicht jedoch auf transiente oder persistente Tabellen. Es gibt keinen anderen Tabellentyp, der auf eine temporäre Tabelle verweisen kann.

Zugehörige Informationen:

Anhang A, „Algorithmus zur Auswahl der Tabellen, die im Speicher gespeichert werden sollen", auf Seite 29

„Temporäre Tabellen" auf Seite 3
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. Temporäre Tabellen sind noch schneller als transiente Tabellen, da sie keine Protokollierung und keinerlei Mechanismus für die Steuerung des gemeinsamen Zugriffs (wie Eintragssperren) verwenden.

„Transiente Tabellen" auf Seite 5
Transiente Tabellen bleiben bis zum Herunterfahren des Servers bestehen. Mehrere Benutzer können dieselbe transiente Tabelle verwenden und jedem Benutzer werden die Daten aller anderen Benutzer angezeigt.

2.2 Speicherinterne und plattenbasierte Tabellen erstellen

Sie können auf zwei Arten explizit angeben, ob Tabellen speicherintern oder auf der Platte gespeichert werden sollen.

1. Verwenden Sie die Klausel STORE MEMORY oder STORE DISK des Befehls CREATE TABLE oder ALTER TABLE.

   ```
   CREATE TABLE Mitarbeiter (name CHAR(20)) STORE MEMORY;
   CREATE TABLE ... STORE DISK;
   ALTER TABLE Netzadressen 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 den Standardwert mit dem Parameter **General.DefaultStoreIsMemory** an.

   Beispiel:

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

   Wenn **General.DefaultStoreIsMemory** auf 'yes' gesetzt ist, werden neue Tabellen so lange als speicherinterne Tabellen erstellt, bis in der Anweisung CREATE TABLE etwas anderes angegeben wird. Wenn dieser Parameter auf 'no' gesetzt ist, werden neue Tabellen als plattenbasierte Tabellen erstellt, bis in der Anweisung CREATE TABLE etwas anderes angegeben wird.

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.

2.3 Temporäre und transiente Tabellen erstellen

Wenn Sie eine speicherinterne Tabelle erstellen, ist dies standardmäßig ein persistente Tabelle. Verwenden Sie zum Erstellen von temporären oder transienten Tabellen das Schlüsselwort TEMPORARY bzw. TRANSIENT.

Temporäre Tabellen erstellen

Verwenden Sie zum Erstellen einer temporären Tabelle den folgenden Befehl:

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

Dabei gilt Folgendes:

GLOBAL wird aus Kompatibilitätsgründen unterstützt. Allerdings sind alle temporären Tabellen global, auch wenn das Schlüsselwort GLOBAL nicht angegeben wird.

<...> kennzeichnet Syntax, die mit der Syntax für einen beliebigen anderen Tabellentyp identisch ist.

Temporäre 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 temporäre Tabelle als eine speicherinterne Tabelle.

Transiente Tabellen erstellen

Verwenden Sie zum Erstellen einer transienten Tabelle:

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

Dabei gilt Folgendes:

<...> kennzeichnet Syntax, die mit der für einen beliebigen anderen Tabellentyp identisch ist.

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.

2.4 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 umbenennen, sodass sie denselben Namen wie die ursprüngliche Tabelle hat:

3 Speicherinterne Datenbank konfigurieren

Sie können solidDB so konfigurieren, dass neue Tabellen standardmäßig als speicherinterne Tabellen (M-Tabellen) erstellt werden, indem Sie den Parameter **General.DefaultStoreIsMemory** auf yes setzen. Die meisten anderen speicherinternen Funktionen werden über die Parameter im Abschnitt [MME] der Datei solid.ini konfiguriert.

Besondere Aufmerksamkeit erfordert die Steuerung der Speicherbelegung, denn wenn die speicherinterne Datenbank oder der Serverprozess den gesamten verfügbaren virtuellen Speicher im System belegt, können Sie keine Daten mehr hinzufügen oder aktualisieren. 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.

3.1 Konfigurationsparameter

Die meisten zur speicherinternen solidDB-Datenbank gehörenden Parameter werden im Abschnitt [MME] der Konfigurationsdatei solid.ini 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.

3.1.1 Abschnitt 'General'

Tabelle 3. MME-bezogene Parameter im Abschnitt 'General'

[General]	Beschreibung	Werkseitige Einstellung	Zugriffsmodus
