IBM solidDB
IBM solidDB Universal Cache

バージョン 6.3

拡張レプリケーション・ユーザー・ガイド

Information Management software

SC88-5815-00
(英文原典：SC23-9827-00)

IBM solidDB
IBM solidDB Universal Cache

バージョン 6.3

拡張レプリケーション・ユーザー・ガイド

Information Management software

SC88-5815-00
(英文原典：SC23-9827-00)

注記

本書および本書で紹介する製品をご使用になる前に、 195 ページの『特記事項』に記載されている情報をお読みください。

本書は、バージョン 6、リリース 3 の IBM solidDB (プロダクト番号 5724-V17) および IBM solidDB Universal Cache (プロダクト番号 5724-W91)、および新しい版で明記されていない限り、以降のすべてのリリースおよびモディフィケーションに適用されます。

お客様の環境によっては、資料中の円記号がバックスラッシュと表示されたり、バックスラッシュが円記号と表示されたりする場合があります。

原典： SC23-9827-00
IBM solidDB
IBM solidDB Universal Cache
Version 6.3
Advanced Replication User Guide

発行： 日本アイ・ビー・エム株式会社

担当： ナショナル・ランゲージ・サポート

第1刷 2009.2

© Solid Information Technology Ltd. 1993, 2008

目次

図

表

本書について

IBM® solidDB® 拡張レプリケーションは、アプリケーションのニーズに合わせて、ネットワーク内の異種コンピューター・ノード間でデータを複製する機能です。拡張レプリケーションでは、待ち時間、パフォーマンス、または可用性の要件が示すように、データをアプリケーションに近づけることが可能です。拡張レプリケーションの中核となるのは、パブリッシュ/サブスクライブ・モデルに基づく非同期レプリケーションです。solidDB のコンテキストでは、拡張レプリケーションで使用するレプリケーション・タイプは「同期」と呼ばれています。

この「*拡張レプリケーション・ユーザー・ガイド*」では、同期の概念とアーキテクチャーを紹介し、拡張レプリケーションを使用して solidDB をセットアップ、使用、および管理する方法を説明します。

本書は、読者が DBMS に関して一般的な知識を持っていること、また SQL に精通していることを前提としています。また、solidDB に精通していることも前提としています。本書を読む前に、必要に応じて「*スタートアップ・ガイド*」と「*管理者ガイド*」も参照してください。

ヒント: solidDB は、InfoSphere™ CDC レプリケーション・テクノロジーを使用する別のレプリケーション・ソリューションも提供しています。これは、「*IBM InfoSphere Change Data Capture レプリケーション・ユーザー・ガイド*」に説明されています。

IBM solidDB 製品によるレプリケーションの概要

IBM® solidDB 製品ファミリーでは、拡張レプリケーション、CDC レプリケーション、および solidDB 高可用性 (HotStandby) の 3 種類のテクノロジーを使用してデータ・レプリケーションをインプリメントできます。

- solidDB HotStandby レプリケーションは、1+1 トポロジーを使用する、非常に高速なフェイルオーバーおよびリカバリー機能を必要とする高可用性システムを対象としています。
- 拡張レプリケーションは、組み込みの SQL 拡張機能を使用しており、N+M トポロジーによる不定期またはイベント・ベースの非同期レプリケーションを対象としています。
- CDC テクノロジー・ベースのレプリケーションは、トランザクション・ログの読み取りを使用しており、不定期または連続的なレプリケーション・フローを対象とし、異種混合環境もサポートします。

規則

書体の規則

solidDB の資料では、以下の書体の規則を使用します。

表 1. 書体の規則

フォーマット	用途
データベース表	このフォントは、すべての通常テキストに使用します。
NOT NULL	このフォントの大文字は、SQL キーワードおよびマクロ名を示しています。
solid.ini	これらのフォントは、ファイル名とパス式を表しています。
`SET SYNC MASTER YES;` `COMMIT WORK;`	このフォントは、プログラム・コードとプログラム出力に使用します。SQL ステートメントの例にも、このフォントを使用します。
run.sh	このフォントは、サンプル・コマンド行に使用します。
TRIG_COUNT()	このフォントは、関数名に使用します。
java.sql.Connection	このフォントは、インターフェース名に使用します。
LockHashSize	このフォントは、パラメーター名、関数引数、および Windows® レジストリー項目に使用します。
argument	このように強調されたワードは、ユーザーまたはアプリケーションが指定すべき情報を示しています。
管理者ガイド	このスタイルは、他の資料、または同じ資料内の他の章の参照に使用します。新しい用語や強調事項もこのように記述します。
ファイル・パス表示	ファイル・パスは、UNIX® フォーマットで示します。スラッシュ (/) 文字は、インストール・ルート・ディレクトリーを表します。
オペレーティング・システム	資料にオペレーティング・システムによる違いがある場合は、最初に UNIX フォーマットで記載します。UNIX フォーマットに続いて、小括弧内に Microsoft® Windows フォーマットで記載します。その他のオペレーティング・システムについては、別途記載します。異なるオペレーティング・システムに対して、別の章を設ける場合があります。

構文表記法の規則

solidDB の資料では、以下の構文表記法の規則を使用します。

表 2. 構文表記法の規則

フォーマット	用途
`INSERT INTO table_name`	構文の記述には、このフォントを使用します。置き換え可能セクションには、この フォントを使用します。
solid.ini	このフォントは、ファイル名とパス式を表しています。
[ ]	大括弧は、オプション項目を示します。太字テキストの場合には、大括弧は構文に組み込む必要があります。
|	垂直バーは、構文行で、互いに排他的な選択項目を分離します。
{ }	中括弧は、構文行で互いに排他的な選択項目を区切ります。太字テキストの場合には、中括弧は構文に組み込む必要があります。
...	省略符号は、引数が複数回繰り返し可能なことを示します。
. . .	3 つのドットの列は、直前のコード行が継続することを示します。

1 拡張レプリケーションを使用したデータ同期の概要

この章では、IBM solidDB 拡張レプリケーション・データ同期について説明します。この機能は、データベース間での、データの保管、管理、および同期に使用できます。これは、メディア配信ネットワーク、通信デバイス、通信アプリケーションなどの各種業界で役に立ちます。

拡張レプリケーション・データ同期機能は、solidDB 内部にインプリメントされています。この機能は、トランザクション、SQL など、solidDB に備わっているあらゆる機能を使用して、さまざまな種類のデータ分散機能を実現します。

簡単な例として、例えば、支社をいくつも抱える企業を運営する立場であるとします。各支社のユーザーの応答時間をできるだけ短くしたいので、データベースのローカル・コピーを各支社または各ユーザーに配布することを検討します。しかし、各支社または各ユーザーが独自にデータベースのコピーを所有することになり、システム上の多数のデータベースで、各種データが更新されるため、時間を追うごとに、データベースの整合性が失われる危険性が高くなります。solidDB 拡張レプリケーションでは、データベースのローカル・コピーを提供して柔軟性およびその高速アクセスに対応し、同時に分散システムのデータの整合性を維持できるようになります。

solidDB 拡張レプリケーションでは、「マスター」データベースと、そのローカル・コピー (レプリカ) をいくつも持つことができます。拡張レプリケーションでは、マスターとレプリカ間のデータの同期を頻繁に実行することができます。例えば、ラップトップ、PDA などにあるデータベースのローカル・コピーをネットワークから切断し、後で再接続することさえできます。再接続するときに毎回、双方向の同期を実行できます。このとき、最新バージョンのローカル・データをマスター・サーバーに送信し、マスターの最新データをローカル・レプリカ・サーバーにダウンロードします。

solidDB のマスター/レプリカ同期テクノロジーの他のメリットは、各ユーザーまたは各支社にその作業に該当する情報の「一部」のみを与えることが可能であるということです。したがって、ネットワークを介してデータベースを同期するときに必要な帯域幅を削減できます。このため、アクセスを制限することにより、すべてのデータを単一の安全な場所に保管しなくても、データ・セキュリティーの向上を図ることができます。

拡張レプリケーションの同期には、周期的に実行されるという性質があります。したがって、更新の競合が起こり得ます。例えば、現場の技術者が、ローカル・レプリカで顧客の住所を更新し、本社でも顧客サービス担当者が、マスター・データベースでその顧客の住所を更新することがあります。レプリカがネットワークに再接続したとき、両方のデータが一致しなかった場合に、どちらを優先するかという問題があります。このような競合が発生した場合にそれを解決するためのアプリケーション固有のロジックを記述することができます。solidDB 拡張レプリケーション機能では、このような競合解決のロジックの記述がサポートされています。

solidDB 拡張レプリケーションについて

solidDB 拡張レプリケーションは、データを同期する新しいアプローチを備えています。過去の世代のデータ・レプリケーション・ソリューションとは基本的に異なる概念を用いています。

基本的な違いは、完全にコア・データベース・エンジンの内部に組み込まれたアーキテクチャーにあります。リレーショナル・データ管理機能の拡張を使用して、アプリケーションの要件に合ったデータ同期機能を構築できます。この新しいモデルは、従来のレプリケーション・ソリューションとは異なり、データ同期にあるアプリケーション固有の側面を扱えるようになりました。適切に機能するには、ビジネス・アプリケーションの要件に合わせて、同期機能が調整可能である必要があります。例えば、アプリケーションは、固有のビジネス・ルールおよびビジネス・ロジックに基づいて長期にわたり競合を正確に検出し、その競合をマルチデータベース・システムで解決する必要があります。

solidDB 拡張レプリケーション・ソリューションによって、アプリケーション開発者は、最小の労力で堅固なデータ同期機能をアプリケーションに組み込めるデータ管理機能のセットを使用できるようになりました。

「マスター」データベースには、データのマスター・コピーが含まれています。1 つ以上のレプリカ・データベースには、マスターのデータの完全なまたは部分的なコピーが含まれています。レプリカ・データベースには、他のデータベースと同様に、複数の表を含めることができます。それらの表には複製されたデータ (マスターからコピーされたデータ) だけを含む表、ローカル専用データ (マスターからコピーされたのではないデータ) を含む表、複製されたデータとローカル専用データの混合データを含む表があります。

レプリカは、マスター・サーバーに更新をサブミットすることができ、その場合はマスター・サーバーは、アプリケーション・プログラマーが設定したルールに従ってその更新を検査します。その後、検査したデータは「パブリッシュ」され、すべてのレプリカで使用できるようになります。

solidDB 拡張レプリケーション機能

solidDB 拡張レプリケーションは、以下の機能を提供します。

- システム全体の情報の共用

  solidDB 拡張レプリケーションでは、システム内の各サーバーは、必要なデータのローカル・コピーを独自に持つことができます。ユーザーに、中央のデータ管理リソースへのオンラインでのアクセス権限を付与する必要はありません。さらに、拡張レプリケーション・システムの各レプリカ・データベースは、特定の目的に使用できます。例えば、レプリカの 1 つを意思決定支援またはレポート作成アプリケーション専用とし、もう 1 つをオンライン・トランザクション処理アプリケーション専用とすることができます。

- データ保全性

マルチデータベース・システムでは、複数のデータベースで更新が実行される可能性があり、データ保全性の維持が難題となります。solidDB 拡張レプリケーション独自のトランザクション管理アーキテクチャーでは、トランザクションのアプリケーション開発者がトランザクション妥当性検査機能をトランザクション自体に組み込めるようにすることにより、データ保全性の問題に対処しています。これにより、アプリケーション自体のルールとロジックに基づいて情報の正確性が保証されます。

- 高性能と柔軟性

  拡張レプリケーション・アーキテクチャーにより、ユーザーは、同期化処理を調整してパフォーマンスを最大にすることができます。例えば、使用可能な帯域幅が最適な場合には、ネットワークを通して大量のデータを転送できます。同様に、混雑する時間帯には、緊急のデータを反映した高優先度トランザクションのみの伝搬を指定できます。

目的

solidDB 拡張レプリケーション・コンポーネントを使用して、ユーザーは複数のコンピューターの間でデータを同期させることができます。各コンピューターには、solidDB と拡張レプリケーション・コンポーネントで管理されるデータベースがあります。拡張レプリケーション対応 solidDB は、別の拡張レプリケーション対応 solidDB にデータを送信でき、受信側の solidDB は、それ自身のデータベースにデータを保管できます。

使用できるデータ分散モデルは、いくつかあります。例えば、地理的モデルまたは概念モデルを使用して、データの分散方法を制御できます。

- どのコンピューターも、すべてのデータの正確なコピーを持つようにする必要が生じることがあります。例えば、現場で作業をするどの修理担当者も、修理部品のリストと価格がすべて揃った最新のコピーを持つようにする必要が生じることがあります。
- または、データを「スライス」して、コンピューターごとに異なる部分を分散する必要が生じることがあります。この分散は、地理的分担に基づいて行うことができます。例えば、本社のコンピューターは全データのコピー (すべての顧客勘定など) を持ち、各支社は、その支社に該当するデータのコピーだけを保持することができます。または、通信回線カードのラックを使用して、それぞれが特定の接続またはアドレスを処理するようにする必要が生じることがあります。単一の「マスター」コンピューターが接続の完全なセットを保持し、それを個別の回線カードに割り当てます。

長期にわたってデータを分散する方法を制御することもできます。

-

例えば、週に 1 回、1 日に 1 回、1 分に 1 回、5 秒に 1 回など、定期的にデータを更新することを選択できます。

-

  または、時計もしくはカレンダーに従うのではなく、変更の発生時にすぐに更新されたデータを伝搬することを選択できます。

-

  特定のタイプのデータ (更新された E メール・アドレスなど) は更新されたときにすぐに伝搬し、別のタイプのデータ (請求のサマリーなど) は月に 1 回伝搬するなど、混合して使用する必要が生じることがあります。

拡張レプリケーション同期テクノロジーを使用して、データの分割方法、および別の拡張レプリケーション対応 solidDB にそのデータを送信するタイミングを決定できます。

マスター/レプリカ・モデル

solidDB の同期テクノロジーは、「マスター/レプリカ」モデルに依存します。単一のコンピューターがデータの「マスター」(「正式な」) コピーを保持します。システムにあるその他のすべてのコンピューター (「レプリカ」) も、マスター・データの一部またはすべてのコピーを持つことができます。レプリカがデータを変更した場合、そのデータはマスターに送信され、受け入れられるまで、「正式でない」データです。データがマスターに受け入れられた後、そのデータを送信したレプリカ (および、その他のレプリカ) は新しい正式なデータのコピーを要求できます。このようにして、レプリカは一時的にマスターとの同期から外れることがありますが、この差分はすぐに訂正できます。

solidDB のソリューションでは、ほぼ無数のレプリカがデータにアクセスできます。ローカルな書き込みが正式なデータになるには、マスターで受け入れられる必要があるということをユーザーが認識している限り、各レプリカでデータの読み取りと書き込みができます。レプリカは、ネットワークに常時接続している必要はありません。レプリカは、一時的にネットワークに接続されるだけの PDA およびその他のコンピューターに保管できます。そのためレプリカは、マスターから独立して長時間動作できます。これによって、柔軟性が提供されます。また、マスター・データベース・サーバーが、保守のため、または問題を訂正するためにシャットダウンされたというだけで、システム全体が使用不可になることはありません。solidDB の同期テクノロジーに基づく分散システムは、切り分けられた障害に関して、本質的に堅固です。

レプリカ・データベースは、マスター・データの一部またはすべてのコピーだけではなく、マスターと共用しない「ローカル」データも保持することができます。単一のコンピューターが、ローカル・データと共用データの混合データを保持することができます。

典型的なアプリケーション

以下は、solidDB の同期テクノロジーの使用例です。

銀行は中央のデータベースにすべての口座情報を保持しています。支店は、その支店を利用する顧客データのサブセットを取得します。

移動中の営業員または修理担当者は、モバイル・コンピューティング・デバイスに顧客情報や製品情報のサブセットのコピーを保持し、本店がデータの完全なセットを保持します。

通信機器のラックには、それぞれ独自のメモリーを搭載した回線カードが多く設置されています。1 枚のカードを、特定の回線カードへの接続を割り当てる「マスター」として機能させることができます。他の回線カードは、それぞれが接続のサブセットを処理するレプリカとして機能させることができます。例えば、各カードは、特定の範囲のネットワーク・アドレスを担当します。

コンテンツ・プロバイダーはファイルの完全なセット (映画や音楽など) を保持し、各顧客で使用のためにダウンロードすることができます。このタイプのアプリケーションは「1 対多数」の関係には限定されない点に注意してください。顧客は複数のコンテンツ・プロバイダーと関係しうるので、コンテンツ・プロバイダーと顧客との間には「複数対複数」の関係があります。

データの交換方法

solidDB では、双方向のデータ・フローが可能です。マスターはレプリカにデータを送信でき、レプリカはマスターにデータを送信できます。

マスターからレプリカへのデータ送信は、「パブリッシュおよびサブスクライブ」と呼ばれるモデルを使用して行われます。これは、マスターが「パブリッシュ」するデータを、レプリカが「サブスクライブ」することができるという概念に大まかに基づいているからです。

レプリカからマスターにデータを送信することを、データの「伝搬」と呼んでいます。

マスターからレプリカへのデータの送信

マスター・データベースは、必要に応じて、多くのデータを共用することも少ないデータを共用することもできます。マスターのユーザーは、レプリカが要求できるデータ集合である「パブリケーション」を作成します。パブリケーションは、ビューに似ています。これは、データの取得元の表と、組み込むデータ部分を指定することで定義されるデータ集合です。

このようなパブリケーションが作成されると、レプリカはそれらパブリケーションに「登録」できるようになります。

レプリカは、パブリケーションに登録すると、そのパブリケーションからリフレッシュを取得できるようになります。レプリカは、パラメーターを使用して、パブリケーションのサブセットだけを要求できます。例えば、マスターが、全顧客の請求のサマリーをパブリッシュすることを決定するとします。支社にあるレプリカには、その支社が担当する顧客の請求のサマリーだけが必要であるとします。レプリカに必要な請求のサマリーを指定するために、レプリカは、パブリケーションのデータの検索基準として機能する入力パラメーターを指定できます。

手順の概要は、以下のとおりです。

1. 指定されたデータ (全顧客の請求のサマリーなど) で、マスターがパブリケーションを作成。

2. そのパブリケーションにレプリカが登録。
3. レプリカがそのパブリケーションからのリフレッシュを要求し、全データまたはデータの一部を取得。

「パブリッシュ・アンド・サブスクライブ」のモデルは、雑誌の出版と購読に似ていますが、同じではないことに注意してください。大手の出版社は、さまざまな月刊誌 (SQL 関連、C 関連、Java™ 関連など) を販売しています。出版社は、各号に載せる情報を決定します。購読者は内容を規制しません。購読者として、これらの出版物の 1 つ以上を受け取るように登録します。例えば、SQL 雑誌の購読料を送付します。

雑誌出版の世界と拡張レプリケーション同期の世界との主要な違いは、solidDB 同期では、パブリッシャーではなく受信者が、いつ新しいデータを取得するかを決定する点にあります。雑誌出版では、雑誌社がいつ雑誌を送るかを決定します。購読者が月刊のコンピューター雑誌を購読する場合、出版社は、購読者に読む時間があるかどうかにかかわらず、購読者に毎月雑誌を送ります。しかし、拡張レプリケーション同期の世界では、サブスクライバーが、いつ新しいデータを要求するかを決定します。このように、solidDB 同期は、印刷物としての出版よりも Web 出版に似ています。Web 出版では、購読者が出版社に支払いを行い、出版社は好きなときに Web サイトにデータを配置し、購読者は好きなときに Web サイトにアクセスします。

受信者にいつデータを送信するかをパブリッシャーが決定する状態の説明に、「プッシュ」という単語を使用します。パブリッシャーからいつデータを取得するかを受信者が決定する状態の説明に、「プル」という単語を使用します。Web 出版 (および solidDB 同期) では、読者がデータを「プル」します。

一部のアプリケーションでは、データが更新されるとすぐに、マスター (パブリッシャー) がそのデータを送信することが重要です。solidDB の同期テクノロジーには、サブスクライブの価値がある新しいデータがあるときに、マスターがレプリカに通知できるプル同期通知機能があります。 solidDB のプル同期通知機能は、「プッシュ」テクノロジーと同等ですが、より柔軟性が高い機能です。

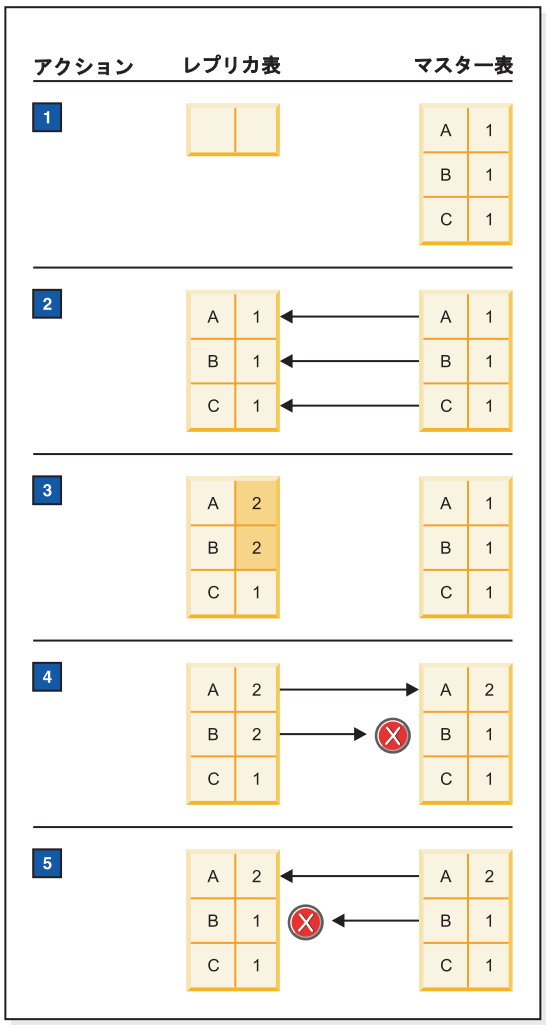
solidDB 同期は雑誌の出版と似ていて、顧客 (レプリカ) は複数のマスター・データベース (複数の雑誌) をサブスクライブできます。3 つの異なる雑誌を購読したい、すなわち 3 つの異なるマスター・データベースをサブスクライブしたい場合は、そのようにできます。ただし、各マスターからのデータは、レプリカで個別のカタログ (論理データベース) に保管する必要があることに注意してください。 (1 つのデータベース・カタログには、1 つのマスター・データベースからのデータだけを含めることができます。ただし、単一物理データベースに複数のデータベース・カタログを含めることができます。そのため、レプリカ・データベース・サーバーには、複数のマスター・データベースからのデータを含めることができます。1 台のサーバーに、マスターとレプリカの両方のデータベース・カタログを含めることもできます。これらのトピックについては、本書で後述します。)

solidDB 同期は、別の点で、雑誌の出版と購読に似ています。solidDB 同期では、サブスクライバーは情報の「フル」または「インクリメンタル」ダウンロード (またはリフレッシュ) を要求できます。インクリメンタル・リフレッシュには、最新のリフレッシュ以降に行われた変更だけが含まれます。これは、雑誌の最新号を受け取ること、または新しい記事だけを Web サイトで見ることに似ています。あるい

は、solidDB サブスクライバーは、フル・リフレッシュを要求できます。これは、雑誌のバック・ナンバーを全巻注文すること、または現行の Web サイト全体を見ることに似ています。当然のこととして、solidDB システムのレプリカは、必ずフル・ダウンロードから開始します。その後、その方が望ましければ、インクリメンタル・リフレッシュだけを要求できます。(注: 「リフレッシュ」という用語は、前のデータの更新という意味ですが、ここでは、「リフレッシュ」を広い意味で使用して、後続のダウンロードだけではなく、初期ダウンロードも指します。)

レプリカからマスターに送信されたデータは、マスターで自動的には受け入れられません。solidDB の「インテリジェント・トランザクション」機能によって、マスターは、有効なデータだけがマスターに保管されるように、データを拒否または変更できます。これについて詳しくは、 21 ページの『インテリジェント・トランザクション』を参照してください。

以下の図で、マスターと単一レプリカ間の伝搬およびリフレッシュのプロセスの概要を示します。この図には、マスターに受け入れられたレコードと、(マスターで実施されたビジネス・ルールによると、更新が無効であるなどの理由で) マスターに拒否された別のレコードの例が含まれています。



1. 初期状態
2. フル・リフレッシュ
3. Anne Anderson (A) と Barry Barrymore (B) が自分のレコードを更新し、後で伝搬するために保存します。
4. 更新をマスターに伝搬します。この例では、マスターは、Anne の変更を受け入れますが、Barry の変更は拒否します。
5. インクリメンタル・リフレッシュ。マスターは、Anne の更新は返信しますが、Barry の更新を拒否したので、Barry の更新は返信しません。Barry のレコードは、最後の正式値のまま残ります。

本書で、後ほど、インクリメンタル・リフレッシュ中に発生する処理を詳しく説明します。具体的には、事前更新された Barry のレコードを最後の「正式」値にリストアする処理について説明します。

図 1. 伝搬およびリフレッシュ

solidDB システムは、各レプリカにとってどのデータが「新規」かを追跡するように構成できます。ユーザー自身は、インクリメンタル更新だけを要求するために、何も追跡する必要はありません。さらに、レプリカごとのインクリメンタル・データの必要性は、独立して追跡されます。1 週間、リフレッシュしなかったレプリカは、最近 1 週間分の変更を受信します。1 時間前にリフレッシュしたレプリカには、最近 1 時間分の変更だけが送信されます。

レプリカからマスターへのデータの送信

データがレプリカからそのマスターに送信されるとき、これを、データがマスターに「伝搬」されると言います。REFRESH 操作の場合と同様に、データ送信のタイミングと量は、レプリカが決定します。

データがレプリカからマスターに伝搬されるとき、レプリカからマスターに送信される「メッセージ」を作成することで伝搬が実行されます。このメッセージには、データの行ではなく、SQL ステートメントが含まれます。単一メッセージに複数のステートメントを含めることができ、実際には、複数のトランザクションを含めることもできます。 (トランザクションのフラグメントをマスターに送信する方法はありません。メッセージで送信した SQL ステートメントは、そのメッセージでコミットする必要があります。1 つのメッセージを複数のトランザクションに分割することはできますが、1 つのトランザクションを複数のメッセージに分割することはできません。)

使用できる SQL ステートメントには、ほとんど制限事項がありませんが、実際問題として、マスターに送信するステートメントは通常、レプリカで実行するステートメントと同じです。つまり、マスターに生データを送信する代わりに、レプリカで実行したのと同じ操作をマスターで実行する、一連のステートメントを送信します。例えば、レプリカで以下の手順を実行したとします。

```
INSERT INTO employees (id, name) VALUES (12, 'Michelle Uhuru');
UPDATE employees SET department = 'Telecommunications' WHERE id = 12;
```

この場合、同じコマンドをマスターに送信して、マスターで同じ手順を繰り返します。

注意:

レプリカで影響を与えたレコードよりも、マスターで影響を与えるレコードが多くなる可能性があるステートメントを実行するときは、注意が必要です。例えば、WHERE 節でユニークな従業員 ID を使用するコマンドは、マスターで、レプリカの場合と同じ効果を持つと考えられます。しかし、「適用範囲が広い」WHERE 節を使用するコマンドは、レプリカにマスターのデータのサブセットだけがある場合、マスターではレプリカよりも多くのレコードに影響を与える可能性があります。

メッセージを作成するには、

```
SAVE
```

ステートメントを使用します。疑似コードでは、以下のようになります。

```
INSERT INTO employees (id, name) VALUES (12, 'Michelle Uhuru');
UPDATE employees SET department = 'Telecommunications' WHERE id = 12;
-- 後でマスター・データベースに伝搬するために
-- ステートメントを保存。
SAVE INSERT INTO employees (id, name) VALUES (12, 'Michelle Uhuru');
SAVE UPDATE employees SET department = 'Telecommunications' WHERE id = 12;
COMMIT WORK;
```

(詳しくは、 109 ページの『後で伝搬するためのトランザクションの保存』 および「*solidDB SQL ガイド*」の付録の構文を参照してください。)

伝搬される SQL ステートメントにストアード・プロシージャー呼び出しを含めることができることに注意してください。

伝搬されたデータのマスターでの受け入れ

レプリカがデータをマスターに伝搬するときに、マスターでそのデータを受け入れる必要はありません。マスター/レプリカ・システムでは、マスターだけがデータを「正式」と宣言する権限を持ちます。データベース・ルール (参照整合性制約など) またはビジネス・ルール (顧客が一定のクレジット制限を超えるのを禁止するなど) に違反するデータをマスターが受け取った場合、マスターはデータを拒否または変更できます。

マスターには、データを変更して適合させるなど、別のオプションもあります。例えば、顧客が在庫数よりも多くの製品をオーダーした場合、マスターは、在庫として残っている製品すべてを送るように顧客のオーダーを変更することもできます。このようにすれば、顧客のオーダーを完全には断らずに済みます。

要約

solidDB の拡張レプリケーション・テクノロジーは、マスター/レプリカ・モデルを使用します。マスターにはデータの「正式な」コピーが格納されており、レプリカは、そのデータにサブスクライブできます。また、レプリカは、データのローカル・コピーを変更し、そのトランザクションをマスターに伝搬することができます。ただし、マスターは、マスター・データの整合性を維持するために、そのトランザクションのデータを変更または拒否する権限を持っています。

レプリカがパブリケーションからのデータのリフレッシュを要求すると、データがマスターからレプリカに送信されます。レプリカは、マスターに対して、「REFRESH」コマンドを含むメッセージを作成します。

レプリカが、マスターにトランザクションを伝搬するためのメッセージを作成および送信すると、データがレプリカからマスターに送信されます。

メッセージ内に、伝搬トランザクションと REFRESH 要求の両方が含まれる場合、双方向の同期が行われます。

以降の各章では、マスターとレプリカ間で双方向にデータを送信する操作を実行する実際の構文を示します。 29 ページの『3 章 データ同期の概要』に、同期の実例が記載されており、関連する各ステップを参照できます。

2 solidDB データ同期アーキテクチャー

solidDB 拡張レプリケーション・アーキテクチャーの概念

solidDB 拡張レプリケーション・アーキテクチャーは、複数層データ冗長モデルに基づいています。同じシステム内の複数データベース内に同じデータが存在する場合、そのデータは冗長であるといわれます。このことは、同じデータ項目に対して複数の、場合によっては一時的に異なるバージョンが共存可能なことを意味しています。

複数層冗長モデル

複数層のデータ冗長モデルは、最上位のマスター・データベース 1 つと、その下位に属する複数のレプリカ・データベースから構成されます。レプリカは更新可能ですが、レプリカのデータは、マスター・データベースにコミットされるまでの間は、常に*一時的* なデータです。レプリカは、階層でそれより下位に属する他のレプリカのマスターとして役割を果たすことができます。

このモデルでは、データベース間で双方向の非同期データの同期メカニズムのインプリメンテーションが可能であるため、マルチデータベース・システムのデータベースの整合性とスケーラビリティーに関する問題に完全に対処します。

複数層のデータ冗長モデルは、以下の原則に基づきます。

- データ項目 1 つに対して、マスター・バージョン が 1 つ存在します。これは正式なバージョンのデータとして考えられます。その他のコピー項目は、*一時的*なバージョン、つまり*レプリカ* です。
- レプリカは、それより下位に属する他のレプリカのマスターとして役割を果たすことができます。マスターでもあるレプリカには、それより上位の マスター・バージョン のデータのサブセット (またはフル・セット) が格納されます。
- システムのどのデータベースのデータも更新可能です。
- マスター・データベースに直接変更を加えた場合、その変更内容は正式なものです。
- レプリカ・データベースでコミットされたトランザクションは、マスター・データベースに正常に伝搬され、そこでコミットされるまでは、一時的なものです。
-

マスター・データベースからレプリカに変更データを送信すると、レプリカ・データベースはリフレッシュされます。

基本的な 2 層アーキテクチャー

このモデルの最も単純なインプリメンテーションは、 図 2 に示す 2 層の同期アーキテクチャーです。トランザクションは常に、コミットされるマスター・データベースに送信されることに注意してください。次に、変更されたマスター・データがレプリカに送信されます。

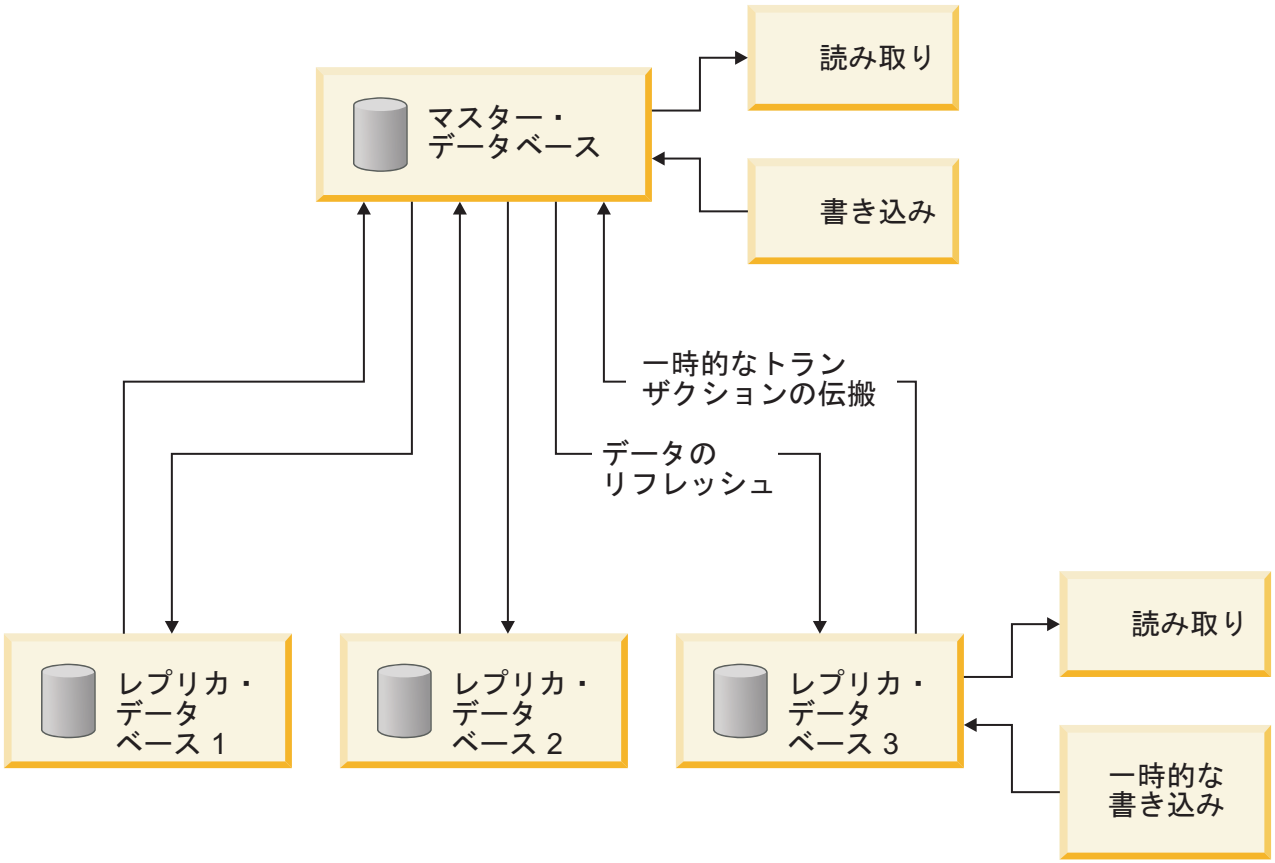


図 2. 2 層データ冗長アーキテクチャー

拡張複数層アーキテクチャー

このモデルのより拡張されたインプリメンテーションによって、2 つより多くの層で、マスターでもあるレプリカから、階層でそれより下にある別のレプリカに情報が流れるようにすることができます。このアーキテクチャーは、ローカル領域へのシステム情報の流れと、さまざまなローカル領域から特定のエンド・ノードへの情報の流れが必要なシナリオでよく使用されます。

注:

3 つ以上の層を持つシステム、すなわち、マスターとレプリカの両方であるノードを持つシステムを説明する場合は、「多重層」という用語を使用します。2 つ以上の層を持つシステムを説明する場合は、「複数層」という用語を使用します。

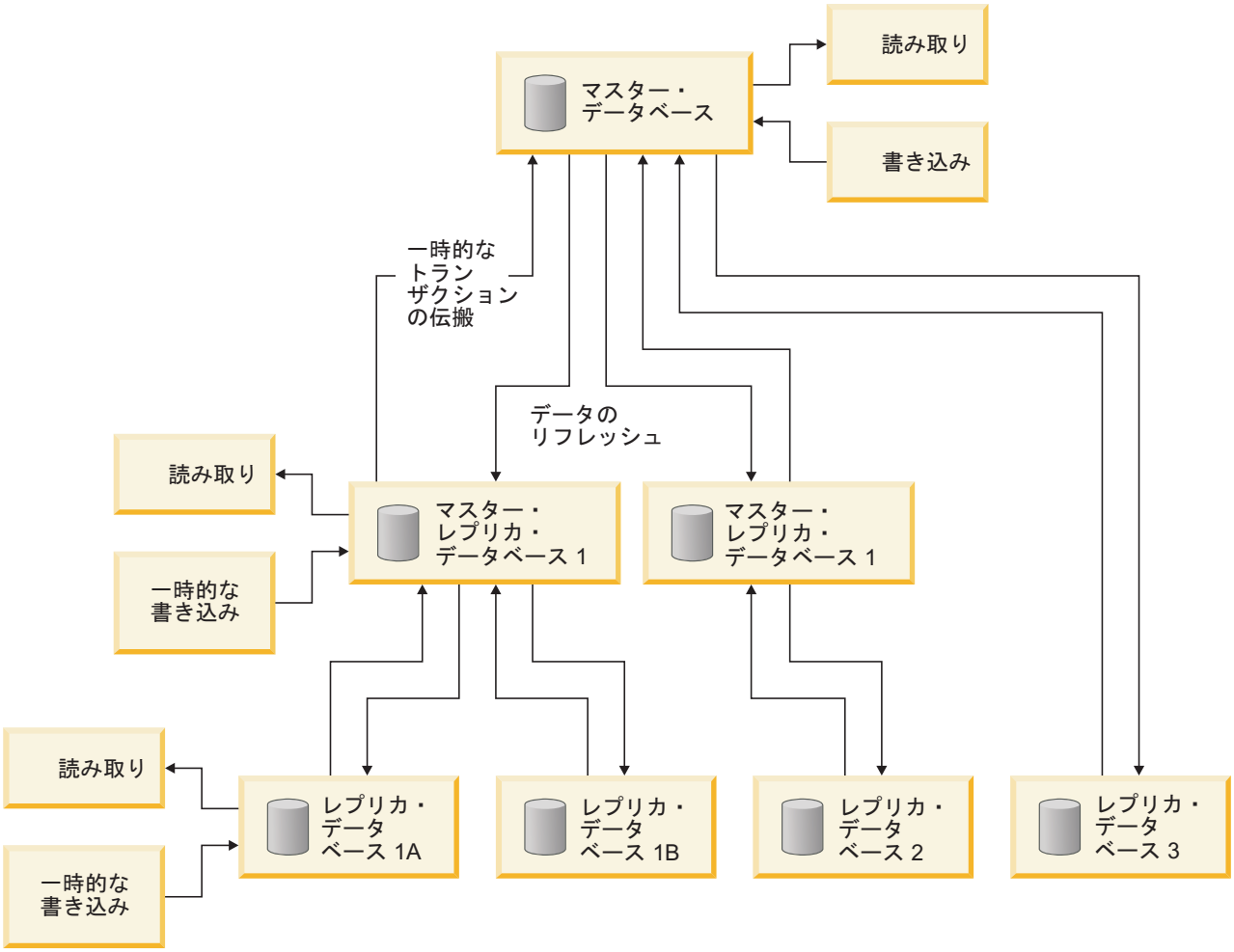


図 3. 複数層データ冗長アーキテクチャー

solidDB 拡張レプリケーションのトランザクション・モデル

2 層アーキテクチャーと複数層アーキテクチャーの両方で、再検証されたレプリカ・データを変更するトランザクションは、常に一時的なものです。一時的なトランザクションは、マスター・データベースに受け入れられた時点で正式なものになります。このことは、トランザクションのライフ・サイクル全体は、レプリカのコミット時点からマスターのコミット時点まで拡張されていることを意味しています。この段階の間、レプリカによって既にコミットされたトランザクションを無効にする可能性があるアクティビティーが、アプリケーション内で行われることがあります。したがって、マスター・データベースに伝搬されるすべてのトランザクションは、マスター・データベースの保全性を保証するのに十分な妥当性検査ロジックを、それ自身の内部に組み込んでおく必要があります。

同期マルチデータベース・システムのデータ保全性の問題に対処するため、solidDB 拡張レプリケーションは、新しいトランザクション・モデルである solidDB インテリジェント・トランザクションを導入しています。このモデルは、開発者に対して、マスター・データベースを常に整合した状態に保つようトランザクションをインプリメントする方法を提供します。詳しくは、21 ページの『インテリジェント・トランザクション』を参照してください。

レプリカ・データベースでコミットされたトランザクションは、そのレプリカのマスター・データベースに対してのみ伝搬されることに注意してください。これらのトランザクションは、他のレプリカに直接には伝搬されません。その代わり、他のレプリカは、1 つ以上のパブリケーションからのリフレッシュを要求することにより、マスター・データベースに変更データを要求することができます。(「パブリケーション」は、マスター上のデータ集合です。クライアントは、パブリケーションからリフレッシュして、マスターから更新データを取得できます。)

マルチマスター同期モデル

今日の分散型ネットワーク環境では、1 つのシステムが複数のアプリケーションから構成される可能性があります。これらのアプリケーションは、それぞれが独自のデータベースを所有していることがあります。solidDB 拡張レプリケーションでは、複数のマスター・データベースのデータを 1 つの物理データベースに格納できます。例えば、請求ホスト・システムとネットワーク構成ホスト・システムのレプリカを 1 つのローカル・データベースに格納可能です。2 層と多重層のアーキテクチャーでは、複数のマスター・データベースを収容できるように拡張が容易になっています。15 ページの図 4 に示すように、2 層のアーキテクチャーでは、マルチマスター同期を利用しています。

15 ページの図 4 は、表レベルでのマルチマスター同期の概念も表しています。この図では以下の点に注意してください。

- データベース・サーバーは、複数のマスターからのレプリカ・データベースを格納できます。
- システム A およびシステム B は、それぞれ分離し、独立しています。
- レプリカ・データベースごとに、データベース・サーバーにデータベース・カタログが 1 つ作成されます。
- レプリカ A はマスター A と同期し、レプリカ B はマスター B と同期します。
- データベース・サーバーは、カタログに、1 つまたは複数のマスター・データベースまたはローカルのみのデータベースを入れることも可能です。

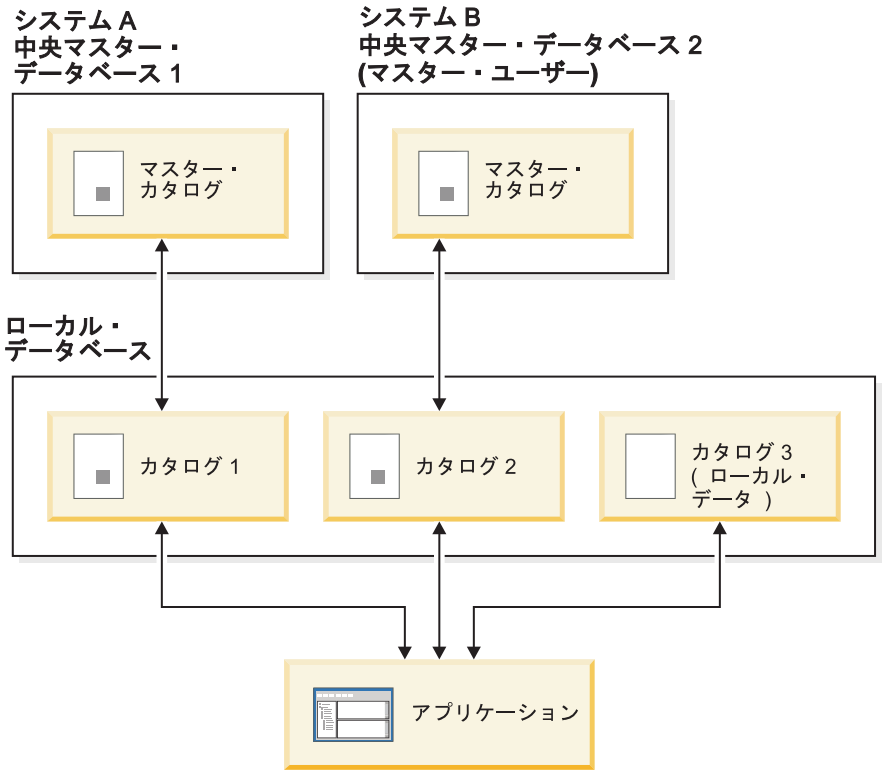


図 4. マルチマスター・モデル

マルチマスターの機能

solidDB 拡張レプリケーションのマルチマスター・モデルの機能は、以下のとおりです。

- 登録によってレプリカは solidDB 拡張レプリケーションの複数のマスターとデータを同期できます。
- カタログを使用して、マスターが異なるレプリカのデータを区別します。
- ローカル・データと共用データを分離します。

これらの機能を以下のセクションで説明します。

マルチマスター環境におけるレプリカ・データの管理

solidDB データベースは、複数の独立したパーティション (カタログ) に分割することができます。また、そのカタログ自体も、複数の独立したスキーマ に分割できます。データベースの中に複数のトピックが含まれる場合、またはデータベースが複数のアプリケーションで使用される場合に、カタログ単位にデータベースを分割できると便利です。一般に、各アプリケーションは、個別のデータベース・カタログにデータを保管するものです。

マルチマスター環境の場合、複数のカタログを設定できるため、1 つのデータベース・サーバー内の複数のデータベース (マスターまたはレプリカ) を同期化用として指定することができます。例えば、構成管理データベースのレプリカ用に 1 つ、サブスクライバー・プロビジョニング・システム用に 1 つというように、アクセス・ルーターの solidDB サーバーに 2 つのカタログを設定することが可能です。

以下のセクションでは、solidDB データベースをパーティション化し、同期できるようにする際の概念と背景について説明します。

solidDB データベース、カタログ、およびスキーマ

solidDB は、ファイル (またはファイル集合) にデータを保管します。これらのファイルは、*物理データベース* と呼ばれています。 これらのファイルの場所は、solid.ini 構成ファイル内に指定します。これらのファイルは、ユーザー指定の 1 つまたは複数のディレクトリーに格納できます。

各 solid.ini 構成ファイルは 1 つの物理データベースの場所を指定するので、理論的には、単一の solidDB プログラムを異なる solid.ini ファイルで始動するだけで、このサーバーは異なる時点で異なる物理データベース上で動作できます。複数の物理ファイル (および solid.ini ファイル) を作成して複数の物理データベースを持つことができますが、1 つの solidDB インスタンスは、一度に 1 つの物理データベースのみに対して、「参照」と「処理」を行います。異なる時点で複数のデータベースで単一の solidDB プログラムを使用することは、めったにありません。

1 つの物理データベース・ファイルは、複数の*論理データベース* を含むことができます。 各論理データベースは、表、索引、プロシージャー、トリガーなどのデータベース・オブジェクトからなる完全かつ独立したグループです。各論理データベースは、*カタログ* と呼ばれています。ただし、 solidDB カタログは、索引 (項目の完全な内容を含むことなく、その項目を見つける機能を持つ、従来の意味でのライブラリー・カード・カタログにおいて見られるのと同様) だけでなく、多彩なデータ・オブジェクトを含むことに注意してください。solidDB について詳しくは、「*solidDB 管理者ガイド*」を参照してください。

原則として、「カタログ」という用語は「論理データベース」を意味するものとし、「データベース」という用語は「物理データベース」を意味するものとします。

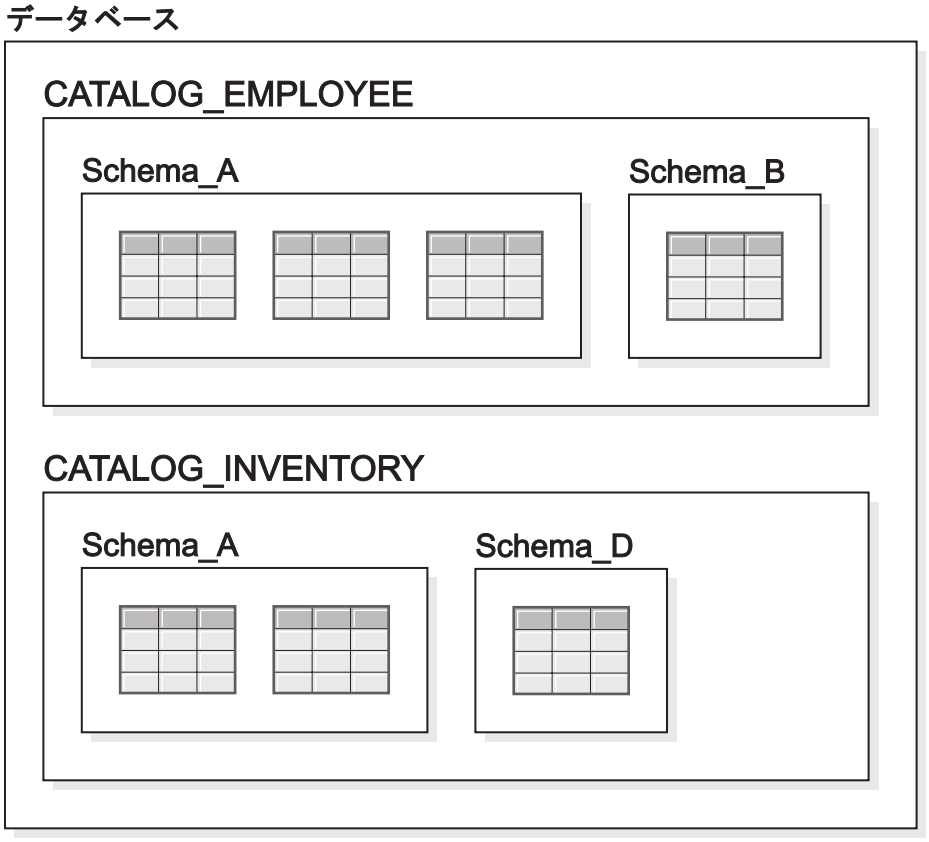


図 5. データベース、カタログ、およびスキーマ

カタログとその中のオブジェクトは、階層内に配置されています。図 5 に示すように、solidDB カタログは、スキーマ と呼ばれるデータベース・オブジェクトを持つことができます。各カタログ内には、複数のスキーマが存在する場合があります。次に、各スキーマは、表、ビュー、シーケンスなど、複数のデータベース・オブジェクトを持つことができます。カタログは階層の最上部にあり、表などのオブジェクトは階層の最下部にあります。

スキーマ内では、各オブジェクトの名前はユニークでなければなりません。例えば、同じスキーマ内に「table1」という名前の表を 2 つ持つことはできません。同様に、カタログ内では、各スキーマの名前はユニークでなければなりません。例えば、同じカタログ内に、「smith_schema」という名前のスキーマを 2 つ持つことはできません。

1 つのスキーマ内ではオブジェクト名はユニークでなければなりませんが、異なるスキーマ内の異なるオブジェクトは同じ名前を持つことができます。例えば、以下の場合は両方とも正当です。

```
smith_schema.table1
jones_schema.table1
```

同様に、1 つのカタログ内ではスキーマ名はユニークでなければなりませんが、異なるカタログ内には、同じ名前のスキーマが存在しても構いません。例えば、以下の場合は有効です。

```
employee_catalog.smith_schema.table1
inventory_catalog.smith_schema.table1
```

上記のように、オブジェクト名の前にスキーマ名とカタログ名が付く場合、そのオブジェクト名は「完全修飾されている」といわれます。つまり、曖昧さがありません。データベース・オブジェクト名は、DML ステートメント内で以下のように修飾されます。

```
catalog_name.schema_name.database_object
```

または

```
catalog_name.user_id.database_object
```

一般に、カタログ内の各ユーザーは、独自のスキーマ (複数可) を持つことが許可されます。例えば、ユーザーは、

```
employee_catalog
```

内に

```
smith_schema
```

を所有することができます。

単一ユーザーが、複数のカタログ内にスキーマを持つことができます。例えば、上記に示すように、ユーザー「smith」は、「smith_schema」という名前のスキーマを複数のカタログ内に持つことができます。各カタログ内のデータベース・オブジェクトが完全修飾名 (つまり、オブジェクト名、スキーマ名、およびカタログ名を含む名前)で指定されている限り、どのデータベース・オブジェクトが必要なオブジェクトか混乱することはありません。カタログおよびスキーマ内でデータを適切に編成することにより、「コンテキスト」を「制限」して、ユーザーやアプリケーションが、自分のタスクに関係するデータベース・オブジェクトだけを参照するようにできます。これらの概念は、同期のためにカタログとスキーマを作成する際に適用されます。

カタログと同期

カタログを使用して、複数の論理データベースを同期するようにインプリメントできます。ローカル物理データベースに複数のマスターのレプリカ・データがある場合、そのレプリカには、コピー (または部分コピー) があるマスター・データベースごとに 1 つのカタログが必要です。レプリカのカタログには、同期データだけではなくローカル・データも含まれていることに注意してください。ローカル・データは、ローカル・データベースだけに属し、他のデータベースに複製されたり、他のデータベースから複製されたりはしません。

このようにして、表はレプリカ・データベースとマスター・データベースの両方で定義され、共用データからローカル・データが区別されます。共用データはマスター・データベースと同期されますが、レプリカにだけまたはマスターにだけ属するデータは、同期のときに変更されません。

マルチマスター環境では、カタログで、異なるマスターのデータを分離しておきます。カタログ間でのオブジェクト名の競合は起きません。異なるマスターで、同じ表名と異なるオブジェクト名が使用されている場合でも、各マスターの個別のカタログ名によって表名とオブジェクト名が修飾され、また、どのオブジェクトがどのマスターに属するかを指定できます。また、solidDB 拡張レプリケーションでは、単一カタログに異なるマスターのオブジェクトが含まれることはありません。ただ

し、カタログの使用ではスキーマのすべてのオブジェクトが同期オブジェクトである必要はないため、カタログにローカル表を含めることができます。

マスターおよびレプリカのスキーマ名は、同じである必要があります。これは、基本的な 2 層アーキテクチャーと一貫性があり、単一マスターに登録された単一レプリカと違いはありません。データベースを作成するときに、デフォルトのスキーマ名が作成されますが、これはデータベース所有者のユーザー ID です。CREATE SCHEMA ステートメントで、データベース内に分離したスキーマが作成されます。スキーマを使用したデータベース・オブジェクト管理について詳しくは、「*solidDB 管理者ガイド*」を参照してください。

solidDB 拡張レプリケーションでは、同期にカタログとスキーマを使用することで、柔軟で拡張が容易なアーキテクチャーが提供されます。1 つのレプリカは、1 つのマスターだけに登録されます。つまり、1 つのレプリカ・カタログは 1 つのマスター・カタログにマップされます。ただし、単一物理データベースは複数のカタログを持つことができます。同じローカル・データベースに、同じまたは異なるデータベース・サーバーの新しいレプリカ・カタログにマップするマスター・カタログをさらに作成すると、追加のマスター・データベースが含まれます。さらに、マスター・カタログとレプリカ・カタログの両方が同じ物理データベースに存在することができます。

13 ページの図 3 で示すように、マスター/レプリカ階層には 2 つより多くの層を入れることができ、階層のいくつかのカタログは、レプリカ・カタログとマスター・カタログの両方として機能することができます。マスターとレプリカの 2 つの役割を持つカタログの定義について詳しくは、57 ページの『マスター・データベースとレプリカ・データベースの定義』を参照してください。

カタログのインプリメントについて詳しくは、58 ページの『マルチマスター・トポロジーのガイドライン』を参照してください。

マルチマスター環境でのトランザクション

マルチマスター環境では、トランザクションは 2 つの異なるマスターにまたがることはできません。例えば、以下は無効になります。

```
SAVE UPDATE table A in CATALOG A
SAVE UPDATE table B in CATALOG B
COMMIT WORK
```

トランザクションは特定のマスター・データベースに伝搬されます。このマスターはトランザクションの途中で変更できないので、トランザクションに保存されたすべてのステートメントは 1 つのマスターだけに伝搬されることになります。solidDB は、1 つのトランザクションが 2 つの異なるマスターのデータを更新しているケースを検出できます。このような場合、エラー・メッセージと共に操作が失敗します。

solidDB 拡張レプリケーションでは、カタログ内の同一トランザクション内でのローカル・データの変更が可能です。SET CATALOG コマンドは、拡張レプリケーションに関連するすべての操作に使用するマスターを明示的に定義します。SET CATALOG コマンドは、どの同期コマンドよりも前に実行され、データベース内で複数のカタログが定義されている場合に必要です。

solidDB 拡張レプリケーション・アーキテクチャーのコンポーネント

solidDB 拡張レプリケーションは、アプリケーション・プログラマーに対して豊富なデータ分散機能およびデータ管理機能のセットを提供します。これらの機能は、アプリケーションの固有のニーズを満足する、信頼性が高く、柔軟で、かつ堅固なデータ分散システムをサポートします。

solidDB 拡張レプリケーション・アーキテクチャーは、以下の機能コンポーネントから構成されています。

- 正式なバージョンと一時的なバージョンのデータを格納するためのマスターおよびレプリカ のデータベース。
- マスター・データベースからレプリカ・データベースに、新しいデータおよび変更データを転送するためのパブリケーションとサブスクリプション。
- レプリカ・データベースからマスター・データベースに変更を伝搬するためのインテリジェント・トランザクション。
- マスターとレプリカ間の安全で信頼性の高い通信をインプリメントするための*非同期ストア・アンド・フォワード・メッセージング*。

このコンポーネントについては、データを同期する上でそれぞれが果たす役割も含めて、以降のセクションで詳細に説明します。

マスター・データベースとレプリカ・データベース

solidDB 拡張レプリケーションを使用する分散システムでは、マスター・データベースは正式なバージョンのデータを格納します。このデータは、ビジネス・アプリケーションのデータおよび同期定義データを含みます。同期定義データは、カタログ、データベース・スキーマ、パブリケーション定義、登録、レプリカ・データベースのサブスクリプションおよびユーザー・アクセス定義などを含みます。

レプリカ・データベースは、ローカル・データ、およびマスターに伝搬されるトランザクションを格納します。すべてのレプリカ・データまたはレプリカ・データの該当部分をマスター・データベースからリフレッシュする必要がある場合は、

```
REFRESH
```

コマンドを 1 つ以上のパブリケーションに送信することによって、そのリフレッシュが可能です。ローカル・データは、ビジネス・アプリケーションのデータ (通常はマスター・データベースのサブセット)、および特定のデータベースに固有の情報を格納しているシステム表のデータを含みます。

複数層の同期環境では、同期データベースは、それぞれがマスターおよびレプリカの二重の役割を果たすように設定可能です。このような役割を設定するには、カタ

ログを作成し、そのカタログがレプリカとマスターの両方になるように定義します。 14 ページの『マルチマスター同期モデル』を参照してください。

パブリケーション、サブスクリプション、およびリフレッシュ

マルチデータベース・システムの同期アーキテクチャーでは、アプリケーションがマスター・データベースからレプリカ・データベースにデータをダウンロードし、このレプリカ・データを必要に応じてリフレッシュできる方法が必要です。

パブリケーション は、レプリカにダウンロードできるマスター・データ集合の定義です。レプリカ・データベースは、サブスクリプション を使用して、マスターからの特定のパブリケーションへのインタレストを登録します。パブリケーションは、レプリカに登録されます。ユーザーは、登録されているパブリケーションからのみデータをリフレッシュ できます。このようにしてパブリケーション・パラメーターが検証され、ユーザーが誤って不要な、または存在しないパブリケーションからリフレッシュしたり、アドホック・リフレッシュ・コマンドを作成することを防ぎます。

初期ダウンロード (「リフレッシュ」) では、常に、フル・パブリケーションのデータが返されます。検索基準 (パブリケーション・パラメーターとして指定) に一致するパブリケーションのすべてのデータがレプリカ・データベースに送信されます。パブリケーションとサブスクリプションについて詳しくは、 100 ページの『パブリケーションの作成』を参照してください。

初期ダウンロードの後、同じパブリケーションに対する後続のリフレッシュ (同じパラメーター値を使用) では、前のリフレッシュ以降に変更されたデータのみを受信します。これを*インクリメンタル・リフレッシュ* と呼びます。 一般に、最新の変更を含むパブリケーションの更新のみをレプリカに送信する必要があります。パブリケーションの作成と、そのインクリメンタルとしての指定は、solidDB SQL の拡張である solidDB 拡張レプリケーション・コマンドで行います。

```
CREATE PUBLICATION
```

コマンドについて詳しくは、 77 ページの『5 章 solidDB 拡張レプリケーション・アプリケーションのインプリメント』を参照してください。

インテリジェント・トランザクション

拡張レプリケーションを理解する上での基本的な考え方は、どのレプリカ・データベースのデータも正式でないため、レプリカで変更されたデータはどれも一時的なものであるということです。変更は、正常に検証され、マスター・データベースでコミットされて初めて正式となります。

この「レプリカで今作成し、後でマスターでコミットする」という条件のため、トランザクションのライフ・サイクルは、数分の 1 秒から、不明確な期間にまで広がります。マルチデータベース・システムでは、トランザクションがレプリカからマスター・データベースに伝搬されるまでの時間は、数秒間から数週間にまで及ぶ可能性があります。この種のトランザクションの課題は、トランザクションが検証され、マスター・データベースにコミットされるときに、マスター・データベースの変更前でも変更後でも、必ず状態の一貫性を確保することです。

データベース整合性の保証

データベースの内容を変更するトランザクションが、そのコミット時に以下の基準を満たしている場合に、データベースは整合性があります。

- 参照整合性規則など、DBMS 固有のルールに違反していないこと。
- ビジネス・トランザクションおよびそれぞれのデータベース・トランザクションに適用するビジネス・ルールに違反していないこと。

伝搬したレプリカ・トランザクションが最終的にマスター・データベースでコミットされたときに、マスター・データベースの状態がレプリカ・データベース (トランザクションの作成元) と一致しないことがあります。他のレプリカから伝搬されたトランザクションにより、またはレプリカの最後のリフレッシュ後にマスター・データベースで直接行われた更新により、マスター・データベースの状態が変更されている可能性があります。そのため、マスター・データベースでは、レプリカ・トランザクションは元の内容で使用できない場合があります。

2 層レプリケーション・モデルでの整合性要件に対応するには、ライフ・サイクル中に無効になる可能性がある各トランザクションに、マスター・データベースでトランザクションをコミットした時に、そのマスター・データベースの整合性が維持されるように、ビジネス・ロジックを必ず組み込む必要があります。データベースでトランザクションの元の動作との不整合が発生すると、トランザクションはこれを検出し、データベースの整合性が維持されるように動作を変更する必要があります。

solidDB 拡張レプリケーションのインテリジェント・トランザクション・モデルは、長期的なトランザクションをインプリメントするためのフレームワークを提供します。拡張レプリケーション・アーキテクチャーにおけるトランザクションの伝搬は、solidDB インテリジェント・トランザクション・テクノロジーに基づいています。マスターで受け入れられるレコードと、受け入れられないレコードがある場合に、データの伝搬とリフレッシュを行う簡単な例を 8 ページの図 1 に示します。

インテリジェント・トランザクションのシナリオ

インテリジェント・トランザクションのインプリメンテーションを説明するために、受注アプリケーションに、顧客がクレジット制限を超過してはならないというビジネス・ルールがあるとします。クレジット制限を超過した場合には、新しいオーダーはできません。

マルチデータベース・システムでは、レプリカ・データベースで顧客がクレジット制限を超過していなくても、マスター・データベースでは同じデータがクレジット制限を超過するということが起こり得ます。この状態では、クレジット制限を超過しているという情報がレプリカ・データベースにはまだ届いていないため、顧客はそのまま、レプリカ・データベースにオーダーを入力できます。しかし、この「新規オーダーの追加」トランザクションがレプリカ・データベースからマスター・データベースに伝搬されたとき、それは「クレジット制限」ビジネス・ルールの違反となるため、そのままの状態ではコミットされません。この場合、トランザクションの動作を変更しなければそれは有効になりません。例えば、マスター・データベ

ース側で、そのオーダーの「状況」列の値を「invalid」に設定して、そのオーダーを有効なオーダーと区別しなければなりません。リフレッシュ時に、無効なオーダーをレプリカ・データベースに戻し、そのレプリカ・データベースのユーザーにトランザクションを失敗したことを通知できます。

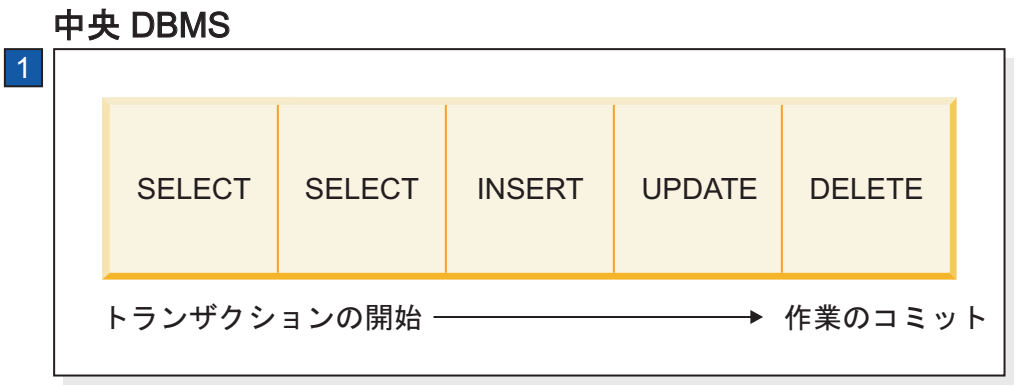
マルチデータベース・システムと集中型システム

中央のデータベースを使用している従来型のクライアント/サーバー・システムでは、各トランザクションの妥当性検査ロジックは、一般的にクライアント・アプリケーションまたはアプリケーション・サーバーのサービスの中にあります。例えば、受注アプリケーションでは、トランザクションのコミット前に、アプリケーション・ロジックにより、新規オーダーが顧客のクレジット制限を超過していないことを検査する必要があります。

マスター・データベースにトランザクションを伝搬するときも、データベースの保全性を確保するために同様の妥当性検査が必要です。唯一の違いは、同期メカニズムが、アプリケーションのトランザクション妥当性検査ロジックを利用できないことです。したがって、ロジックをトランザクション自体に組み込む必要があります。以下の種類の妥当性検査ロジックが各トランザクションに必要です。

- 更新の競合検出
- ビジネス・ルールを使用した妥当性検査
- DBMS エラー処理

集中型システムのトランザクションとマルチデータベース・システムのトランザクションでは大きく異なります。集中型システムでは、トランザクションの存続期間は通常、数分の 1 秒であり、DBMS にロック方式メカニズムが装備されているため、更新の競合が発生する可能性がありません。



1. ローカル・ユーザーは、マスター・アクセス権限を持たない場合、同期操作を実行できません。

図 6. 中央のデータベースのトランザクション

上の図は、典型的なトランザクションの例です。トランザクションでは、トランザクションの内容を検証するために、書き込み操作の前に照会が何件か行われます。例えば、受注システムは、顧客の新規オーダーを作成する前に、顧客のクレジット制限を超過していないことを検査することが可能です。トランザクション中は、サーバーの並行性制御メカニズムがデータの同時使用による更新の競合などの問題を処理します。

マルチデータベース・システムでは、トランザクションは最初にレプリカ・データベース内で作成されて保存されますが、後でデータベース同期化処理の一部としてトランザクションがマスター・データベースに伝搬されたときに、最終的にそこでコミットされます。一時的にコミットされたトランザクションは、期間が制限されずに、システムの中に存在することができます。つまり、トランザクションのライフ・サイクルがまったく異なります。

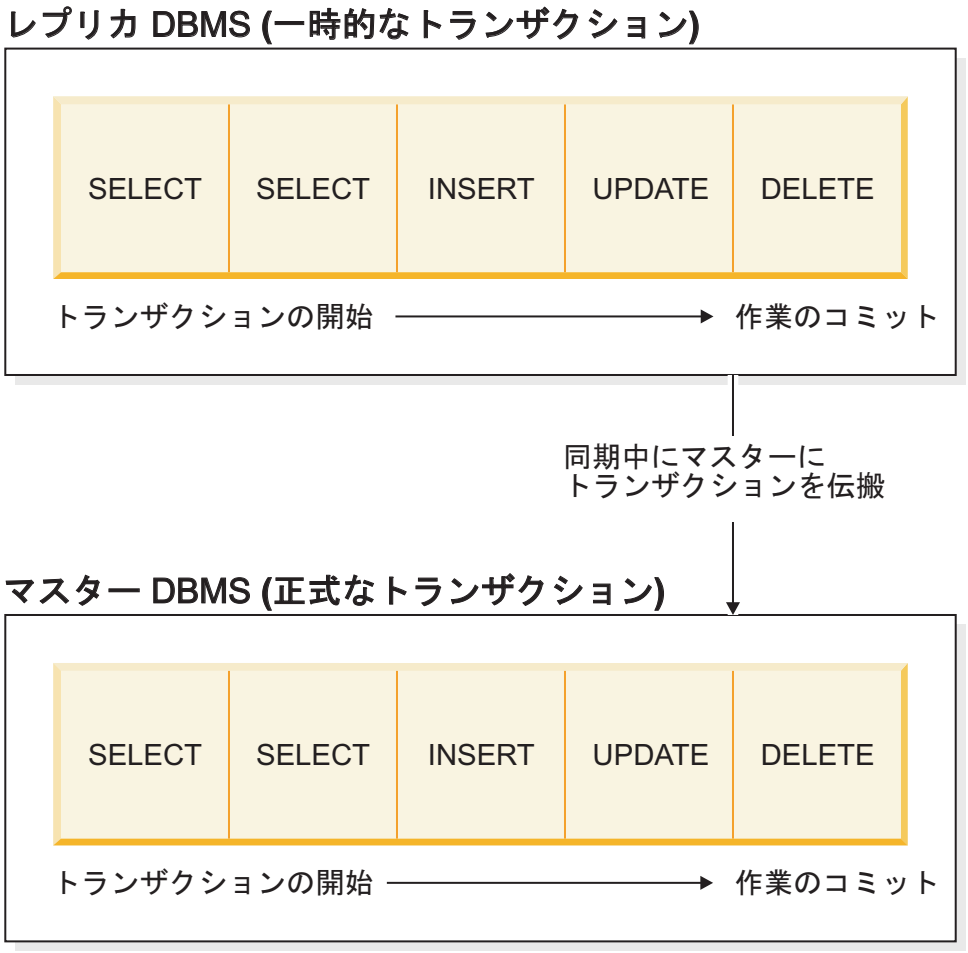


図 7. 同期データベースのトランザクション

マルチデータベース・システムにおけるインテリジェント・トランザクション

マルチデータベース・システムでは、トランザクションには、「2 つの段階」があります。トランザクションは、まずレプリカ・データベースの中に一時的なデータとして作成された後、ビジネス・アプリケーションによって検証され、コミットされます。後でマスター・データベースに伝搬できるように、トランザクションはレプリカ・データベースに保存されます (トランザクション・キューに入れられます)。「2 段階」目でトランザクションは、マスター・データベースに伝搬されます。ここで、マスター・データベースには、レプリカ・データベースと同じ妥当性検査ルーチン (つまり、レプリカ・データベースで実行された照会) の実行が必要になり

ます。例えば、トランザクションの有効性を確認するためにレプリカ・データベースで顧客のクレジット制限が検査された場合、トランザクションがコミットされる前に、通常マスター・データベースでも同じ操作が行われなくてはなりません。そうでなければ、トランザクションの有効性は、マスター・データベースでは保証されません。

2 層のデータ冗長モデルのライフ・サイクルの延長をサポートするために、データベース・トランザクションの拡張機能が開発されました。solidDB インテリジェント・トランザクションでは、トランザクションの有効性を保証するため、マスター・データベースにおけるトランザクション自体の検証と、トランザクションの動作の調整が可能です。

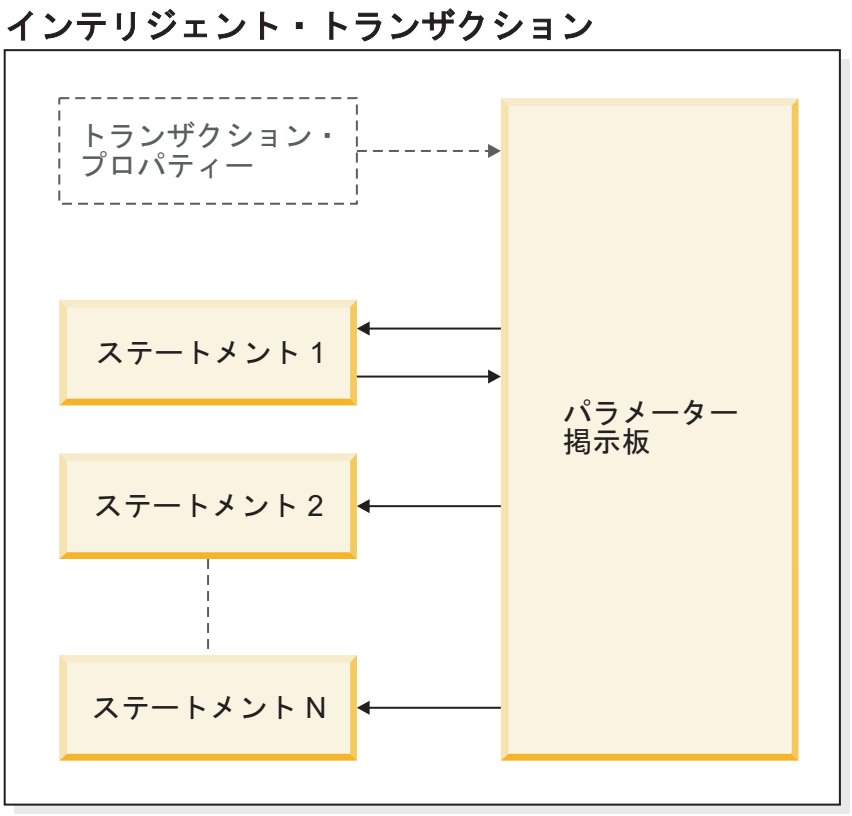


図 8. インテリジェント・トランザクションの構成

インテリジェント・トランザクション機能の仕組み

solidDB インテリジェント・トランザクション機能を使用すると、トランザクションは現行のデータベース内で自身の検証を行えるだけでなく、元の動作が無効な場合は、その動作 (データベース操作) を変更することができます。

トランザクションのステートメントには任意の SQL ステートメントを使用することができますが、ストアード・プロシージャーに対する呼び出しが一般的です。ステートメントには、異なる環境や状況でのステートメントの妥当性を確保するために必要なロジックを含める必要があります。

マスター・データベース内でトランザクションを実行する場合、トランザクションのステートメントは、トランザクション・パラメーター を、同じトランザクションの後続のステートメントが読み取れるようにパラメーター掲示板に入れることによって、お互いに通信を行うことができます。 トランザクション・パラメーターは、ステートメントのこの通信機能によって、自身の検証と、現行の環境に応じた動作の調整が可能なトランザクションを作成することができます。

例を以下に示します。

この例は、このセクションの冒頭で紹介したインテリジェント・トランザクションのシナリオに適用されます。このシナリオでは、マスター・データベースに伝搬されたトランザクションが、クレジット制限を超えた顧客に新しいオーダーを追加しようとします。この例では、トランザクションの背後の SQL ステートメント・ロジックを紹介します。

トランザクションは以下の操作を行います。

- CUST_ORDER 表に行を挿入します。
- CUSTOMER 表の CREDIT 列を更新します。

以下の処理が発生します。

1. 新しい行を CUST_ORDER 表に挿入する前に、 INSERT_ORDER プロシージャーが顧客のクレジットに問題がないことを確認します。この例では、問題があることにします。
2. したがって、インテリジェント・トランザクションが CUST_ORDER 表に、別の STATUS 値 (例えば STATUS = 'Not approved'など) を設定した新しい行を挿入します。
3. オーダーが無効なので、CREDIT 列の更新操作を実行してはなりません。したがって、INSERT_ORDER プロシージャーは、名前が「ORDER_FAILED」で値が「YES」であるパラメーターを掲示板に入れます。
4. UPDATE_CUST_CREDIT プロシージャーは掲示板を確認し、このプロシージャーに適用される情報があることを検出します。
5. これで UPDATE_CUST_CREDIT プロシージャーは、クレジット金額を更新してはならないことを認識します。
6.