DefaultStoreIsMemory	Wenn dieser Parameter auf `yes` gesetzt ist, werden neue Tabellen als speicherinterne Tabellen erstellt, außer sie werden ohne explizite Angabe der Klausel STORE in der Anweisung CREATE TABLE erstellt. Wenn der Parameter auf `no` gesetzt ist, werden neue Tabellen standardmäßig auf der Platte gespeichert. Sie können den werkseitig festgelegten Wert überschreiben, indem Sie die Klausel STORE in der Anweisung CREATE TABLE angeben. **Anmerkung:** Systemtabellen werden auf Platte gespeichert, auch wenn dieser Parameter auf `yes` gesetzt ist.	yes	RW (Schreib-/Lesezugriff)

Tabelle 3. MME-bezogene Parameter im Abschnitt 'General' (Forts.)

[General]	Beschreibung	Werkseitige Einstellung	Zugriffsmodus
MultiprocessingLevel	Dieser Parameter definiert die Anzahl der Verarbeitungseinheiten (Prozessor, Kerne), die im Computersystem zur Verfügung stehen. In der Regel kann der gemeinsamen Zugriff von Schreiboperationen in der Datenbank verbessern werden, wenn der Wert mit der Anzahl der in Ihrem System vorhandenen Prozessoren (Kerne) übereinstimmt. Ab Version 6.5 Fix Pack 4 wird der werkseitig festgelegte Wert als Anzahl logischer Verarbeitungseinheiten aus dem System gelesen. Der automatisch ermittelte Wert wird beim Serverstart in `solmsg.out` ausgegeben. Bei einigen Prozessorarchitekturen stimmt die Anzahl der logischen Verarbeitungseinheiten möglicherweise nicht mit der Anzahl der physischen Kerne überein. In solchen Fällen liegt der optimale Wert für diesen Parameter normalerweise zwischen der Anzahl physischer Kerne und der Anzahl logischer Verarbeitungseinheiten. In Releases vor Version 6.5 Fix Pack 4 ist der werkseitig festgelegte Wert für diesen Parameter 4. **Anmerkung:** Ab Version 6.5 Fix Pack 4 wird der Wert des Parameters **`MME.RestoreThreads`** standardmäßig auf den Wert dieses Parameters gesetzt, sofern nicht explizit ein anderer Wert angegeben wird.	Read from system (Lesen aus System)	RW/Startup (Schreib-/Lesezugriff/Start)

3.1.2 Abschnitt 'MME'

Tabelle 4. MME-Parameter

[MME]	Beschreibung	Werkseitige Einstellung	Zugriffs-modus

ImdbMemoryLimit	Dieser Parameter setzt eine Obergrenze für die Speichermenge (virtueller Speicher), die der Server für speicherinterne Tabellen und Indizes für speicherinterne Tabellen zuordnet. In-memory tables includes Temporary Tables and Transient Tables, as well as persistent in-memory tables.
	
	Der Grenzwert kann in Byte, Kilobyte (KB), Megabyte (MB) oder Gigabyte (GB) angegeben werden. Beispiel:
	`ImdbMemoryLimit=1073741824`
	`ImdbMemoryLimit=1048576kb`
	`ImdbMemoryLimit=1024mb`
	`ImdbMemoryLimit=1gb`
	
	Wert 0 bedeutet "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 plattenbasierte 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.
	Wichtig: 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)

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.	90	RW (Schreib-/ Lesezugriff)
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 plattenbasierte Tabellen.	80	RW (Schreib-/ Lesezugriff)

Tabelle 4. MME-Parameter (Forts.)

[MME]	Beschreibung	Werkseitige Einstellung	Zugriffs-modus
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 dieser Parameter 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. Mögliche Werte sind `yes` und `no`. **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)
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)

Tabelle 4. MME-Parameter (Forts.)

[MME]	Beschreibung	Werkseitige Einstellung	Zugriffs-modus
MaxBytesCachedInPrivateMemoryPool	Dieser Parameter definiert die maximale Anzahl Byte, die in der Liste der freien Blöcke des privaten Hauptspeicherpools der MME gespeichert werden (der private Hauptspeicherpool ist für jeden Hauptspeicherindex privat). Ist im privaten Pool mehr freier Speicher vorhanden, wird dieser zusätzliche Speicher in globale Pools aufgenommen. Der Wert 0 bedeutet, dass der Speicher sofort in den globalen Pool aufgenommen wird. Diese Einstellung verringert generell die Leistung, minimiert jedoch den Speicherbedarf. Es gibt keinen Maximalwert. Der Standardwert 100.000 bietet eine gute Leistung bei nur geringem Speicheraufwand.	100000	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)