後でレプリカが最新のデータを取得するためにリフレッシュされる際に、顧客オーダーに関するレプリカ自体の情報が更新され、レプリカのデータはオーダーが処理されなかったことを正しく示します。このようにして「ループを閉じ」、レプリカが要求/更新を実行しようとしてできなかった場合でも、マスターとレプリカの両方に、正しいデータが確実に含まれるようにします。

非同期ストア・アンド・フォワード・メッセージング

solidDB 拡張レプリケーションのマスターとレプリカの通信は、非同期ストア・アンド・フォワード・メッセージングに基づいて行われます。各メッセージは動的に組み立てられ、多数の同期タスクを含むことができます。例えば、1 つのメッセージで、複数のトランザクションをレプリカからマスターに伝搬して、マスター・データベースからの多数のパブリケーションにリフレッシュを要求することが可能です。

レプリカからのメッセージはマスター・データベースに非同期に送信されます。solidDB 拡張レプリケーション・アーキテクチャーに組み込まれているメッセージ・キューイング機能では、メッセージ全体が受信ノードに到達する前に、送信ノードからメッセージが削除されることはありません。

3 データ同期の概要

solidDB は IBM Corporation が特許権を有するデータ分散テクノロジーを特長としており、このテクノロジーは拡張レプリケーションと呼ばれています。以降の各章で拡張レプリケーションという用語を頻繁に使用します。

この章では、基本的な拡張レプリケーション機能を示すサンプル・スクリプトの実行について詳細に説明します。この章は、「*solidDB 管理者ガイド*」で説明している solidDB の基本的な管理機能を理解していることを前提としています。以下に関する知識が前提となります。

- solidDB 基本的な管理 (インストール、開始、シャットダウン、ネットワーク名の構成など)。詳しくは、「*solidDB 管理者ガイド*」の関連する章を参照してください。
- solidDB の操作 (データベースへの接続、SQL ステートメントの実行)
- solidDB SQL ストアード・プロシージャーのプログラミング
- solidDB 拡張レプリケーション・アーキテクチャーの基礎。この章を使用する前に、必ず 11 ページの『2 章 solidDB データ同期アーキテクチャー』を参照してください。

始める前に

拡張レプリケーションでは、SQL ステートメントで同期がインプリメントされます。この章で説明するデータ同期 SQL ステートメントは、SQL エディター (テレタイプ)、ODBC、JDBC、Light Client API で使用できます。API を使用した solidDB への SQL の受け渡しについて詳しくは、その API のパブリケーションを参照してください。(API のパブリケーションについては、「*solidDB プログラマー・ガイド*」を参照してください。)

サンプル・スクリプトの使用を始める前に、スクリプトを実行できるように solidDB 環境を準備する必要があります。solidDB 評価環境のセットアップについては、「*solidDB スタートアップ・ガイド*」という別のパブリケーションを参照してください。

拡張レプリケーションのインプリメンテーション

この章では、solidDB をセットアップ、構成、および使用して、拡張レプリケーション・テクノロジーを使用する例を順を追って簡潔に説明します。

表 3. 章の要約

ステップ	アクション	説明
1	始めに	2 台の solidDB データベース・サーバーのインストールと始動について説明します。
2	サーバーの構成とパブリケーションの作成	サーバーを、マスターおよびレプリカとして構成します。拡張レプリケーション設計要件を満足する表を (マスター・データベースおよびレプリカ・データベースの両方で) 定義します。マスター・データベース内にサンプル・パブリケーションを定義します。
3	トランザクションの提供	マスター・データベースおよびレプリカ・データベースの両方に、solidDB ストアード・プロシージャーとしてインプリメントされた 2 つのトランザクションを含めます。
4	同期機能の使用	マスター・データベースへの伝搬のために、サンプル・パブリケーションを登録し、レプリカ・データベースを更新し、トランザクションを保存します。
5	同期メッセージのインプリメント	データベースを同期するために、コンソールから実行する solidDB 拡張レプリケーション SQL コマンドを含めます。

スクリプトについて

この章では、2 つのデータベース間で基本的な solidDB 拡張レプリケーション操作を実行する方法を説明します。この例は、完全なアプリケーションではありません。データベース・スキーマには 1 つの表だけが含まれ、この章のトランザクションは基本的なトランザクションです。これらの目的は、拡張レプリケーション・アーキテクチャーの基本機能を示すことにあります。

この章のすべての SQL スクリプトは、solidDB パッケージのディレクトリー samples/smartflow/eval_setup にもあります。

始めに

サンプル・スクリプトの実行の開始前に、以下を行っておいてください。

- solidDB Development Kit のインストール。詳しくは、solidDB Development Kit に含まれているリリース・ノート・ファイルを参照してください。
- 以下で説明する環境セットアップの準備。

  solid/samples/smartflow/eval_setup/readme.txt。

環境のセットアップ

solidDB 拡張レプリケーション環境をセットアップするには、マスター・データベースおよびレプリカ・データベース用に solidDB を始動します。マスター・データベースおよびレプリカ・データベースの開始については、readme.txt に記載されています。

データベース表の設計

同期アプリケーションで使用されるデータベース表の一般的な要件は以下のとおりです。

- 行がグローバルにユニークになるように、ユニークな主キーが必要です。
- 更新の競合を処理する手段を提供するために、行の状況列が必要になる場合があります。
- 更新の競合を検出できるようにするために、バージョン番号、または最後に更新した時刻を含めるようにします。

これらの要件について詳しくは、63 ページの『データベース表の定義』 を参照してください。後で使用する SQL ステートメントによって、これらの基準を満たす表が作成されます。

以下は列の説明です。

- REPLICAID には、データベースのユニーク ID が含まれています。このサンプル・スクリプトでは、値 1 がレプリカ用に、値 2 がマスター用に予約済みである点に注意してください。
- ID は行が作成されたデータベース内でユニークな行 ID です。
- 状況の値は次のとおりです。-1 はマスターによって無効になった更新、1 は一時的なレプリカ・データ、2 は正式なマスター・データです。
- INTDATA および TEXTDATA は、表の「ビジネス・データ」を示します。

表に対して SYNCHISTORY プロパティーを ON に設定することで、インクリメンタル・パブリケーションが可能になります。つまり、表データの同期化の際に、変更された行だけがマスターからレプリカに転送されます。 SYNCHISTORY プロパティーを設定することで、SYNCDEMO のメイン表の「シャドー表」が作成されます。「シャドー表」の名前は、メイン表に「_SYNCHIST_」接頭部を追加したもの

です。更新され、削除された行の旧バージョンは、シャドー表に移動されます。SYNCHISTORY はマスター・データベースおよびレプリカ・データベースの*両方*でアクティブでなければなりません。

同期化する他の表と同様に、マスター・データベースとレプリカ・データベースの両方で表を定義します。

サーバーの構成とパブリケーションの作成

このセクションでは、以下を行います。

1. 1 台のサーバーをマスターとして構成します。
2. もう 1 台のサーバーをレプリカとして構成します。
3. 各サーバー上に表を作成します。
4. マスター上で、パブリケーションを作成します。
5. レプリカ上で、パブリケーションに登録します。

2 つのスクリプトで、これらの手順を実行します。スクリプトの 1 つはマスター上で実行し、もう 1 つはレプリカ上で実行します。

マスター上

マスターに接続し、master1.sql SQL スクリプトを実行します。実行方法の 1 つを以下に説明します。

solidDB のインストール・ルート・ディレクトリーに移動し、以下のコマンドを (すべてを 1 行に) 入力します。

./bin/solsql -O eval.out ″tcp 1315″ dba dba_password ./samples/smartflow/eval_setup/master1.sql

上記の要素についての説明

- *-O eval.out* は、結果の出力ファイルを定義するオプション・パラメーターです。
- *"tcp 1315"* は、マスター・サーバーのネットワーク・プロトコルとネットワーク・アドレスです。必要に応じて、コマンドのこの部分をカスタマイズしてください。
-

dba と *dba_password* は、それぞれユーザー名とパスワードです。

- *master1.sql* は実行対象の SQL スクリプトです。

eval.out 内の結果は任意のテキスト・エディターで表示できます。

注:

拡張レプリケーションのスクリプトを実行する場合には、自動コミット・モードを OFF に設定する必要があります。SQL エディター (テレタイプ) では、デフォルトで自動コミット・モードは OFF に設定されています。

```
--*****************************************
-- master1.sql
-- MASTER データベースでこのスクリプトを実行する。
-- MASTER というノード名のマスターを初期化する。
-- 表とパブリケーションも作成する。
--*****************************************

-- 「sync_demo_catalog」という名前のカタログを作成する。
-- このノードに「master_node」という名前を付ける。
-- このノードは、レプリカとしてではなく、マスターとして登録する。

CALL SYNC_SETUP_CATALOG (
   'sync_demo_catalog',
   'master_node',
   1,
   0);
COMMIT WORK;

-- カタログが現行カタログになるように設定する。
SET CATALOG sync_demo_catalog;
COMMIT WORK;

-- 同期する表を作成する。
CREATE TABLE SYNCDEMO
(
   REPLICAID INTEGER NOT NULL,
   ID INTEGER NOT NULL,
   STATUS INTEGER NOT NULL,
   INTDATA INTEGER,
   TEXTDATA CHAR(30),
   UPDATETIME TIMESTAMP,
   PRIMARY KEY (REPLICAID, ID, STATUS)
);
ALTER TABLE SYNCDEMO SET SYNCHISTORY;
COMMIT WORK;

-- SyncDemo 表のデータをすべてパブリッシュするパブリケーションを作成する。
-- CREATE PUBLICATION コマンドは二重引用符で囲む必要があることに注意する。
"CREATE PUBLICATION PUB_DEMO
   BEGIN
   RESULT SET FOR SYNCDEMO
   BEGIN
      SELECT * FROM SYNCDEMO ;
   END
END";
COMMIT WORK;
```

このパブリケーションには、syncdemo 表の行がすべて格納されます。このパブリケーションからのリフレッシュは、マスター・データベースにおいてもレプリカ・データベースにおいても、この表に SYNCHISTORY プロパティーが設定されている

ため、インクリメンタル・パブリケーションになります。
SYNC_SETUP_CATALOG ストアード・プロシージャーについて詳しくは、「*solidDB SQL ガイド*」を参照してください。

レプリカ上

solidDB のインストール・ルート・ディレクトリーに移動し、以下のコマンドを (すべてを 1 行に) 入力します。

./bin/solsql -O eval.out ″tcp 1316″ dba dba_password ./samples/smartflow/eval_setup/replica1.sql

上記の要素についての説明

- *-O eval.out* は、結果の出力ファイルを定義するオプション・パラメーターです。
- *″tcp 1316* は、マスター・サーバーのネットワーク・プロトコルとネットワーク・アドレスです。必要に応じて、コマンドのこの部分をカスタマイズしてください。
- *dba* と *dba_password* は、それぞれユーザー名とパスワードです。
- master1.sql は実行対象の SQL スクリプトです。

eval.out 内の結果は任意のテキスト・エディターで参照できます。

readme.txt ドキュメントに記載されている指示に従った場合には、replica1.sql を変更する必要はありません。そうでない場合は、スクリプトの以下の部分を必要に応じて変更してください。

- SYNC_REGISTER_REPLICA() の呼び出しで、必要に応じて、最初にマスター・データベースに接続する際のユーザー ID とパスワードを指定します。
- SYNC_REGISTER_REPLICA() の呼び出しで、必要に応じて、マスター・データベースの接続ストリングを 'tcp localhost 1315' とは違う値に設定します。

注:

solidDB 拡張レプリケーションのスクリプトを実行する場合には、自動コミット・モードを OFF に設定する必要があります。SQL エディター (テレタイプ) を使用する場合、デフォルトで自動コミット・モードは OFF に設定されています。

replica1.sql

```
--*******************************************************
-- replica1.sql
-- 「replica_node_01」というノード名のレプリカ・データベースを
```

```
-- 初期化する。
-- (レプリカごとにノード名をユニークにする必要がある。)
-- REPLICA データベースでこのスクリプトを実行する。
-- メッセージ処理のために自動コミットはオフに設定する必要があることに注意する。
--*********************************************************
-- 'sync_demo_catalog' という名前のレプリカ・カタログを作成し、
-- 'master_node' という名前のマスター・データベース・サーバーに
-- それを登録する。さらに、マスター・ノードのネットワーク・アドレスと
-- 名前を指定し、使用するユーザー ID とパスワードを指定する。
CALL SYNC_REGISTER_REPLICA (
   'replica_node_01',
   'sync_demo_catalog',
   'tcp localhost 1315', -- マスターのネットワーク・アドレス。カスタマイズすること。
   'master_node', -- マスターのノード名。
   'dba',
   'dba_password');
COMMIT WORK;

SET CATALOG sync_demo_catalog;
COMMIT WORK;

-- 表を作成する。
CREATE TABLE SYNCDEMO
(
   REPLICAID INTEGER NOT NULL,
   ID INTEGER NOT NULL,
   STATUS INTEGER NOT NULL,
   INTDATA INTEGER,
   TEXTDATA CHAR(30),
   UPDATETIME TIMESTAMP,
   PRIMARY KEY (REPLICAID, ID, STATUS)
);
-- この表で、インクリメンタル・パブリケーションを使用可能にする。
ALTER TABLE SYNCDEMO SET SYNCHISTORY;
COMMIT WORK;

-- マスターに既に定義されているパブリケーションを登録する。
CALL SYNC_REGISTER_PUBLICATION (
   'sync_demo_catalog',
   'pub_demo');
COMMIT WORK;
```

ストアード・プロシージャー SYNC_REGISTER_REPLICA および SYNC_REGISTER_PUBLICATION について詳しくは、「*solidDB SQL ガイド*」を参照してください。

トランザクションの提供

通常、同期アーキテクチャーの書き込み操作は、ストアード・プロシージャーを使用してインプリメントされます。これによって、アプリケーションのデータ保全性およびビジネス・ルールに違反せずに、発生する可能性がある競合を処理するビジネス・ロジックをトランザクションにインプリメントできます。ストアード・プロシージャーについて詳しくは、「*solidDB プログラマー・ガイド*」を参照してください。

以下のスクリプトのプロシージャー・ロジックは、できるだけ単純になるように設計されています。更新プロシージャーには、競合を処理するロジックだけがインプリメントされています。更新が失敗した場合、状況の値が -1 で行が挿入されます。状況フィールドのその他の値は、レプリカでの一時的な書き込みの場合は 1 で、マスターが受け入れた正式なデータの場合は 2 です。

proced1.sql スクリプトと proced2.sql スクリプトは、レプリカ・データベースとマスター・データベースの両方で実行します。

以下のプロシージャーは、表 syncdemo に行を挿入します。単純にするために、エラーの処理は行いません。実際のアプリケーションでは、ユニーク・キー制約違反や、発生する可能性があるその他のエラー状態を処理するロジックが必要です。

PROCED1.SQL

```
--*********************************************************
-- proced1.sql
-- サンプル表 SYNCDEMO にデータを挿入するプロシージャーを
-- 作成。
--
-- MASTER DB と REPLICA DB の両方で実行。
--
-- 注: この例には妥当性検査ルールがありません。重複は、重複挿入を
-- 無視することで処理されます。
--*********************************************************

   SET CATALOG sync_demo_catalog ;

"CREATE PROCEDURE SYNCDEMO_INSERT
   (MACHINEID INTEGER,
   ID INTEGER,
   INTDATA INTEGER,
   TEXTDATA CHAR(20),
   UPDATETIME TIMESTAMP,
   TARGETDB CHAR(1))
RETURNS
   (SUCCESS INTEGER, ROWS_AFFECTED INTEGER)

BEGIN

   DECLARE STATUS INTEGER ;
   IF TARGETDB = 'R' THEN
      STATUS := 1 ;
         ELSE IF TARGETDB = 'M' THEN
         STATUS := 2;
      ELSE
         STATUS := -1;
      END IF;
   END IF;

   EXEC SQL PREPARE SYNCDEMO_INS
   INSERT INTO SYNCDEMO
   (REPLICAID,ID,STATUS,INTDATA,TEXTDATA,UPDATETIME)
   VALUES(?,?,?,?,?,?);

   EXEC SQL EXECUTE SYNCDEMO_INS USING
   (MACHINEID,ID,STATUS,INTDATA,TEXTDATA,UPDATETIME);

   SUCCESS := SQLSUCCESS;
   ROWS_AFFECTED := SQLROWCOUNT;

   EXEC SQL CLOSE SYNCDEMO_INS;
   EXEC SQL DROP SYNCDEMO_INS;

END";
COMMIT WORK;
```

以下のプロシージャーは、 syncdemo 表の行を更新します。このプロシージャーには、競合解決ロジックの単純なインプリメンテーションが含まれています。プロシージャーは、行が存在しないことを検出した場合、代わりに状況の値が *-1* の行を

挿入します。実際のアプリケーションのロジックでは、ほとんど無数の競合およびその他のトランザクション妥当性検査エラーの処理が可能である必要があることに注意してください。

PROCED2.SQL

```
--*********************************************************
-- proced2.sql
-- サンプル表 SYNCDEMO のデータを更新するプロシージャーを
-- 作成。
--
-- MASTER DB と REPLICA DB の両方で実行。
--
-- 妥当性検査ルール: タイム・スタンプが変更されている場合、更新は
-- パラメーター「TARGETDB」が 'F' の挿入に変更されます。
-- その結果、呼び出されるプロシージャー SYNCDEMO_INSERT で、
-- 状況の値が -1 の挿入になります。
--*********************************************************
SET CATALOG sync_demo_catalog ;

"CREATE PROCEDURE SYNCDEMO_UPDATE
(
MACHINEID INTEGER,
ID INTEGER,
INTDATA INTEGER,
TEXTDATA CHAR(20),
UPDATETIME TIMESTAMP,
TARGETDB CHAR(1)
)
RETURNS
(SUCCESS INTEGER, ROWS_AFFECTED INTEGER)
BEGIN
DECLARE TMPSTR VARCHAR;
DECLARE TNOW TIMESTAMP;
DECLARE DSTAT INTEGER;
DECLARE STATUS INTEGER;
IF TARGETDB = 'R' THEN
STATUS := 1;
ELSE
STATUS := 2;
END IF;

TNOW := NOW();
TMPSTR := 'R';
DSTAT := -1;

EXEC SQL PREPARE SYNCDEMO_UPD
UPDATE SYNCDEMO SET
   STATUS = ?,
   INTDATA = ?,
   TEXTDATA = ?,
   UPDATETIME = ?
WHERE
   REPLICAID = ? AND
   ID = ? AND
   UPDATETIME = ? ;

EXEC SQL EXECUTE SYNCDEMO_UPD USING
   (STATUS, INTDATA, TEXTDATA, TNOW, MACHINEID, ID, UPDATETIME);

SUCCESS := SQLSUCCESS ;
ROWS_AFFECTED := SQLROWCOUNT;

IF (SUCCESS = 1) AND (ROWS_AFFECTED = 0) THEN
   TMPSTR := 'F' ;
```

```
   EXEC SQL PREPARE SYNC_UPD1 CALL
      SYNCDEMO_INSERT(?,?,?,?,?,?);
   EXEC SQL EXECUTE SYNC_UPD1 USING
      (MACHINEID, ID, INTDATA, TEXTDATA, TNOW, TARGETDB);
   EXEC SQL FETCH SYNC_UPD1 ;

   SUCCESS := SQLSUCCESS;
   ROWS_AFFECTED := SQLROWCOUNT;
END IF ;

END";
COMMIT WORK;
```

注:

このウォークスルーをできるだけ単純にするため、上記のプロシージャーでは、シーケンス値を使用する代わりに、列 ID のパラメーター値をとっています。また、データベース ID にもパラメーター値が提供されています。実際のアプリケーションでは、列 ID にシーケンス値を含める必要があります。シーケンスの使用について詳しくは、「*solidDB プログラマー・ガイド*」を参照してください。

solidDB 拡張レプリケーション機能の使用

基本的な solidDB 拡張レプリケーション機能を示す以下のスクリプトを実行することができます。これには以下が含まれます。

- パブリケーションをレプリカに登録し、パブリケーションのリフレッシュを要求すると、レプリカがそれを受信できるようにします。パブリケーションを登録することで、パブリケーション・パラメーターを検証することができます。これによって、ユーザーが間違って不要なリフレッシュを要求したり、アドホック・リフレッシュを要求するのを防止します。
- レプリカをトランザクションで更新します。このトランザクションは、後でマスター・データベースに伝搬するために、レプリカに保存されます。

同期を行うには以下のようにします。

1. 以下の SQL ステートメントをレプリカとマスターの両方のデータベースで実行します。

   select.sql

   ```
   --*********************************************************
   -- select.sql
   -- サンプル表の状況を確認するための SQL ステートメント
   --*********************************************************
   SET CATALOG sync_demo_catalog ;
   COMMIT WORK ;
   SELECT * FROM SYNCDEMO ;
   ```

   表に行を挿入していないので、行がまったく返されていないことが分かります。

2.

レプリカで以下のステートメントを実行して、パブリケーションを登録し、2 行をレプリカに挿入します。

REPLICA2.SQL

```
--*********************************************************
-- replica2.sql
-- このスクリプトはパブリケーション PUB_DEMO に登録し、2 行を
-- REPLICA データベースに挿入し、
-- MASTER に伝搬するトランザクションを保存する。
--
-- REPLICA データベースで実行する。
--*********************************************************
-- パブリケーションに登録する。
SET CATALOG sync_demo_catalog ;
MESSAGE REG_PUBL BEGIN;
MESSAGE REG_PUBL APPEND REGISTER PUBLICATION
PUB_DEMO;
MESSAGE REG_PUBL END;
COMMIT WORK;
MESSAGE REG_PUBL FORWARD TIMEOUT FOREVER;
COMMIT WORK;

CALL SYNCDEMO_INSERT (1,1,100,'First row','1998-05-15 12:00:00','R');
SAVE CALL SYNCDEMO_INSERT (1,1,100,'First row','1998-05-15 12:00:00','M');
CALL SYNCDEMO_INSERT (1,2,101,'Second row','1998-05-15 12:00:01','R');
SAVE CALL SYNCDEMO_INSERT (1,2,101,'Second row','1998-05-15 12:00:01','M');

COMMIT WORK;
```

3.

select.sql を使用してレプリカに 2 行が含まれており、マスターには行が含まれていないことを検証します。

この時点では、レプリカ・データベースには、マスター・データベースへの伝搬を待機している 1 つのトランザクションに、保存された 2 つのステートメントもあります。

同期メッセージのインプリメント

同期メッセージは、レプリカ・データベースで実行される SQL ステートメントで作成されます。

1.

レプリカで replica3.sql を実行して、マスター・データベースとレプリカ・データベースを同期します。

replica3.sql は、my_msg というメッセージを作成しますが、このメッセージの処理内容は以下の 2 つです。

a.

すべてのローカル・トランザクションをマスターに伝搬する。

b.

pub_demo パブリケーションからリフレッシュを実行して、更新データを取得する。

スクリプトはメッセージを作成すると、そのメッセージをマスター・データベースに送信し、応答が戻るまで待機します。

REPLICA3.SQL

```
--*********************************************************
-- replica3.sql
-- my_msg という名前のメッセージを新規作成する。
-- メッセージ my_msg に以下のタスクを追加する。
--              トランザクションの伝搬
--              パブリケーションからリフレッシュ
--
-- REPLICA データベースで実行する。
--
-- 注: 自動コミットは必ずオフにすること。
--*********************************************************
SET CATALOG replica_catalog ;
MESSAGE my_msg BEGIN ;
MESSAGE my_msg APPEND PROPAGATE TRANSACTIONS ;
MESSAGE my_msg APPEND REFRESH PUB_DEMO ;
MESSAGE my_msg END ;
COMMIT WORK ;

-- メッセージをマスターに送信し、
応答が戻るまで待機しない。
MESSAGE my_msg FORWARD;
COMMIT WORK;
-- メッセージに対する応答を個別にマスターに
要求する。
MESSAGE my_msg GET REPLY TIMEOUT DEFAULT ;
COMMIT WORK;
```

select.sql を使用して、レプリカとマスターの行が両方とも 2 行になっていることを確認します。

2.

以下のステートメントをマスターで実行して、マスターから行を 1 行削除します。

MASTER2.SQL

```
--*************************************************
-- master2.sql
-- サンプル表から行を 1 行削除する。
-- MASTER データベースで実行する。
--*************************************************
SET CATALOG sync_demo_catalog ;
DELETE FROM SYNCDEMO WHERE ID = 2;
   COMMIT WORK ;
```

select.sql を使用して、レプリカの行は 2 行で、マスターの行は 1 行になっていることを確認します。

3.

レプリカで replica3.sql のステートメントをすべて実行して、同期を行います。

select.sql を使用して、レプリカとマスターの行が両方とも 1 行になっていることを確認します。

4.

レプリカで replica4.sql のステートメントをすべて実行して、レプリカに行を 2 行挿入します。

REPLICA4.SQL

```
--*******************************************************
-- replica4.sql
-- このスクリプトは、REPLICA データベースに行を 2 行挿入し、
-- MASTER に伝搬するトランザクションを保存する。
--
-- REPLICA データベースで実行する。
--
*******************************************************
SET CATALOG sync_demo_catalog; CALL SYNCDEMO_INSERT
(1,3,102,'Third row','1998-05-15 12:10:00','R');
SAVE CALL SYNCDEMO_INSERT (1,3,102,'Third
row','1998-05-15 12:10:00','M');
CALL SYNCDEMO_INSERT (1,4,103,'Fourth
row','1998-05-15 12:10:01','R');
SAVE CALL SYNCDEMO_INSERT (1,4,103,'Fourth
row','1998-05-15 12:10:01','M');
COMMIT WORK;
```

5.

レプリカで replica3.sql のステートメントをすべて実行して、同期を行います。

select.sql を使用して、レプリカとマスターの行が両方とも 3 行になっていることを確認します。

6.

レプリカで replica5.sql を実行して、レプリカの行を 1 行更新します。

REPLICA5.SQL

```
--*******************************************************
-- replica5.sql
-- このスクリプトは、REPLICA データベースの行を 1 行更新し、
-- MASTER に伝搬する行を保存する。
--
-- REPLICA データベースで実行する。
--*******************************************************
SET CATALOG sync_demo_catalog ;

CALL SYNCDEMO_UPDATE (1,1,201,'Row 1
changed','1998-05-15 12:00:00','R');
SAVE CALL SYNCDEMO_UPDATE (1,1,201,'Row 1
changed','1998-05-15 12:00:00','M');
COMMIT WORK;
```

7.

レプリカで replica3.sql のステートメントをすべて実行して、同期を行います。

select.sql を使用して、レプリカ・データベースとマスター・データベースの行が両方とも 3 行になっていることと、更新内容がマスター・データベースに伝搬していることを確認します。

8.

マスターで master3.sql を実行して、マスターの行を 1 行更新します。

MASTER3.SQL

```
--*********************************************************
-- master3.sql
-- このスクリプトは、MASTER データベースの行を 1 行更新する。
--
-- MASTER データベースで実行する。
--*********************************************************
SET CATALOG sync_demo_catalog ;

CALL SYNCDEMO_UPDATE (1,3,203,'Row 3
masterchange','1998-05-15
12:10:00','M');
COMMIT WORK ;
```

レプリカにデータを同期しないでください。

select.sql を使用して、レプリカとマスターの行が両方とも 3 行になっていることと、最新の更新がマスター・データベースのみで発生していることを確認します。

9.

レプリカで replica6.sql を実行して、レプリカの同じ行を更新します。

REPLICA6.SQL

```
--*********************************************************
-- replica6.sql
-- このスクリプトは、REPLICA データベースの行を 1 行更新し、
-- MASTER に伝搬する行を保存する。
--
-- REPLICA データベースで実行する。
--*********************************************************
SET CATALOG sync_demo_catalog ;
CALL SYNCDEMO_UPDATE (1,3,203,'Row 3
replicachange','1998-05-15 12:10:00','R');
SAVE CALL SYNCDEMO_UPDATE (1,3,203,'Row 3
replicachange','1998-05-15 12:10:00','M');
COMMIT WORK;
```

select.sql を使用して、マスターとレプリカの更新内容が違っていることを確認します。

10.

レプリカで replica3.sql のステートメントをすべて実行して、同期を行います。

select.sql を使用して、同じ行が更新されたため発生していた競合が、正しく処理されたことを確認します。

マスターとレプリカは両方とも 4 行になるはずです。このうち 1 行は、無効な状況 (*-1*) で、その理由はレプリカ側の最後の更新操作によって、マスター・データベースで競合が発生するためです。

同期リフレッシュ

非同期データ・リフレッシュは大量のメモリーを消費し、ディスク I/O のオーバーヘッドをもたらす可能性があります。これを回避するため、同期式のメッセージレス拡張レプリケーション・インターフェースを使用することができます。 このモードでは、関連するデータがデータ・ストリームとして送信されるので、メモリーが

節約されます。またこのモードでは、ディスクへのメッセージの書き込みがないので、必要なディスク I/O 帯域幅が減少します。

同期リフレッシュとその使用法について詳しくは、84 ページの『同期リフレッシュの使用』、および「*solidDB SQL ガイド*」の付録 *B* の『*solidDB SQL 構文*』の REFRESH コマンドの説明を参照してください。

プル同期通知

ここまで、データの同期について「プル」モデルを中心に説明してきました。このモデルでは、更新データのコピーをマスターから「プル」するタイミングをレプリカが決定します。ただし、多くの状況で、「プッシュ」モデルの方が望ましいことがあります。このモデルでは、レプリカにデータを送信するタイミングをマスターが決定します。例えば、「プッシュ」モデルでは、マスターは、レプリカからの更新データのコピーの要求を待つことなく、データが更新されるとすぐにそのデータを送信できます。

「プル」モデルと「プッシュ」モデルの違いは、「ポーリング」と、「シグナル」または「割り込み」への応答の違いに似ています。「プル」モデルでは、レプリカは、マスターに新しいデータがいつ到着したかを認識しません。したがって、レプリカは通常、定期的、ネットワークに再接続された時点、またはレプリカの同期管理者に手動で指示された時点のいずれかでマスターに「ポーリング」を行います。一方、「プッシュ」モデルでは、マスターは新しいデータまたは更新データの到着を認識し、そのデータをレプリカに送信できます。

「プル」モデルの利点は、マスターに対してある程度ランダムに要求を配布することが多いことです。これにより、マスターは、一度に多数の更新要求が発生して過負荷になるということはありません。また、「プル」モデルは、レプリカが不定期にしかネットワークに接続されない場合にも十分に機能します (例えば、データを PDA (携帯情報端末) に保持し、たまにしか PDA を同期しない場合)。「プッシュ」モデルの利点は、当然のことながら、最新でないデータが迅速に更新されることであり、また新しいデータが到着していないときに、レプリカが「ポーリング」で時間を無駄にしないで済むことです。

solidDB は真の「プッシュ」機能を提供していませんが、solidDB は、プッシュ同期に類似しているが、さらに柔軟なプル同期通知機能を提供しています。プル同期通知方式では、マスターは、更新データが存在することをレプリカに通知し、次にレプリカは、更新データをダウンロードするかどうか選択できます。

プル同期通知に基づいてシステムをインプリメントするには、システムが以下の要件を満足する必要があります。

- 更新データは、コミットされるまでレプリカに「プッシュ」してはいけません。基本的には、データは、トランザクションの途中ではなく終了時にプッシュされます。
- システムは、マスター内でデータが変更され、レプリカへのプッシュが必要になった時点を検出できなければなりません。

solidDB 拡張レプリケーションでは、プル同期通知は、1 つの機能としてはインプリメントされていません。その代わり、プル同期通知は、「START AFTER COMMIT」、「リモート・ストアード・プロシージャー」、および「レプリカ・プロパティー名」の組み合わせでインプリメントされています。以下に、これらの機能を簡単に説明します。詳細な説明が、別途提供されています (レプリカ・プロパティー名については本書で後述し、リモート・ストアード・プロシージャーと START AFTER COMMIT については、「*solidDB SQL ガイド*)」に説明されています)。

リモート・ストアード・プロシージャー機能は、その名の示すとおり、別のデータベース内のストアード・プロシージャーを呼び出す方法です。具体的には、レプリカがストアード・プロシージャーを作成し、マスターがそのストアード・プロシージャーを呼び出します。例えば、レプリカが REFRESH コマンドを含むストアード・プロシージャーを作成した場合、マスターがそのストアード・プロシージャーを呼び出すことができ、それによってレプリカがリフレッシュを要求します。

ユーザーは、「START AFTER COMMIT」機能を使用して、トランザクションがコミットされたときに実行するアクションを指定できます。例えば、特定のトランザクションがマスター上の情報を更新する場合、START AFTER COMMIT でマスターに対して、更新が正常にコミットされた後で、レプリカへのリモート・プロシージャー・コールを実行するよう指示できます。

START AFTER COMMIT とリモート・ストアード・プロシージャーの組み合わせにより、プル同期通知をインプリメントできます。マスターは、いずれかのデータを更新する場合、各レプリカ上のリモート・ストアード・プロシージャーを呼び出す START AFTER COMMIT を作成して、サブスクライバーに通知することができます。 ストアード・プロシージャーは更新データを取得するためにリフレッシュします。

次の問題は、どのレプリカに通知するか判断することです。言い換えれば、データの更新時に、どのレプリカ上のストアード・プロシージャーを呼び出すか決定することです。これを制御するために、以下に説明する「レプリカ・プロパティー名」機能を使用します。 (また、ノード名を使用して、単一のレプリカを参照することもできます。)

注:

START AFTER COMMIT 機能とリモート・ストアード・プロシージャー機能については、「*solidDB SQL ガイド*」に詳しく説明されています。プル同期通知の説明を引き続き参照する前に、これらの機能の詳細を参照することを推奨します。

レプリカ・プロパティー名

プロパティー名を使用して、レプリカにラベルを付けることができます。ラベルを付けられたレプリカは、グループ化できます。これは、START AFTER COMMIT 機能を使用するときに重要になります (START AFTER COMMIT ... を参照してください)。例えば、自転車業界に関連したレプリカと、サーフボード業界に関連したレプリカがあり、これらのレプリカのグループのそれぞれを別々に更新したいとし

ます。プロパティー名を使用して、これらのレプリカをグループ化できます。グループのすべてのメンバーが、同じプロパティーを持ち、そのプロパティーに対して同じ値を持ちます。

プロパティーおよび値は、ほとんどどんな名前およびリテラルにでもすることができます。例を以下に示します。

表 4. レプリカ・プロパティーと値

プロパティー	**可能な値の例**
Region	north、south、east、west
Color	red、yellow、green、blue、rainbow、transparent
Mood	upbeat、blue
Philosophy	realist、nihilist、purple

プロパティーと値はサーバーにとって意味はなく、単に任意のラベルなので、「意味のある」値にする必要はありません。「Philosophy」カテゴリーで「purple」が実用的な値であると判断した場合は、その判断に従って使用します。さらに、異なるプロパティーでオーバーラップする値を使用できます。「blue」が Color と Mood の両方で実用的であると判断した場合は、両方のプロパティーの値として使用できます。(Mood が blue であるものとしてカテゴリー化されたサーバーは、Color が blue であるサーバーと同じカテゴリーになりません。プロパティーは完全に独立しています。)

サーバーは複数のプロパティーを持つことができるため、複数のグループに属することができます。例えば、レプリカは以下のプロパティーを持つことができます。

Region = west

Color = green

サーバーが、すべての可能なプロパティーの設定を持っている必要はありません。このレプリカは、Philosophy プロパティーも Mood プロパティーもまったく持たないことがあります。

プロパティーの使用例については、START AFTER COMMIT コマンドのセクションを参照してください。

レプリカ・プロパティー名は、いつでも追加できます。また、新しいレプリカもいつでも追加できます。プロパティー名を使用してレプリカをグループ化することによって、レプリカのリモート・ストアード・プロシージャーを呼び出すようにマスターのロジックを書き直す必要がなくなります。例えば、特定のデータが更新されたときに、Region が「north」のすべてのオフィスに通知したいとします。新しいオフィス (レプリカ) を追加する場合、その新しいオフィスにプロパティー「Region=north」を指定するだけで、関係のあるデータが更新されたときに、新しいオフィスは自動的に通知を受けるようになります。新しいレプリカにプロパティー「Region=north」が設定されたことをマスターに通知する必要があることを除いて、

マスターで必要な変更はありません。もちろん、新しいレプリカには、マスターが呼び出す予定であるストアード・プロシージャーのコピーが必要です。

すべてのレプリカが、「name」という暗黙的プロパティーを持っています。これは、レプリカが登録されたときにレプリカに割り当てられるノード名です。

レプリカのプロパティーは、マスターがレプリカのプロパティーを認識している場合にのみ、プル同期通知で有効です。マスターとレプリカのどちらも、レプリカのプロパティーを設定できます。レプリカがそれ自身のプロパティーを設定した場合は、そのプロパティーと値をマスターに通知する必要があります。レプリカは、SAVE を使用して、メッセージでマスターにプロパティーを送信できます。マスターは、SET コマンドを使用して、指定したレプリカのプロパティーを設定できます。

マスターでの構文は、以下のとおりです。

```
SET SYNC PROPERTY <propertyname> = { 'value' | NONE } FOR REPLICA
<replicaname>
```

レプリカでの構文は、以下のとおりです。

```
SAVE SET SYNC PROPERTY <propertyname> = { 'value' | NONE; }
```

例

マスター

```
SET SYNC PROPERTY color = 'red' FOR REPLICA replica_node_01;
SET SYNC PROPERTY color = NONE FOR REPLICA replica_node_01;
```

レプリカ

```
SAVE SET SYNC PROPERTY color = 'red';
SAVE SET SYNC PROPERTY color = NONE;
```

プル同期通知の概要

プル同期通知について詳しく説明する前に、プル同期通知とその他のデータ複製方式を比較します。

「プッシュ同期」、「プル同期」、「プル同期通知」の対比

データを複製するとき、サーバーは以下を使用できます。

- 「プル」アプローチ、または
- 「プッシュ」アプローチ、または
- 「プル同期通知」などのハイブリッド

solidDB 拡張レプリケーションは、「プル」アプローチと「プル通知」アプローチをサポートします。

「プル」アプローチでは、レプリカがデータを要求します。これは、solidDB 拡張レプリケーションの「リフレッシュ」操作に対応します。レプリカは、リフレッシュするときに、マスター・サーバーが特定のパブリケーションのすべてのデータ (または、最後のリフレッシュ操作以降に変更されたデータ) を送信するように要求します。

「プッシュ」アプローチでは、マスターが選択した時間に、マスターがレプリカにデータを送信します (これは通常、マスターでデータが更新された直後に行われますが、必ずしもそうであるとは限りません)。マスターがデータをプッシュするとき、レプリカはデータを受け入れる必要があります。solidDB サーバーは、真の「プッシュ」方式を使用しません。

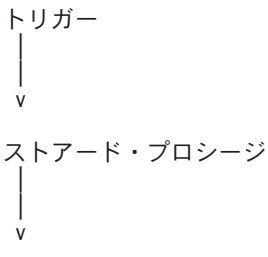
solidDB は、「プル同期通知」というハイブリッド・アプローチを使用します。このアプローチでは、新しいデータが使用可能になったことがレプリカに通知されます。そうするとレプリカは、REFRESH コマンドを実行して新しいデータを「プル」できます。REFRESH はオプションです。新しいデータがあると通知された後、レプリカは、すぐにリフレッシュするか、時間をおいた後でリフレッシュするか、通知を無視して全くリフレッシュしないかを選択できます。詳しくは、次のセクションを参照してください。

プル同期通知のインプリメント

solidDB サーバーは、プル同期通知を 2 つのステップのプロセスとしてインプリメントします。まず、マスター・サーバーがレプリカ・サーバーに通知し、次に、レプリカ・サーバーが REFRESH を実行します。

ほとんどの場合、マスター・サーバーは、リモート・ストアード・プロシージャーを呼び出してレプリカへ通知します。レプリカは、このストアード・プロシージャーを作成済みである必要があります。通常、このプロシージャー自体に、適切な REFRESH コマンドを含めます。(より間接的な方法を使用することもできますが、この方法が最も短く、簡潔な方法です。)

どのようにしてマスターはリモート・ストアード・プロシージャーを呼び出すでしょうか。その方法の 1 つは、トリガーを作成することです。例えば、「employees」という名前の表がマスターで更新されるたびに、replica_node_01 がその表のデータをリフレッシュするとします。マスター側の employees 表に、INSERT、UPDATE、および DELETE のトリガーを作成することが考えられます。それぞれのトリガーは、データのリフレッシュを要求する replica_node_01 のリモート・ストアード・プロシージャーを呼び出します。



レプリカに通知するプロセスは単純ですが、通知対象のレプリカを自動的に決定する方法はありません。マスターでトリガーを作成または更新するユーザーは、通知対象の各レプリカの呼び出しストアード・プロシージャーの名前を知っている必要があります。

同様に、レプリカには適切なパブリケーションを指定する REFRESH ステートメントが必要です。どのレプリカがどのパブリケーションにサブスクライブするのかをマスター・サーバーで識別して、そのレプリカのプロシージャーを呼び出すトリガーを自動的に作成する自動化された方法はありません。(これでも十分ではなく、マスター・サーバーも該当するストアード・プロシージャーを作成し、またレプリカにそのストアード・プロシージャーの作成を強制する必要があるからです。)

マスターでデータが更新されたことをレプリカに通知する手段として、トリガー、START AFTER COMMIT、あるいはその他の方式のうちどれを使用するにせよ、通知対象はあくまでも指定したノードに限定されることに注意してください。

注:

プル同期通知の開始にトリガーを使用すること、または、頻繁な同期の実行に他のメカニズムを使用することに決定する場合は、可能性のあるパフォーマンス上の結果を常に意識するようにしてください。同期メッセージングは、データをディスクに書き込んでからネットワーク経由で送信するストア・アンド・フォワード方式に基づいているため、同期メッセージングに関係したオーバーヘッドは常に多少生じます。メッセージングの相対オーバーヘッドが大きくなると、それだけ同期されるデータの量が減ります。例えば、solidDB サーバーには、1 秒間に数百または数千の更新操作を実行するだけの能力がありますが、同期メッセージについては 1 秒間に数十件程度しか処理できない場合があります。更新が発生するごとに同期を開始するために、更新トリガーを更新頻度の高い表に書き込んだ場合、同期メッセージングのパフォーマンスにより、更新のパフォーマンスは制限されます。 1 秒間に 1000 件の更新パフォーマンスを達成するのではなく、1 秒間に実行可能な更新が 10 件のみになる場合があります。どの更新でも、更新データを 1 行しか含まない同期メッセージが 1 つ生じるからです。この問題に対する対処法は、例えば同期の頻度を数秒間に 1 回のみにすることです。こうすることで、それぞれの同期メッセージに含まれる行数は 1 行よりも多くなるため、同期メッセージのオーバーヘッドは、最悪のケースの何分の 1 かになります。

リモート・ストアード・プロシージャーの DEFAULT キーワード

リモート・ストアード・プロシージャーの DEFAULT キーワードを理解するには、リモート・ストアード・プロシージャーと START AFTER COMMIT コマンドの両方について理解する必要があります。

START AFTER COMMIT コマンドには、オプションで以下の節を指定することができます。

```
FOR EACH REPLICA [WHERE ... ];
```

例えば、以下のように指定します。

```
START AFTER COMMIT FOR EACH REPLICA WHERE region = 'west'
UNIQUE CALL my_proc;
```

コマンドをすべてのレプリカに適用する場合には、以下のように指定します。

```
START AFTER COMMIT FOR EACH REPLICA UNIQUE CALL my_proc;
```

このステートメントを実行するサーバーは、WHERE 節に一致するすべてのレプリカのリストを効果的に作成し、リストのレプリカごとに指定プロシージャー (my_proc) を呼び出します。そのストアード・プロシージャーの実行中に、DEFAULT 節が現在処理中のレプリカを識別します。例えば、WHERE 節に一致する 3 つのレプリカ、すなわち、region = 'west' である以下の 3 つのレプリカがあるとします。

California

Oregon

Washington

すると、my_proc プロシージャーが 3 回、呼び出されます。DEFAULT は、1 回目の呼び出しのときには「California」、2 回目の呼び出しのときには「Oregon」、3 回目の呼び出しのときには「Washington」です。my_proc という名前のストアード・プロシージャーから 3 つのレプリカのそれぞれに対してリモート・ストアード・プロシージャーを呼び出すには、以下の構文を使用します。

```
CALL remote_proc_name AT DEFAULT;
```

my_proc が呼び出されるごとに、DEFAULT は 3 つのレプリカのいずれかの名前に設定されるので、以下のそれぞれの呼び出しを行うことになります。

```
CALL remote_proc_name AT California;
CALL remote_proc_name AT Oregon;
CALL remote_proc_name AT Washington;
```

マスターからリフレッシュするコマンドが「remote_proc_name」に含まれる場合、更新データがあり、リフレッシュする必要があることを 3 つのレプリカすべてに対し通知することになります。以下の組み合わせを使用することで、データをリフレッシュする必要があることをすべてのレプリカに通知できるようになります。

START AFTER COMMIT

および

リモート・プロシージャー・コール

このような保守を簡単にする方法の 1 つとして、前述の「プロパティー」機能を使用する方法があります。例えば、California、Oregon、および Washington という名前の 3 つのレプリカ・サーバーがあり、そのすべてが特定のパブリケーションからリフレッシュされるとします。ここで Arizona という名前のレプリカを新しく追加し、同じパブリケーションからリフレッシュし、したがって California、Oregon、および Washington サーバーと同じ状況で通知を受けるようにするとします。この場合、新しいサーバーのプロパティーを他の 3 つのサーバーのプロパティー (これは START AFTER COMMIT の WHERE 節で使用されているプロパティーです) と一致するように設定するだけで済みます。例えば、以下のように指定します。

```
SET SYNC PROPERTY region = 'west'; -- レプリカ上
SET SYNC PROPERTY region = 'west' FOR REPLICA arizona; -- マスター上
```

プル同期通知を使用するタイミング

プル同期通知機能は、マスターの情報更新とレプリカの情報更新の間の遅延を少なくします。ただし、ネットワーク・トラフィックが増加する場合があります。

プル同期通知によって、ネットワークの負荷が増える場合があります。現在、1 時間ごとに各レプリカを同期するという方法を使用しており、通常、1 時間に複数回の更新がある場合は、1 つのレプリカに対して 1 時間に 1 セットのネットワーク・メッセージのみを使用して同期を行います。しかし、プル同期通知機能の使用に切り替えると、1 つのレプリカに対して 1 時間に、更新と同数のセットのネットワーク・メッセージを使用します。