MaxTransactionSize	Dieser Parameter definiert die maximale ungefähre Größe (in Byte) einer Transaktion. Einige MME-Transaktionen (wie z. B. DELETE FROM <Tabelle>) führen möglicherweise dazu, dass solidDB für die Operation eine große Speichermenge zuordnet. Dies kann zu einer Speicherknappheit führen, bei der solidDB keinen Speicher aus dem Betriebssystem mehr zuordnen kann und daraufhin eine Notabschaltung auslöst. Damit dies verhindert wird, definieren Sie mit diesem Parameter die maximale näherungsweise berechnete Speicherkapazität (in Byte) für jede MME-Transaktion. Überschreitet die Speicherkapazität für die Transaktion den mit diesem Parameter festgelegten Wert, schlägt die Transaktion mit solidDB-Fehler 16509 fehl, der darauf hinweist, dass die maximale Größe der MME-Transaktion überschritten wurde. Wert 0 bedeutet unbegrenzt.	0	RW (Schreib-/ Lesezugriff)

MemoryPoolScope	Dieser Parameter legt den Bereich des Hauptspeicherpools fest. Mögliche Werte sind `Global` und `Table`.
	
	Wird der Parameter auf `Table` gesetzt, werden nur Objekte zugeordnet, die zu derselben Datenbanktabelle eines einzelnen Speichersegments gehören. Dadurch wird z. B. sichergestellt, dass durch das Löschen einer ganzen Tabelle das Speichersegment wieder für das Betriebssystem freigegeben wird. Nur ungenutzte Speichersegmente können an das System zurückgegeben werden.
	
	Wird der Parameter auf `Global` gesetzt, werden Hauptspeicherpools von allen MME-Daten gemeinsam genutzt.
	
	Wenn **MME.MemoryPoolScope** auf `Table` gesetzt wird, können Sie die Anweisung DESCRIBE <Tabelle> zur Anzeige des Speicherbedarfs für die Tabelle verwenden. Beispiel:
	`DESCRIBE tmemlimit_tab;`
	`RESULT`
	`------`
	`Catalog: DBA`
	`  Schema: DBA`
	`    Table: TMEMLIMIT_TAB`
	`      Table type: in-memory`
	
	`      Memory usage: 7935 KB (total), 7925 KB (active),`
	`                    6192 KB (rows), 1733 KB (indexes).`
	
	`...`
	`1 rows fetched.`
		Global	RW/Startup (Schreib-/ Lesezugriff/ Start)

Tabelle 4. MME-Parameter (Forts.)

[MME]	Beschreibung	Werkseitige Einstellung	Zugriffs-modus
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)
ReleaseMemoryAtShutdown	Wenn dieser Parameter auf `yes` gesetzt ist, gibt der Server beim Herunterfahren den von den M-Tabellen belegten Speicher explizit frei, anstatt sich darauf zu verlassen, dass das Betriebssystem den gesamten, diesem Prozess zugeordneten Speicher bereinigt. Bei einigen Betriebssystemen müssen Sie diesen Parameter möglicherweise auf `yes` setzen, um sicherzustellen, dass der gesamte Speicherbereich freigegeben wird. Mögliche Werte sind `yes` und `no`. Die werkseitige Einstellung ist `no`, da der Server so schneller heruntergefahren wird.	no	RW/Startup (Schreib-/ Lesezugriff/ Start)
RestoreThreads	Dieser Parameter definiert die maximale Anzahl Threads, die beim Durchführen eines Restores für eine speicherinterne Datenbank während des Datenbankstarts verwendet wird. Wenn Sie diesen Parameter nicht explizit festlegen, wird der Wert des Parameters auf denselbem Wert wie **`General.MultiprocessingLevel`** gesetzt. Mögliche Werte liegen zwischen 1 und 65536. Der Wert 1 bedeutet, dass das Laden mit Einzelthread ausgeführt wird. Bei ungültigen Werten wird dieser Parameter standardmäßig auf **`General.MultiprocessingLevel`** gesetzt. Der Restore der speicherinternen Datenbank ordnet pro Tabelle einen Thread zu, wenn die Anzahl der Tabellen kleiner oder gleich der Zahl des Parameterwerts ist. Der maximale gemeinsame Zugriff wird erreicht, wenn der Parameterwert kleiner als die folgenden beiden Werte ist: Anzahl der Kerne/Prozessoren und die Anzahl der Tabellen in der Datenbank.	Wie **`General.Multi-processing-Level`**	RW/Startup (Schreib-/ Lesezugriff/ Start)

3.2 Speicherbelegung

Die Hauptspeicherbelegung der speicherinternen Datenbank unterscheidet sich erheblich vom solidDB-Standardprodukt. Die speicherinterne Datenbank befindet sich in ihrem eigenen Hauptspeicherpool.

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.

3.2.1 Speicherbelegung überwachen

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

ADMIN COMMAND 'info imdbsize'

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.

ADMIN COMMAND 'info processsize'

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. Dieser Befehl gibt die Menge des belegten virtuellen Speichers zurück, nicht die Menge des belegten physischen Speichers.

ADMIN COMMAND 'pmon mme'

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

Der Befehl **ADMIN COMMAND 'pmon mme';** erstellt die folgende Liste mit aktuellen Werten der Leistungsdaten.

```
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.

ADMIN COMMAND 'memory'

Der Befehl **ADMIN COMMAND 'memory';** listet 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.

3.2.2 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.

Datenbankspeicher

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

Der Standardwert für **MME.ImdbMemoryLimit** ist 0. Dies bedeutet "kein Grenzwert". Sie sollten nicht den Standardwert verwenden, sondern den Parameter auf einen Wert setzen, der sicherstellt, dass die speicherinternen Daten vollständig in den physischen Hauptspeicher passen. Berücksichtigen Sie außerdem die folgenden Faktoren:

- 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 100% des mit **MME.ImdbMemoryLimit** angegebenen Speichers erreicht ist, 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. Weitere Details finden Sie in „MME.ImdbMemoryLowPercentage" auf Seite 24.

Beispiel:

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

MME.ImdbMemoryLowPercentage: Der Parameter **MME.ImdbMemoryLowPercentage** legt einen unteren Grenzwert für die Menge an virtuellem Speicher fest, die speicherinternen Tabellen zugeordnet werden kann. Der Grenzwert wird als Prozentsatz des Parameterwerts von **MME.ImdbMemoryLimit** ausgedrückt.

Wenn der Server den mithilfe von **MME.ImdbMemoryLowPercentage** angegebenen Prozentsatz des Speichers belegt hat, beginnt er, Aktivitäten einzuschränken, damit kein weiterer Speicher mehr belegt wird. Wenn **MME.ImdbMemoryLimit** zum Beispiel 1000 MB und **MME.ImdbMemoryLowPercentage** 90 % ist, beginnt der Server, Aktivitäten einzuschränken, wenn der den speicherinternen Tabellen zugeordnete Speicher 900 MB übersteigt. Der Server wird insbesondere Folgendes tun:

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

Wenn die mit **ImdbMemoryLimit**) gesetzte Grenze erreicht ist, verhindert der Server auch Aktualisierungen (UPDATE-Operationen) für Datensätze in speicherinternen Tabellen.

Gültige Werte für **MME.ImdbMemoryLowPercentage** liegen im Bereich von 60 bis 99 (Prozent).

MME.ImdbMemoryWarningPercentage: Der Parameter **MME.ImdbMemoryWarningPercentage** legt einen Grenzwert für die Ausgabe eines Systemereignisses fest, das Sie warnt, wenn der Maximalwert des virtuellen Speichers erreicht wird, der speicherinternen Tabellen zugeordnet werden kann.

Der Warngrenzwert wird als Prozentsatz des Parameterwerts von **MME.ImdbMemoryLimit** ausgedrückt. Wenn der Grenzwert von **MME.ImdbMemoryWarningPercentage** überschritten wird, wird ein Systemereignis ausgegeben.

Fehlerbehebung für MME.ImdbMemoryLimit:

Wenn in einer Fehlernachricht angezeigt wird, dass der mit **MME.ImdbMemoryLimit** festgelegte Grenzwert erreicht wurde, müssen Sie sofort Maßnahmen ergreifen.

Sie müssen sowohl die dringenden Probleme als auch langfristige Probleme beheben. 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 dasselbe Speicherproblem nicht erneut auftritt. Auf Lange Sicht müssen Sie sicherstellen, dass bei einem Anwachsen der Tabellen der Speicherplatz nicht erneut knapp wird.

Dringende Probleme beheben

Bei dringenden Problemen gehen Sie in der Regel wie folgt vor:

1. Benachrichtigen Sie die Benutzer, dass sie die Verbindung zum Server trennen müssen. Dadurch werden zwei Dinge erreicht: 1. Weniger Benutzer sind von möglichen negativen Auswirkungen betroffen. 2. 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 wurden, um Speicherplatz freizugeben, löschen Sie einige Indizes von transienten Tabellen oder einige transiente Tabellen.