当然、逆の場合もあります。現在、リフレッシュは頻繁に行うが、更新はそれほど頻繁に行わない場合は、プル同期通知によってネットワーク・トラフィックが実際に減少します。これは、データが実際に必要な場合にのみリフレッシュするためです。データが変更されたかどうかを確認するために、頻繁に「ポーリング」する必要はありません。更新があまり頻繁に行われないが、更新があったときは、すぐにそのことを知る必要がある場合は、プル同期通知は非常に優れたソリューションです。

「プル」と「プッシュ」（プル同期通知）による方法は、相互に排他的なものではないことに注意してください。この 2 つを組み合わせて使用することができます。例えば、毎日特定の時間に、マスターがレプリカに対して、リフレッシュするタイミングであることを通知するようにシステムを設計することができます。ただし、レプリカ・データベースを持って現場に出る修理担当者が、オフィスを出る直前に「REFRESH」コマンドを発行することで、最新のデータを確実に取得することもできます。

システムをプル同期通知用に設計する際、マスター内のデータの変更方法が以下の 3 つしかないので、更新データを対象のレプリカに「プッシュ」する必要が生じる可能性がある場合が、3 つしかないことを知っておくと役立ちます。

1. データをマスターで直接変更できます。すなわち、クライアントはマスター上の表でレコードを挿入、更新、または削除できます。
2. マスターはレプリカからデータを受信できます。
3. マスター・サーバーがマスターとレプリカの両方を兼ねている場合 (例えば、3 つ以上のレベルがある階層の「中間」層の場合)、サーバーはそのマスターにリフレッシュを要求し、そこからデータを取得することができます。

REFRESH またはプル同期通知のスケジューリング

場合によっては、REFRESH やプル同期通知などの操作を定期的に (30 秒ごとなど) 実行したいことがあります。そのためには、ストアード・プロシージャーで SLEEP() コマンドを使用します。以下に、SLEEP() コマンドを使用して定期的にタスクを実行する 2 つの例を示します。これらの例はストアード・プロシージャーとしてインプリメントされ、このストアード・プロシージャーはおそらく、START

AFTER COMMIT コマンドの本体から呼び出されることによってバックグラウンドで実行されることに注意してください。SLEEP() 関数に引き渡されるパラメーターは、所望の期間であることに注意してください (ミリ秒単位)。

以下に SLEEP を使用する単純な例を示します。

```
CREATE PROCEDURE SIMPLE_CLOCK
RETURNS (T TIMESTAMP)
BEGIN
   -- 「永久に」ループ。
   WHILE 1 LOOP
      T := NOW();
      RETURN ROW;
      EXEC SQL COMMIT WORK;
      SLEEP(1000);
   END LOOP
END
```

以下はプロシージャー・コードで sleep() を使用して、REFRESH コマンドをスケジュールする例です。

```
CREATE PROCEDURE REFRESH_SCHEDULER
BEGIN
   DECLARE I INTEGER;
   I := 0;
   WHILE I = 0 LOOP
      EXEC SQL COMMIT WORK;
      SLEEP(10000); -- ここで、プロシージャーは 10 秒 (10000ms) スリープ。
      EXEC SQL EXECDIRECT call refresh_now;
   END LOOP
END
```

このアプローチを拡張して、プル同期通知に適用できます。

SLEEP 関数は、ストアード・プロシージャーから呼び出すことができます。期間はミリ秒で測定されます。期間は概算であることに注意してください。ご使用のプラットフォームの、クロックとタイマーのレゾリューションが、ミリ秒の精度をサポートしていない場合があります。また、正確なタイミングは、コンピューターのビジー状況に部分的に依存します。さらに、START AFTER COMMIT ステートメントで実行される SQL ステートメントまたはプロシージャー呼び出しは、バックグラウンドで非同期に実行され、その実行タイミングはあまり正確ではありません。最後に、スリープ以外のアクティビティーの期間もタイミングに影響を与えます。例えば、10 秒間継続する SLEEP() と、実行に 2 秒かかる SQL ステートメントがループに含まれている場合、ループは、実際には 10 秒ごとではなく約 12 秒ごとに実行されます。

4 拡張レプリケーション・アプリケーションの計画および設計

この章では、solidDB 拡張レプリケーションのデータ同期テクノロジーを使用するアプリケーションのインストールおよびインプリメントの前に検討する必要がある設計と計画の問題について説明します。 29 ページの『3 章 データ同期の概要』では、拡張レプリケーション機能の簡単な概要を示します。これで、solidDB 拡張レプリケーション・アプリケーションの開発計画と、それを自分の独自のビジネス・ニーズに合わせるカスタマイズを開始できます。この章では、それを行うためにマルチデータベース・システムを計画し、設計する方法を示します。設計の問題が当てはまるアプリケーションとデータベースのさまざまな領域を指摘します。

拡張レプリケーション・インストールの計画

solidDB 拡張レプリケーションを使用する分散データベース・システムをインストールする前に、アプリケーションの同期のニーズを判別、分析、評価する必要があります。これらのニーズは、システムのリソース要件およびアプリケーション要件に影響を与えます。さらに、データの分散、データ伝搬の開始、同期のスケジュール、インフラストラクチャーの作成、コンピューターおよびネットワークのリソースの割り振りを行うことをどのように決定するかについて、パフォーマンスの考慮が影響を与えることがあります。

データの分散

レプリカで必要なローカル・データの量や内容は、同期化処理のリソース要件に影響を及ぼします。スケーラビリティーを高めるため、データをさまざまなレプリカ・データベースにパーティション化し、各レプリカ・データベースにはマスター・データのサブセットだけが含まれるようにしてください。一般に、データのパーティション化が適切であるほど、システム全体でのスケーラビリティーが高くなります。システムの論理データ・モデルおよび物理データ・モデルを設計する場合は、パフォーマンスの要求を必ず考慮してください。

同期化処理の調整

分散拡張レプリケーション・システムでは、システムのオフピーク時間を利用することができます。solidDB 拡張レプリケーション・アーキテクチャーでは、同期化処理を完全に調整することができます。使用可能なインフラストラクチャーの容量を必ず考慮してください。例えば、使用可能な帯域幅が最適な場合は、ネットワーク経由で行う大量のデータの転送を調整できます。一方、ラッシュ時には、優先度の高いトランザクションの伝搬など、最も緊急を要する同期化タスクのみの転送を許可することができます。

データの同期化にはある程度のオーバーヘッドが伴うため、多くの場合、データベースの全体的なパフォーマンスとデータの適時性との間の妥協が必要です。データの適時性の要求が高いほど (すなわち、同期メッセージが小さいほど)、同期によるオーバーヘッドが増加し、したがってシステム全体のスケーラビリティーが低くなります。

パフォーマンスとスケーラビリティーの評価

パフォーマンスとスケーラビリティーを計画する際、I/O 処理、フォールト・トレランス、および同期メッセージの転送のために十分な容量をインフラストラクチャーで提供する必要があります。このセクションでは、キャパシティー・プランニングのために考慮する必要のある、パフォーマンスやスケーラビリティーに影響を及ぼす各コンポーネントについて説明します。

マスター・データベース

マスター・データベースは、システムの重要なコンポーネントです。システムで作成されたすべての同期トランザクションは、最終的にマスター・データベースにコミットされます。同様に、パブリケーション・データはマスター・データベースからリフレッシュされます。システムのインフラストラクチャーという観点からすると、これには以下の 2 つの意味があります。

- レプリカのトランザクションとリフレッシュの処理にかかる CPU およびディスク I/O のロードを管理するのに十分な、マスター・サーバーの容量が必要です。同期アーキテクチャーがストア・アンド・フォワード・メッセージングであるため、多少、ディスク I/O が増えます。ストア・アンド・フォワード・メッセージングが原因でディスク I/O が増加すると問題になる場合には、42 ページの『同期リフレッシュ』に説明されている同期リフレッシュ方式を使用することを検討してみてください。また、「*solidDB SQL ガイド*」の*付録 B の『solidDB SQL 構文*』に説明されている REFRESH コマンドの説明を参照してください。
- マスター・サーバーのフォールト・トレランスが、十分なレベルに達している必要があります。レプリカ・データベースはマスターのみを介して互いに連絡するため、マスター・サーバーは単一障害点になります。マスター・サーバーが停止すると、レプリカ間の同期が停止します。(solidDB の高可用性コンポーネントを使用して、マスター・データベース・サーバーをミラーリングする方法を検討する必要があります。)

マスター・データベースの負荷の最適化

標準のシステムでは、データベースにかかる負荷の大半は、データベースをオンラインで読み取り中心で使用することによる読み取り I/O 負荷です。マルチデータベース・システムの場合、この負荷は、多数のデータベースに分散可能です。したがって、マスター・データベースには、システムのレプリカ・データベースから伝搬されてきたトランザクションの処理用に容量が残されます。

システムの「共用」トランザクションまたは同期トランザクションはすべてマスター・データベースにコミットされるため、マスター・データベースのリソースをできるだけ効果的に使用することが重要です。以下の処置を行うことで、マスター・データベースのリソースの使用を最適化することが可能です。

- 可能であれば、マスター・データベースを同期使用のみの専用にします。大量の同期化処理が同時に実行される場合は、このデータベースにオンラインでアクセスするときの応答時間は、予測不能になる可能性があります。

- 同期使用のみのために、データベースの索引付けを最適化します。例えば、データベースをオンラインで使用しない場合は、検索基準とパブリケーションの結合で使用される索引、およびトランザクションに必要な索引のみにするようにします。
- オンラインで集中的にデータベースを使用する必要があるときは、その用途に合うように、マスター・データベースの完全なレプリカを用意することが多くの場合に望ましいことです。このようなデータベースでは、オンラインの使用に合わせて最適化された索引付けをすることができます。
- パブリケーションを単純にします。表間でいくつも結合してパブリケーションが複雑になると、照会も複雑になり、サーバーのリソースがさらに必要になります。

  一般的には、インクリメンタル・リフレッシュの方がフル・リフレッシュよりもリソースの消費量が少ないため、パブリケーションの表の同期履歴プロパティーを設定して、インクリメンタル・リフレッシュを有効にできます。これにより、マスター・サーバーは、フル・パブリケーションではなく、パブリケーションで変更があったマスター・データのみを送信できます。詳しくは、100 ページの『パブリケーションの作成』を参照してください。
- 必要以上に頻繁に同期を実行しないようにします。
- オフピークの時間帯に同期化処理を使用します。システムのオンライン使用が最小の時間帯に、「緊急性の高くない」データをまとめて同期します。

レプリカ・データベース

システムのレプリカ・サーバーの使用パターンは、通常、かなり「従来型」です。サーバーは、データベースに対する照会操作および書き込み操作を実行するアプリケーションからアクセスされます。レプリカ・データベースには、サーバーの通常のオンライン使用に十分な容量が必要です。データベース同期で発生するオーバーヘッドをカバーするために、追加容量を予約してください。

システムのデータ・セキュリティーのレベルを最大にするために、可能な場合、物理データベース・ファイルへのユーザー・アクセスを拒否します。

ネットワーク

マスター・サーバーは、できる限りスループットの高いマシンに置いてください。データベースの同期を処理できる十分なネットワーク帯域幅を確保するために、同期中の転送データの最大量を注意して算定し、テストしてください。

同期メッセージの伝送についてネットワークをテストしてください。同期メッセージの構成要素は以下のとおりです。

- ヘッダー・データ (わずか)
- トランザクション (以下を含む)
  - プロシージャー呼び出し (ストリング)
  - パラメーター (バイナリー・データ)
- パブリケーションへのリフレッシュ (以下を含む)
  - マスター・データベースからのすべての挿入行および更新行
  - マスター・データベースから削除する行の主キー

同期のためのデータベースの設計および準備

solidDB Development Kit のリリース・ノート・ファイルの説明に従って、各マシンに solidDB をインストールしたら、同期のためにデータベースの準備と設計を行うことができます。これには以下の作業が必要になります。

- マスター・データベースとレプリカ・データベースを定義する。
- 拡張レプリケーションのガイドラインに従って、データベース・スキーマを作成する。
- 環境がマルチマスター環境である場合、またはマスター・データベースとレプリカ・データベースで異なるスキーマ名を使用する場合に、カタログを作成する。
- 同期表での並行性競合の処理を定義する、つまり、表がオプティミスティック並行性制御とペシミスティック並行性制御のどちらを採用するのかを決める。
- 同期に必要なユーザー・アクセス権限を付与する。
- マスター・データベースと大規模レプリカのバックアップをセットアップする。
-

データ同期のためにアプリケーションを設計する。

上記の各トピックについては、以下のセクションで説明します。

マスター・データベースとレプリカ・データベースの定義

データベース・スキーマを作成する前に、SET SYNC コマンドを使用して、データベース・カタログを「マスター」、「レプリカ」、またはその両方として設定する必要があります。セットアップに必要なコマンドを入力する際に、solidDB SQL エディター (テレタイプ) を使用することができます。

カタログを専用「マスター」データベースとして指定するには、データベースが存在するカタログに以下のコマンドを入力します。

```
SET SYNC MASTER YES;
COMMIT WORK;
```

カタログを 2 つの役割を持つよう指定する (つまり、複数層からなる同期階層の中間層のデータベースとして指定する) には、データベースが存在するカタログに以下のコマンドを入力します。

```
SET SYNC MASTER YES;
SET SYNC REPLICA YES;
COMMIT WORK;
```

専用のレプリカが存在するカタログごとに、以下に示すように、カタログを「レプリカ」データベースとして指定します。

```
SET SYNC REPLICA YES;
COMMIT WORK;
```

現行のデータベース・カタログは、SET CATALOG コマンドを使用して定義することができます。カタログを何も指定しなければ、基本カタログが使用されます。

データベース・スキーマの作成

マルチデータベース・システムでは、データベースの使用法は非常に多彩です。したがって、物理的にデータベースをインプリメントしチューニングするときに、ご使用のシステム内でのデータベースの使用方法を検討する必要があります。

以下は、スキーマとカタログを使用する際のガイドラインです。ご使用の拡張レプリケーション・アーキテクチャーに該当するセクションを参照してください。

2 層トポロジーのガイドライン

2 層データ冗長モデルには、1 つのマスター・データベースと複数のレプリカ・データベースがあります。マスターとレプリカの両方のデータベースには、データベース所有者のユーザー ID であるデフォルト・スキーマ名を使用して、異なるスキーマを割り当てることができます。この場合、スキーマを明示的に定義する必要はなく、かわりに、サーバーがユーザー ID に対して自動的にスキーマを割り当てます。マスター・データベースおよびレプリカ・データベースに同じスキーマ名を使用することを推奨します。マスターとレプリカで異なるスキーマ名を使用することもできますが、異なるスキーマ名を使用すると、アプリケーション・プログラミングが複雑化する場合があるので注意してください。

スキーマを使用するには、スキーマに関連付けるデータベース・オブジェクトの作成前に、スキーマ名を作成する必要があります。スキーマを作成するには、CREATE SCHEMA コマンドを使用します。詳しくは、「*solidDB SQL ガイド*」を参照してください。

複数層トポロジーのガイドライン

複数層トポロジーでは、同期データベース階層に 3 つ以上の層が含まれています。トポロジーの最上層がシステム全体のマスター・データベースです。トポロジーの中間層のデータベースは、マスター・データベースおよびレプリカ・データベースという 2 つの役割を持ちます。最下層には、レプリカだけが含まれます。

複数層トポロジーは、地理的に広く分散されたシステムで、レプリカ・データベースの数が多い可能性があり、さらにそのレプリカ・データベースにローカル・データ (最上層のマスターとの同期を必要としないデータ) が存在する場合に特に有用です。このタイプのシステムのデータは通常、特定のレプリカへのデータ・アクセスを制限するために、パーティション化されています。例えば、大規模な管理対象ネットワークの構成およびイベント情報を管理するデータベースが含まれたネットワーク管理システムは、複数層トポロジーの基準を満たします。

マルチマスター・トポロジーのガイドライン

solidDB の物理データベース・ファイルには、複数の論理データベースを含めることができます。各論理データベースは、表、索引、プロシージャー、トリガーなどのデータベース・オブジェクトから成る、完全で独立したグループです。各論理データベースはデータベース・カタログとしてインプリメントされます。これらのカタログはそれぞれ、独立したマスター・データベースまたはレプリカ・データベースとして機能できます。これにより、例えば、複数の独立したレプリカ・データベースを 1 つの物理ローカル・データベースに作成することができます。また、それぞれにマスター・データベースを含む 1 つ以上のカタログを、この同一ローカル・データベース内に入れることもできます。

マルチマスター・トポロジーは、複数のアプリケーションが solidDB データベースを使用する環境で有用です。例えば、ローカル・データベースに、構成管理アプリケーション用と使用状況モニター・アプリケーション用にそれぞれ 1 つずつ、合計 2 つのマスターのレプリカを含めることができます。複数層トポロジーおよびマルチマスター・トポロジーは組み合わせられる点に注意してください。

カタログの作成

以下は、同期に使用する複数のカタログの設計とインプリメンテーションに関するガイドラインです。

- データベースの作成時、solidDB はデータベース用のデフォルト・カタログを作成します。
- デフォルト・カタログ以外にも、単一の solidDB データベースに、必要な任意の数のカタログを含めることができます。
- CREATE および SET でカタログを明示的に作成および設定しない場合は、データベースのデフォルト・カタログを使用することになります。
- データベースの各カタログは、マスター、レプリカ、またはその両方とすることができます。

- 各カタログには複数のスキーマを含めることができます。トランザクションは任意のカタログのデータベース・オブジェクトにアクセスできます。
- カタログにはローカル表と、マスターと同期する表を含めることができます。単一のトランザクションでローカル表とマスター表を組み合わせて使用することができます。
- 物理データベースには、ローカル・データ管理機能にアクセス可能な一連のローカル・ユーザーが定義されています。データ同期機能にアクセスできるようにするため、各カタログにはレプリカの登録の一部としてダウンロードされた 1 つ以上のマスター・ユーザーがあります。

図 9 はこれらのガイドラインを示しています。

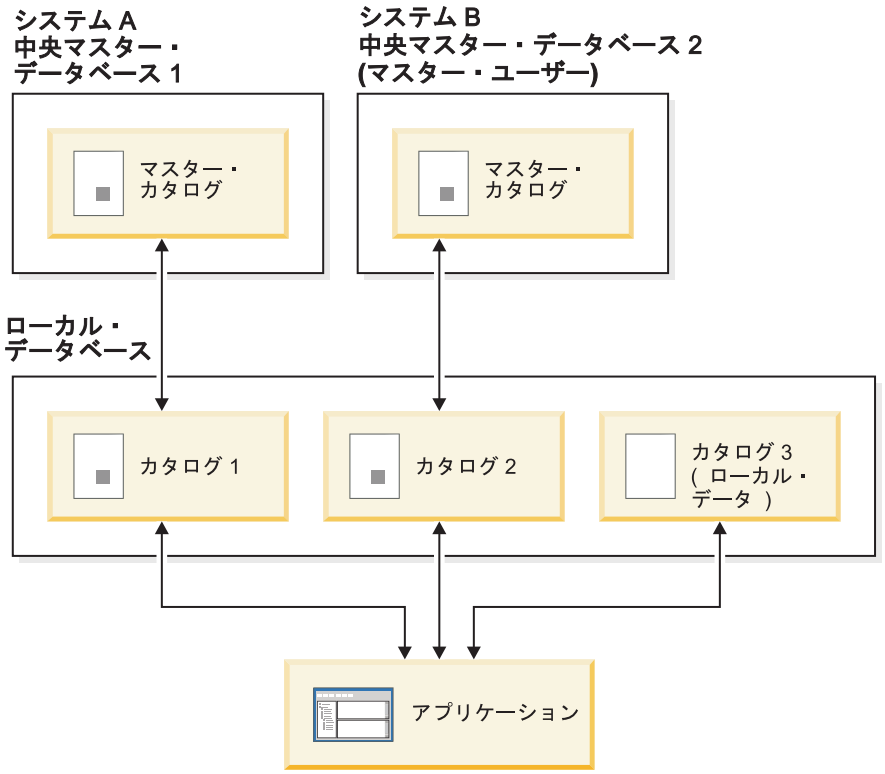


図 9. マルチマスター・モデル

マスターとレプリカ用にカタログを作成し、設定するには、CREATE CATALOG コマンドおよびSET CATALOG コマンドを使用します。カタログの作成と設定について詳しくは、「*solidDB SQL ガイド*」の CREATE CATALOG コマンドおよび SET CATALOG コマンドの説明を参照してください。

マスター上では以下のように指定します。

```
CREATE CATALOG INVENTORY;
SET CATALOG INVENTORY;
COMMIT WORK;
```

レプリカ上では以下のように指定します。

```
CREATE CATALOG INVENTORY;
SET CATALOG INVENTORY;
COMMIT WORK;
```

注:

1. カタログ名は、マスターとレプリカで同じにする必要はありません。 (完全修飾名を指定しない場合)
2. 作成後に実際にカタログを使用する際に、(カタログ名を含む) 完全修飾表名を使用して指定することも、SET CATALOG コマンドを使用して、どのカタログを使用するかを指定することもできます (完全修飾名を指定しない場合)。

カタログ内でのスキーマの使用

データベースを論理的にパーティション化するには、スキーマの作成前に最初にカタログを作成します。カタログとスキーマの作成後、スキーマに関連付けるデータベース・オブジェクトを作成します。スキーマを指定せずにデータベース・オブジェクトを作成すると、スキーマがユーザー ID になります。

カタログ内で複数のスキーマを使用することができます (ただし、単一のスキーマで十分な場合もあります)。複数のスキーマがある場合は、表名の一部にスキーマを含めてスキーマを指定することも、SET SCHEMA コマンドを使用して、どのスキーマを使用するかを指定することもできます。

CREATE および SET でスキーマ名を明示的に作成および設定しない場合は、ユーザー ID であるデフォルト・スキーマ名を使用することになります。

注: カタログの場合は、データベース全体に 1 つのデフォルトがありますが、スキーマの場合は、ユーザーごとに 1 つのデフォルトがあります。データベース全体には 1 つのデフォルトはありません。

スキーマの作成と設定について詳しくは、「*SQL ガイド*」の CREATE SCHEMA コマンドおよび SET CATALOG コマンドの説明を参照してください。

同期のためのデータのセットアップ

このセクションは、2 層と複数層の両方のマルチレベル・アーキテクチャーに適用されます。カタログとスキーマ名 (必要な場合) が作成されていると想定しています。

以下に、スキーマを設計およびインプリメントするためのガイドライン、および CREATE TABLE コマンドを使用してマスター・データベース表およびレプリカ・データベース表を作成するためのガイドラインを示します。

同期に必要な表であり、かつパブリケーションで使用される表を定義します。パブリケーションは、マスター・データベースからレプリカ・データベースにダウンロードされるデータ集合です。スキーマを作成するときには、以下の各項目を定義する必要があります。

- マスター・データベースの表
-

レプリカ・データベースの表

- レプリカ・データベースには、マスター・データベースのすべての表を含めることも、そのサブセットを含めることも可能です。
- また、レプリカ・データベースには、ローカルにのみ使用する表を含めることができます。
- レプリカ表には、マスター表からの列のサブセットを含めることができます。
- レプリカ表の名前は、マスター表とは異なっても構いません。 CREATE PUBLICATION コマンドを使用してパブリケーションを作成するときには、マスター表名を、異なる名前を持つレプリカ表に関連付けることができます。パブリケーション定義は、マスター表とレプリカ表のマッピングを処理します。

表の作成時には、以下の点に注意してください。

- スキーマ内のすべての表には、ユーザー定義の主キーが必要です。マスター表およびレプリカ表内の主キーは同一であり、かつユニークであることがグローバルに保証されていなければなりません。レプリカの主キー内の列数の方が多いと、マスター・データベースにトランザクションを伝搬するときに競合が発生します。マスターの主キー内の列数の方が多いと、同様に、レプリカにデータをリフレッシュするときに競合が発生します。
- ALTER TABLE SET SYNCHISTORY コマンドを適用して、マスター表とレプリカ表に対するインクリメンタル・パブリケーションを使用可能にします。その他の場合、マスターは、インクリメンタル・パブリケーションではなく、フル・パブリケーション (より多くのリソースを使用) をレプリカに送信します。

マスターとレプリカの両方のデータベース内のパブリケーションの各表に対して SYNCHISTORY プロパティーを設定することにより、インクリメンタル・パブリケーション用のデータ更新を追跡するシャドー履歴表の作成を可能にします。ALTER TABLE 構文について詳しくは、 100 ページの『インクリメンタル・パブリケーションの作成』を参照してください。

論理データベースの設計

マルチデータベース・システムのデータ・モデルは、集中型システムのデータ・モデルとはやや異なります。レプリカ・データが一時的なデータであるという特性、および同じデータ項目 (行) の複数の異なるバージョンが共存する可能性に、論理データ・モデルで適切に対処する必要があります。したがって、マルチデータベース・システムの論理データベースを設計する場合に考慮すべき経験法則がいくつかあります。

ユニークな代理主キー

マスター・データベースで実行されるすべての書き込み操作は、正常に終了しなければなりません。「ユニーク制約違反」があると、同期化処理全体が停止します。したがって、行の主キー (およびユニーク索引) は、システム全体でユニークでなくてはなりません。データベース内のすべての表で、グローバルにユニークである代理主キー値を使用することを推奨します。このようなキーは、例えば、データベース ID とシーケンス番号を組み合わせたものにすることができます。以下に例を示します。

```
CREATE TABLE CUST_ORDER
DB_ID VARCHAR NOT NULL, ID INTEGER NOT NULL,
... 表のその他の列,
(PRIMARY KEY (DB_ID, ID));
```

更新の競合の検出

更新の競合を検出する方法としては、システムの各表の中にある「更新時間」列を使用するのが最も簡単です。行が更新されるときに、毎回、更新時間列の現行の (更新前の) 値が WHERE 節に追加されます。行が見つからなければ、更新時間が変更されたことになります。つまり、行が更新されたため、競合が発生していることになります。このメカニズムをオプティミスティック・ロック方式と呼びます。

同期エラーの報告

同期中にエラーが発生する可能性が常にあります。そのエラーの中には、マスター・データベースで発生する更新の競合のように、アプリケーション・レベルのエラーもあります。このようなエラーを自動的に解決することができない場合には、エラーを手作業で解決できるようにログに記録する必要があります。

これを実施する 1 つの方法としては、各エラーのエントリーを入れる同期エラー・ログ表を作成する方法があります。同期中にアプリケーション・レベルのエラーが発生したときには必ず、トランザクションのストアード・プロシージャーによってエラー・ログ表に行が挿入されるようにする必要があります。エラー・ログの作成方法については、114 ページの『アプリケーションに対する同期エラー・ログ表の作成』を参照してください。

エラーの中には、同期メッセージの処理を停止するサーバー・レベルのエラーもあります。この場合には、メッセージング・レベルでエラー・リカバリーを実行します。詳しくは、149 ページの『7 章 拡張レプリケーションによる solidDB の管理』を参照してください。

データベース・スキーマの設計

マスター・データベース とレプリカ・データベースとでは、使用パターンがまったく違うことがあります。マスター・データベースとレプリカ・データベースで実行される照会もまったく違うことがあります。

したがって、データベースの索引付けは、使用パターンが違うことに注意しながら、慎重に設計する必要があります。レプリカの索引付けは、データベースを使用するアプリケーションの要件に合わせる必要があります。マスター・データベースの索引を設計する際に、 ユニーク索引がグローバルにユニークでなければならない

ことに注意してください。パブリケーションとトランザクションについては、以下の索引に関するガイドラインも検討してください。

パブリケーション

パブリケーション定義から派生する照会には、索引を付けるようにしてください。solidDB は、ネストされたパブリケーションを結合として扱います。パブリケーション操作を効率よく実行するには、パブリケーションの表の結合列に索引を付ける必要があります。

以下の例では、CUSTOMER 表の SALESMAN_ID 列を使用して、CUSTOMER 表と SALESMAN 表を結合しています。このパブリケーションからのリフレッシュを効率よく実行するために、副次索引を使用して SALESMAN_ID 列に索引を付けます。

```
CREATE PUBLICATION pub_customers_by_salesperson (sperson_area varchar)
BEGIN
   RESULT SET FOR salesman
   BEGIN
      SELECT * FROM salesman where area = :sperson_area
      DISTINCT RESULT SET FOR customer
      BEGIN
         SELECT * FROM customer WHERE salesman_id = salesman.id
      END
   END
END;
```

ユーザーが作成する索引に加えて、solidDB は、SYNCHISTORY プロパティーがオンに設定されている表に対して、自動的にシステム索引を 2 つ作成します。また、メイン表の履歴表に対しても、同じ索引が自動的に作成されます。

トランザクションによる書き込み負荷

マスター・データベースの書き込み負荷により、拡張レプリケーション・システムのスケーラビリティーの実際の制限が決まります。拡張レプリケーション・システムでは、すべての伝搬トランザクションは最終的にマスター・データベースにコミットされます。それぞれの索引により、その表に対するすべての書き込み操作 (挿入、更新、削除) で追加のディスク I/O が発生します。よって、書き込みのパフォーマンスが重要な要因である場合には、マスター・データベースの副次索引の数を最小限にしてください。

データベース表の定義

以下の CREATE TABLE SQL コマンドは、同期用にセットアップされたデータベースの典型的な表を作成します。また、ALTER TABLE SET SYNCHISTORY 拡張レプリケーションの拡張機能により、この表でインクリメンタル・パブリケーションに対する準備が行われることに注意してください。また、小さくて更新頻度が高い表の場合と違って、大きい表を使用する場合には、必ずインクリメンタル・パブリケーション用に表をセットアップしてください。

注:

ALTER TABLE SET SYNCHISTORY ステートメントを実行する前に、SET SYNC MASTER コマンドと SET SYNC REPLICA コマンドを使用して、マスターとして、およびレプリカとしてデータベースを定義しておいてください。マスターとレ

プリカを定義しておかないと、ALTER TABLE コマンドを使用しようとしたときにエラー・メッセージが表示されます。詳しくは、 97 ページの『同期のためのデータベースのセットアップ』を参照してください。

注:

レプリカが読み取り専用の場合 (パブリケーションの複製部分の変更が禁止されている場合)、ALTER TABLE ... SET SYNCHISTORY ステートメントは不要です。その際、以下の Flow Replica 常駐パラメーターを設定する必要があります。

```
set sync parameter SYS_SYNC_KEEPLOCALCHANGES 'Yes';
```

また、この場合は、ALTER TABLE ... SET HISTORY COLUMNS を使用できないことに注意してください。

```
CREATE TABLE CUST_ORDER (
ID VARCHAR NOT NULL,
SYNC_STATUS CHAR(3) NOT NULL,
CUST_ID VARCHAR NOT NULL,
PRODUCT_ID VARCHAR NOT NULL,
QUANTITY INTEGER NOT NULL,
PRICE DECIMAL(10,2) NOT NULL,
UPDATETIME TIMESTAMP NOT NULL,
PRIMARY KEY (ID, SYNC_STATUS));
ALTER TABLE CUST_ORDER SET SYNCHISTORY;
COMMIT WORK;
```

以下は、上記の例に関する注釈です。

- ID 列は、新しい行の主キーの生成値 (代理キー) です。

  ```
  ID            VARCHAR NOT NULL,
  ```

  値はできるだけ、行が最初に作成されたデータベースのユニーク ID と、そのデータベース内のシーケンス番号の 2 つのパートの複合となるようにしてください。

  代理キーの使用を推奨する理由は、キーがグローバルにユニークである必要があるからです。2 つの異なるレプリカ・データベースに同じキー値で行を挿入することを、*許可してはいけません*。許可した場合、次回のトランザクション伝搬タスクで、ユニーク制約違反のエラーがマスター・データベースに作成されます。このようなエラーが発生するたびに、同期化処理が停止し、データベースの中から重複している行を削除して問題を解決するまで、継続不可能になります。

- SYNC_STATUS 列には、行の同期状況に関する情報が格納されます。

  ```
  SYNC_STATUS   CHAR(3) NOT NULL,
  ```

  行が有効であれば、例えば「OK」のような値になります。しかし、行がマスター・データベースでコミットされたときに、更新の競合などのトランザクションの妥当性検査エラーが発生した場合には、例えば「C01」(この行で最初に発生した更新の競合) などの値が設定された行がデータベースに挿入されます。この列があるため、同じ行の複数のバージョンをデータベースに保管することが可能です。これらのバージョンのうち、1 つは正式なバージョンであり (状況が「OK」

の行)、その他は競合解決およびエラー・リカバリーを目的とした複数の追加バージョンです。表の主キーは、ID 列と SYNC_STATUS 列から構成されます。

- UPDATETIME 列には、行の最新の更新日時を示すタイム・スタンプが格納されます。

  ```
  UPDATETIME TIMESTAMP     NOT NULL,
  ```

  アプリケーション・ロジック (トランザクションを形成するストアード・プロシージャーを含む) はシステム内の更新の競合の検出にこの列を使用することができます。

UPDATE トリガーの処理

UPDATE トリガーに特殊な設計やコーディングが必要な場合があります。

マスターでレコードに対して UPDATE を実行し、その UPDATE 操作によって UPDATE トリガーが起動されると、レプリカにレコードが (REFRESH によって) 送信される際に、マスターでも UPDATE トリガーが起動されると予想するかもしれません。しかし、実際にはそうではありません。レコードがマスターで更新され、レプリカで同じレコードが REFRESH 操作によってマスターで「更新」されると、レプリカ・サーバーは、実際には UPDATE トリガーではなく、DELETE+INSERT のトリガー・ペアを起動します。これはレプリカが直接レコードを更新するのではなく、古いレコードを削除してから新しいレコードを挿入するからです。

トリガー起動の考えられる原因について

サーバーは、同期に関係しトリガー起動の原因となる、1 組の挿入/削除操作を追加で実行することもできます。マスター・データをレプリカの表にコピーする前に、レプリカは、前回の同期以降にレプリカに書き込まれたローカル・データ変更をすべて「取り消し」ます。これは、レプリカのデータは、マスターによって処理されて正式なデータになるまでは、常に「一時的な」ものであるからです。レプリカに一時的なデータが存在する場合、レプリカはマスターから正式なデータを受信すると必ず、レプリカ内に存在する正式でないデータをすべて破棄し、マスターからの正式なデータのみを保管します。データの一時的な変更を「取り消す」プロセスにより、レプリカ・サーバーは追加の削除と挿入を実行します。このことについて、以下で詳しく説明します。

REFRESH がインクリメンタルの場合、処理はいくつかのステップで実行されます。最初のステップでは、実際上は、前回の正式なデータを受信した後の一時的な変更をすべて取り消して、前回の正式なデータをリストアします。2 番目のステップでは、新しい正式なデータを処理します。したがって、前回のリフレッシュ時点の古い正式なデータがすべて存在し、そのリフレッシュ以降の正式な変更がすべて存在しているため、データはマスターのデータとまったく同じになります。

ここでは例を示して段階的に説明します。この例では、マスターからフル・リフレッシュを受信し、レプリカで何らかの変更を加え、マスターからインクリメンタル・リフレッシュを受信します。マスターからリフレッシュを受信するプロセスでは、レプリカのデータをすべて破棄し、マスターから送信されるデータをすべて保管します。

午前 10:00、フル・リフレッシュを受信。フル・リフレッシュの内容は、Anne Anderson、Barry Barrymore、および Carrie Carlson のデータ・レコードです。

午前 10:01、Anne Anderson が自分のレコードを更新。

午前 10:02、Barry が自分のレコードを更新。

午前 10:03、変更を伝搬。この場合は、Anne と Barry のレコードの変更が伝搬されます。この例では、Anne の変更はマスターに受け入れられ、Barry の変更は拒否されるものとします。

午前 10:04、インクリメンタル・リフレッシュを受信。インクリメンタル・リフレッシュの処理時に、レプリカは前回の正式なリフレッシュ (この場合は午前 10:00 のリフレッシュ) 以降の変更をすべて破棄します。Anne と Barry の変更がレプリカから破棄されます。

この時点で、レプリカの内容は、前回のリフレッシュ操作後の内容と同じであり、したがってレプリカはインクリメンタル・サブスクリプションの受信が可能な状態になっています。このサブスクリプションには、前回のリフレッシュ以降にマスターに承認された変更のみが含まれています。

インクリメンタル・リフレッシュの実行時に、Anne の更新レコードを受信します (伝搬したデータはマスターに受け入れられ、REFRESH の実行時に返されました)。Barry の変更は、マスターに拒否されたため、永久に消失します。最終結果として、データベースには 3 件のレコードすべてに正式な値が存在します。すなわち、Anne については新規/承認済みの変更であり、Barry と Carrie についてはそれより古い (直前の正式な) データです。

(上記の説明では、単純化のためにいくつかの想定をしていることに注意してください。例えば、Barry と Carrie のレコードは、このレプリカによる前回の正式なリフレッシュ以降にマスターでは変更されていません。)

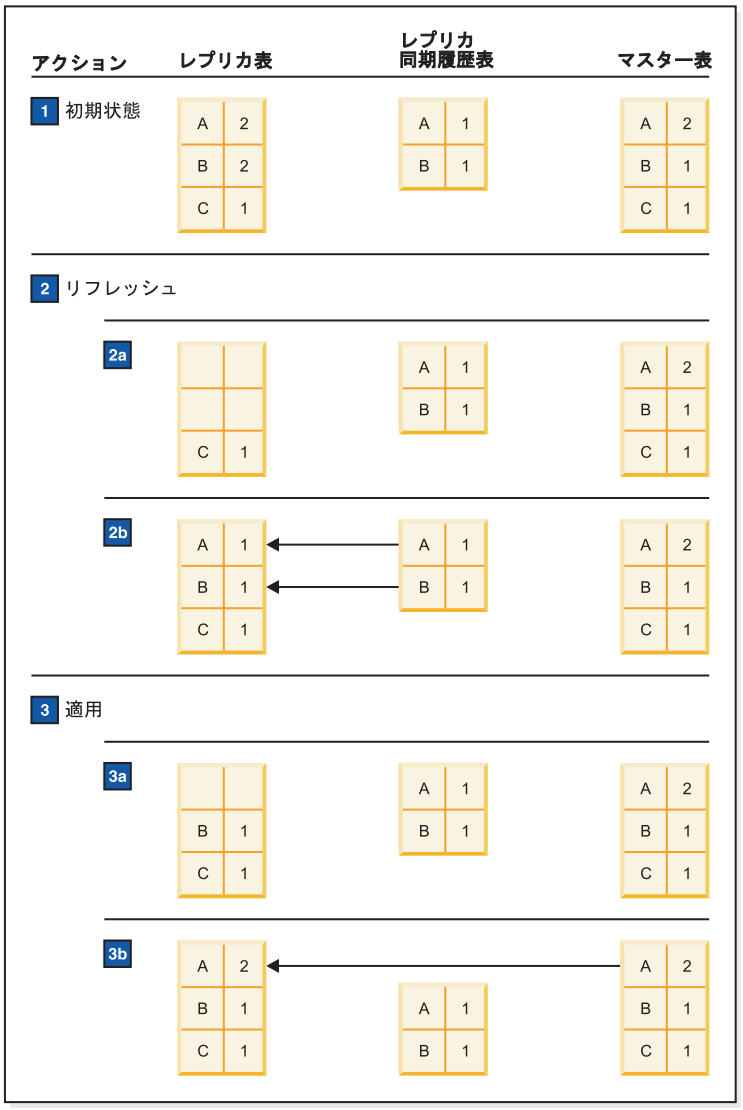
ここでは例を挙げて、レプリカの更新から、マスターからのリフレッシュの受信までの一連の処理全体を説明し、この一連の処理の間に実行される可能性があるすべてのトリガーを理解できるようにします。この例では、レプリカがローカル更新を実行しますが、その更新はマスターに伝搬され、そこで実行されます。次に、同期の結果セットとして、更新のマスター・バージョンがダウンロードされてレプリカに戻されます。この一連の処理の間に、以下に示すように、マスター・データベースとレプリカ・データベースでトリガーが起動されます。

1. レプリカで更新が行われると、レプリカで通常の UPDATE トリガー (BEFORE トリガーと AFTER トリガーの両方) が起動されます。レプリカに古いマスターの値が含まれる場合 (まったく新しいレプリカである場合を除いて、おそらく古い値が含まれており、このレプリカではリフレッシュをまだ行っていません)、それらマスターからのデータ値はレプリカの履歴表に書き込まれます。
2. 次にレプリカはそのデータをマスターに伝搬します。マスターで伝搬トランザクションを実行するときに、マスターで UPDATE トリガーが起動されます。更新対象の行の古い値は、マスター・データベースの同期履歴表に書き込まれます。行の新しい値はメイン表に書き込まれます。
3. レプリカがリフレッシュを要求すると、マスター・サーバーはサブスクリプションの結果セットを組み立てます。その中には、古いバージョンの更新行の主キー

値が含まれます。このような行は、レプリカ表から削除されます。この行が削除された後、同じ行の新しい値が挿入されます。(レプリカに送信される結果セットには、最初に削除対象の主キーのリストが含まれ、その後に挿入対象の行が含まれます。)

4. レプリカはマスターから同期メッセージを受信すると、メイン表から「一時的な」データをすべて削除します。このときに、削除トリガー (ローカルの一時的な行の削除) が起動されます。その後にレプリカ・サーバーによって、同期履歴表から、存在すると考えられる古いマスター・バージョンの行が挿入されます。このときに、挿入トリガー (古い正式な行の挿入) が起動されます。その後にサーバーによって、同期メッセージから削除が適用されます。このときに、削除トリガー (古い正式な行の削除) が起動されます。最後にサーバーによって、同期メッセージから挿入が適用されます。このときに、挿入トリガー (新しい正式な行の挿入) が起動されます。

上記のステップ 4 の図については、68 ページの図 10 を参照してください。この図には、インクリメンタル・リフレッシュの実行中に行われる処理の詳しい内容が示されています。8 ページの図 1 にある前の方の図は、レコードの伝搬とリフレッシュの処理全般を示しています。後の方の図は、前の図のインクリメンタル・リフレッシュ部分だけの詳細を示しています。図から分かるように、「Anne」が更新したレコードについては、レプリカ・サーバーによって実際には複数の削除および挿入操作が行われます。また、「Barry」が更新したレコードも、単に更新されるのではなく、いったん削除されてから挿入されます。



1. 初期状態は、伝搬後で、かつインクリメンタル・リフレッシュの直前の状態を示しています。Anne と Barry は、最後のリフレッシュ後にデータを変更しました。マスターは、Barry の更新を受け入れなかったことに注意してください。
2. リフレッシュ状態では、1) 最後のリフレッシュ後にロールバック・レプリカを変更、および 2) マスターから最新または更新したレコードを挿入、という 2 つのステップが存在します。これらの 2 つのステップは、それぞれ 2 つのサブステップ (削除と更新) で構成されています。
   a. 最後のリフレッシュ後に変更されたレコードを削除します。
   b. 同期履歴からコピーすることにより、最新の「正式」レコードを挿入します。これで、レプリカは、最後の更新の直後の状態になったように見えます。
3. マスターから受け取った新しいレコード値を適用します。この場合、マスターから "Anne Anderson 2" が送信されています。
   a. マスターから送信された新しい値に対するレコードを削除します。
   b. マスターからの最新の「正式」レコードを挿入します。

図 10. インクリメンタル・リフレッシュの詳細

レプリカ・データベースとマスター・データベースの両方で行われる可能性がある更新操作を同期する場合、以下に示す最大 4 つの同期関連トリガーがレプリカで起動される可能性があります。

- 現行の一時的な行の削除
- 古い正式な行の挿入
- 古い正式な行の削除
- 新規の正式な行の挿入

(また、マスターのいくつかのトリガーも起動される可能性があります。)

これで、マスターで UPDATE が実行されたときに、拡張レプリケーション操作で起動される可能性があるすべてのトリガーについて説明しました。次に、新しい掲示板パラメーターで可能なすべての値について説明します。可能な値の詳細な内容は、169 ページの『付録 A. 掲示板パラメーター』に記載されています。以下は要約です。

DELETE トリガーの SYS_SYNC_OPERATION_TYPE パラメーターで可能な値は、以下のとおりです。

- CURRENT_TENTATIVE_DELETE (レプリカで応答メッセージを実行する前に、行のローカルに更新された現行値を削除する場合に設定)
- OLD_OFFICIAL_DELETE (マスターで削除された行を削除する場合に設定)
- OLD_OFFICIAL_UNIQUE_DELETE (追加する行をマスターがレプリカに送信したが、同様の行が既にレプリカに存在する場合に設定。このパラメーター値を指定すると、新しい行がレプリカに追加される前に、古い正式な行が削除されます。)
- OLD_OFFICIAL_UPDATE (マスターの更新結果として作成された削除を実行する場合に設定)

INSERT トリガーの SYS_SYNC_OPERATION_TYPE パラメーターで可能な値は、以下のとおりです。

- OLD_OFFICIAL_INSERT (レプリカで応答メッセージを実行する前に、古いマスターの値をリストアする場合に設定)
- NEW_OFFICIAL_INSERT (マスターに挿入された行を挿入する場合に設定)
- NEW_OFFICIAL_UPDATE (マスターの更新結果として作成された挿入を実行する場合に設定)

トリガーを起動したのがローカル・トランザクションである場合 (つまり同期ロジックではない場合) には、このパラメーターの値は NULL です。

上記の説明は、インクリメンタル・リフレッシュの場合にのみ当てはまります。結果セットがフルの場合には、ローカル・データはすべて削除され、マスター・データのフル・セットが適用されます。この場合、マスター・データベースからレプリカに DELETE 操作は送信されません。リフレッシュがフルとインクリメンタルのどちらなのかを判断するには、掲示板パラメーター SYS_SYNC_RESULTSET_TYPE の値を読み取ります。このパラメーターで可能な値は以下のとおりです。

- FULL
- INCREMENTAL

UPDATE トリガー用のコードの実行

残念ながら、マスター上での UPDATE 操作がレプリカでも UPDATE として必ず実行されるようにするのは容易ではありません。レプリカで UPDATE トリガーを直接、強制的に実行しようとするのではなく、solidDB サーバー (マスター) は拡張レプリケーション・パラメーター掲示板に 2 つのパラメーターを通知しています。これらのパラメーターをレプリカ上のトリガー・ロジックが読み取り、マスターでそのレコードが当初どのように処理されたのかを判断することができます。そして、レプリカは、必要に応じて、UPDATE 操作に適したロジックを実行できます。

掲示板パラメーターは、レコードが当初どのように処理されたのか (例えばマスター上で、当初は UPDATE 操作の一部だったかどうかなど) を示します。当然のことながら、INSERT または DELETE トリガーは、それ自体を停止してから、UPDATE トリガーを強制的に起動することが簡単にはできないので、元のコマンドが UPDATE トリガーだったことを知るだけでは不十分です。トリガー内のコードを変更し、元のステートメントが何であったか (マスターでのオリジナル・ステートメントが更新だったのか、DELETE/INSERT のペアだったのかなど) に応じて、異なるアクションを行うようにする必要があります。必要に応じて、DELETE または INSERT トリガーによって、UPDATE トリガーで呼び出すはずであったコードと同じものを呼び出します。レプリカでは以下を行う必要があります。

表において考えられる以下の 6 つのトリガーのそれぞれに対して、

```
BEFORE UPDATE,
AFTER UDPATE,
BEFORE INSERT,
AFTER INSERT,
BEFORE DELETE,
AFTER DELETE,
```

すべてのトリガー・ロジックを、トリガー内部から呼び出せるストアード・プロシージャーに抽出します。

したがって、例えば、BEFORE UPDATE トリガーが以下のようになる場合は、

```
CREATE TRIGGER trigger_name ON table1 BEFORE UPDATE
   BEGIN
   stmt1;
   stmt2;
   stmt3;
  END;
```

以下のような結果になります。

```
CREATE PROCEDURE before_update_on_table1
   BEGIN
   stmt1;
   stmt2;
   stmt3;
   END;
CREATE TRIGGER trigger_name ON table1 BEFORE UDPATE...
   BEGIN
   CALL before_update_on_table1;
   END;
```

上記の処理の後、レコードが元々 UPDATED と INSERTED のどちらだったのかを考慮するように、トリガーを書き換えることができます。例えば、BEFORE INSERT トリガーのロジックは以下のようになります。

```
CREATE TRIGGER trig1 ON table1 BEFORE INSERT...
 BEGIN;
 IF (get_param('SYS_SYNC_OPERATION_TYPE') = 'NEW_OFFICIAL_UPDATE') THEN
  IF (get_param('SYS_SYNC_RESULTSET_TYPE') = 'INCREMENTAL') THEN
   CALL before_update;
  ELSE ...
  END IF;
 ELSE IF (get_param('SYS_SYNC_OPERATION_TYPE') = 'NEW_OFFICIAL_INSERT') THEN
  CALL before_insert; - do what I normally do in my INSERT trigger.
 ELSE ...
 END IF;
 END;
```

トリガーとストアード・プロシージャーの言語は基本的に同じなので、トリガーの本体をストアード・プロシージャーに移動することは比較的簡単です。

重要:

レプリカのトリガーでキーワード「BEFORE」を使用した場合でも、マスターの「Before」値は使用できません。

レプリカの同期表における並行性競合の処理

solidDB は、オンライン照会や書き込み操作などのデータ管理機能の処理の並行性制御メカニズムを使用します。デフォルトの並行性制御方法として、すべての表に対して、オプティミスティック並行性制御が自動的に設定されます。したがって、2つのユーザーが同時に同じデータを変更しようとすると、後から変更しようとしたほうが失敗し、そのユーザーにエラーが戻されます。

同期化中は、以下の例に示すような一連のイベントによって、並行性競合が発生することがあります。

1. パブリケーションからリフレッシュを行う同期スクリプトをレプリカが作成し、実行する。
2. 同時に、レプリカの別のユーザーがリフレッシュ対象の行を更新する。
3. ユーザーがトランザクションをコミットする前に、同期メッセージの応答メッセージがレプリカに到着し、エンジンがリフレッシュ・データのデータベースへの適用を開始する。
4. ユーザーがオンライン・トランザクションをコミットする。
5. オンライン・ユーザーが既に変更済みの行をリフレッシュで変更しようとする。
6. 並行性競合により、同期応答メッセージの実行が失敗する。

以下の表には、この例に示す、または類似の状況下で発生する並行性競合を処理するための各種方法が示されています。

表 5. 並行性競合の処理

リカバリーのための基準と方法	リカバリーに使用するコマンド
並行性競合が頻繁に発生することが予想されない場合は、レプリカで失敗した応答メッセージを再実行して、この問題からリカバリーすることができます。	失敗した応答を再実行するには、以下のコマンドを使用します。 `MESSAGE msgname EXECUTE`
並行性競合が頻繁に発生することが予想され、並行性競合によってメッセージの再実行が失敗する場合は、ペシミスティック表レベル・ロック方式を使用して、メッセージを実行できます。これによって、メッセージが正常に実行されます。 このモードでは、同期メッセージの完了まで、影響を受ける表への他のすべての並行アクセスがブロックされます。	ペシミスティック・モードでメッセージを実行するには、以下のコマンドを使用します。 `MESSAGE msgname EXECUTE PESSIMISTIC`
応答メッセージを、最初の実行時にペシミスティック表レベル・ロック方式を使用するように定義することができます。	ペシミスティック・モードの応答メッセージをマスターに要求するには、以下のコマンドを使用します。 `MESSAGE msgname GET REPLY PESSIMISTIC`

MESSAGE FORWARD 操作の一部として、応答メッセージの最初の実行時に、ペシミスティック表レベル・ロック方式を使用することができます。	ペシミスティック・モードの応答メッセージを要求するには、以下のコマンドを使用します。
	`MESSAGE msgname FORWARD`
	`TIMEOUT seconds PESSIMISTIC`
	レイアウト用に構文が 2 行に配置されています。 コマンドは 1 行に入力します。

solidDB では、行レベル・ロック方式を使用して、表のペシミスティック・ロックを行うように定義することもできます。表に対して多数の更新が競合すると予想される場合に、この方法は有用です。	ペシミスティック・ロック方式を使用するように表を設定するには、以下のコマンドを使用します。 `ALTER TABLE tablename SET PESSIMISTIC`

ユーザー・アクセス要件の決定

データベース・スキーマを作成後、以下を決定する必要があります。

- 各レプリカ・データベースで、パブリケーション内の特定の表の使用許可、および実行しなければならないプロシージャーの実行権限が必要なローカル・ユーザー。
- マスター・データベースおよびレプリカ・データベースの両方で同期データの管理許可が必要なマスター・ユーザー。例えば、パブリケーションを定義するマスター・ユーザーは、そのパブリケーションで参照する表のアクセス権限が必要です。これによって、パブリケーションをドロップすることもできます。
- 管理タスク用に特に作成された SYS_SYNC_ADMIN ロールで同期化操作を実行するための権限が必要な、システムのマスター・ユーザー。
- SYS_SYNC_REGISTER_ROLE で同期のためにレプリカ・データベースを登録するための権限が必要な、マスター・ユーザーまたはローカル・ユーザー。

同期アクセス権限のインプリメントについて詳しくは、 85 ページの『アクセス権限およびロールによるセキュリティーのインプリメント』を参照してください。

フォールト・トレランスのためのバックアップの作成

拡張レプリケーション・システムの重要なコンポーネントに、マスター・データベースをホストするサーバーがあります。マスター・データベースとそのスキーマを作成した後、テープや CD-ROM などの信頼性の高いストレージ・メディアを使用して、複数のバックアップ・バージョンを使用できるようにしてください。標準データベースに対して行う通常のバックアップ手順の作業を、solidDB データベースにも適用できます。詳しくは、 155 ページの『バックアップおよびリカバリーの実行』を参照してください。

さらに、ご使用のシステム内で RAID ディスクとハードウェア・クラスタリングを使用することにより、マスター・データベースをフォールト・トレラントにすることができます。

マスター・データベースから新しいレプリカにデータをリフレッシュすることにより、マスター・データベースからレプリカ・データベースを再構成することができます。災害状態からの高速リカバリーを保証するため、大規模レプリカは、通常の solidDB バックアップ手順を使用して別途バックアップしてください。

拡張レプリケーション・システムが完全にインプリメントされており、トランザクションとメッセージを送受信している場合、システムに対する定期的なバックアップを確実に行ってください。 156 ページの『マスター・データベースおよびレプリカ・データベースのバックアップおよびリストア』を参照してください。

同期のためのアプリケーションの設計

solidDB 拡張レプリケーションが従来のデータ・レプリケーション・ソリューションと異なる点は、アプリケーション内でのデータ同期機能を構築するという原則にあります。以下のセクションでは、クライアント・アプリケーションの機能に関するいくつかの考慮事項について説明します。

ユーザー・インターフェースでの「一時的なデータ」状況の提供

レプリカ・データが一時的なものであるという性質は、実際には、レプリカでコミットされたトランザクションがマスター・データベースでは変更される可能性があることを意味しています。これは、場合によってはアプリケーション自体に明らかな影響を及ぼします。例えば、受注システムでは、オーダーの状況が最初、「一時的に OK」となることがあり、これはそのオーダーがレプリカでは受け入れられているが、マスターではまだ受け入れられていないことを意味します。このような一時的なデータであるという状況をユーザーにも示すことが適切である場合があります。

同期を管理するユーザー・インターフェースの提供

スタンドアロン・レプリカ・データベースで、データベースのデータ内容が変更されることがあります。ユーザーの要求に基づいて、ローカル・データベースにデータをダウンロードしたり、ローカル・データベースからデータを削除したりできま

す。例えば、ある販売担当者が、今日は西部地域の顧客情報を必要とし、翌日は東部地域の顧客情報を必要とすることがあります。動的にレプリカ・データベースにデータを設定できるようにするには、新しい (および既存の) データのリフレッシュを行うほか、対応するサブスクリプションのドロップにより不要なローカル・データの削除を行うユーザー・インターフェースが必要になることがあります。

同期化処理の管理

solidDB 拡張レプリケーション・アーキテクチャーの同期化処理は、アプリケーション・レベルでインプリメントする必要があります。同期化処理の管理内容には、以下のタスクが含まれます。

- 同期化処理の内容を定義する。つまり、単一の同期メッセージにつき、どのトランザクションをマスターに伝搬するのか、およびどのパブリケーションをレプリカにリフレッシュするのかを定義します。
- 処理を実行する。つまり、要求メッセージを送信し、戻りの応答メッセージを取得します。アプリケーションのニーズによって、実行する処理ステップは、1 回の「同期」操作になる場合も、2 回の「非同期」操作になる場合もあります。
- 処理の状況をモニターする。
- 同期中に発生したシステム・レベルおよびアプリケーション・レベルのエラーを解決する。

このようなタスクは、ユーザー・インターフェース、あるいは自動処理で設計できます。例えば、タスクのモニターと実行を手動で行うためのユーザー・インターフェースをアプリケーションにインプリメントできます。また、ユーザーの対話が何も必要ないように、アプリケーション内部にタスクを完全に自動化することもできます。

同期中に、エラーが発生することがあります。エラーはアプリケーション・レベルでもシステム・レベルでも発生する可能性があります。

アプリケーション・レベルのエラー

これは、トランザクションの妥当性検査ロジックがエラーを検出し、状態の修正に手動アクションが必要な場合に発生するエラーです。例えば、受注アプリケーションの「オーダー挿入」トランザクションが、マスター・データベースで顧客のクレジット制限の超過を検出した場合、そのオーダーを手動で承認する必要があります。エラーを追跡し解決するために、通常、クライアント・アプリケーションから参照可能なアプリケーション・レベルのエラー・ログ表が必要です。

システム・レベルのエラー

このエラーの原因は、通常、ストア・アンド・フォワード・メッセージングの障害です。例えば、同期を試みる間に、ネットワークがダウンした場合などです。この

ため、同期化処理の実行中に、適切なエラー処理をインプリメントすることが重要です。同期化処理のモニターと管理に必要な情報は、solidDB のシステム表で参照できます。

アプリケーションのニーズに基づいたインテリジェント・トランザクションの提供

solidDB インテリジェント・トランザクションは、従来のトランザクション・モデルの拡張機能です。開発者は、現行のデータベースの中でトランザクション自身を検証し、必要に応じてその内容を、トランザクションのルールに合わせて変更する機能を持つトランザクションをインプリメントすることができます。

ユーザーは、レプリカ・データベースでトランザクションを作成します。これらのトランザクションは、データの「正式な」バージョンが含まれているマスター・データベースにまだコミットされていないため、一時的なものです。レプリカ・トランザクションは、後でマスター・データベースに伝搬するために保存されます。このモデルでは、トランザクションの存続時間は長く、データ項目の複数のインスタンスがシステムの個別のデータベースに存在することがあります。

レプリカ・トランザクションがマスター・データベースに伝搬されるときに、更新の競合など、トランザクション妥当性検査エラーが発生することがあります。トランザクションは、アプリケーションに要求されるビジネス・ルールに適合するように、応答する必要があります。これは、データベースの整合性と信頼性を保証する最善の方法です。

開発者は、必要なビジネス・ルールを評価し、拡張レプリケーションの使いやすいモデルに基づいてトランザクションを構築する必要があります。solidDB インテリジェント・トランザクションを使用したトランザクションの作成について詳しくは、107 ページの『インテリジェント・トランザクションの設計およびインプリメント』 を参照してください。

5 solidDB 拡張レプリケーション・アプリケーションのインプリメント

この章では、同期のインプリメントに必要な基本タスクについて説明します。これらのタスクについては、 29 ページの『3 章 データ同期の概要』で簡単に紹介していますが、この章ではさらに詳しく説明します。また、solidDB 拡張レプリケーションの同期ステートメントについての紹介もあります。これらのステートメントは SQL の拡張機能で、このステートメントを使用して、データの同期化用に拡張レプリケーションのインストール済み環境をセットアップし定義することができます。

この章のトピックは以下のとおりです。

- 拡張レプリケーション・ステートメントの使用
- 同期メッセージのインプリメント
- 同期アクセスの権限とロールのインプリメント
- 同期化用データベースの指定
- パブリケーションとサブスクリプションの作成
- solidDB インテリジェント・トランザクションのインプリメント

拡張レプリケーションのデータ同期ステートメントの使用

拡張レプリケーション同期ステートメントは、solidDB SQL の拡張機能です。これらを使用すると、同期化操作を指定して、分散データを管理することができます。拡張レプリケーション・ステートメントは、solidDB SQL エディター (テレタイプ) を使用して実行できます。詳しくは、「*solidDB SQL ガイド*」を参照してください。

拡張レプリケーション・ステートメントのタイプ

拡張レプリケーションのデータ同期ステートメントによって、レプリカ・データベースの登録、アクセス権限のインプリメント、パブリケーションの作成などのタスクを実行できます。さらに、これらのステートメントの多くを使用して、拡張レプリケーション機能を管理します。例えば、MESSAGE コマンドを使用して同期メッセージの作成と管理を行い、DROP コマンドを使用してパブリケーションやサブスクリプションなどの同期オブジェクトを除去します。

拡張レプリケーション・ステートメントは以下のカテゴリーにグループ化されており、その使用法をこの章で説明します。拡張レプリケーション操作で使用されるものを含む、全 SQL ステートメントのアルファベット順リストについては、「*solidDB SQL ガイド*」を参照してください。

データベース構成ステートメント

以下のステートメントは、拡張レプリケーション・システムで使用するデータベースのセットアップおよび構成に使用します。

```
DROP MASTER
DROP REPLICA
REGISTER REPLICA
SET SYNC master_or_replica
SET SYNC master_or_replica
SET SYNC CONNECT
SET SYNC NODE
SET SYNC PARAMETER
SET SYNC USER
```

セキュリティー・ステートメント

以下のステートメントは、複数層環境およびマルチマスター環境のセキュリティーをセットアップするために使用されます。

```
ALTER USER SET MASTER
ALTER USER SET PUBLIC | PRIVATE
```

パブリケーション・ステートメント

以下のステートメントは、パブリケーションの作成および保守、およびパブリケーションからのデータのリフレッシュに使用されます。

```
ALTER TABLE SET { SYNCHISTORY | NOSYNCHISTORY }
CREATE SYNC BOOKMARK
CREATE PUBLICATION
DROP PUBLICATION
DROP PUBLICATION REGISTRATION
DROP SUBSCRIPTION
DROP SYNC BOOKMARK
EXPORT SUBSCRIPTION
GRANT REFRESH ON
IMPORT
MESSAGE APPEND REGISTER | UNREGISTER PUBLICATION
MESSAGE APPEND REFRESH
REVOKE REFRESH ON
REFRESH
```

インテリジェント・トランザクション制御

インテリジェント・トランザクションの実行制御に使用する機能およびステートメントを以下に示します。

- マスターに伝搬するトランザクションをレプリカ・データベースに保存する
- 掲示板パラメーター (特定メッセージのトランザクションのうちどれを伝搬するかを制御するためなど) を定義する
-

トランザクション内のパラメーター掲示板にパラメーターを設定する

- パラメーター掲示板からパラメーターを読み取る

```
GET_PARAM( )
PUT_PARAM( )
SAVE
SAVE PROPERTY
```

MESSAGE ステートメント

MESSAGE ステートメントは、レプリカおよびマスター・データベースの間で送信される同期メッセージを作成および実行するために使用します。

```
MESSAGE APPEND PROPAGATE TRANSACTIONS
MESSAGE APPEND REFRESH
MESSAGE APPEND REGISTER | UNREGISTER PUBLICATION
MESSAGE APPEND REGISTER | UNREGISTER REPLICA
MESSAGE APPEND SYNC_CONFIG
MESSAGE BEGIN
MESSAGE DELETE
MESSAGE DELETE CURRENT TRANSACTION
MESSAGE END
MESSAGE EXECUTE
MESSAGE FORWARD
MESSAGE FROM REPLICA EXECUTE
MESSAGE GET REPLY
```

同期用メッセージの構築

2 つの solidDB サーバー間のデータ同期は、solidDB 拡張レプリケーション・メッセージング・アーキテクチャーに依存します。これは、solidDB の内部に構築されるストア・アンド・フォワード・メッセージング・アーキテクチャーです。レプリカ・データベースとマスター・データベースの間で、確実なメッセージ転送が可能です。

拡張レプリケーション・アーキテクチャーの同期化処理は、2 つの異なるタスクで構成されています。

- マスター・データベースへのトランザクションの伝搬
- レプリカへのパブリケーションのリフレッシュ

これらのタスク (伝搬コマンド、リフレッシュ、またはその両方を含む) の組み合わせが、同期メッセージでグループ化されます。特定の表を参照するトランザクションは、常に、その表のデータをリフレッシュする前に、マスターに伝搬される必要があることに注意してください。下の例で示すように、これらのアクションの両方を同じ同期メッセージに入れることができます。

同期メッセージング

```
MESSAGE my_msg BEGIN ;
MESSAGE my_msg APPEND PROPAGATE TRANSACTIONS ;
```

```
MESSAGE my_msg APPEND REFRESH ORDERS_BY_SALESPERSON ('1') ;
MESSAGE my_msg APPEND REFRESH PARTS_IN_INVENTORY ;
MESSAGE my_msg END ;
COMMIT WORK ;
```

メッセージの開始

レプリカ・データベースからマスター・データベースに送信される同期メッセージは、それぞれ明示的に、MESSAGE BEGIN ステートメントで開始する必要があります。構文は、以下のとおりです。

```
MESSAGE unique_message_name BEGIN [TO master_node_name]
```

メッセージには、レプリカ・データベース内でユニークな名前を付けます。自動コミットはオフに設定してください。このステートメントを実行する前に、接続にオープン・トランザクションがあれば、それをコミットまたはロールバックする必要があります。

注:

REGISTER REPLICA コマンドを含むメッセージを送信して、デフォルト・カタログ以外のデータベース・カタログにあるマスターにレプリカを登録する場合は、オプションの TO 節を使用します。登録について詳しくは、98 ページの『マスター・データベースへのレプリカの登録』を参照してください。

ストアード・プロシージャーから同期メッセージを作成および実行する場合について、ユニークなメッセージ名で同期メッセージを作成する方法の例を以下に示します。

```
DECLARE Autoname VARCHAR;
DECLARE MsgBeginStr VARCHAR;
Autoname := GET_UNIQUE_STRING('MSG_') ;
MsgBeginStr := 'MESSAGE ' + autoname + ' BEGIN';
```

SQL ステートメントをストリングとして構成した後、2 つの方法のいずれかで、ストアード・プロシージャー内で実行できます。1 つは EXECDIRECT 機能を使用する方法で、もう 1 つは SQL ステートメントを準備して実行する方法です。

```
EXEC SQL EXECDIRECT MsgBeginStr;
```

または

```
EXEC SQL PREPARE cursor1 MsgBeginStr;
EXEC SQL EXECUTE cursor1;
EXEC SQL CLOSE cursor1;
EXEC SQL DROP cursor1;
```

レプリカからマスターへのトランザクションの伝搬

MESSAGE APPEND PROPAGATE TRANSACTIONS ステートメントを使用して、レプリカ・データベースからマスター・データベースにトランザクションを伝搬できます。SAVE <sql-statement> ステートメントで明示的にレプリカ・データベースのトランザクション・キューに入れられたステートメントだけが、伝搬されます。構文は、以下のとおりです。

```
MESSAGE unique_message_name APPEND
   [PROPAGATE TRANSACTIONS [WHERE {property_name {=|<|<=|>|>=|<>}
   'value_string' | ALL}]]
```

property_name が特定の基準と一致するトランザクションだけを伝搬するには、WHERE 節を使用します。 SAVE PROPERTY ステートメントを使用して、レプリカ・データベースで現在アクティブなトランザクションにプロパティーを設定できます。プロパティーがないステートメントを含めて、すべてのステートメントを伝搬するには、キーワード ALL を使用します。

キーワード ALL は、SAVE DEFAULT PROPAGATE PROPERTY WHERE コマンドで既に設定されている可能性があるデフォルトの伝搬条件をオーバーライドします。このコマンドは、パラメーター掲示板で、トランザクションの別のステートメントがパラメーターを使用できるようにするために使用します。

マスターからレプリカへのパブリケーション・データのリフレッシュ

MESSAGE APPEND REFRESH ステートメントを使用して、マスター・データベースにあるパブリケーションからリフレッシュできます。構文は、以下のとおりです。

```
MESSAGE unique_message_name APPEND
   [REFRESH publication_name [(publication_parameters)] [FULL]]
```

このステートメントを使用して、パブリケーションにパラメーターを提供し (パブリケーション定義でパラメーターを使用している場合)、レプリカのリフレッシュ範囲を狭めることができます。例えば、特定の支社に関連するデータだけをリフレッシュするように指定できます。

メッセージの終了

MESSAGE END ステートメントを使用して、各同期メッセージを明示的に終了させることを推奨します。このステートメントをトランザクション・コミット操作と併用すると、レプリカ・データベースの同期システム表にメッセージを保存することにより、レプリカ内でメッセージが永続的なものになります。メッセージを終了させるための構文は以下のとおりです。

```
MESSAGE unique_messsage_name END
```

メッセージの終了後、COMMIT WORK ステートメントを指定して、メッセージを必ずコミットする必要があることに注意してください。

マスター・データベースへのメッセージの転送

MESSAGE END ステートメントでメッセージを終了し、コミットして永続的にした後、MESSAGE FORWARD ステートメントを使用して、そのメッセージをマスター・データベースに送信します。送信された各メッセージに対して、マスター・データベースから応答メッセージが発行され、各メッセージは、クライアント・アプリケーションを介してフェッチする必要がある結果セットを戻します。構文は、以下のとおりです。

```
MESSAGE unique_message_name FORWARD
   [TIMEOUT {FOREVER | seconds }]
```

例を以下に示します。

```
MESSAGE mymsg FORWARD TIMEOUT 60;
```

TIMEOUT オプションを設定して、レプリカ・データベースでの応答メッセージの待機時間を定義できます。この時間を過ぎると、応答メッセージが期限切れになり、以下のセクションで説明する MESSAGE GET REPLYステートメントを使用して、応答メッセージを要求する必要があります。

注:

マスターは、完全なメッセージを受信しなかった場合は、受信したメッセージ部分を実行しません。

マスター・データベースからの応答メッセージの要求

MESSAGE FORWARD ステートメントで TIMEOUT が定義されていない場合、メッセージがマスターに転送されるだけで、レプリカは応答を待機しません。この場合、レプリカ・データベースで、別の MESSAGE GET REPLY 呼び出しを使用して応答をリトリーブできます。構文は、以下のとおりです。

```
MESSAGE unique_message_name GET REPLY
   [TIMEOUT {FOREVER | seconds }]
```

拡張レプリケーション・メッセージの構成

同期化処理の内容は、アプリケーション設計者が完全に定義できます。このようにして、アプリケーションのニーズを最適に考慮します。同様に、同期化処理を調整することにより、現在使用可能なインフラストラクチャーの容量と特性を効果的に使用できます。solidDB 拡張レプリケーション・アーキテクチャー自体はデフォルト処理を提供しませんが、カスタム・ビルド処理の内容に対して制限を設定することもありません。

最大メッセージ・サイズの設定

単一同期メッセージの最大サイズは、データベース・レベルのシステム・パラメーターで設定できます。SYS_R_MAXBYTES_OUT パラメーターは、レプリカ・データベースからマスター・データベースに送信するメッセージの最大長を設定し、SYS_R_MAXBYTES_IN パラメーターは、レプリカ・データベースで受信できるマスター・データベースからのメッセージの最大長を設定します。

両方のパラメーターのデフォルト・メッセージ長は 2 GB です。両方のパラメーターで有効な値は、0 GB から 2 GB です。0 を指定すると、2 GB が使用されます。

これらのパラメーターを設定するには、レプリカ・データベース内で SET SYNC PARAMETER ステートメントを使用してください。構文は、以下のとおりです。

```
SET SYNC PARAMETER parameter_name value_as_string
```

以下に例を示します。

```
SET SYNC PARAMETER SYS_R_MAXBYTES_OUT '1048576000';
```

両方のパラメーターに関して、メッセージが想定よりも長いとエラー・メッセージが発行されることに注意してください。

コミット・ブロック・サイズの設定

デフォルトでは、パブリケーション・リフレッシュのすべてのデータは、単一トランザクションでレプリカ・データベースに書き込まれます。送信されたメッセージの応答に大きなパブリケーションのリフレッシュが含まれる場合、MESSAGE FORWARD ステートメントまたは MESSAGE GET REPLY ステートメントの COMMITBLOCK オプションを使用して、1 つのトランザクション内でコミットされる行の数を調整できます。これにより、単一の大きなトランザクションを、より小さな複数のトランザクションに分割できます。COMMITBLOCK は、以下の構文で使用します。

```
MESSAGE unique_message_name FORWARD
[COMMITBLOCK block_size_in_rows]
```

または

```
MESSAGE unique_message_name GET REPLY
[COMMITBLOCK block_size_in_rows]
```

以下に例を示します。

```
MESSAGE mymsg FORWARD TIMEOUT 300 COMMITBLOCK 1000
MESSAGE mymsg GET REPLY TIMEOUT 300 COMMITBLOCK 1000
```

コミット･ブロックの最大サイズを設定すると、レプリカ・データベースのパフォーマンスが向上します。ただし、複数のトランザクションでデータを送信し、レプリカ上のアクティブ・ユーザーが同時にデータを変更する場合は、データ保全性を保証することはできません。したがって、COMMITBLOCK オプションを使用するときには、レプリカ・データベースからすべてのオンライン・ユーザーを切断することを推奨します。

注:

レプリカが HSB 構成で実行されている場合、COMMITBLOCK 節は、MESSAGE GET REPLY、DROP SUBSCRIPTION、MESSAGE FORWARD、および MESSAGE GET REPLY の各コマンドでは正しくありません。COMMITBLOCK 節を使用すると、「25083 Commit block can not be used with HotStandby」というエラーが発生します。

同期化処理の実行

同期化処理は通常、レプリカ・データベースによって開始され、ほとんどの制御もレプリカ・データベースから行います。マスターから同期化処理を開始する必要がある場合は、プル同期通知機能を使用できます。この機能によってマスターは、同期化処理を開始するタイミングをレプリカに通知できます。プル同期通知について詳しくは、43 ページの『プル同期通知』を参照してください。

同期化処理の作成と実行は以下のパターンで行われます。

1.

   ユニークな名前を付けてメッセージを作成します。例えば、以下のようにします。

   ```
   MSGNAME := GET_UNIQUE_STRING('MSG');
   ```

   自動コミット・モードは、オフにする必要がある ことに注意してください。

2.

同期化タスク (トランザクションの伝搬とパブリケーションからのリフレッシュ) をメッセージに付加します。メッセージには任意の数のタスクを含めることができます。

3.

メッセージを終了し、トランザクションをコミットしてそのメッセージを永続的なものにします。これ以降、ストア・アンド・フォワード・メッセージング・アーキテクチャーによって、メッセージに含まれるデータが失われないようになります。

4.

マスター・データベースにメッセージを転送 (送信) します。

転送コマンドの一部として、応答メッセージを受信できます。これは、例えば 1 分以内など、妥当な時間内に応答メッセージがあると予想される場合に有用な方法です。あるいは、例えば翌日の朝など、応答がかなり後になると予想される場合は、個別の GET REPLY コマンドを使用して応答を要求することができます。

典型的な同期化処理の実行

```
-- 'my_msg' という名前でメッセージを作成します。
-- AUTOCOMMIT はオフにする必要がある。また、それより前のあらゆるトランザクションは、
-- MESSAGE ステートメントの実行前に完了しておく必要がある。
MESSAGE my_msg BEGIN ;
-- メッセージにタスクを付加する (トランザクションの伝搬)。
MESSAGE my_msg APPEND PROPAGATE TRANSACTIONS ;
-- メッセージにタスクを付加する (パブリケーションからのリフレッシュ)。
MESSAGE my_msg APPEND REFRESH ORDERS_BY_SALESPERSON ('1') ;
MESSAGE my_msg APPEND REFRESH PARTS_IN_INVENTORY ;
-- メッセージを終了し、永続的なものにする。
MESSAGE my_msg END ;
-- メッセージ作成操作をコミットする。
COMMIT WORK ;
-- メッセージをマスターに送信し、応答は待機しない。
MESSAGE my_msg FORWARD ;
-- メッセージに対する応答を個別にマスターに要求する。
-- 最大 100 秒間、応答メッセージを待機する。
MESSAGE my_msg GET REPLY TIMEOUT 100 ;
COMMIT WORK ;
```

同期リフレッシュの使用

非同期データ・リフレッシュは大量のメモリーを消費し、ディスク I/O のオーバーヘッドをもたらす可能性があります。これを回避するため、同期式のメッセージレス拡張レプリケーション・インターフェースを使用することができます。このモードでは、関連するデータがデータ・ストリームとして送信されるので、メモリーが節約されます。またこのモードでは、ディスクへのメッセージの書き込みがないので、必要なディスク I/O 帯域幅が減少します。

ヒント:

また、ReplicaRefreshLoad パラメーターを使用して、リフレッシュ操作に使用可能なリソースを制限することもできます。

非同期データ・リフレッシュと組み合わせて、レプリカ表をどのようにロックするかのモードを定義できます。 以下のオプションがあります。

- OPTIMISTIC モード (デフォルト値) は、並行性制御方法が、表タイプおよび分離レベルによって決まることを定義します。OPTIMISTIC モードのディスク・ベース表では、REFRESH は常に正常に実行されます。一般的なインメモリー表と PESSIMISTIC モードのディスク・ベース表では、行レベル・ロック方式が使用されます。ロックがかけられない場合は、PESSIMISTIC が失敗し、エラーを返します。
- PESSIMISTIC は、選択した表タイプおよび分離レベルに関係なく、リフレッシュの際に表が排他的にロックされることを定義します。ロックがかけられない場合は、リフレッシュ要求が失敗し、エラーを返します。

デフォルトでは、パブリケーション REFRESH のすべてのデータは、単一トランザクションでレプリカ・データベースに書き込まれます。REFRESH 要求への応答に大規模なパブリケーションの REFRESH が含まれている場合、REFRESH のコミット・ブロックのサイズ、すなわち 1 つのトランザクションでコミットされる行の数は、COMMITBLOCK プロパティーで定義できます。

コミット・ブロックの最大サイズを設定すると、レプリカ・データベースのパフォーマンスが向上します。ただし、複数のトランザクションでデータを送信し、レプリカ上のアクティブ・ユーザーが同時にデータを変更する場合は、データ保全性を保証することはできません。したがって、COMMITBLOCK オプションを使用するときには、レプリカ・データベースからすべてのオンライン・ユーザーを切断することを推奨します。

TIMEOUT プロパティーは、レプリカ・サーバーが応答メッセージを待機する時間を定義します。TIMEOUT が定義されていない場合は、FOREVER が使用されます。

同期リフレッシュの例については、以下を参照してください。

同期リフレッシュ

```
REFRESH my_table;
COMMITBLOBK 1000;
COMMIT WORK;
```

同期リフレッシュの構文について詳しくは、「*solidDB SQL ガイド*」の付録 *B* の『*solidDB SQL 構文*』の REFRESH コマンドの説明を参照してください。

アクセス権限およびロールによるセキュリティーのインプリメント

solidDB 拡張レプリケーションでは、アクセス権限とロールのインプリメントにより、システム全体のセキュリティーが強化されます。このセクションでは、拡張レプリケーションのセキュリティーの基本的な原理と、セキュリティーをセットアップするために拡張レプリケーション・コマンドを使用する方法について説明します。