Wenn zu wenige temporäre und transiente Tabellen verwendet wurden, um genügend Speicherplatz freizugeben, gehen Sie wie folgt vor:

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 ohne Index haben und alle Tabellen kritische Daten enthalten), erhöhen Sie den Wert von **MME.ImdbMemoryLimit** geringfügig, 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.

Langfristige Probleme beheben

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.

Zur Vermeidung zukünftiger Probleme 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.

- Wurde das Problem lediglich aufgrund intensiver Nutzung temporärer Tabellen (oder transienter Tabellen) verursacht, stellen Sie sicher, dass nicht zu viele Sitzungen zu viele umfangreiche temporäre Tabellen (oder transiente Tabellen) gleichzeitig erstellen.
- Wurde das Problem verursacht, weil zu viel Speicher für normale speicherinterne Tabellen verwendet wurde und Sie die für den Server verfügbare Speichermenge nicht erhöhen können, verschieben Sie eine oder mehrere Tabellen aus dem Hauptspeicher auf die Platte.

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 eine plattenbasierte Tabelle.

   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, dass die gesamte Operation über eine einzige Transaktion ausgeführt wird. 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 sollten Sie 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 kopieren.

   Anmerkung: Für die Zwischentabelle sind keine Indizes erforderlich. Die Indizes sollten in der neuen Tabelle erneut erstellt werden, nachdem die Daten erfolgreich kopiert wurden.
3. Löschen Sie die speicherinterne Tabelle.
4. Benennen Sie die plattenbasierte Tabelle um, sodass sie den ursprünglichen Namen der gelöschten speicherinternen Tabelle hat.

Tipp:

- Sie sollten den Wert von **MME.ImdbMemoryLimit** auf einen etwas niedrigeren Wert als Ihren tatsächlich zur Verfügung stehenden Maximalwert setzen. Falls der Speicher knapp wird und Sie keine entbehrlichen speicherinternen Tabellen oder Indizes haben, die gelöscht werden können, können Sie den Wert von **MME.ImdbMemoryLimit** etwas erhöhen und den Server mit ausreichend freiem Speicher für den langfristigen Bedarf erneut starten.
- Nutzen Sie den Parameter **MME.ImdbMemoryWarningPercentage** zur Ausgabe einer Warnung über die steigende Speicherbelegung.
- 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.

Prozessspeicher

Srv.ProcessMemoryLimit:

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

Die Werkseinstellung für **Srv.ProcessMemoryLimit** ist 0; es gibt keinen Grenzwert für den Prozessspeicher. Wenn Sie den Parameter verwenden, setzen Sie ihn auf einen Wert, der sicherstellt, dass der speicherinterne Datenbankprozess vollständig in den physischen Hauptspeicher passt. Die folgenden Faktoren beeinflussen die erforderliche Speichermenge:

- Größe des physischen Hauptspeichers im Computer
- Vom Betriebssystem belegter Speicher
- Von den speicherinternen Tabellen (einschließlich temporären Tabellen und transienten Tabellen) sowie den Indizes für diese speicherinternen Tabellen verwendeter Speicher
- Für den Cache des solidDB-Servers vorgesehene Speichermenge (Parameter **IndexFile.CacheSize**)
- 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 **Srv.ProcessMemoryLimit** angegebenen Speicher vollständig verwendet, akzeptiert der Server nur noch ADMIN COMMAND-Befehle). Mithilfe der Parameter **Srv.ProcessMemoryWarningPercentage** und **Srv.ProcessMemoryLowPercentage** können Sie sich eine Warnung über die steigende Speicherbelegung ausgeben lassen.

Anmerkung:
- Die Parameter **Srv.ProcessMemoryLimit** und **Srv.ProcessMemoryCheckInterval** sind verkettet, d. h., wenn der Parameter **ProcessMemoryCheckInterval** auf 0 gesetzt ist, ist der Parameter **ProcessMemoryLimit** nicht wirksam, und es gibt demnach keinen Grenzwert für den Prozessspeicher.
- Sie sollten den Parameter **Srv.ProcessMemoryLimit** nicht setzen, wenn Sie den gemeinsamen Speicherzugriff (SMA - Shared Memory Access) verwenden. Falls Sie den vom SMA-Server verwendeten Speicher begrenzen müssen, verwenden Sie den Parameter **SharedMemoryAccess.MaxSharedMemorySize**.