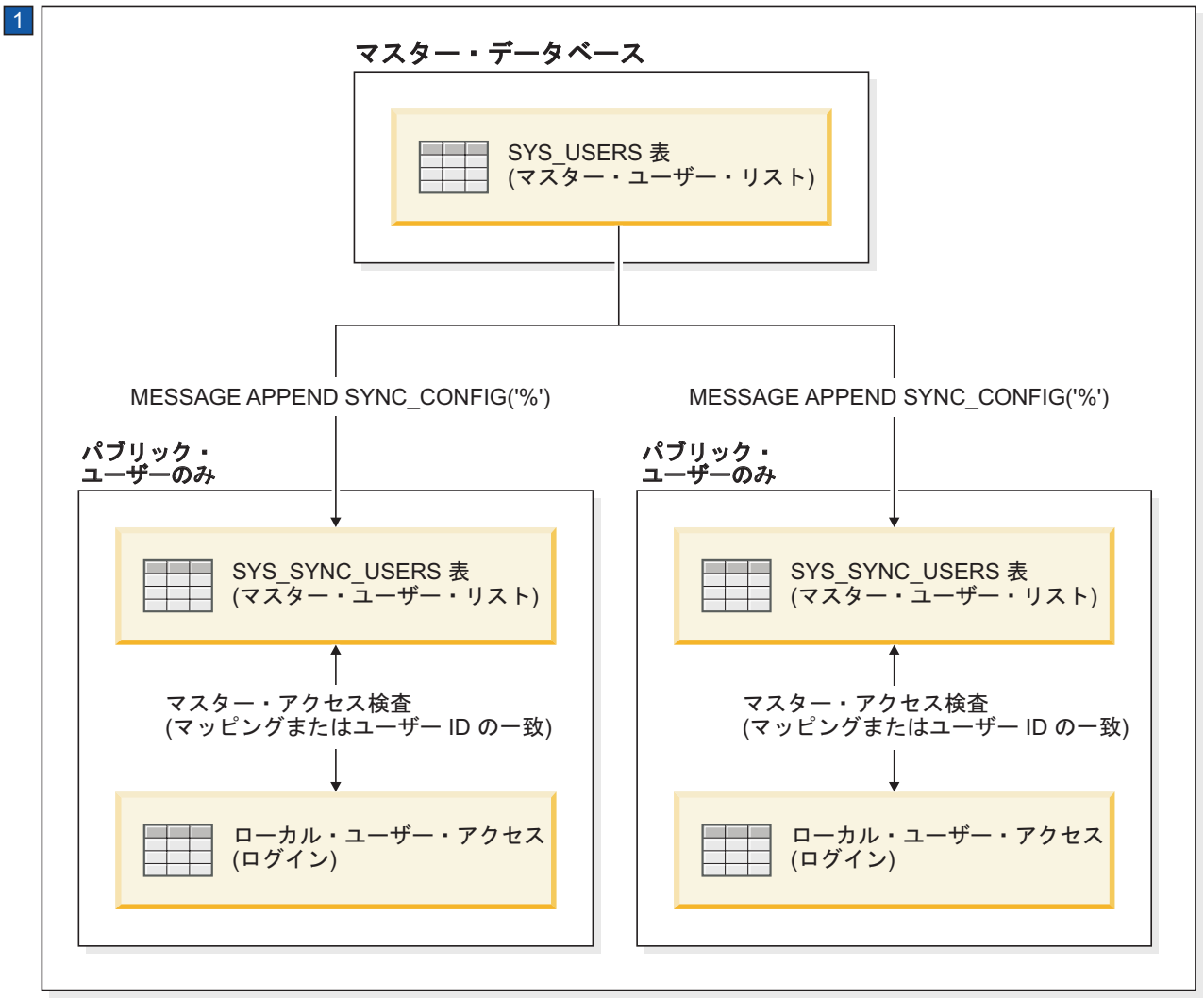
拡張レプリケーション・セキュリティーの仕組み

solidDB 拡張レプリケーション・セキュリティー・モデルは、以下の原理に基づいています。

- ユーザーには、ローカル・ユーザーとマスター・ユーザーの 2 種類があります。
- ローカル・ユーザーは、レプリカ・データベースへのアクセス権限を持っています。
- マスター・ユーザーは、マスター・データベースへのアクセス権限を持っています。
- ローカル・ユーザーが同期化関連のタスクを実行できるようにするには、ローカル・ユーザーをマスター・データベースで対応するマスター・ユーザーにマップする必要があります。
- マスター・ユーザーのアクセス権限は、マスター・データベースで同期メッセージを実行する際に使用します。
- ローカル・ユーザーとマスター・ユーザーの両方が、拡張レプリケーション・システムのレプリカ・データベース内に存在しています。
- ローカル・ユーザーは、そのローカル・ユーザーに対して定義されたアクセス権限に基づき、照会の実行、表の作成、ストアード・プロシージャーの呼び出しなどのローカル・データベース操作を行うことができます。例えば、ローカル・データベースの管理者は、ローカル・データベースに対するあらゆる操作を実行できます。しかし、ローカル・ユーザーの場合は、SAVE *sql_statement* または MESSAGE ステートメントなど、同期化関連のステートメントにはアクセス権限を持ちません。ローカル・ユーザーがトランザクションをマスターに伝搬できるようにするには、ローカル・ユーザーをマスター・ユーザーにマップする必要があります。
- マスター・ユーザーは、マスター・データベースで定義され、レプリカ登録プロセスの一部としてレプリカ・データベースにダウンロードされているユーザーです。すべての同期化操作では、ローカル・ユーザー ID をマスター・ユーザー ID にマップすることで、現行のマスター・ユーザーを定義する必要があります。

  マスター・ユーザーの名前とパスワードを、各レプリカ・データベースの SYS_SYNC_USERS 表で個別に定義することで (以下のセクションで説明)、マスター・ユーザーに、許可を得ている表にトランザクションを保存する権限を付与します。ユーザー・アクセスは、同期化中にマスター・データベースでも検証されます。
- マスターとローカルの両方のユーザーが、レプリカ登録が可能なロールや、同期機能の管理ロールなどの同期化固有のロールを持つことができます。

各コマンドに必要なアクセス権限の概要については、 94 ページの『アクセス権限の要約』を参照してください。



1. ローカル・ユーザーは、マスター・アクセス権限を持たない場合、同期操作を実行できません。

図 11. 拡張レプリケーションのユーザー・アクセス権限

マスター・データベースに対するレプリカのアクセス権限の変更

任意のサーバーでトランザクションを実行するとき、そのトランザクションは、適切な特権 (INSERT、DELETE、UPDATE などの表に対する特権) で実行する必要があります。トランザクションがレプリカ・サーバーからマスター・サーバーに伝搬されるとき、そのトランザクションは、マスターで適切な特権で実行する必要があります。(レプリカで適切な特権で実行しても、そのトランザクションがマスターで適切な特権で実行されるとは限りません。)

伝搬トランザクションをマスターで実行できるようにするには、レプリカ・ユーザーを、適切な特権を持つ対応するマスター・ユーザーにマップする必要があります。例えば、ユーザー kathy_on_master が適切な特権を持つ場合に、ユーザー kathy_on_replica1 をユーザー kathy_on_master にマップします。

マスター上のユーザーを変更するときは、レプリカ上の「マッピング」情報も更新する必要があります。更新しないと、適切でなくなったマスター・ユーザーまたは既に存在しないマスター・ユーザーにレプリカ・ユーザーがマップされることがあります。更新された情報を該当するレプリカにダウンロードするには、レプリカで MESSAGE APPEND SYNC_CONFIG コマンドを実行する必要があります。更新されたマスター・ユーザー情報をダウンロードしたら、ALTER USER SET MASTER ステートメントを使用して、レプリカ・ユーザー ID を適切なマスター・ユーザー ID にマップし直す必要があります (後述の『*レプリカ・ユーザー ID のマスター・ユーザー ID へのマッピング*』を参照してください)。

MESSAGE APPEND SYNC_CONFIG コマンド自体が、適切な特権を必要とします。新しいレプリカを作成すると、そのレプリカには特権がなく、そのため、マスターに接続できません。レプリカ登録ユーザーを作成して、最初にマスター・データベースから SYS_SYNC_USERS 表にマスター・ユーザーのリストを初期設定する必要があります。その後、必要に応じてマスター・ユーザーのリストをマスター・データベースからダウンロードできるようになります。

更新されたユーザー・データをマスターから取得するために必要な特権が既にあるレプリカのマッピングを更新する方法については、以下で詳しく説明します。

拡張レプリケーション操作のためのマスター・ユーザーの更新

レプリカでマスター・ユーザーを更新するには以下のようにします。

レプリカから、以下のコマンドを使用して、別のメッセージでマスター・データベースからユーザー情報をサブスクライブします。

```
MESSAGE unique_message_name APPEND SYNC_CONFIG
   (sync_config_arg)
```

sync_config_arg は、マスター・データベースからレプリカに返されるユーザー名の検索パターンを定義します。すべての名前をレプリカに送信する場合は、入力引数として SQL ワイルドカードである % を指定します。

以下に例を示します。

```
MESSAGE CFG2 BEGIN;
MESSAGE CFG2 APPEND SYNC_CONFIG('%');
MESSAGE CFG2 END;
COMMIT WORK;
```

マスター・ユーザーの管理

solidDB のデータ同期機能へのすべてのアクセスは、マスター・ユーザーが制御します。レプリカ・データベースのデータがマスター・データベースと同期可能にするには、レプリカがマスター・データベースからマスター・ユーザー情報をダウンロードし、1 つ以上のローカル・ユーザー ID をマスター・ユーザー ID にマップする必要があります。

レプリカ・ユーザー ID のマスター・ユーザー ID へのマッピング

レプリカ・ユーザー ID をマスター・ユーザー ID にマップするには、ALTER USER SET MASTER ステートメントを使用します。ALTER USER SET MASTER ステートメントを使用する場合には、マスター・ユーザーにマップするローカル・

ユーザー用にマスター・ユーザー名とパスワードを付与します。同じローカル・ユーザーを複数のマスター・ユーザーにマップすることもできます。

マッピングの完了後に、ローカル・ユーザーがレプリカ・データベースにログインしようとすると、solidDB は、そのローカル・ユーザーがマスター・ユーザー ID にマップされているかどうかを検査します。マッピングが指定されていない場合、デフォルトでは、solidDB はマスターとレプリカの両方に同じユーザー ID とパスワードがないか探します。つまり、マッピングを使用していない場合には、マスターとレプリカのユーザー ID とパスワードが同じである必要があります。

MESSAGE APPEND SYNC_CONFIG コマンドを使用すると、レプリカ表 SYS_SYNC_USERS を最新のマスター・ユーザー名で更新できます。この概念については、87 ページの図 11 を参照してください。

パブリック・ユーザーとプライベート・ユーザーの設定

データベース管理者は、ユーザーをプライベートまたはパブリックのいずれかに設定するために、マスター・データベースの SYS_USERS 表のユーザーを変更することができます。PRIVATE オプションをユーザーに設定した場合、このユーザーの ID とパスワードが、レプリカに対するパブリケーションのサブスクリプション中に、レプリカに送信されることはありません。

PRIVATE ユーザーが MESSAGE APPEND SYNC_CONFIG コマンドの指定されたサブスクリプション要求と一致したとしても、そのユーザーが PRIVATE に設定されている限り、ユーザーの情報はマスターの SYS_USERS 表の外には出ません。SYS_SYNC_USERS 表のサブスクリプション要求を実現するために、マスターからレプリカには、PUBLIC ユーザーのみがダウンロードされます。デフォルトでは、ユーザーは PUBLIC に設定されます。ユーザーのパブリックまたはプライベートへの設定について詳しくは、「*solidDB SQL ガイド*」を参照してください。

ユーザーをパブリックとプライベートの間で切り替えるには、以下のコマンドを使用します。

```
ALTER USER SET { PRIVATE | PUBLIC }
```

注:

同じユーザーを、あるレプリカではプライベートに設定し、他のレプリカではパブリックに設定することはできません。ユーザーは、スマート・フロー・システム全体でパブリックかプライベートのいずれかに指定されなくてはなりません。

管理権限 (DBA) を持つユーザーは全員、PRIVATE に設定することを推奨します。こうしておけば、DBA のパスワードはレプリカに送信されず、一般に知られる心配もないため、セキュリティーの追加対策となります。万が一、DBA のパスワードが公開された場合に、セキュリティーの回復が必要であれば、マスター・データベースでパスワードを変更した後で、システムのレプリカをすべてドロップしてから再作成しなくてはなりません。

また、レプリカに必要でないユーザーがあれば、そのユーザーも PRIVATE に設定しておいた方がよいと考えられます。

注:

マスター・データベースでプライベート・ユーザーとして設定されているユーザーを使用して、レプリカがメッセージを作成またはトランザクションを実行した場合、メッセージの受信時またはトランザクションの実行時に、マスター側での操作は失敗し、セキュリティー・エラーとなります。

トランザクションおよびリフレッシュ・コマンドのアクセス権限の決定

同期メッセージには、そのメッセージの作成者のマスター・ユーザー名でラベル付けされます。solidDB は、このマスター・ユーザー名を使用して、メッセージが実行されるアカウントを指定します。このアカウントを使用して、すべてのサブスクリプションが実行されます。各トランザクションでは、レプリカにステートメントを保存したマスター・ユーザーが使用されます。

アクセス権限の設定

以下のセクションでは、拡張レプリケーション・システムのインプリメントに必要なアクセス権限を示します。

アクセス権限の付与

ローカル・ユーザーは、トランザクションに使用する表に対して (マスター・データベースとユーザーのローカル・レプリカ・データベースの両方で) 適切なアクセス権限が必要であり、実行するプロシージャーに対して実行権限が必要です。プロシージャーを使用してレプリカ・データベースで同期機能を実行する場合は、プロシージャーを作成したローカル・ユーザーをマスター・ユーザーにマップする必要がある点に注意してください。

マスター・データベースの DBA は、マスター・ユーザーに対して以下の権限を付与する必要があります。

- パブリケーションやトランザクションに使用する表に対する、マスター・データベースでの適切なアクセス権限、および
- 実行するプロシージャーに対する実行権限

注:

アクセス権限が付与されると、それが付与されたユーザーがデータベースにログオンした時に、そのアクセス権限が有効になります。権限の付与時にユーザーが既にデータベースにログオンしている場合は、以下の場合にのみ権限が有効になります。

- 権限が設定されている表やオブジェクトにユーザーが初めてアクセスした場合、または
- ユーザーがデータベースから切断後、再接続した場合

該当するレプリカ・データベース内で、ALTER USER SET MASTER コマンドを使用して、レプリカ・ユーザー ID をマスター・ユーザー ID にマップすることにより、現在アクティブなマスター・ユーザーを指定します。

アクセス権限をセットアップする際、以下の拡張レプリケーション SQL コマンドを使用できます。

```
CREATE USER username IDENTIFIED BY password
GRANT rolename TO username
GRANT [SELECT | UPDATE | INSERT | DELETE] ON tablename TO username
GRANT EXECUTE ON procedure_name TO username
```

詳しくは、「*solidDB SQL ガイド*」を参照してください。

ユーザーにパブリケーションへのアクセス権限を付与するには、以下を使用します。

```
GRANT REFRESH ON publication_name TO username
```

詳しくは、『REFRESH アクセス権限の付与』を参照してください。

レプリカ・ユーザー ID をマスター ID にマップするには、以下を使用します。

```
ALTER USER replica_user SET MASTER master_name USER user_specification
```

詳しくは、「*solidDB SQL ガイド*」を参照してください。

REFRESH アクセス権限の付与

ローカル・ユーザー、ユーザー・ロール (create role ステートメントで作成)、または全ユーザーに対してパブリケーションへのアクセス権限を付与するには、マスター・データベースで GRANT REFRESH ステートメントを使用します。構文は、以下のとおりです。

```
GRANT REFRESH ON publication_name TO {PUBLIC | user_name,
   [, user_name] ... | role_name [, role_name ] ...
```

例を以下に示します。

```
GRANT REFRESH ON customers_by_area TO salesman_jones
```

REFRESH アクセス権限の取り消し

ローカル・ユーザー、ユーザー・ロール (create role ステートメントで作成)、または全ユーザーの、パブリケーションに対するアクセス権限を取り消すには、マスター・データベースで REVOKE REFRESH ステートメントを使用します。構文は、以下のとおりです。

```
REVOKE REFRESH ON publication_name FROM {PUBLIC | user_name,
   [user_name] ... | role_name [, role_name, [, role_name]...}
```

例を以下に示します。

```
REVOKE REFRESH ON customers_by_area FROM salesman_jones
```

レプリカでのトランザクションの保存

ローカル・ユーザーがレプリカでトランザクションのステートメントを保存するとき、ステートメントのトランザクションには現行のマスター・ユーザーのユーザー名でラベルが付けられます。トランザクションをマスター・データベースで再実行するときは、マスター・ユーザーに定義されているアクセス権限が使用されます。

マスターは、トランザクションの伝搬中にユーザー・アクセス権限違反を検出した場合、同期メッセージの実行を終了します。これにより、ローカル・レプリカ・ユーザーは、マスター・データベースで無許可のステートメントを実行できなくなります。

異なるマスター上のアプリケーションに対するアクセス権限の作成

マルチマスター環境では、単一ユーザー ID を、各カタログ内の別のマスター・ユーザーにマップできます。SET CATALOG コマンドを使用してアクティブなレプリカ・カタログを変更すると、現行マスター・ユーザーが自動的に変更されます。例えば、レプリカ・データベース内には、ローカル・ユーザーが 1 つ存在すると想定されています。このユーザーは、カレンダー・アプリケーションの CALENDARUSER マスター・ユーザー、およびニュース・アプリケーションの NEWSUSER マスター・ユーザーにマップされます。SET CATALOG コマンドを使用して、現行カタログを CALENDAR または NEWS のどちらかに設定します。CALENDAR カタログを設定すると、現行マスター・ユーザーは、自動的に CALENDARUSER マスター・ユーザーに設定されます。同様に、NEWS を設定すると、現行マスター・ユーザーは NEWSUSER に設定されます。

マッピングが定義されていない場合、最初の拡張レプリケーション・データ同期化操作 (例えば、SAVE ステートメントまたは MESSAGE ステートメント) で、「no active master user」エラーが戻されます。

パブリケーションおよび表に対するユーザー権限の作成

パブリケーションを定義するマスター・ユーザーには、そのパブリケーションが参照する表に対する読み取りアクセス権限と書き込みアクセス権限が必要です。

マスター・データベースでは、REFRESH *publication* 節を含むメッセージの作成者のマスター・ユーザー名を使用してサブスクリプションを実行することに注意してください。

同期に関係する表を使用するには、ローカル・ユーザーには、レプリカ・データベースの実際のサブスクリプション表に対する権限が必要になり、対応する (マップされている) マスター・ユーザーには、マスター・データベースのパブリケーションに対するサブスクライブのアクセス権限が必要になります。サブスクリプション行をレプリカに挿入 (またはレプリカから削除) するときには、solidDB によって、表に対する INSERT 権限および DELETE 権限がサブスクライバーにあるかどうか検証されます。

パブリケーションを定義するユーザーには、そのパブリケーションをドロップする権限もあります。

パブリケーションに対するアクセス権限の付与について詳しくは、90 ページの『アクセス権限の設定』を参照してください。

レプリカ登録ユーザーの作成

拡張レプリケーションのレプリカ・データベースを新しく作成する場合、新しいデータベースの SYS_SYNC_USERS 表は空です (何のデータも入っていません)。新し

いレプリカ・データベースをマスター・データベースに登録し、表に最初にデータを設定するには、登録権限を持つ、マスター・データベースのユーザー名が必要です。

マスター・データベースのマスター・ユーザーに登録権限を付与するには、GRANT *rolename* TO *user* コマンドを使用して、ユーザーを SYS_SYNC_REGISTER_ROLE または SYS_SYNC_ADMIN_ROLE によって指定します。

マスターに登録するレプリカ・サイトに、この登録ユーザー名とパスワードを指定する必要があります。こうすれば、レプリカ・サイトで、以下のコマンドを使用して、レプリカ側の登録ユーザーを明示的に設定できるようになります。

```
SET SYNC USER username IDENTIFIED BY password
```

ユーザー名はマスター・データベースに存在するため、登録ユーザーは、このコマンドを使用して明示的にレプリカを登録することができます。次に、レプリカ登録プロセスの一環として、マスター・データベースのパブリック SYS_USERS 情報を SYS_SYNC_USERS レプリカ表に設定することができます。

レプリカの登録を正常に実行したら、以下のステートメントを実行します。

```
SET SYNC USER NONE
```

このステートメントを実行しないと、SET SYNC USER *username* がアクティブであり、ユーザーがステートメントを保存するか、パブリケーションに伝搬、リフレッシュ、または登録を行った場合に、以下のエラー・メッセージが返されます。

```
User definition not allowed for this operation.
```

注:

SET SYNC USER コマンドの用途は、レプリカの登録に限定されています。登録以外の場合、すべての同期化操作には SYNC_CONFIG タスクを使用してレプリカ・データベースにダウンロードした有効なマスター・ユーザー ID が必要です。レプリカに対して異なるマスター・ユーザーを指定する場合には、レプリカ・データベースのレプリカ ID をマスター・データベースのマスター ID にマッピングする必要があります。詳しくは、この章の『レプリカ・ユーザー *ID* のマスター・ユーザー *ID* へのマッピング』を参照してください。

マスター・ユーザーがデータ同期機能のユーザー・アクセスを制御する方法について詳しくは、88 ページの『マスター・ユーザーの管理』を参照してください。

拡張レプリケーションの特殊なロールのインプリメント

solidDB には、同期化操作を実行するための以下の 2 つのロールがあります。

- SYS_SYNC_ADMIN_ROLE

  これは、メッセージの削除などの、拡張レプリケーションのデータ同期化操作を実行するための管理用のロールです。この特権を持つ人には、すべての同期ロールが自動的に付与されます。

-

SYS_SYNC_REGISTER_ROLE

これは、レプリカ・データベースをマスターに登録するか、マスターからレプリカ・データベースを登録抹消するためのみのロールです。

SYS_SYNC_ADMIN_ROLE には、自動的に SYS_SYNC_REGISTER_ROLE が含まれます。

これらのロールを付与するには、マスター・データベースで以下の構文の GRANT ステートメントを使用します。

```
GRANT role_name TO user_name
```

注:

いったん付与されたユーザー・ロールが有効になるのは、そのロールを付与されたユーザーがデータベースにログオンしたときです。既にユーザーがデータベースにログオンしていた場合にロールを有効にするには、そのユーザーはデータベースから切断して再接続する必要があります。

アクセス権限の要約

以下は、レプリカ・データベースおよびマスター・データベースで各拡張レプリケーション・コマンドを実行するために必要なアクセス権限の包括的な要約です。

レプリカでのアクセス権限

以下の表で、同期化操作をレプリカ・データベースで実行するために必要なアクセス権限をリストします。

表 6. レプリカでのアクセス権限

コマンド	タスク	アクセス権限の要件
ALTER TABLE SET SYNCHISTORY | NOSYNCHISTORY	表をインクリメンタル・パブリケーション用にセットアップするかどうかを指定する。	SQL ALTER TABLE コマンドと同じ (表の所有者または DBA)
ALTER USER SET MASTER	レプリカ・ユーザー ID をマスター・ユーザー ID にマップする。	SYS_SYNC_ADMIN_ROLE
GET_PARAM( )	PUT_PARAM( ) で掲示板に入れたパラメーターをリトリーブする。	任意のユーザー
PUT_PARAM( )	パラメーターを掲示板に入れる。	任意のユーザー
SAVE	後でマスターに伝搬させるために、トランザクションのステートメントをレプリカ・データベースに保存する。	有効なマスター・ユーザー
SAVE PROPERTY	プロパティーを現行のアクティブなトランザクションに割り当てる。	有効なマスター・ユーザー
MESSAGE BEGIN	新規同期メッセージを開始する。	有効なマスター・ユーザー、SYS_SYNC_ADMIN_ROLE、または SYS_SYNC_REGISTER_ROLE

表 6. レプリカでのアクセス権限 (続き)

コマンド	タスク	アクセス権限の要件
MESSAGE APPEND REFRESH	パブリケーションからリフレッシュする。	有効なマスター・ユーザー
MESSAGE APPEND PROPAGATE TRANSACTIONS	トランザクションを伝搬する。	有効なマスター・ユーザー
MESSAGE APPEND { REGISTER | UNREGISTER } REPLICA	マスター・データベースにレプリカを登録または登録抹消する。	SYS_SYNC_ADMIN_ROLE または SYS_SYNC_REGISTER_ROLE
MESSAGE APPEND{ REGISTER PUBLICATION | UNREGISTER PUBLICATION }	レプリカ内でパブリケーションを登録または登録抹消する。パブリケーションを登録すると、ユーザーがパブリケーションからリフレッシュできるようになります。	パブリケーションに対するリフレッシュ・アクセス権限
MESSAGE APPEND SYNC_CONFIG	SYS_SYNC_USERS 表のデータをレプリカにダウンロードする。	SYS_SYNC_ADMIN_ROLE または SYS_SYNC_REGISTER_ROLE
MESSAGE FORWARD	保存したメッセージをマスター・データベースに送信する。	有効なマスター・ユーザーまたは SYS_SYNC_ADMIN_ROLE
MESSAGE GET REPLY	送信したメッセージに対する応答を取得する。	有効なマスター・ユーザーまたは SYS_SYNC_ADMIN_ROLE
MESSAGE DELETE [FROM REPLICA]	エラーからリカバリーするために、メッセージ全体 (すべてのトランザクション) をレプリカ・データベースから削除する。	SYS_SYNC_ADMIN_ROLE
MESSAGE DELETE [FROM REPLICA] CURRENT TRANSACTION	エラーからリカバリーするために、同期メッセージから現行トランザクションを削除する。	SYS_SYNC_ADMIN_ROLE
DROP MASTER	マスター定義をドロップする。	SYS_SYNC_ADMIN_ROLE
DROP SUBSCRIPTION	レプリカ・データベースでサブスクリプションをドロップする。	有効なマスター・ユーザー
DROP PUBLICATION REGISTRATION	レプリカ・データベースでパブリケーション登録をドロップする。	SYS_SYNC_ADMIN_ROLE
IMPORT	マスター・データベースで EXPORT SUBSCRIPTION コマンドを使用して作成したデータ・ファイルからデータをインポートする。	有効なマスター・ユーザー
SET SYNC CONNECT *listen_name* TO MASTER *master_name*	マスター・データベースに関連付けられているネットワーク名を変更する。	SYS_SYNC_ADMIN_ROLE
SET SYNC NODE *node_name* | NONE	ノード名を登録の一部としてデータベースに割り当てるか、ノード名を削除する (登録を解除して同期データベースをドロップするときなど)。	SYS_SYNC_ADMIN_ROLE
SET SYNC PARAMETER	同期データベース・カタログで同期関連データベース・パラメーターを設定する。	SYS_SYNC_ADMIN_ROLE

表 6. レプリカでのアクセス権限 (続き)

コマンド	タスク	アクセス権限の要件
SET SYNC { REPLICA | MASTER } { YES | NO }	データベース・カタログをレプリカおよびマスター、またはそのいずれかとして指定する。	SYS_SYNC_ADMIN_ROLE
SET SYNC USER NONE	現行の登録ユーザーを現行のデータベース接続で非アクティブにする。	任意のローカル・ユーザー
SET SYNC USER *username* IDENTIFIED BY *password*	登録プロセスで使用する現在アクティブなマスター・ユーザー名とパスワードを定義する。	SYS_SYNC_ADMIN_ROLE

マスターでのアクセス権限

以下の表で、拡張レプリケーション・コマンドをマスター・データベースで実行するために必要なアクセス権限をリストします。

表 7. マスターでのアクセス権限

コマンド	タスク	アクセス権限の要件
ALTER TABLE SET SYNCHISTORY | NOSYNCHISTORY	表をインクリメンタル・パブリケーション用にセットアップするかどうかを指定する。	SQL ALTER TABLE コマンドと同じ (表の所有者または DBA)
ALTER USER SET { PUBLIC | PRIVATE }	レプリカの SYS_SYNC_USERS 表へのサブスクリプション・ダウンロードにユーザー ID を含めるか、このダウンロードからユーザー ID を除外する。	DBA または SYS_SYNC_ADMIN_ROLE
GET_PARAM( )	PUT_PARAM( ) で掲示板に入れたパラメーターをリトリーブする。	任意のユーザー
PUT_PARAM( )	パラメーターを掲示板に入れる。	任意のユーザー
CREATE PUBLICATION	マスター・データベースでパブリケーションを定義する。	パブリケーションの表に対して全アクセス権限を持つ有効なマスター・ユーザー
CREATE SYNC BOOKMARK	マスター・データベースでブックマークを作成する。	DBA または SYS_SYNC_ADMIN_ROLE
DROP SYNC BOOKMARK	マスター・データベースでブックマークをドロップする。	DBA または SYS_SYNC_ADMIN_ROLE
GRANT REFRESH ON	マスター・データベースで定義されているユーザーまたはロールに、パブリケーションに対するアクセス権限を付与する。	パブリケーションの作成者または DBA
REVOKE REFRESH ON	マスター・データベースで定義されているユーザーまたはロールから、パブリケーションに対するアクセス権限を取り消す。	パブリケーションの作成者または DBA
DROP PUBLICATION	マスター・データベースでパブリケーションをドロップする。	パブリケーションの作成者または DBA

表 7. マスターでのアクセス権限 (続き)

コマンド	タスク	アクセス権限の要件
EXPORT SUBSCRIPTION	マスター・データをファイルにエクスポートする。	パブリケーションへのサブスクライブ・アクセス権限を持つマスター・ユーザー
MESSAGE DELETE FROM REPLICA	エラーからリカバリーするために、同期メッセージ全体 (すべてのトランザクション) を削除する。	SYS_SYNC_ADMIN_ROLE または DBA
MESSAGE DELETE CURRENT TRANSACTION	エラーからリカバリーするために、同期メッセージの現行 (失敗) トランザクションを削除する。	SYS_SYNC_ADMIN_ROLE または DBA
MESSAGE FROM REPLICA EXECUTE	マスター・データベースにあるレプリカからの失敗したメッセージを実行する。	SYS_SYNC_ADMIN_ROLE または DBA
DROP SUBSCRIPTION REPLICA	マスターのパブリケーションに対するレプリカのサブスクリプションをドロップする。	SYS_SYNC_ADMIN_ROLE または DBA
DROP REPLICA	マスター・データベースからレプリカ・データベースをドロップする。	SYS_SYNC_ADMIN_ROLE または DBA
SET SYNC {MASTER | REPLICA}{YES | NO}	データベース・カタログをマスターおよびレプリカ、またはそのいずれかとして指定する。	SYS_SYNC_ADMIN_ROLE または DBA
SET SYNC USER NONE	現行のマスター・ユーザーを現行のデータベース接続で非アクティブにする。	SYS_SYNC_ADMIN_ROLE または DBA
SET SYNC PARAMETER	マスター・データベース・カタログで同期関連データベース・パラメーターを設定する。	有効なマスター・ユーザー
SET SYNC NODE { *node_name* | NONE}	ノード名を登録の一部としてマスター・データベースに割り当てるか、ノード名を削除する (登録を解除して同期データベースをドロップするときなど)。	SYS_SYNC_ADMIN_ROLE または DBA

同期のためのデータベースのセットアップ

データベースを (マスター、レプリカ、またはその両方として) 定義し、データベース・スキーマおよびカタログ (必要な場合) を作成し、ユーザー・アクセス権限をインプリメントすると、同期のためにデータベースを構成する準備が整います。このセクションでは、各データベース用にデータベース・ノード名を割り当てる必要があります。solidDB SQL エディター (solsql) を使用して、セットアップに必要な拡張レプリケーション・ステートメントを入力します。

マスター・データベース (複数可) の構成

処理を開始する前に、マスター・データベース (複数可) を定義したことを確認してください。詳しくは、57 ページの『マスター・データベースとレプリカ・データベースの定義』を参照してください。

各マスター・データベースに対して、ドメイン内でユニークなノード名を指定します。例えば、以下のように指定します。

```
SET SYNC NODE "MASTER";
COMMIT WORK;
```

マスター・データベースへのレプリカの登録

始める前に、マルチステートメント・メッセージを作成できるように、自動コミットをオフに設定するようにします。アクティブ・トランザクションは、すべてコミットまたはロールバックするようにします。また、レプリカをマスター・データベースに登録するためのマスター・ユーザー名とパスワードを定義し、ご使用の環境のカタログ名 (複数可) を確認しておくようにします。さらに、レプリカ・データベースを定義しておくようにします。詳しくは、 57 ページの『マスター・データベースとレプリカ・データベースの定義』を参照してください。

各レプリカ・データベースで、以下のステップを実行します。

1. ローカル・データベースが複数のマスター・データベースとデータの同期をとる場合 (ローカル・データベースに複数のレプリカがある場合)、レプリカごとにカタログを作成します。以下に例を示します。

   ```
   CREATE CATALOG CAT_FOR_REP1;
   COMMIT WORK;
   ```

2. このレプリカ・カタログに、このレプリカのマスター・データベースのレプリカ全体でユニークなノード名を付けます。カタログのノード名を設定する前に、このカタログを現行カタログに設定しておく必要があります。以下に例を示します。

   ```
   SET CATALOG CAT_FOR_REP1;
   COMMIT WORK;
   SET SYNC NODE "REPLICA1";
   COMMIT WORK;
   ```

   注:

   多くのレプリカがある場合は、例えばサーバーの論理名や場所から派生した論理名をレプリカに付けることを推奨します。

   また、マスター内のカタログとそれに対応するマスター内のノードには、異なる名前を付けることができる点に注意してください。

3. レプリカ登録用のマスター・ユーザーを設定します。以下に例を示します。

   ```
   SET SYNC USER REG_USER IDENTIFIED BY SECRET;
   ```

4. 登録メッセージを送信して、レプリカをマスター Master1 に登録します。例を以下に示します。

```
MESSAGE CFG1 BEGIN TO "MASTER";
MESSAGE CFG1 APPEND REGISTER REPLICA;
MESSAGE CFG1 END;
COMMIT WORK;
MESSAGE CFG1 FORWARD TO 'tcp 1315' TIMEOUT FOREVER;
COMMIT WORK;
```

注:

REGISTER REPLICA コマンドを使用して、マスター・サーバーのデフォルト・カタログ以外のカタログにレプリカを登録する場合、そのカタログの、該当するマスター・ノード名を MESSAGE BEGIN コマンドで指定する必要があります。これによって、solidDB は、レプリカのマスター・データベースで正しいカタログを解決できます。構文は、以下のとおりです。

```
MESSAGE message_name BEGIN TO master_node_name
```

MESSAGE FORWARD コマンドは、MESSAGE END コマンドでメッセージが永続的になった後、マスター・データベースにメッセージを送信します。メッセージ受信者のネットワーク listen 名は、MESSAGE FORWARD コマンドで指定することに注意してください。これは、新しいレプリカからマスター・データベースに最初のメッセージを送信するときにのみ必要です。TIMEOUT が定義されていない場合、レプリカは応答をフェッチしません。別の MESSAGE GET REPLY 呼び出しでリトリーブする必要があります。

5.

別のメッセージで MESSAGE APPEND SYNC_CONFIG コマンドを使用して、マスター・データベースからのマスター・ユーザー名情報をサブスクライブします。この例では、SQL ワイルドカード '%' を使用して、マスター・データベースからすべてのユーザー名を送信するように要求します。

```
MESSAGE CFG2 BEGIN;
MESSAGE CFG2 APPEND SYNC_CONFIG('%');
MESSAGE CFG2 END;
COMMIT WORK;
MESSAGE CFG2 FORWARD TIMEOUT FOREVER;
COMMIT WORK;
```

6.

マスター・ユーザー名情報が正常にサブスクライブされたら、レプリカ・データベース接続で登録ユーザーがアクティブでなくなるように、同期ユーザーを「none」にリセットします。以下に例を示します。

```
SET SYNC USER NONE;
```

登録ユーザーには、登録以外の権限はありません。登録ユーザーがアクティブなままの場合は、通常、後続のコマンドで以下のエラー・メッセージが発生します。

```
User definition not allowed for this operation
```

パブリケーションの作成

パブリケーション は、マスター・データベースからサブスクライブ・レプリカ・データベースにダウンロードするデータ集合の定義です。これは、データを変更するトランザクションから完全に分離されています。「従来」のレプリケーション方法では、通常マスターからレプリカにトランザクション (挿入、更新、および削除) を送信する方法に基づいていましたが、solidDB では、その代わりに、更新されたデータのスナップショットをレプリカに送信することに注意してください。

拡張レプリケーション・パブリケーションの定義には、以下の内容が含まれます。

- 1 つまたは複数の表からのデータ — パブリケーションの表の間の関係を定義できます。
- 表のすべての行またはサブセットの行 — パブリケーションには、パブリケーション用データを選択するための通常の SELECT ステートメントを含めることができます。
  - パラメーターで表のサブセット行を制限することにより、パブリケーションに関して固定検索基準または動的検索基準を指定できます。
- 表のすべての列または定義された列 — パブリケーションには、パブリケーション用の列を選択するための通常の SELECT ステートメントを含めることができます。
- フル・データまたはインクリメンタル・データ — フル・パブリケーションは、パブリケーションに含まれるすべてのデータを送信します。インクリメンタル・パブリケーションは、前回のリフレッシュ以降に変更されたデータだけを送信します。

注:

リソースを節約し、パフォーマンスを向上させるため、インクリメンタル・パブリケーションを使用することを推奨します。実際のパブリケーションを作成する前に、インクリメンタル・パブリケーション用に表をセットアップする必要があります。詳しくは、後述するセクションの『インクリメンタル・パブリケーションの作成』を参照してください。

インクリメンタル・パブリケーションの作成

サーバーが表のインクリメンタル・リフレッシュを行うためには、その表に対して行った最新のリフレッシュに関する情報を保管する必要があります。このリフレッシュ情報は、同期履歴データと呼ばれています。 (各表の同期履歴データは、同期履歴表に保管されます。パブリケーション内の各表に、同期履歴表が 1 つ必要です。)

パブリケーションをインクリメンタル・パブリケーションにするためには、各表のSYNCHISTORYプロパティーを設定し、サーバーに対してその表の同期履歴データを収集するよう指示する必要があります。この処理は、以下のコマンドで行います。

```
ALTER TABLE table_name SET SYNCHISTORY
```

このコマンドは、パブリケーション内の各表に対して実行する必要があります。また、マスター・データベースとレプリカ・データベースの両方に対して実行する必要があります。

表の同期履歴設定は、その表の「プロパティー」と解釈されます。デフォルトでは、このプロパティーは各表で NOSYNCHISTORY に設定されます。マスター・データベースとレプリカ・データベースの両方において、このプロパティーが SYNCHISTORY に設定されている場合、最初のリフレッシュ後は、特定のパブリケーションに対する後続のリフレッシュにより、表内のデータが同期されたときに、新規の行と変更された行だけがレプリカ・データベースに送信されます。

いずれかのパブリケーションが表を参照する前に、その表の SYNCHISTORY プロパティーを設定することを推奨します。表がパブリケーションの一部として組み込まれた後に、その表の SYNCHISTORY プロパティーを変更したい場合には、同期保守モードを使用する必要があります。詳しくは、135 ページの『分散システムのスキーマのアップグレード』を参照してください。

例えば、SYNCDEMO という名前の表の SYNCHISTORY プロパティーを設定するには、以下のコマンドを使用してください。

```
ALTER TABLE SYNCDEMO SET SYNCHISTORY;
COMMIT WORK;
```

このステートメントは、履歴データを保管するシャドー表を作成します。 シャドー表は、メイン表で変更または削除された行を追跡します。リフレッシュのためにデータを要求するレプリカ・データベースがもう存在しない場合には、シャドー表の不要なデータは自動的に削除されます。

履歴データ管理の最適化について詳しくは、159 ページの『8 章 パフォーマンスのモニターおよびチューニング』を参照してください。

CREATE PUBLICATION ステートメントの使用

パブリケーションを作成する際、どの表 (マスター内) からデータを読み取るのか、どの表 (レプリカ内 (複数可)) にデータを書き込むのかを指定します。

パブリケーションを作成するには、マスター・データベース内で CREATE PUBLICATION ステートメントを実行します。構文は、以下のとおりです。

```
"CREATE PUBLICATION publication_name
   [( parameter_definition [,parameter_definition ...])]
BEGIN
   main_result_set_definition...
END";

main_result_set_definition ::=
RESULT SET FOR main_replica_table_name
   BEGIN
      SELECT select_list
```

```
   FROM master_table_name
   [ WHERE search_condition ] ;
      [[DISTINCT] result_set_definition...
   END

result_set_definition ::=
RESULT SET FOR replica_table_name
   BEGIN
      SELECT select_list
   FROM master_table_name
   [ WHERE search_condition ] ;
   [[DISTINCT] result_set_definition...]
   END
```

CREATE PUBLICATION ステートメントによって、マスター・データベースからレプリカ・データベースに新規および変更済みデータをインクリメンタル「ダウンロード」するためのパブリケーションを指定できます。

パブリケーションのデータは、必ず 1 つのトランザクションでマスター・データベースから読み取ります。これによって、パブリケーションから読み取るデータが内部的に整合したものになります。

注意:

マスターが READ COMMITTED トランザクション分離レベルを使用しない限り、パブリケーションから読み取るデータは内部的に整合性のあるものになります。リフレッシュのトランザクション分離レベルは、システム・デフォルトと異なっていてもかまいません。詳しくは、「*solidDB 管理者ガイド*」の付録 A の『サーバー・サイド構成パラメーター』に記載のパラメーター RefreshIsolationLevel を参照してください。

デフォルトでは、パブリケーション・データはまた 1 つのトランザクションでレプリカ・データベースに書き込まれ、整合性が維持されます。ただし、レプリカ側では、COMMITBLOCK を使用してデフォルトの動作をオーバーライドして、リフレッシュを複数のトランザクションに分割できます。COMMITBLOCK を使用すると、内部の整合性は保証されなくなります。COMMITBLOCK について詳しくは、83 ページの『コミット・ブロック・サイズの設定』を参照してください。

search_condition は、*parameter_definition* または前の (上位の) レベルで定義されたレプリカ表の列、またはその両方を参照することができます。SELECT 節の検索条件には、パブリケーションの入力引数をパラメーターとして含むことができます。パラメーター名には、接頭部としてコロンを含む必要があります。

パブリケーションには複数の表のデータを含むことができます。これらの表は独立させることも、関連させることもできます。表に関連性がある場合は、結果セットをネストしなければなりません。パブリケーションの内部結果セットの SELECT ステートメントにある WHERE 節は、外部結果セットの表の列を参照しなければなりません。

パブリケーションに複数の表が含まれており、表に関連性がある場合でも、データを書き込む表 (レプリカ内) の数は、データを読み取った表 (マスター内) の数と同じである点に注意してください。2 つの表を単一のビューに結合したり、SUM() な

どの集約関数を使用してデータを要約したりするような方法で、マスター内の複数の表のレコードがレプリカ内の単一のレコードに要約されたり、結合されたりすることはありません。

以下は典型的なパブリケーションです。

```
CREATE PUBLICATION ORDERS_BY_SALESPERSON
(SALESPERSON_ID VARCHAR)
BEGIN
   RESULT SET FOR CUST_ORDER
   BEGIN
      SELECT * FROM CUST_ORDER
      WHERE SM_ID = :SALESPERSON_ID AND STATUS = 'ACTIVE';
      RESULT SET FOR ORDER_LINE
     BEGIN
        SELECT * FROM ORDER_LINE
        WHERE ORDER_ID = CUST_ORDER.ID;
      END
      DISTINCT RESULT SET FOR CUSTOMER
      BEGIN
         SELECT * FROM CUSTOMER
         WHERE ID = CUST_ORDER.CUSTOMER_ID ;
      END
   END
END;
```

上記のサンプル・パブリケーションは、マスター・データベースの 3 つの表からデータをリトリーブします。

- パブリケーションのメイン表は CUST_ORDER です。SALESPERSON_ID を検索基準として使用して、表から行がリトリーブされます。
- 各オーダーに対して、ORDER_LINE 表の行がリトリーブされます。CUST_ORDER と ORDER_LINE の多重度は 1-N です。CUST_ORDER 表の ID 列と ORDER_LINE 表の ORDER_ID 列を使用して、データがリンクされます。
- 各オーダーに対して、CUSTOMER 表の行もリトリーブされます。CUST_ORDER 表と CUSTOMER 表の多重度は N-1です。これは、顧客が複数のオーダーを行えることを意味しています。CUST_ORDER 表の CUSTOMER_ID 列と CUSTOMER 表の ID 列を使用して、データがリンクされます。キーワード DISTINCT によって、同じ顧客情報がレプリカ・データベースに 1 度だけ取り込まれるようになります。

パブリケーションのガイドライン

結果セットの間で 1-N および N-1 の関係を持つパブリケーションを作成できます。結果セットは、ネストさせることもできます。例えば、CUST_ORDER が ORDER LINES (1-N) を持ち、各 ORDER LINE が PRODUCT (N-1) を持つことができます。

パブリケーションの外部と内部の結果セットの関係が N-1 の関係である場合、結果セット定義でキーワード DISTINCT を使用する必要があります。

ネストされた各結果セットは、内部的には結合として扱われます。パブリケーションの結果セットの層が多くなるほど、データをリトリーブする照会は複雑になります。そのため、パフォーマンスを向上させるために、結果セットの大規模なネストは避けてください。以下の例では、CREATE PUBLICATION ステートメントを、ネストのない同等のステートメントに書き換えることで、パフォーマンスを向上させています。

ネストされたパブリケーションのバージョン

```
CREATE PUBLICATION NESTED (IN_ORDER_ID INTEGER)
BEGIN
RESULT SET FOR CUST_ORDER
   BEGIN
      SELECT * FROM CUST_ORDER
      WHERE ID = :IN_ORDER_ID;
      RESULT SET FOR ORDER_LINE
      BEGIN
         SELECT * FROM ORDER_LINE
         WHERE ORDER_ID = CUST_ORDER.ID;
      END
   END
END;
```

ネストされていないパブリケーションのバージョン

```
CREATE PUBLICATION UNNESTED (IN_ORDER_ID INTEGER)
BEGIN
RESULT SET FOR CUST_ORDER
   BEGIN
      SELECT * FROM CUST_ORDER
      WHERE ID = :IN_ORDER_ID;
   END
RESULT SET FOR ORDER_LINE
   BEGIN
      SELECT * FROM ORDER_LINE
      WHERE ORDER_ID = :IN_ORDER_ID;
   END
END;
```

作成した各パブリケーションは、他のパブリケーションから完全に独立しています。つまり、パブリケーション間の従属関係は定義できません。

レプリカでオーバーラップするパブリケーション定義を使用しないでください。これには、表と WHERE 条件がオーバーラップするパブリケーション定義も含まれます。同じ表についてオーバーラップするサブセットを作成する可能性がある (一部またはすべての行が両方のサブセットに同時に含まれる可能性がある) 場合に、パブリケーション定義がオーバーラップします。

例えば、パブリケーション「ORDERS_BY_SALESPERSON」が顧客情報をリトリーブする場合、マスター・データベースからサブスクライブ・レプリカに同じ行をリトリーブする可能性がある「CUSTOMERS_BY_AREA」のような別のパブリケーションを使用することは推奨しません。このようにすると、パブリケーションのサブスクリプションをドロップするときなどに競合状態が発生し、別のサブスクリプションがそれらの行を参照しているかどうかにかかわらず、サブスクリプションのレプリカ・データ全体が削除されるという結果になります。

また、1 つのレプリカ内で、同じパブリケーションについてオーバーラップするリフレッシュを行わないように注意してください。例えば、前の REFRESH 操作の応

答メッセージを処理し終わる前に、新しい REFRESH を開始すると、レプリカ・データベースに不正なデータが含まれることがあります。

CREATE PUBLICATION を使用した後、レプリカがパブリケーションにサブスクライブする前に、トランザクションをコミットすることが重要です。パブリケーションを定義するトランザクションがマスターでコミットされていない場合、レプリカがそのパブリケーションにサブスクライブしようとすると、システムは、パブリケーションが存在しないというエラー・メッセージを発行します。

パブリケーション・データは、MESSAGE APPEND REFRESH *publication_name* ステートメントを使用してマスター・データベースに要求されます。詳しくは、次の章を参照してください。

パブリケーションへのサブスクライブ

レプリカ・データベースは、リフレッシュを使用してマスターにパブリケーション・データを要求します。リフレッシュは、マスター・データベース内のパブリケーション定義によって異なります。レプリカがパブリケーションを登録し、そこからリフレッシュ可能であることを確認してください。レプリカに登録されていないパブリケーションからは、リフレッシュできません。パブリケーションを登録することで、パブリケーション・パラメーターを検証することができます。

パブリケーションは、MESSAGE APPEND REGISTER PUBLICATION ステートメントを使用してレプリカに登録されます。構文は、以下のとおりです。

```
MESSAGE APPEND REGISTER PUBLICATION publication_name
```

以下に例を示します。

```
MESSAGE MyMsg0001 APPEND REGISTER PUBLICATION publ_customer;
```

ユーザーがパブリケーションにアクセスするには、そのパブリケーションに対して REFRESH 特権を持つ必要があり、パブリケーションで使用される表に対して特権を持つ必要があります。表の所有者 (または DBA) は、パブリケーションで使用される表に対する特権を付与する必要があり、パブリケーションの作成者 (または DBA) は、GRANT REFRESH を実行して、パブリケーションに対するユーザー・アクセス権限を付与する必要があります。詳しくは、 85 ページの『アクセス権限およびロールによるセキュリティーのインプリメント』を参照してください。

注:

レプリカは、マスターで定義されたパブリケーションからのみリフレッシュできます。レプリカは、レプリカ・データベース自身に定義されたパブリケーションは使用できません。CREATE PUBLICATION コマンドをレプリカ・データベースで実行すると、パブリケーション定義がレプリカに格納されますが、そのレプリカが 3 層以上の階層内で別の層のマスターでもある場合を除いて、パブリケーション定義は使用されません。

パブリケーション・データは、一連の入力パラメーター値 (パブリケーションで使用されている場合) を持つパブリケーション呼び出しとして、マスター・データベースに要求します。構文は、以下のとおりです。

```
MESSAGE unique_message_name APPEND
      [REFRESH publication_name [( publication_parameters )]
      [FULL]]
```

以下に例を示します。

```
MESSAGE my_msg APPEND
      REFRESH ORDERS_BY_SALESMAN ('SMITH') ;
```

初期「リフレッシュ」は常にフル・パブリケーションであり、パブリケーションの検索基準に一致するすべてのデータがレプリカ・データベースに送信されます。同じパブリケーションに対するそれ以降のリフレッシュには、前のリフレッシュ以降に変更されたデータのみが含まれます。これは、*インクリメンタル・パブリケーション* と呼ばれています。リソースを節約し、パフォーマンスを向上させるため、インクリメンタル・パブリケーションを使用することを推奨します。一般に、最新の変更を含むパブリケーションの更新のみをレプリカに送信する必要があります。インクリメンタル・パブリケーション用に変更を追跡するための表の設定について詳しくは、100 ページの『インクリメンタル・パブリケーションの作成』を参照してください。

REFRESH にキーワード FULL を指定すると、フル・パブリケーション・データがレプリカにフェッチされます。パブリケーションが大きい場合、レプリカ・データベースへのデータの初期 (非インクリメンタル) ダウンロードによって、大きなトランザクションが作成されます。このような場合、同期メッセージの単一トランザクションのサイズは、COMMITBLOCK オプションで制限することができます。COMMITBLOCK について詳しくは、83 ページの『コミット・ブロック・サイズの設定』を参照してください。

ヒント:

また、ReplicaRefreshLoad パラメーターを使用して、リフレッシュ操作に使用可能なリソースを制限することもできます。

サブスクライブ・データとローカル・データの結合

レプリカ上の表には、サブスクライブ・データだけでなく、「ローカル・データ」も含まれていることがあります。レプリカでローカルにのみ使用する行を保持する場合には、パブリケーションにローカル行を除外する 'where' 制約を指定する必要があります。この状態では、レプリカは、表内にローカル行を保持し、マスターからサブスクライブ・データを追加します。

サブスクリプションのドロップ

サブスクライブ・データがレプリカで使用されなくなった場合、そのレプリカで DROP SUBSCRIPTION コマンドを使用してサブスクリプションをドロップすることにより、サブスクライブ・データを削除することができます。詳しくは、134 ページの『パブリケーションおよびパブリケーション内の表の変更』を参照してください。

パブリケーションの登録抹消またはドロップ

同期メッセージで以下のコマンドを使用して、レプリカで登録済みのパブリケーションを登録抹消できます。

```
MESSAGE APPEND UNREGISTER PUBLICATION publication_name
```

以下に例を示します。

```
MESSAGE MyMsg0001 APPEND UNREGISTER PUBLICATION publ_customer;
```

これはマスターに伝搬されるメッセージの一部である必要があります。

メッセージを送信せずに、登録済みのパブリケーション定義をレプリカでドロップすることもできます。構文は、以下のとおりです。

```
DROP PUBLICATION publication_name REGISTRATION
```

以下に例を示します。

```
DROP PUBLICATION publ_customer REGISTRATION;
```

DROP PUBLICATION REGISTRATION コマンドは、レプリカがマスターと通信できない場合にのみ使用します。マスターに通知せずにサブスクリプションをドロップすると、そのサブスクリプションのシステム情報がマスター上に残り、無制限にスペースを消費します。何よりも重要なのは、このレプリカの同期履歴データが、レプリカで使用しないにもかかわらず収集されることです。これによって、パブリケーションに関連する「シャドー表」が膨張します。可能であれば、マスター・データベースに移動し、以下のコマンドを使用してサブスクリプションをドロップして、該当するシステム情報を手動で解除してください。

```
DROP SUBSCRIPTION publication_name(parameters) FROM REPLICA replica_name
```

インテリジェント・トランザクションの設計およびインプリメント

トランザクションとは、従来、データベースをある有効な状態から別の有効な状態に変化させる、データベース操作のアトミック・セットです。「データベースの有効な状態」とは、そのデータベースの中で保全性および整合性のルールにまったく違反していない状態を指します。このようなルールは、データベース固有 (参照整合性) である場合も、アプリケーション固有である場合もあります。

solidDB インテリジェント・トランザクション は、従来のトランザクション・モデルの拡張機能です。現行データベースの中でトランザクション自身を検証し、必要に応じてその内容をトランザクションのルールに合わせて変更する機能を持つトランザクションをインプリメントすることができます。

例えば、あるオーダー処理システムのアプリケーション・ルールでは、顧客がクレジット制限を超過しない場合に限って、オーダーの作成が可能であるとします。また、「オーダー作成」トランザクションの処理内容が、CUST_ORDER 表に行を挿入し、INVOICE 表に別の行を挿入することであるとします。顧客がクレジット制限を超過していたため、オーダーを挿入できなかった場合には、そのオーダーに関する請求書も挿入できません。INSERT_ORDER プロシージャーから INSERT_INVOICE プロシージャーに対して、妥当性検査エラーが発生したことを通知する必要があります。こうすることで、INSERT_INVOICE プロシージャーは動作を変更し、それによってトランザクションを有効な状態に維持することができます。

アプリケーション固有の整合性ルールがあれば、トランザクション設計原則は以下のようになります。

solidDB インテリジェント・トランザクション は、SQL ステートメントの集合であり、通常 solidDB ストアード・プロシージャーとしてインプリメントされるビジネス・ロジックを入れることができます。 インテリジェントなトランザクションの動作と特性を以下に示します。

- トランザクションは複数の操作から構成されます。つまり、複数のストアード・プロシージャーの呼び出しで構成されます。
- トランザクションは、作成され、一時的にコミットされ、レプリカ・データベースに保存されますが、最終的にマスター・データベースにコミットされるため、長時間にわたって存続します。したがって、マスター・データベースの各トランザクションの妥当性検査は、すべてトランザクション自身で行う必要があります。
- トランザクションは、マスター・データベースの整合性を維持する必要があります。

『オーダー・トランザクションの作成』では、単純な受注アプリケーションで、「オーダー作成」トランザクションを作成します。以下のセクションでは、この例を使って、solidDB インテリジェント・トランザクション のインプリメント方法について説明します。

オーダー・トランザクションの作成

```
-- ローカル・データベースを変更します
CALL INSERT_ORDER(...) ;
CALL UPDATE_CUSTOMER_CREDIT(...) ;
-- プロパティーをトランザクションに保存します
SAVE PROPERTY priority VALUE '1';
-- 後でマスターに伝搬するために、ステートメントを保存します
SAVE CALL INSERT_ORDER(...) ;
SAVE CALL UPDATE_CUSTOMER_CREDIT(...) ;
-- ローカルの変更と保存したトランザクションを
-- そのまま維持します
COMMIT WORK;
```

ローカル・データの更新

『オーダー・トランザクションの作成』では、トランザクションの最初の部分 (ローカルでの変更) で、標準的な SQL 節を直接実行します。

```
-- ローカル・データベースを変更します。
CALL INSERT_ORDER(...) ;
CALL UPDATE_CUSTOMER_CREDIT(...) ;
```

solidDB 拡張レプリケーション・アーキテクチャーでは、後で伝搬するためにトランザクションのステートメントとパラメーターを明示的に保存しない限り、ローカルでの変更はローカルのまま維持されます。

後で伝搬するためのトランザクションの保存

108 ページの『オーダー・トランザクションの作成』からの抜粋である以下の内容には、ローカルな変更の後に SAVE ステートメントがあり、後でローカル更新をマスター・データベースに伝搬することを指定しています。

```
-- 後でマスターに伝搬するために、ステートメントを保存します。
SAVE CALL INSERT_ORDER(...) ;
SAVE CALL UPDATE_CUSTOMER_CREDIT(...) ;
```

後で伝搬するためにステートメントを保存する構文は、以下のとおりです。

```
SAVE sql_statement
```

ローカル・データベースをまったく更新せずに、後で伝搬するためにステートメントの保存だけを行うこともできることに注意してください。この場合、レプリカは情報をマスターに伝搬し、マスターの更新データからリフレッシュした後、更新情報を取得します。

重要:

保存されたステートメントは、マスターで「そのまま」実行されます。このステートメントには、レプリカで実行されたときにレプリカのどのレコードが影響を受けたかが記憶されていません。例えば、以下のような一連のステートメントを実行したとします。

```
UPDATE employee_table SET salary = salary * 1.10
WHERE state = 'California';
SAVE UPDATE employee_table SET salary = salary * 1.10
WHERE state = 'California';
```

また、マスター・データベースにはカリフォルニアで働く 200 人の従業員が含まれ、レプリカにはカリフォルニアのサンフランシスコにある支社で働く 100 人の従業員だけが含まれているとします。この場合、レプリカで実行される UPDATE コマンドは、レプリカのデータベースに含まれている 100 人の従業員にだけ適用されます。しかし、保存されるとこのステートメントは、マスターのデータベースにリストされている 200 人のカリフォルニアの従業員すべてに適用されます。コマンドをマスターに伝搬するときは、コマンドが適切なレコードにだけ適用されるようにする WHERE 節を使用するように注意してください。

拡張レプリケーション・パラメーター掲示板の使用

solidDB 拡張レプリケーションでは、新しいパラメーター受け渡し方法として「パラメーター掲示板」が導入されており、これをトランザクションが各種の目的で使用することができます。パラメーター掲示板は、各種のパラメーターを追加でき、トランザクションの操作 (プロシージャー) でこれらのパラメーターを読み取ることができるメモリー領域です。

パラメーター掲示板に表示されるパラメーターには 3 種類あります。

- *VOLATILE* トランザクション・パラメーター。トランザクション内で、そのトランザクションの異なるプロシージャー間で情報を転送するために使用します。
-

トランザクション・プロパティー (永続的なトランザクション・パラメーター)。トランザクションがレプリカ・データベースで作成される際に、そのトランザクションにいくつかのプロパティーを指定する (すなわちトランザクションを記述する) ために使用します。

- *永続的な、カタログ・レベルの同期パラメーター*。トランザクションの実行場所であるカタログを記述するために使用します。

パラメーターのタイプによって、パラメーターを指定する方法や、パラメーター掲示板にパラメーターを配置するタイミングと方法のメカニズムが異なります。ただし、掲示板からパラメーターを読み取るためのメカニズムは、パラメーターのタイプが異なってもすべてに共通しています。

トランザクション内でのプロシージャー間のパラメーターの引き渡し

トランザクションのプロシージャーは、PUT_PARAM() 関数を使用して VOLATILE パラメーターをパラメーター掲示板に入れ、GET_PARAM() 関数を使用してそのパラメーターを読み取ることで、相互に通信できます。各パラメーターは、名前と値のペアです。パラメーターが既に掲示板にある場合、PUT_PARAM() 関数は現行値を新しい値に置き換えます。パラメーターが掲示板にない場合、GET_PARAM() は NULL を返します。

以下に、パラメーターの書き込みと読み取りの例をいくつか示します。

```
-- プロシージャー P1 が掲示板の同期パラメーターを設定する。
"CREATE PROCEDURE P1()
BEGIN
PUT_PARAM('CreditLimitExceeded', 'Y');
...
END";

-- プロシージャー P2 が掲示板の同期パラメーターを読み取る。
"CREATE PROCEDURE P2()
BEGIN
DECLARE cred_lim_exceeded CHAR(1);
cred_lim_exceeded := GET_PARAM('CreditLimitExceeded');
...
END";
```

GET_PARAM() は、PUT_PARAM() コマンドで設定された同期パラメーター値の読み取りだけではなく、SAVE PROPERTY コマンドで設定されたトランザクション・プロパティーの読み取りにも使用できることに注意してください。

パラメーター掲示板は、トランザクションのすべてのステートメントで可視です。そのため、例えば異なるストアード・プロシージャーが、互いを呼び出さない場合でも、相互に通信できます。あるプロシージャーでエラーを検出した場合、エラーのあるデータの処理をスキップするように、同じトランザクションの後続のプロシージャーに通知するフラグを設定できます。

パラメーターは、トランザクションがマスター・データベースで実行されるときに、トランザクションのパラメーター掲示板に表示されます。レプリカでトランザクションが実行されているときは、レプリカでもパラメーターが可視になります。

ほとんどの場合、トランザクション・プロパティーは、マスターでトランザクションが実行されるとき、つまり、トランザクションがレプリカからマスターに伝搬された後で使用されます。ただし、値はレプリカでもマスターでも (またはその両方で) 使用できます。

注: 競合解決のためにインテリジェント・トランザクションをインプリメントするときは、トランザクション・プロパティーが失われないように、必ず自動コミットをオフにします。トランザクション・パラメーターのライフ・サイクルは、1 トランザクションです。つまり、その値を設定したトランザクション内でのみ可視です。自動コミットがオンの場合は、各ステートメントが別個のトランザクションになり、掲示板の値がすぐに失われます。

PUT_PARAM() および GET_PARAM() について詳しくは、「*IBM solidDB SQL ガイド*」を参照してください。

複製されたトランザクションへのプロパティーの割り当て

トランザクション・プロパティーは、複製されたトランザクション全体を記述するために使用されます。これらのパラメーターは、SAVE PROPERTY ステートメントを使用してレプリカ・データベースで定義される永続的なパラメーターで、アタッチ先のトランザクションが正常にマスター・データベースに伝搬され、実行されるまで持続します。SAVE PROPERTY コマンドは、伝搬可能なトランザクションのパラメーターをレプリカ・データベースに保管します。伝搬されたトランザクションが後でマスターで実行されるとき、これらのパラメーターがトランザクションの初めにパラメーター掲示板 (マスター・データベース内) に入れられます。トランザクションのすべてのプロシージャーが、GET_PARAM() 関数を使用して、このパラメーターの値を照会できます。

トランザクション・プロパティーは、2 つの目的で使用できます。

- 同期メッセージで伝搬するトランザクションを選択する選択基準として機能
- マスター・データベースでトランザクションが実行されるときに、トランザクションのプロシージャーが内部的に使用

プロパティーをトランザクションに保存する構文は、以下のとおりです。

```
SAVE PROPERTY property_name VALUE property_value
```

108 ページの『オーダー・トランザクションの作成』からの抜粋では、トランザクションに、名前が「priority」で値が「1」の保存されたプロパティーが 1 つあります。

```
-- Save a parameter to the transaction
SAVE PROPERTY priority VALUE '1';
```

このパラメーターは、トランザクション伝搬プロセスの検索基準として使用できます (「パラメーター「priority」の値が「1」のトランザクションだけを伝搬する」など)。以下に例を示します。

```
MESSAGE APPEND PROPAGATE TRANSACTIONS WHERE priority = '1';
```

トランザクションがマスター・データベースに伝搬されると、トランザクションがマスター・データベースで実行されるときに、このトランザクションのすべての定義済みプロパティーの値がトランザクションのパラメーター掲示板に表示されます。よって、GET_PARAMI() 関数を使用して、プロシージャーから「priority」プロパティーの値を照会できます。

```
DECLARE priority_value CHAR(1);
priority_value := GET_PARAM('priority');
```

この情報は、アプリケーションの独自の目的に使用できます。例えば、発生する可能性がある更新の競合を、このトランザクションでどのように解決するかを判別するために使用できます。

カタログ・レベルの永続的な同期パラメーターの定義

カタログ単位のスコープを持つパラメーターを定義するには、SET SYNC PARAMETER コマンドを使用します。このコマンドは、そのカタログ内で実行するどのトランザクションからも (GET_PARAM() 関数を使用して) 読み取ることが可能なパラメーターを指定します。そのカタログの各トランザクションは、パラメーターの値を変更することもできますが、更新後の値は現行トランザクションの中でしか認識されません。後続のトランザクションからは更新後の値は認識されず、「元」の値のみが認識されます。

拡張レプリケーション・システム・パラメーター

拡張レプリケーション・システム自身で、「システム・パラメーター」と呼ばれるパラメーターをいくつか事前定義しています。システム・パラメーターは、サーバーが認識しているものであり、これに従って、サーバーは動作することができます (例えば、トランザクションを終了します)。(総称カタログのスコープを持つパラメーターは、アプリケーション・レベルのインテリジェント・トランザクションが認識すべき対象です。サーバー自体の動作には影響しません。)システム・パラメーターの例として、SYS_ROLLBACK があります。 SYS_ROLLBACK を 'YES' に設定すると、サーバーは、そのトランザクションを終了させます。SYS_ROLLBACK の値は、次のトランザクションのために、デフォルトの 'NO' にリセットされます。システム・パラメーターは、単にシステムによって予約された名前を使用するパラメーターです。その他のパラメーターと同様に、システム・パラメーターも掲示板に表示されます。ユーザーのトランザクションは、その他のパラメーターの読み取りや書き込みと同じように、システム・パラメーターの読み取りや書き込みを行うことができます。

ストアード・プロシージャーの作成

データベースの物理的な整合性と論理的な整合性を保証するため、solidDB SQL ストアード・プロシージャー言語を使用してストアード・プロシージャーを作成する必要があります。これらのプロシージャーは、パラメーター掲示板を使用して互いに通信することができます。 113 ページの『INSERT_ORDER ストアード・プロシージャー』では、新しいオーダーの挿入に失敗した場合、勘定残高を更新してはいけません。

以下に、 108 ページの『オーダー・トランザクションの作成』で呼び出されるプロシージャーの簡単な例を示します。

INSERT_ORDER ストアード・プロシージャー

```
"CREATE PROCEDURE INSERT_ORDER
   (ORDER_ID VARCHAR, CUST_ID VARCHAR, ...)
BEGIN
DECLARE ORDER_FAILED VARCHAR ;
DECLARE CUST_OK INTEGER ;
DECLARE STATUS VARCHAR ;
ORDER_FAILED := 'N' ;
STATUS := 'OK' ;

-- オーダーを検証
-- 例えば、有効な顧客が指定されていなければなりません。
EXEC SQL PREPARE CHECK_CUST
CALL CHECK_CUSTOMER (?) ;
EXEC SQL EXECUTE CHECK_CUST
USING (CUST_ID)
INTO (CUST_OK) ;
IF CUST_OK = 0 THEN
ORDER_FAILED := 'Y' ;
STATUS := 'FAIL' ;
END IF ;
-- ここで、その他の妥当性検査を行います...
-- ...
-- 妥当性検査の終わり

-- 妥当性検査が失敗した場合、掲示板にパラメーターを記入し、
-- 以降のストアード・プロシージャーに妥当性検査が失敗したことを通知します。
IF ORDER_FAILED = 'Y' THEN
PUT_PARAM('ORDER_FAILED', 'Y');
END IF;

-- データベースにオーダー行を挿入します。行内の STATUS 値は、
-- 'OK' または 'FAIL' のどちらかになります。
EXEC SQL PREPARE INS_ORD
INSERT INTO CUST_ORDER (ORD_ID, CUST_ID, STATUS, ...)
VALUES (?,?,? ...);
EXEC SQL EXECUTE INS_ORD
USING (ORD_ID, CUST_ID, STATUS...);
EXEC SQL CLOSE INS_ORD;
EXEC SQL DROP INS_ORD;
END";
```

以下のプロシージャーは、指定された勘定の残高を更新します。

```
"CREATE PROCEDURE UPDATE_CUST_CREDIT (ACC_NUM VARCHAR, AMOUNT FLOAT)
BEGIN
DECLARE ORDER_FAILED VARCHAR;
-- 掲示板から、オーダーが正しく挿入されたか、
-- または変更されたかを確認します。
-- 失敗した場合、勘定残高を更新しないでください。
ORDER_FAILED := GET_PARAM('ORDER_FAILED');
IF ORDER_FAILED = 'Y' THEN
   RETURN
END IF;

EXEC SQL PREPARE UPD_CREDIT
   UPDATE ACCOUNT
   SET BALANCE = BALANCE + ?
   WHERE ACC_NUM = ?;
EXEC SQL EXECUTE UPD_CREDIT
USING (AMOUNT, ACC_NUM);
EXEC SQL CLOSE UPD_CREDIT;
EXEC SQL DROP UPD_CREDIT;
END";
```

アプリケーションに対する同期エラー・ログ表の作成

非集中型システムのデータベース・スキーマの一部として、同期エラー・ログ表を作成することを強く推奨します。アプリケーション開発者は、ログ表を使用して、同期中に発生する可能性のあるアプリケーション・レベルのエラーに関する情報を保管できます。

エラーを手動で解決する必要がある場合、エラー・ログは、エラーの訂正に必要な情報源としての役割を果たします。例えば、更新の競合が発生した場合、ログには、競合が発生した行の情報、およびその行に対してトランザクションが実行した処理に関する情報を含めることができます。以下に、エラー・ログ表の例を示します。

```
CREATE TABLE ERRLOG
   (ID CHAR(20) NOT NULL,
   LOG_TITLE CHAR(80) NOT NULL,
   LOG_DESCRIPTION VARCHAR NOT NULL,
   UPDATETIME DATETIME NOT NULL,
   PRIMARY KEY (ID));
```

ID は、ログ・エントリーのグローバルなユニーク・キーとして生成されます。

LOG_TITLE 列と LOG_DESCRIPTION 列には、更新の競合などのエラーに関する情報が含まれます。

UPDATETIME 列には、エラー・ログ行の最終更新操作のタイム・スタンプが含まれます。

インテリジェント・トランザクションの検証

データ同期によって、ビジネス・アプリケーションの機能性に新たな側面が生まれます。集中型システムと非集中型システムの主な違いは、非集中型システムでは一時的なデータが存在するという点です。システム内に同じデータ項目の異なるバージョンが複数存在することがあります。

マスター・データベースには、正式な正しいバージョンのデータが存在します。レプリカには、同じデータの正式でない、異なるバージョンが存在することがあります。レプリカ・トランザクション (データの正式でないレプリカ・バージョンに基づく) がマスター・データベースに伝搬されると、更新の競合などのトランザクション妥当性検査エラーがマスター内で発生することがあります。このような状態では、トランザクションはアプリケーションのビジネス・ルール要件に準拠するように動作する必要があります。

トランザクション妥当性検査エラーが発生した場合、いくつかの処理方法があります。

- ユーザー介入なしで自動的にエラーを解決する方法。例えば更新の競合の場合、最新の更新を選択します。
-

データのマスター・バージョンをそのままにしておき、競合している操作またはエラーになっているその他の操作に関する十分な情報を保存して、手動でのユーザー介入によるエラー修正を行えるようにする方法。

最初の方法では、データ・モデルから何も特別のことは必要としません。トランザクション妥当性検査エラーは発生すると自動的に解決され、手動での介入が不要だからです。

ただし、最初の方法は、考えられるトランザクション妥当性検査エラーを必ずしもすべて考慮しているわけではないので、いくつかのエラーは自動的には解決できません。例えば、受注システムのマスターとレプリカの両方のデータベースでオーダーが更新された場合、どちらの更新が正しいのかを自動的に判断することは困難です。場合によっては、エラーを修正するためにユーザー介入が必要になります。

ユーザーがエラーを修正できるようにするために、失敗したトランザクションに関する十分な量の情報を維持しておく必要があります。この情報を別のエラー・ログ表に格納することも、同じデータ項目の複数のバージョンに対応できるようにデータ・モデルを設計することもできます。

複雑な妥当性検査ロジックの設計

ビジネス・アプリケーションのトランザクションは、通常、読み取りおよび書き込みのアトミック操作の小規模なグループですが、複雑なものにすることもできます。例えば、非集中受注システムでは、トランザクションに以下の操作を含めることができます。

- CUST_ORDER 表に行を挿入する
- ORDER_LINE 表に複数行を挿入する
- CUSTOMER 表の USED_CREDIT 情報を更新する
- アプリケーションの帳簿部分の ACCOUNT_TRANSACTION 表にエントリーを挿入する
- オーダーされた各タイプの製品について、PRODUCT 表の行の STOCK_BALANCE 列を更新し、オーダーされた製品の在庫量を更新する

この例では、トランザクションをマスター・データベースで実行する際に、各種の問題が発生する可能性があります。例えば、以下が考えられます。

- マスター・データベースによると、顧客には、オーダーに使用できるクレジットが不足しています。したがってオーダー全体が無効であるか、または別途承認を受ける必要があります。

- オーダーされた製品の総数に対して、在庫が不足しています。その結果、オーダーの一部が未完了になります。オーダー全体を保留にすることが必要になるか、不足している製品に対して別のオーダー (バック・オーダー) が必要になります。
- 前月へのエントリーができなくなったときに、アカウンティング・トランザクションが月末後に伝搬されたため、このトランザクションは無効です。これによって、企業の帳簿システムとオーダー・システム間で不一致が発生します。

solidDB インテリジェント・トランザクションを使用しているときは、これらの問題を解決するためのさまざまな方法があります。そのうちの 2 つは、ここで説明している事前妥当性検査と補正という方法です。

事前妥当性検査

トランザクションの各操作は、妥当性検査操作と書き込み操作の 2 つの部分に分けられます。

書き込み部分のいずれかの前に妥当性検査部分を実行するようにトランザクションをインプリメントできます。妥当性検査部分は、パラメーター掲示板に、書き込み部分が正しく動作するために必要なすべての情報を残します。

上記の例で、トランザクションの妥当性検査部分は以下のようになります。

```
VALIDATE_ORDER
VALIDATE_ORDER_LINE (multiple)
VALIDATE_CREDIT_UPDATE
VALIDATE_ACCOUNT_TRANSACTION
VALIDATE_STOCK_UPDATE
```

この時点で、書き込み操作はまだ行われていませんが、パラメーター掲示板には、トランザクション全体を有効にするために書き込み操作が必要とする、すべての情報が存在しています。トランザクションの残りの部分は、以下のようになります。

```
INSERT_ORDER
INSERT_ORDER_LINE (multiple)
UPDATE_CUSTOMER_CREDIT
INSERT_ACCOUNT_TRANSACTION
UPDATE_PRODUCT_STOCK_BALANCE
```

補正

上記の問題を解決するもう 1 つの方法として、トランザクションの最後に補正操作を追加する方法があります。例えば、オーダー・ラインのいずれかでオーダーされた製品が、既に存在しない場合にこの操作を使用できます。結果として、オーダー全体が未完了になります。しかし、ORDER 表に対する行は、STATUS 列の値が 'OK' の状態で既に挿入されています。

この場合、VALIDATE_AND_INSERT_ORDER_LINE は、オーダー状態を 'INVALID' に変更するのに必要な、トランザクションの最後の操作 (COMPENSATE_ORDER) を通知するパラメーターを掲示板に残す必要があります。この例では、インテリジェント・トランザクションのインプリメンテーション全体は、以下のようになります。

```
VALIDATE_AND_INSERT_ORDER
VALIDATE_AND_INSERT_ORDER_LINE (複数)
VALIDATE_AND_UPDATE_CUSTOMER_CREDIT
VALIDATE_AND_INSERT_ACCOUNT_TRANSACTION
VALIDATE_AND_UPDATE_PRODUCT_STOCK_BALANCE
COMPENSATE_ORDER
```

アプリケーションにおけるエラー処理

データベースの同期はアプリケーションが行うので、アプリケーション、すなわちストアード・プロシージャー内でエラー処理もインプリメントされている必要があります。トランザクションは、システム・レベルやアプリケーション・レベルで表れるエラーを生成する可能性があります。システム・レベルのエラーは通常致命的なもので、自動的にはリカバリーできません。

アプリケーション・レベルのエラーは、トランザクションの元の動作が無効になった場合に発生します。一般的には、競合が検出された場合に発生します。例えば、マスター・データベースに既に存在しない顧客に対して、オーダーが挿入された場合などです。

この種類のエラーからリカバリーするには、以下に示すいくつかの方法があります。

- 「現行マスター・バージョンを優先する」といった競合解決ルールを使用して、トランザクション内でエラーを自動的に解決します。
- エラーの解決を、ユーザーまたはシステム管理者に委ねます。エラー・ログ表は、エラーに関する十分な情報を保管するために使用できます。
- 上記の方法を組み合わせます。例えば、競合を自動的に解決し、エラー・ログ表を使用した解決についてユーザーに通知します。

使用する方法は、アプリケーションとその要件によって異なります。

致命的エラーからのリカバリーの指定

エラー処理の最も重要なルールは、マスター・データベース内ですべてのトランザクションをコミットする必要があるということです。 すべての DBMS エラーは致命的エラーであり、デフォルトでは、同期メッセージの実行が停止します。これらのエラーは、システム･レベルのエラーです。 例えば、ユニーク制約違反により、同期中にマスター・データベース内で書き込み操作が失敗した場合、メッセージの実行が停止し、レプリカ・データベースにエラー・コードが返されます。

トランザクションのビジネス・ロジックで致命的エラーが検出された場合、そのトランザクションが異常終了し、トランザクションの掲示板にロールバック要求を書き込むことにより、それ以降の同期メッセージの実行が停止される可能性があります。 伝搬トランザクションでは、COMMIT WORK ステートメントと ROLLBACK WORK ステートメントは使用できないことに注意してください。ただし、以下のシステム掲示板パラメーターで、ロールバック要求を発行できます。

```
SYS_ROLLBACK = 'YES'
SYS_ERROR_CODE = user_defined_error_code
SYS_ERROR_TEXT = user_defined_error_text
```

SYS_ROLLBACK パラメーターは、システムに認識されるパラメーターです。トランザクションがこの値を 'YES' に設定すると、サーバーは、トランザクションを自動的にロールバックします。同じ同期メッセージのトランザクションのうち、既にコミットされたトランザクションは、コミットされたままになります。残りの同期メッセージに対する処理は、PROPAGATE TRANSACTIONS 操作のモードによって異なります。 (PROPAGATE TRANSACTIONS 操作のエラー処理モードについて詳しくは、『トランザクションの伝搬用コマンドの IGNORE_ERRORS、FAIL_ERRORS、および LOG_ERRORS フラグ』を参照してください。)

以下に、シナリオの例を示します。

トランザクションが、データベースの参照整合性の違反を検出したと想定します。例えば、オーダーの顧客がマスター・データベース内に存在しません。トランザクションは、ロールバックを要求してアプリケーション固有のエラー・コードを返すために、掲示板に以下のパラメーターを記入することができます。

```
Put_param('SYS_ROLLBACK', 'YES');
Put_param('SYS_ERROR_CODE', '90001');
Put_param('SYS_ERROR_TEXT', 'Referential integrity violation detected');
```

各トランザクションについてのトランザクション管理はプロシージャーの外部で実行されるため、プロシージャーの内部で直接 ROLLBACK WORK コマンドや COMMIT WORK コマンドを発行することは許されないことに注意してください。

トランザクションの伝搬用コマンドの IGNORE_ERRORS、FAIL_ERRORS、および LOG_ERRORS フラグ

トランザクションの伝搬中にエラーが発生した場合は、デフォルトの動作として、サーバーがメッセージの処理を停止し、現行のトランザクションを異常終了します。メッセージ内の以前のトランザクションは有効なままです。このため、メッセージの実行済み部分だけで終了することになる場合があります。

solidDB は、伝搬されるメッセージに対して 3 つのエラー処理モードをサポートしています。

- IGNORE_ERRORS - このオプションは、エラーが発生すると、トランザクションが異常終了することを意味します。次のトランザクションの実行は継続されます。つまり、エラーによってメッセージ全体が異常終了することはありません。
- LOG_ERRORS - IGNORE_ERRORS と同様に、このオプションは、エラーが発生するとトランザクションが異常終了し、次のトランザクションの実行が継続されることを意味します。さらに、失敗したトランザクションのステートメントは、後で実行または調査するために SYS_SYNC_RECEIVED_STMTS システム表に保存されます。
-

FAIL_ERRORS - このオプションは、ステートメントが失敗すると、現行のトランザクションがロールバックされ、サーバーが同じメッセージ内で後続のトランザクションの処理を継続しないことを意味します。(既にコミットされているトランザクションが取り消されることはありません。) これは伝搬に対するデフォルトのエラー処理モードです。

特定のメッセージやトランザクションに対して、上記のどのエラー処理モードを適用するのかを、以下の 3 つの方法で指定することができます。

- SAVE コマンドで適切なキーワードを使用します。SAVE コマンドでエラー処理モードを指定すると、指定されたモードが、(トランザクション全体やメッセージ全体ではなく) 保存されたステートメントにのみ適用されます。
- トランザクション掲示板パラメーターを使用して、エラー処理モードを指定します。この場合、指定したモードは現行のトランザクションに適用されます。
- MESSAGE APPEND PROPAGATE TRANSACTIONS コマンドを使用して、エラー時の動作を指定します。この場合、指定されたモードはメッセージ全体に適用されます。

エラー処理オプションを SAVE と PROPAGATE TRANSACTIONS ステートメントの両方で指定すると、PROPAGATE TRANSACTIONS ステートメントで指定したエラー処理オプションが優先されます。

失敗したメッセージは、SYNC_FAILED_MESSAGES システム・ビューを使用して検査できます。また、ステートメント MESSAGE <*msg_id*> FROM REPLICA <*replica_name*> RESTART <*error_options*> を使用して、そこから再実行することもできます。

パラメーター掲示板に値を設定するための構文を以下に示します。

エラー処理のためのパラメーター名は以下のとおりです。

```
SYNC_DEFAULT_PROPAGATE_ERRORMODE
```

以下の値を指定できます。

```
IGNORE_ERRORS
LOG_ERRORS
FAIL_ERRORS
```

マスターへの自動保存に使用するパラメーター名は以下のとおりです。

```
SYNC_DEFAULT_PROPAGATE_SAVEMODE
```

以下の値を指定できます。

```
AUTOSAVE
AUTOSAVEONLY
```

自動保存は、3 つ以上のレベルがある階層で使用されます。レプリカは、トランザクションを直接のマスターではなく、その上のマスターに伝搬する必要がある場合、AUTOSAVE を使用することができます。AUTOSAVE はこの章の後半で詳しく説明します。

例を以下に示します。

```
PUT_PARAM('SYNC_DEFAULT_PROPAGATE_ERRORMODE', 'LOG_ERRORS');
PUT_PARAM('SYNC_DEFAULT_PROPAGATE_SAVEMODE', 'AUTOSAVE');
```

メッセージに値を設定するための構文を以下に示します。

```
MESSAGE <message_name> APPEND PROPAGATE TRANSACTIONS
[ { IGNORE_ERRORS | LOG_ERRORS | FAIL_ERRORS } ] [WHERE ...]
```