Srv.ProcessMemoryLowPercentage:

Der Parameter **Srv.ProcessMemoryLowPercentage** setzt einen Warngrenzwert für die Gesamtprozessgröße. Der Grenzwert wird als Prozentsatz des Parameterwerts von **Srv.ProcessMemoryLimit** ausgedrückt.

Bevor der Grenzwert überschritten wird, wurde eine Warnung ausgegeben, da Sie bereits den mit dem Parameter **ProcessMemoryWarningPercentage** definierten Warngrenzwert überschritten haben. Wenn der Grenzwert von **Srv.ProcessMemoryLowPercentage** überschritten wird, wird ein Systemereignis ausgegeben.

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

Srv.ProcessMemoryWarningPercentage:

Der Parameter **Srv.ProcessMemoryWarningPercentage** setzt den ersten Warngrenzwert für die Gesamtprozessgröße. Der Warngrenzwert wird als Prozentsatz des Parameterwerts von **Srv.ProcessMemoryLimit** ausgedrückt.

Wenn der Grenzwert von **Srv.ProcessMemoryWarningPercentage** überschritten wird, wird ein Systemereignis ausgegeben.

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

Srv.ProcessMemoryCheckInterval:

Der Parameter **Srv.ProcessMemoryCheckInterval** definiert das Intervall, in dem die Grenzwerte für die Prozessgröße überprüft werden. Das Intervall wird in Millisekunden angegeben.

Der Mindestwert ungleich null für **Srv.ProcessMemoryCheckInterval** ist 1000 (ms). Es sind nur die Werte 0, 1000 oder größer als 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.

Die Parameter **Srv.ProcessMemoryLimit** und **Srv.ProcessMemoryCheckInterval** sind verkettet, d. h., wenn der Parameter **ProcessMemoryCheckInterval** auf 0 gesetzt ist, ist der Parameter **ProcessMemoryLimit** nicht wirksam, und es gibt keinen Grenzwert für den Prozessspeicher.

Anhang A. Algorithmus zur Auswahl der Tabellen, die im Speicher gespeichert werden sollen

In diesem Abschnitt wird eine Strategie beschrieben, die Ihnen bei der Auswahl der Tabellen hilft, die in den Speicher gestellt werden sollen.

In erster Linie muss die "Zugriffsdichte" für die Tabelle berücksichtigt werden. 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 was hier in Zeilen/MB/s dargestellt wird. (Der Einfachheit halber wird ein Zugriff pro Zeile angenommen.)

Beispiel 1:

Wenn Sie eine 1-MB-Tabelle haben und in einem Zeitraum von 10 Sekunden auf 300 Zeilen zugreifen, ergibt sich eine Dichte von`30 Zeilen/MB/s = 300 Zeilen / 1 MB / 10 Sekunden`.

Beispiel 2:

Wenn Sie eine 500-KB-Tabelle haben und pro Sekunde auf 300 Zeilen zugreifen, ist die Zugriffsdichte `600 Zeilen/MB/s = 300 Zeilen / 0,5 MB / Sekunde`.

Die Tabelle im zweiten Beispiel hat eine deutlich höhere Zugriffsdichte als die im ersten Beispiel. Falls nur eine dieser Tabellen in den Speicher passt, sollten Sie die zweite Tabelle in den Speicher stellen.

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 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 noch keiner vorhanden ist), 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. Sie beginnen mit der Tabelle mit der höchsten Dichte und gehen dann in der Liste nach unten und definieren Tabellen so lange als speicherinterne Tabellen, bis Sie den gesamten verfügbaren physischen Hauptspeicher belegt haben.

Dies ist allerdings eine vereinfachte Beschreibung, da hier davon ausgegangen wird, 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. Beim solidDB-Server 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.

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 Prozess des Datenbankverwaltungssystems in den physischen Hauptspeicher des Computers passt.

Anhang B. Maximale BLOB-Größe berechnen

B.1 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.

B.2 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 Zeitreihendaten 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) 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.

B.3 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 5. 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 5. 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 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 6. 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 C. 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 aus den folgenden Gründen nicht präzise sind:

- solidDB komprimiert einige Daten.
- Der für Daten mit variabler Länge (zum Beispiel 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.

Bei den hier verwendeten Formeln wird 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)` bedeutet, dass die Summe der Größen von jedem `x` berechnet wird. Beispiel:

- `Summe_von(Spaltengröße)` bedeutet, dass die Summe der Größen aller Spalten in der Tabelle oder im Index berechnet wird.
- `Summe_von(Indexgrößen)` bedeutet, dass die Summe der Größen aller Indizes für die Tabelle berechnet wird.

C.1 Speicherbedarf für plattenbasierte Tabellen berechnen

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 ist *Primärschlüsselgröße* 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. 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.

Hintergrundinformation

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 Spaltengrößen pro Spaltentyp berechnet werden. Siehe C.3, „Spaltengrößen und Spaltentyp”, auf Seite 40.

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. verwendet.

Der Header für jede Zeile belegt 12 Byte:

Tabelle 7. Byte für Header

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

Wenn eine plattenbasierte Tabelle Indizes enthält, mit Ausnahme des Primärschlüssels, 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
- Ein Zeilenheader (12 Byte)

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

Beispiel:

Sie erstellen zunächst wie folgt eine plattenbasierte Tabelle:

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

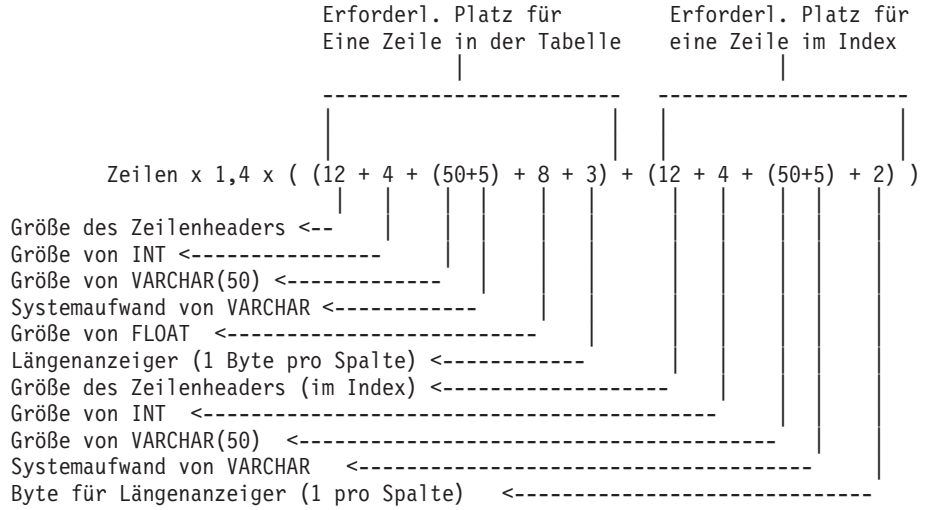
Anschließend erstellen Sie wie folgt einen Sekundärindex:

```
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
```

Tipp: Sie können die oben gezeigte Berechnung auch wie folgt darstellen:



C.2 Speicherbedarf für speicherinterne Tabellen berechnen

Die allgemeine Formel für den erforderlichen Platz in der speicherinternen 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 für 32-Bit-Betriebssysteme und 8 Byte für 64-Bit-Betriebssysteme).
- *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 ist *Verteilungsfaktor* 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.

Hintergrundinformation

Bei der Berechnung des Speicherbedarfs von speicherinternen Tabellen ist die Größe jedes Eintrags die kombinierte Größe der Daten der Tabelle plus drei Speicherzeiger (jeweils 4 Byte bei 32-Bit-Betriebssystemen bzw. jeweils 8 Byte bei 64-Bit-Betriebssystemen) Aufwand pro Zeile. Darüber hinaus fällt ein Aufwand von zwei Byte pro Zeile und zwei Byte pro Spalte der Zeile an. Sie brauchen nicht 1 Byte pro Spalte hinzuzufügen, um den Längenanzeiger zu berücksichtigen, da dieser bereits in den 2 Byte pro Zeile enthalten ist.

Die speicherinternen Tabellen 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 oben angegebene Beispieltabelle und den Beispielindex

Beispiel:

Sie erstellen zunächst wie folgt eine speicherinterne Tabelle:

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

Anschließend erstellen Sie wie folgt einen Sekundärindex:

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

Dann können Sie den Speicherbedarf in einem 32-Bit-Betriebssystem wie folgt schätzen:

```
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
)
```

```
((3 x 4) + 2 + (4 + 2)+(50+5+2)+(8+2)) // Größe der Daten in der Tabelle
  |   |    |    |   |   | | |   | |
  |   |    |    |   |   | | |   |  -> 2 Byte pro Spalte
  |   |    |    |   |   | | |    ---> 8 Byte für FLOAT (Salary)
  |   |    |    |   |   | |  -------> 2 Byte pro Spalte
  |   |    |    |   |   |   ---------> 5 Byte Aufwand pro VARCHAR
  |   |    |    |   |    ------------> 50 Byte für VARCHAR(50)
  |   |    |    |    ----------------> 2 Byte pro Spalte
  |   |    |     --------------------> 4 Byte für INT (ID)
  |   |     ------------------------> 2 Byte pro Zeile
  |    ----------------------------> Zeigergröße (4 Byte auf 32-Bit-Betriebssystemen)
   --------------------------------> 3 Zeiger

((8 x 4) + 4 + 1.2 x 4)     // Größe des Primärschlüsselindex
  |   |    |    |   |
  |   |    |    |    ---------------> 4 Byte für INT
  |   |    |     --------------------> 1,2 Verteilungsfaktor für Indexwert
  |   |     ------------------------> 4 Byte pro Indexeintrag
  |    ----------------------------> Zeigergröße (4 Byte auf 32-Bit-Betriebssystemen)
   --------------------------------> 8: bis zu 8 Zeiger


((8 x 4) + 4 + 1,2 x (50+5))  // Größe des Sekundärindex
  |   |    |    |     | |
  |   |    |    |     |  ------------> 4 Byte für VARCHAR-Aufwand
  |   |    |    |      --------------> 50 Byte für VARCHAR(50)
  |   |    |     --------------------> 1,2 Verteilungsfaktor für Indexwert
  |   |     ------------------------> 4 Byte pro Indexeintrag
  |    ----------------------------> Zeigergröße (4 Byte auf 32-Bit-Betriebssystemen)
   --------------------------------> 8: bis zu 8 Zeiger
```

Verwenden Sie in 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. Sein Wert liegt in der Regel zwischen 1 und 2. Bei der wahlfreien Werteverteilung liegt der Wert näher an 2,0. 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. Die speicherinternen Indizes werden nie auf die Platte geschrieben und belegen daher keinen Plattenspeicherplatz. Die Tabellen selbst werden jedoch bei Prüfpunkten sowie beim Herunterfahren des Servers auf die Platte geschrieben. Aus diesem Grund muss der gesamte Ihnen zur Verfügung stehende Plattenspeicherplatz großgenug sein, um sowohl die plattenbasierten Tabellen als auch die speicherinternen Tabellen zu speichern.

C.3 Spaltengrößen und Spaltentyp

Tabelle 8. Spaltengrößen und Spaltentyp

Spaltentyp	Größe
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

Anmerkung: Die Werte in der Tabelle oben stellen die maximalen Längen dar. Daten mit variabler Länge (VARCHAR) oder komprimierbare Daten erfordern möglicherweise weniger Bytes.

C.4 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.

Index

Sonderzeichen

A

B

C

D

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, 7502796 und 7587429.

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. Anstelle 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.

Aussagen über Pläne und Absichten von IBM unterliegen Änderungen oder können zurückgenommen werden und repräsentieren nur die Ziele von IBM.

Diese Veröffentlichung dient nur zu Planungszwecken. Die in dieser Veröffentlichung enthaltenen Informationen können geändert werden, bevor die beschriebenen Produkte verfügbar sind.

Diese Veröffentlichung enthält Beispiele für Daten und Berichte des alltäglichen Geschäftsablaufs. Sie sollen nur die Funktionen des Lizenzprogramms illustrieren

und 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 Beispielanwendungsprogramme, die in Quellensprache geschrieben sind und Programmiertechniken in verschiedenen Betriebsumgebungen veranschaulichen. Sie dürfen diese Beispielprogramme 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 Beispielprogramme 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. Die Beispielprogramme werden ohne Wartung (auf "as-is"-Basis) und ohne jegliche Gewährleistung zur Verfügung gestellt. IBM übernimmt keine Haftung für Schäden, die durch die Verwendung der Beispielprogramme entstehen.

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

© (Name Ihrer Firma) (Jahr). Teile des vorliegenden Codes wurden aus Beispielprogrammen 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. Weitere Produkt- und Servicenamen können Marken von IBM oder anderen Unternehmen sein. Eine aktuelle Liste der IBM Marken finden Sie auf der Webseite „Copyright and trademark information" unter www.ibm.com/legal/copytrade.shtml.

Java™ und alle auf Java basierenden Marken und Logos sind Marken oder eingetragene Marken der Oracle Corporation und/oder ihrer verbundenen Unternehmen.

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 Produkt- und Servicenamen können Marken von IBM oder anderen Unternehmen sein.

SC12-4630-04