自動保存オプションはこのステートメントでは使用できない点に注意してください。

構文 (保存時):

```
SAVE [NO CHECK] [ { IGNORE_ERRORS | LOG_ERRORS | FAIL_ERRORS } ]
[ { AUTOSAVE | AUTOSAVEONLY } ] <sqlstring>
```

- NO CHECK: このオプションは、ステートメントがレプリカで作成されないことを意味します。コマンドをレプリカで使用しても意味をなさない場合に有用なオプションです。例えば、マスターでは存在しても、レプリカでは存在しないストアード・プロシージャーを SQL コマンドが呼び出す場合は、レプリカでそのステートメントを作成する必要がありません。このオプションを使用すると、ステートメントにはパラメーター・マーカーを設定できません。
- IGNORE_ERRORS: このオプションは、マスターで実行中のステートメントが失敗した場合に、失敗したステートメントが無視され、トランザクションが異常終了することを意味します。ただし、メッセージ全体ではなく、そのトランザクションだけが異常終了します。マスターはメッセージの実行を継続し、失敗したトランザクションの次のトランザクションから実行を再開します。
- LOG_ERRORS: マスターで実行中のステートメントが失敗した場合、失敗したステートメントが無視され、現行のトランザクションが異常終了することを意味します。失敗したトランザクションのステートメントは、後で実行または調査するために SYS_SYNC_RECEIVED_STMTS システム表に保存されます。失敗したトランザクションは、SYNC_FAILED_MESSAGES システム・ビューを使用して検査できます。また、ステートメント MESSAGE <*msg_id*> FROM REPLICA <*replica_name*> RESTART を使用して、そこから再実行することもできます。IGNORE_ERRORS オプションと同様に、トランザクションを異常終了しても、メッセージ全体が異常終了することはない点に注意してください。マスターはメッセージの実行を継続し、失敗したトランザクションの次のトランザクションから実行を再開します。
-

FAIL_ERRORS: このオプションは、ステートメントが失敗すると、マスターがメッセージの実行を停止することを意味します。これはデフォルトの動作です。

- AUTOSAVE: このオプションは、ステートメントがマスターで実行され、そのマスターが他のマスターのレプリカでもある場合 (すなわち中間層のノードの場合)、さらに伝搬するためにそのステートメントが自動保存されることを意味します。
- AUTOSAVEONLY: このオプションは、ステートメントがマスターでは実行されないが、そのマスターが他のマスターのレプリカでもある場合 (すなわち中間層のノードの場合)、さらに伝搬するためにそのステートメントが自動保存されることを意味します。

マスター・データベースがメッセージを伝搬する場合、ノードがレプリカを兼ねていなければ、自動保存設定は無視されます。つまり、自動保存設定は、階層の最上部のマスターでは無視されます。設定の影響を受けるのは 1 ノードのみで、例えば、必要に応じて全ノードを個別に設定する必要があります。

例:

```
SAVE NO CHECK IGNORE_ERRORS insert into mytab values(1, 2)
```

sys_sync_master_msginfo 表には、主キーの一部である新しい列 FAILED_MSG_ID が設定されます。通常のメッセージの場合は、値はゼロになります。LOG_ERRORS オプションが ON でエラーがある場合は、値は msg_id になります。また、SYS_SYNC_RECEIVED_STMTS 表には、実際のエラーがログに記録される errcode 列と err_str 列があります。

自動保存オプションは、SAVE で定義されている場合、またはマスターの掲示板で定義されている場合に有効になります。

注:

レプリカ登録またはパブリケーション登録のメッセージが失敗した場合、ハングしたメッセージがマスターとレプリカの両方から自動的に削除され、メッセージの実行前の状態に戻ります。IGNORE_ERRORS、FAIL_ERRORS、および SAVE_ERRORS の各フラグは、この 2 つのタイプのメッセージには適用されません。

マスターでログに記録されたエラーの再実行または削除

トランザクションが LOG_ERRORS オプションを使用して伝搬された場合、サーバーは、失敗したトランザクション内のステートメントを保存します。これらのステートメントを検査し、場合によっては問題の原因を修正して、失敗したステートメントを再実行できます。

MESSAGE *msgid* FROM REPLICA *replicaname* RESTART <*err-options*>;

ここで、<*err-options*> は、IGNORE_ERRORS または LOG_ERRORS または FAIL_ERRORS です。

問題を訂正できない場合、または失敗したトランザクションを再実行しない場合は、メッセージ全体の削除、またはメッセージ内の 1 つ以上のトランザクションの削除ができます。

MESSAGE *msgid* FROM REPLICA *replicaname* DELETE;

MESSAGE *msgid* FROM REPLICA *replicaname* DELETE CURRENT TRANSACTION;

重要:

以下のコマンドは、削除コマンドとして使用しないでください。

```
MESSAGE message_name FROM REPLICA replicaname DELETE;
```

メッセージが、レプリカで認識されるライフ・サイクルを完了すると、message_name が有効でなくなるからです。*message_name* は、例えば以下のコマンドの後、システムから削除されます。

```
MESSAGE ... GET REPLY
```

ただし、メッセージがマスターでログに記録されている間、*msgid* は適用されます。

伝搬トランザクションからの同期メッセージの作成と送信

伝搬トランザクションから同期メッセージを作成して送信できます。多くの場合、伝搬トランザクション内で明示的なコミットを行うことは許されませんが、伝搬トランザクション内から同期メッセージを作成して送信する場合には明示的なコミットが許されます。伝搬トランザクションがまだ DML ステートメントを実行していない場合で、前のステートメントが以下のいずれかの場合には、伝搬トランザクションの途中で明示的な COMMIT を発行できます。

```
MESSAGE ... END
MESSAGE ... FORWARD、または
MESSAGE ... GET REPLY
```

6 分散システムのスキーマの更新および保守

この章は主要な 2 つのセクションに分かれています。最初のセクションでは、拡張レプリケーションを使用したいくつかの「機構」について説明します。2 番目のセクションでは、分散システム全体でのスキーマのアップグレード、すなわちマスターとそのレプリカ間で共用するスキーマのアップグレードにおける具体的な問題について述べます。

solidDB の表およびデータベースの管理

拡張レプリケーション・システムの最初のインプリメンテーションの後、データベース・スキーマの変更、新規マスターの追加、またはレプリカのドロップが必要になることがあります。このセクションでは、表とデータベースの管理手順を段階的に説明します。データベースの保守を実行する前に、データベース接続をすべて閉じるようにしてください。実行する保守のタイプによっては、すべてのデータベースを確実に同期化することが必要になる場合があります。

データベース・スキーマの変更

マスター・データベースのデータベース・スキーマを変更することができます (CREATE OR REPLACE PUBLICATION コマンドを参照してください。このコマンドは、パブリケーションの新規作成だけでなく既存のパブリケーションの変更にも使用できます)。また、パブリケーションが参照する表であっても、索引付けとユーザー・アクセス権限を変更することもあります。

マスター・データベースまたはレプリカ・データベースの場所の変更

マスターおよびレプリカのデータベースの場所は、データベース・ファイルとログ・ファイルをターゲット・ディレクトリーにコピーすることで、簡単に変更できます。

1.

   サーバーをシャットダウンし、データベース・ファイルとログ・ファイルのオペレーティング・システムのファイル・ロックを解除します。

2.

   データベース・ファイルとログ・ファイルをターゲット・ディレクトリーにコピーします。

3.

   solid.ini ファイルをターゲット・ディレクトリーにコピーします。

4.

データベース・ファイル・ディレクトリー、ログ・ファイル・ディレクトリー、およびバックアップ・ディレクトリーが、solid.ini 構成ファイルで正しく定義されていることを確認します。

5.

移動したデータベースがマスターの場合、すべてのレプリカで SET SYNC CONNECT コマンドを発行します。

これは、マスターに対する次のメッセージの前に接続を確保するために、すべてのレプリカで設定する必要があります。

6.

コマンド行オプション -c *directory_name* でターゲット・ディレクトリーを現行作業ディレクトリーとして使用し、新しい場所で solidDB を始動します。

7.

移動したデータベースがレプリカの場合、MESSAGE FORWARD または MESSAGE APPEND REGISTER REPLICA の接続ストリングを使用して、マスターにアクセスできることを確認します。

レプリカ・データベースの登録抹消

拡張レプリケーション・システムでレプリカ・データベースを使用しない場合は、レプリカ・データベースを登録抹消すること、すなわちレプリカ・データベースとマスター・データベース間の同期関係を解除することを強く推奨します。レプリカの登録抹消後、マスター・データベースは、このレプリカ・データベースの同期履歴データを蓄積する必要がなくなったことを認識します。これによってマスター・データベース内のディスク・スペースが大幅に節約できます。

レプリカ・データベースを登録抹消するには以下のようにします。

1.

レプリカ・データベースでデータが不要になった場合に、レプリカのサブスクリプションをドロップします。

2.

レプリカでは、以下のコマンドを使用してマスター・データベースでレプリカを登録抹消します。

```
MESSAGE message_name BEGIN;
MESSAGE message_name APPEND UNREGISTER REPLICA;
MESSAGE message_name END;
COMMIT WORK;
```

3.

メッセージをマスター・データベースに送信します。

```
MESSAGE message_name FORWARD TIMEOUT seconds;
COMMIT WORK;
```

レプリカ・データベースは、マスター・データベースと同期できなくなります。

以下のコマンドをマスター・データベースで使用して、レプリカ・データベースをマスター・データベースからドロップすることもできます。

```
DROP REPLICA replica_name;
```

DROP REPLICA を使用してレプリカをドロップする場合、DROP MASTER を使用してマスターもレプリカからドロップする必要があります。

レプリカ・データベースからマスター・データベースへのアクセスを拒否する必要がある場合、またはレプリカ・データベースが自身の登録抹消を正常に行えなかった場合に、この方法が必要になります。

大規模レプリカ・データベースの作成

マスター・データベースから 2 GB を超える大規模レプリカを作成する場合には、EXPORT SUBSCRIPTION コマンドと IMPORT コマンドを使用します。ただし、BLOb データの同期時には、同期メッセージごとに 2 GB という制限は適用されません。マスター・データベースから任意のサブスクリプションをファイルにエクスポートして、後でそのファイルをレプリカにインポートすることができます。詳しくは、『同期ブックマークによるデータ管理』を参照してください。

注:

レプリカは、マスター・データベースからのパブリケーションにサブスクライブすると作成できますが、データは 1 つの論理的なチャンクとして送信されるため、データが BLOb データではない場合、2 GB より大きなレプリカではこの方法は確実ではありません。大量のデータをダウンロードするにはかなりの時間を必要とすることがあるため、単一の拡張レプリケーション・メッセージで大規模データを送信するには制約があります。

同期ブックマークによるデータ管理

同期ブックマークとは、solidDB データベースのある状態を後で参照できるように定義することです。ブックマークは、マスター・データベースで以下のコマンドを使用して作成します。

```
CREATE SYNC BOOKMARK bookmark_name
```

このコマンドを実行すると、ブックマークの作成者、作成日時、ユニークなブックマーク ID など、他の属性が自動的に関連付けられます。

ブックマークは、データベースの単なるプレースホルダー以上のものです。ブックマークは、ある意味で、データベースの永続的なユーザー定義のスナップショットと考えることができます。マスターからデータをエクスポートするときにも、レプリカにデータをインポートするときにも使用できます。詳しくは、127 ページの『サブスクリプションのエクスポートとインポート』を参照してください。

ブックマークはマスター・データベースのみに作成することができます。レプリカ・データベースにブックマークを作成することはできません。作成を試みると、エラーが発生します。1 つのデータベースに作成できるブックマークの数には制限がないことに注意してください。

ブックマークの作成には、データベース管理者 (DBA) の権限または solidDB 管理者 (SYS_SYNC_ADMIN_ROLE) の権限が必要です。

ブックマークには、履歴バージョンを定義してあるすべての表の履歴情報が保持されます。そのため、ブックマークは必要なくなったら、除去することを推奨します。そうしないと、余分な履歴バージョンを収容するため、ディスク・スペース使用量が大幅に増えます。

ブックマーク情報のリトリーブ

新しいブックマークの作成または既存のブックマークのドロップを行う前に、solidDB カタログ表 SYS_SYNC_BOOKMARKS に照会して、既存のブックマークのリストを確認できます。例えば、以下の照会で、ブックマークの名前、作成日、および作成者を確認できます。

```
SELECT BM_NAME, BM_VERSION, BM_CREATOR, BM_CREATIME FROM SYS_SYNC_BOOKMARKS;
```

システム・カタログ表からブックマークをリトリーブするのに、管理特権は必要ありません。

ブックマークのドロップ

マスター・データベースでは、以下のコマンドでブックマークがドロップされます。

```
DROP SYNC BOOKMARK bookmark_name
bookmark_name ::= literal
```

注:

ブックマークは、エクスポートしたデータが、1 つのレプリカではなく、該当するすべてのレプリカにインポートされた後でのみドロップする必要があります。インポート後、マスターでブックマークをドロップする前に、レプリカでインポートしたサブスクリプションを一度リフレッシュする必要があります。インポートするレプリカがなくなってからのみブックマークをドロップします。

ブックマークを使用して、ファイルをレプリカに正常にインポートするか、マスター・データベースからデータの最初のリフレッシュを受信した後、データをファイルにエクスポートするために使用したブックマークをドロップすることを推奨します。サブスクリプションのインポートとエクスポートについて詳しくは、以下のセクションを参照してください。

ブックマークを残すと、削除や更新など、それ以降マスターのデータに対して発生する変更が、ブックマークごとにマスター・データベース上で追跡され、インクリメンタル・リフレッシュを行いやすくなります。

ブックマークをドロップすると、サーバーはブックマークで必要になっていた履歴データを削除できます。ブックマークをドロップしない場合は、マスター・データベースで登録されている各ブックマークで消費されるディスク・スペースが多くなります。その結果、パフォーマンスが低下する可能性があります。

詳しくは、「*solidDB SQL ガイド*」の SQL コマンド DROP SYNC BOOKMARK の説明を参照してください。

サブスクリプションのエクスポートとインポート

拡張レプリケーションの EXPORT SUBSCRIPTION コマンドによって、マスター・データベースから、レプリカ・データベースまたはディスク・ファイルにデータの特定のバージョンをエクスポートできます。データをディスク・ファイルにエクスポートした場合は、IMPORT コマンドでレプリカ・データベースにインポートできます。これらのコマンドでは、データベース内でサブスクリプションを作成し、また、エクスポートするデータベースの状態を参照するためのブックマークを作成していることが前提になっています。

エクスポート用サブスクリプションの指定

EXPORT SUBSCRIPTION コマンドの使用に関する概念と手順は、パブリケーションからのリフレッシュに類似しています。以下の環境で、MESSAGE APPEND REFRESH コマンドまたは REFRESH コマンドの代わりに EXPORT SUBSCRIPTION コマンドを使用します。

- 既存のマスターから、大規模なレプリカ・データベースを作成する必要があります。この手順では、まずデータのあるサブスクリプションまたはデータのないサブスクリプションをファイルにエクスポートし、次にそのサブスクリプションをレプリカにインポートする必要があります。詳しくは、129 ページの『データのあるサブスクリプションのエクスポートによるレプリカの作成』または 132 ページの『データのないサブスクリプションのエクスポートによるレプリカの作成』を参照してください。
- 特定バージョンのデータをレプリカにエクスポートする必要があります。
- メタデータ情報だけを、実際の行データなしでエクスポートする必要があります。

EXPORT SUBSCRIPTION コマンドを使用する場合と、MESSAGE APPEND REFRESH または REFRESH でパブリケーションからリフレッシュする場合の違いに注意してください。

- EXPORT SUBSCRIPTION コマンドはマスターで実行されますが、リフレッシュはレプリカから要求されます。
- エクスポート出力はユーザー指定のファイルに保存されますが、REFRESH コマンドの出力は拡張レプリケーション応答メッセージに格納されます。
- エクスポート・ファイルは、データを入れないで (実際の行が出力に含まれない) 作成することも、データを入れて作成することもできます。
-

エクスポート・ファイルは、インクリメンタルなものになることはありません (例えば、エクスポート用データに行が存在する場合、フル・パブリケーションに基づくリフレッシュと同様に、エクスポート・ファイルにすべての行が含まれます)。

- エクスポート・ファイルは、指定されたブックマークに基づいています。これは、エクスポート・データは指定されたブックマークまで整合性を保っており、インクリメンタル・パブリケーションに基づくリフレッシュがそのブックマークから可能であることを意味しています。

EXPORT SUBSCRIPTION コマンド

データは、2 つの異なる方法でマスター・データベースからレプリカ・データベースにエクスポートできます。

後で 1 つまたは複数のレプリカにインポートするために、マスター・データベースからファイルにサブスクリプションのデータをエクスポートする場合には、以下の EXPORT SUBSCRIPTION 構文を使用します。

```
EXPORT SUBSCRIPTION publication_name [( arguments )]
   TO 'filename'
   USING BOOKMARK bookmark_name
   [WITH [NO] DATA];
```

ファイルへのエクスポート操作が完了すると、IMPORT コマンドで、そのファイルのデータをレプリカ・データベースにインポートできます。

マスター・データベースから、指定した既存のレプリカ・データベースにデータを直接エクスポートする場合には、以下の構文を使用します。

```
EXPORT SUBSCRIPTION publication_name [( arguments )]
   TO REPLICA replica_name
   USING BOOKMARK bookmark_name
   [COMMITBLOCK #rows] ;
```

EXPORT SUBSCRIPTION TO REPLICA コマンドは、マスター・データベースからレプリカ・データベースにデータを転送する手段として、ファイルを使用しないことに注意してください。その代わりに、レプリカ・データベースにデータを直接書き込みます。そのため、独立したインポート手順は必要ありません。ネットワーク上にレプリカ・データベースが存在し、使用可能になっている必要があります。

publication_name と *bookmark_name* は、データベース内に存在する必要のある ID です。filename は、単一引用符で囲まれたリテラル値を表しています。1 つのエクスポート・ファイルは、複数のサブスクリプションを含むことができます。サブスクリプションは、「WITH DATA」オプションと「WITH NO DATA」オプションでエクスポートできます。複数のパブリケーションが指定されている場合には、エクスポート・ファイルには、「WITH DATA」と「WITH NO DATA」の組み合わせを指定できます。

使用のルールを含めて、詳しくは、「*solidDB SQL ガイド*」の SQL コマンド「EXPORT SUBSCRIPTION TO <*file*>」と「EXPORT SUBSCRIPTION TO REPLICA」の説明を参照してください。WITH DATA でレプリカを作成する手順については、129 ページの『データのあるサブスクリプションのエクスポートによる

レプリカの作成』、WITH NO DATA でレプリカを作成する手順については、 132 ページの『データのないサブスクリプションのエクスポートによるレプリカの作成』を参照してください。

インポート用サブスクリプションの指定

レプリカ・データベース上で IMPORT コマンドを使用して、EXPORT SUBSCRIPTION コマンドで作成したデータ・ファイルからデータをインポートできます。

IMPORT コマンド

IMPORT コマンドは、以下の構文で指定します。

```
IMPORT 'filename' [COMMITBLOCK #rows]
```

filename は、単一引用符で囲まれたリテラル値を表しています。インポート・コマンドは、単一のファイル名のみを受け入れることができます。レプリカへのインポート用パブリケーション・データは、すべて単一ファイルに格納する必要があります。ただし、複数のインポート・ステートメントを使用すると、複数のファイルをインポートできます。

#rows は、オプションの COMMITBLOCK 節で使用する整数値であり、コミット・ブロック・サイズを指定します。

COMMITBLOCK 節は、データをコミットする前に処理する行の数を示します。COMMITBLOCK を指定しないと、IMPORT コマンドは、パブリケーション内のすべての行を 1 つのトランザクションとして取り込みます。ファイルに多数の行が存在する場合には、COMMITBLOCK の使用を推奨します。

使用ルールを含めて、詳しくは、「*solidDB SQL ガイド*」の「IMPORT」コマンドの説明を参照してください。

データのあるサブスクリプションのエクスポートによるレプリカの作成

マスター・データのサブセットが必要だが、そのデータがまだない既存の (レプリカ) データベースが存在する場合、WITH DATA オプションを使用して EXPORT SUBSCRIPTION コマンドを実行することにより、そのレプリカにデータをエクスポートします。

以下の手順では、エクスポート・ファイル (複数可) にデータを入れ、IMPORT コマンドでそのデータをレプリカにロードする必要があります。

注:

- EXPORT SUBSCRIPTION コマンドを使用すると、同じファイルに 2 回以上データをエクスポートできます。コマンドごとに、データがファイルに追加されます。この作業を計画する場合、各エクスポート・コマンドでエクスポートされるデータを入れるのに十分なディスク・スペースを確保してください。エクスポー

トの途中でディスク・スペースを使い尽くすと、エラーが発生して EXPORT SUBSCRIPTION コマンドが失敗し、そのエクスポート・ファイルは使用できなくなります。

•

solidDB では、EXPORT SUBSCRIPTION コマンドの使用時には、自動コミットをオフに設定する必要があります。

マスターでの手順

マスター・データベースで、以下の手順を実行します。

1.

ブックマークが存在しない場合、ブックマークを作成します。ブックマークが既に存在し、要求を満たしている場合には、それを使用します。ブックマークの作成については、 125 ページの『同期ブックマークによるデータ管理』を参照してください。

また、照会を実行して、ご使用のシステム内に現在存在するブックマークとパブリケーションを確認できます。 126 ページの『ブックマーク情報のリトリーブ』を参照してください。

2.

必要なパブリケーションごとに WITH DATA オプションを指定して EXPORT SUBSCRIPTION コマンドを実行し、エクスポート・ファイル (複数可) を作成します。

マスターで、複数のパブリケーションに関連付けられたブックマークが存在する場合、パブリケーションごとに別々に EXPORT SUBSCRIPTION コマンドを実行するようにしてください。

WITH DATA オプションを指定した各 EXPORT SUBSCRIPTION コマンドで、そのパブリケーションとブックマークに対応するメタデータとバージョン管理されたデータがエクスポート・ファイルに追加されます。

エラー・メッセージ

ディスク・スペースを使い尽くしたことを示すエラーを受け取った場合、以前のファイルを削除し、十分なディスク・スペースを準備してもう一度 EXPORT SUBSCRIPTION コマンドを実行します。

注:

EXPORT SUBSCRIPTION コマンドを中断して、再開することはできません。何らかの理由で実行が完了しなかった場合、EXPORT SUBSCRIPTION コマンドをもう一度実行する必要があります。

以下のエラーが発生する可能性があります。

•

エラー・メッセージ 25067 は、拡張レプリケーション・ブックマークが見つからないことを示しています。ブックマーク名を正しく入力したことを確認してください。

- エラー・メッセージ 25068 は、EXPORT SUBSCRIPTION コマンドまたは IMPORT コマンドで名前を指定したファイルを開くことができないか、または追加モードで開くことができないことを示しています。ファイル名を正しく入力したこと、およびそのファイルが現在使用中でないことを確認してください。

レプリカでの手順

レプリカ・サイトで、以下の手順を実行します。

1. このレプリカをマスター・データベースに登録します。
2. SET CATALOG コマンドを使用して、レプリカ・カタログを現行カタログに設定します。
3. サブスクリプション (複数可) をインポートするパブリケーションを登録します。
4. EXPORT SUBSCRIPTION コマンドで作成したファイルをインポートします。インポートの手順は、 129 ページの『インポート用サブスクリプションの指定』で説明されています。

   IMPORT コマンドは、一度に 1 つのファイルのみを受け入れます。エクスポート・ファイルが複数存在する場合には、ファイルごとに別々に IMPORT コマンドを実行します。1 つのファイルに複数のエクスポートが含まれる場合があることに注意してください。

   FULL パブリケーション・オプションでサブスクライブするのと同じ効果が得られるように、同じレプリカに同じサブスクリプションを複数回インポートできます。

   注:

   IMPORT コマンドを中断して、再開することはできません。何らかの理由で実行が完了しなかった場合、IMPORT コマンドをもう一度実行する必要があります。
5. 拡張レプリケーションの MESSAGE コマンド (例えば MESSAGE APPEND REFRESH) または REFRESH コマンドを使用して、マスター・データベースからパブリケーションをリフレッシュします。

データのないサブスクリプションのエクスポートによるレプリカの作成

場合によっては、データがすべてデータベースに格納されているものの、そのデータベースがまだレプリカとして構成されていないことがあります。例えば、既存のマスターのバックアップ・コピーを格納したサーバーが存在し、そのコピーをレプリカに変換したい場合があります。適切なデータが既にデータベースに存在する場合、そのデータをすべて廃棄してリフレッシュするのは望ましくありません。1 つの解決方法は、マスターからサブスクリプション・メタデータのみをエクスポートし、次にそれをマスターのコピーにインポートすることです (つまり、レプリカに変換したいデータベースにインポートします)。WITH NO DATA オプションを指定して EXPORT SUBSCRIPTION コマンドを実行し、スキーマ (「メタデータ」) をエクスポートします。

注意:

このオプションは、レプリカに有効なデータが存在するのを確認してから使用してください。レプリカにアクセスするアプリケーションが、誤ったデータ集合を使用してしまう恐れがあります。

注:

solidDB では、EXPORT SUBSCRIPTION コマンドの使用時には、自動コミットをオフに設定する必要があります。

以下の手順では、データのないファイル (複数可) をエクスポートし、IMPORT コマンドで、メタデータとパブリケーション情報を格納したファイルをレプリカにロードする必要があります。

マスターでの手順

マスター・データベースで、以下の手順を実行します。

1.

   ブックマークが存在しない場合、ブックマークを作成します。ブックマークが既に存在し、要求を満たしている場合には、それを使用します。ブックマークの作成については、 125 ページの『同期ブックマークによるデータ管理』を参照してください。

   また、照会を実行して、ご使用のシステム内に現在存在するブックマークとパブリケーションを確認できます。 126 ページの『ブックマーク情報のリトリーブ』を参照してください。

2.

   必要なパブリケーションごとに WITH NO DATA オプションを指定して EXPORT SUBSCRIPTION コマンドを実行し、エクスポート・ファイル (複数可) を作成します。

   複数のパブリケーションがエクスポートされる場合でも、ブックマークは 1 つだけ必要です。同じファイル名を指定することにより、複数のパブリケーションを単一ファイルにエクスポートできます。

WITH NO DATA オプションを指定した各 EXPORT SUBSCRIPTION コマンドでは、そのパブリケーションとブックマークに対応するメタデータと履歴データがエクスポート・ファイルに追加されます。

レプリカでの手順

該当する各レプリカ・サイトで、以下の手順を実行します。

1.

既存のマスター・データベースのバックアップを作成します (例えば、ADMIN COMMAND 'backup' コマンドを使用します)。

2.

バックアップ・データベースを開始して、DROP REPLICA コマンドを使用してすべてのレプリカをドロップし、DROP PUBLICATION コマンドを使用してすべてのパブリケーションをドロップします。

3.

SET SYNC NODE *unique_node_name* コマンドを使用してノード名を変更します (そのデータベースは既存のマスター・データベースのバックアップであり、現在、オリジナル・マスターがそのノード名を使用しているため)。

4.

以下のコマンドを実行して、データベースをレプリカとして構成します。

```
SET SYNC MASTER NO
SET SYNC REPLICA YES
```

注:

データベースをレプリカに切り替える場合で、そのデータベースが既にマスター・データベースのバックアップ・コピーである場合には、そのバックアップは既にマスター・データベースとして設定されています。SET SYNC REPLICA YES コマンドを実行すると、データベースは、レプリカのみの専用データベースとしてではなく、2 つの役割 (マスターとレプリカ) を持つデータベースとして定義されます。このデータベースを排他的にレプリカにする場合には、SET SYNC MASTER NO も SET SYNC REPLICA YES も実行する必要があります。

5.

このレプリカをマスター・データベースに登録します。

6.

EXPORT SUBSCRIPTION コマンドで作成したファイルをインポートします。インポートの手順は、 129 ページの『インポート用サブスクリプションの指定』で説明されています。

7.

拡張レプリケーションの MESSAGE コマンドまたは REFRESH コマンドを使用して、マスター・データベースからパブリケーションをリフレッシュします。

新しく作成したレプリカは、すぐに使用できます。これが作成する最後のレプリカである場合、 126 ページの『ブックマークのドロップ』の説明に従ってマスター・データベースからブックマークをドロップします。

パブリケーションおよびパブリケーション内の表の変更

既存のパブリケーションを変更することができます。詳しくは、「*solidDB SQL ガイド*」の CREATE [OR REPLACE] PUBLICATION コマンドの説明を参照してください。このコマンドは、パブリケーションを新規作成できるだけでなく、既存のパブリケーションを変更することもできます。

パブリケーションの中にある表を変更することもあります。詳しくは、「*solidDB SQL ガイド*」の SET SYNC MODE コマンドの説明を参照してください。

加える変更に応じて、各レプリカからの次のリフレッシュがインクリメンタル・リフレッシュまたはフル・リフレッシュのいずれかになる可能性があります。

注:

拡張レプリケーションの既存パブリケーションの内容を変更する際に、CREATE OR REPLACE PUBLICATION を使用する場合は、レプリカから無効な行を削除する必要があります。

インクリメンタル・リフレッシュとフル・リフレッシュ

コマンド「CREATE OR REPLACE PUBLICATION...」を使用してパブリケーションを変更した場合には、レプリカは次回のリフレッシュ時に更新データを受け取ります。パブリケーションの変更内容に応じて、サブスクライバーはフル・リフレッシュまたはインクリメンタル・リフレッシュのいずれかを取得します。以下は、次回のリフレッシュがインクリメンタル・リフレッシュになるのか、またはフル・リフレッシュになるのかを制御するルールの要約です。単一のパブリケーションの中に、複数の結果セットが格納される場合があることに留意してください。それぞれの結果セットは比較的無関係であるため、1 つの結果セットを変更したときに、サブスクライバーが他の結果セットの情報を収集する必要があるとは限りません。

- パブリケーションに新しい結果セットを追加する場合には、インクリメンタル・リフレッシュのみが必要になり、新しい結果セットのデータがサブスクライバーに送信されます。
- 結果セットをドロップする場合には、ネットワークを介してデータを送信する必要はありません。
- 結果セットを変更する場合には、通常、その結果セットの完全なデータが必要になります。
-

パブリケーションをドロップし、再作成する場合、(CREATE OR REPLACE を使用した「置き換え」ではなく)、サブスクライバーは、新しいパブリケーションに再登録し、最初のリフレッシュ時にフル・リフレッシュを取得します。

注:

別の既存のマスター・データベースを使用するためにレプリカを切り替えることはできません。このような操作を行うと、インクリメンタル・パブリケーションの同期に不一致が生じます。これを行う必要がある場合は、データベースを最初から作成し直し、新しいマスター・データベースを定義して、表、プロシージャー、およびパブリケーションを再作成しなくてはなりません。

インテリジェント・トランザクションの SQL プロシージャーの変更

トランザクションの SQL ストアード・プロシージャーは、その呼び出しインターフェースによってのみ識別されます。そのため、呼び出しインターフェースが同じままでさえあれば、プロシージャーを好きなように変更することができます。プロシージャーの呼び出しインターフェース (つまりパラメーター・リスト) を変更する場合は、通常、そのプロシージャーの名前も変更する必要があります。この場合、マスター・データベースで処理中のすべてのトランザクションをそこで正常に実行できるように、前のバージョンのプロシージャーをマスター・データベースの中に使用可能なまま残しておく必要があります。

分散システムのスキーマのアップグレード

概要

拡張レプリケーション・アーキテクチャーでは、マスターとレプリカ (複数可) との間でデータを分散することができます。マスターがパブリケーションを作成し、レプリカがそのパブリケーションにサブスクライブします。

場合によっては、パブリケーションが使用する表を変更してそのパブリケーションのスキーマを変更する必要があります。例えば、新しい列を既存の表に追加するときに必要です。

solidDB には、スキーマの管理および同期化の機能が揃っているため、ご使用のシステムでマスター・データベースとレプリカ・データベースのスキーマを変更できます。solidDB のスキーマ管理機能のスキーマ・アップグレード処理は完全には自動化されていませんが、強力なプログラミング・ツール・セットが揃っているので、アプリケーションのニーズに最適なソリューションを構築できるようになっています。つまり、マスター・データベースで新しい表を作成するか、既存の表を変更したとしても、その表はレプリカ・データベースで自動的に作成または変更されるわけではありません。システム管理者は、solidDB を使用して、表をレプリカ・データベースで変更する必要があるのかどうか、また変更する必要があるとしたら、どのタイミングで変更する必要があるのかを決定することができます。

この章では、「パブリケーションの保守モード」(単に「保守モード」とも言う) で大まかに分類される 4 つの機能について説明します。これらの機能の主な目的は、

分散システムのスキーマのアップグレード (パブリケーションに使用される表の構造の変更) に必要な作業を軽減することです。

保守モードの機能を使用しない場合、その表を参照しているパブリケーションを最初にドロップしない限り、表を変更することはできません。パブリケーションの再作成では、すべてのレプリカにより、その表を含むパブリケーションからの次回のリフレッシュ時に、インクリメンタル・リフレッシュではなく、フル・リフレッシュが強制的に実行されます。これは、パブリケーションがその表の列のサブセットのみを使用しており、実行した ALTER TABLE コマンドがそれら列に影響を与えなかったとしても同じです。

保守モードを使用すると、表の変更後に行わなければならないフル・リフレッシュを多くの場合に回避することができます。また、保守モードでは、データベースに大規模な保守上の更新を行っても、スキーマのアップグレード後に大量のデータの同期化が行われることはありません。保守モードでは、マスター・データベースにデータの書き込み操作を行っても、同期化は一度も実行されません。その代わり、マスター・データベースのスキーマとデータを変更したスクリプトと同種類のスクリプトにより、同じ更新がレプリカ・データベースでも行われることが考えられます。

この章には、スキーマ・アップグレード機能のインプリメント例が記載されています。この章の内容を大体理解できるように、solidDB 同期化機能 (パブリケーションを含む) について理解しておく必要があります。例で使用されている各種の SQL ステートメントについて詳しくは、「*solidDB SQL ガイド*」を参照してください。

主な特徴および機能

以下に示すのは、分散システムのバージョンアップを可能にする機能で、管理者が制御できます。

- 同期モード: 同期モードを保守に設定することで、パブリケーションが参照する表のスキーマ変更 (DDL 操作) が可能になります。また、「同期履歴」トラッキングが一時的に無効になります。
- CREATE PUBLICATION コマンドの「REPLACE」オプション: 既存のパブリケーションを変更するオプションで、変更後にフル・リフレッシュが必ずしも必要ではありません。可能であれば、変更後にインクリメンタル・リフレッシュが許可されることもあります。
- 表レベル・ロック方式: 表全体を明示的にロックすることができます。こうしておけば、スキーマの変更が容易になります。
- スキーマ・バージョン・トラッキング: マスターまたはレプリカのいずれかが永続的なカタログ・レベル・パラメーター SYNC_APP_SCHEMA_VERSION (SET SYNC PARAMETER コマンド使用時) を設定している場合、2 つのサーバーで、

両方のバージョン値が同じでない限り、同期は拒否されます。したがって、レプリカとマスターのスキーマが一致しない場合は、同期は行われません。

これら 4 つの機能について、以下で詳しく説明します。この章の後半部分では、上記の機能を使用した分散システムのスキーマの変更方法について説明します。分散システムのスキーマの変更とは、つまり、マスター・データベースとレプリカ・データベースの両方のスキーマを変更することです。

同期モード

同期モードを「Maintenance」に設定すると、パブリケーションが参照する表に対するスキーマの変更 (DDL 操作) が許可されます。カタログの同期モードが Maintenance ではない場合、サーバーは、パブリケーションが使用する表に対する DDL 操作を禁止します。これは、表に対する変更 (例えば、新しい列の追加) がパブリケーションに影響しない場合であっても、表を変更するためには、パブリケーションをドロップし、表を変更し、パブリケーションを再作成する必要があることを意味しています。パブリケーションのドロップと再作成を行うと、インクリメンタル・リフレッシュではなくフル・リフレッシュをレプリカに強制的に取得させるため、表を変更するたびにレプリカにフル・リフレッシュを強制的に取得させることになります。

同期モードを Maintenance に設定すると、パブリケーションをドロップすることなく、したがって、必ずしもフル・リフレッシュを強制することなく、表を変更できます。

また、同期モードを Maintenance に設定すると、同期履歴が一時的に無効になります。言い換えると、レプリカがデータのリフレッシュを要求したときに、どのレコードをレプリカに送信するかの判断に使用するデータを格納しないように、サーバーに指示します。これにより、あるタイプの主要なデータ変更 (DML) を迅速に行うことができます。マスターとレプリカに同じ更新を行い、同期手順を単純にスキップオーバーできます。

ただし、同期履歴が無効になっているときに誤ってマスターとレプリカを「同期しない」状態にすると、次回レプリカがリフレッシュしたときに、マスターとレプリカは自動的には再同期 (エラーを修正) しません。マスター上で行われた変更を示す同期履歴が存在しないので、マスターには、レプリカに更新を送信する理由がありません。レプリカは、同期しなくなった場合、無期限にまたは次回パブリケーションをフル・リフレッシュするまで、同期しない状態のままになります。同期モードを Maintenance に設定した場合、マスター・データベースとレプリカ・データベースの変更は、十分に注意して行う必要があります。

同期モードを Normal (デフォルト値) に戻すと、サーバーは同期履歴情報の追跡を再開し、またパブリケーション内の表に対する DDL 操作の許可を停止します。

同期モード Maintenance では、表の変更時に、レプリカがフル・リフレッシュを取得することは不要であるという保証がないことに注意してください。表に対する変更 (例えば、パブリケーションで使用される列のドロップ) によっては、その表 (またはパブリケーション) に、レプリカによるフル・リフレッシュの取得が必要となるような影響が発生する場合があります。

CREATE PUBLICATION コマンドの「REPLACE」オプション:

REPLACE オプションの利点は、レプリカに必ずしも強制的に再登録とフル・リフレッシュの取得を行わせなくても、パブリケーションを変更できる点です。場合によっては、レプリカはインクリメンタル・リフレッシュを引き続き取得することができます。

REPLACE オプションを使用しない場合、パブリケーションの変更が必要になるたびに、そのパブリケーションをドロップして再作成する必要があります。パブリケーションがドロップされ、再作成されると、レプリカはそのパブリケーションに再登録し、フル・リフレッシュを取得しなければなりません。

ただし、REPLACE オプションを使用すると、既存のパブリケーションを変更できます。パブリケーションのドロップと再作成を行わなくても、既存のパブリケーション定義を変更できます。パブリケーションのドロップと再作成を行わないので、レプリカはフル・リフレッシュの取得を強制されずに、インクリメンタル・リフレッシュを引き続き取得することができます。

REPLACE オプションでは、パブリケーションの変更時に、レプリカがフル・リフレッシュを取得することは不要であるという保証がない点に注意してください。パブリケーションに対する変更によっては、レプリカによるフル・リフレッシュの取得が必要になるほど重大な場合があります。また、REPLACE ステートメントで実行できない操作もあります。例えば、REPLACE では、パブリケーションの引数リストを変更できません。パブリケーションの引数リストを変更する必要がある場合は、パブリケーションをドロップし、再作成しなければならず、当然、レプリカは再登録とフル・リフレッシュの取得が必要になります。

注: REPLACE オプションと SYNC MODE MAINTENANCE 設定では、いずれもレプリカによるフル・リフレッシュの取得を必ずしも必要としない変更を行うことができます。2 つの違いは、SYNC MODE MAINTENANCE では表のスキーマを変更でき、REPLACE オプションではパブリケーションを変更できるという点です。

表レベル・ロック方式

表レベル・ロック方式では、表全体を明示的にロック (およびロックを解除) することができます (例えば、重要なスキーマのアップグレード操作の間)。これによってスキーマのアップグレード操作が、対象の表に対する他の操作によって妨げられないようにすることができます。

表は EXCLUSIVE モードまたは SHARED モードでロックできます。表のスキーマの変更を計画する場合は、表を EXCLUSIVE モードでロックしておくとよいでしょう。他の排他ロックと同様に、他のユーザーがその表をロックしている場合は、排他ロックをかけることができないことに注意してください。使用中のシステムでは、表に排他ロックをかけるのは非常に困難な可能性があります。他のユーザーに一時的にシステムの使用を停止してもらい (あるいは少なくとも表の使用を停止してもらい)、その表にロックをかけられるようにする必要があります。当然のことながら、排他ロックをかけたら、ロックを解除するまで、他のユーザーがその表を使用することはできません。

ほとんどの場合、トランザクションの最後にロックが解除されます。ただし、LOCK TABLE コマンドを使用することで、トランザクション終了後にロックを維持するこ

ともできます。トランザクション終了後もロックを維持する場合は、明示的に UNLOCK で表のロックを解除する必要があります。そうしないと、ロックをかけたクライアント・アプリケーションが切断するまで、ロックが維持されます。

表レベル・ロック方式は主として、スキーマのアップグレードをさらに簡単で安全に行うために使用しますが、他の目的でも使用できる点に注意してください。同期モードが Maintenance に設定されているときだけでなく、表ロックはいつでも使用することができます。

EXCLUSIVE でロックをかけたり、現行のトランザクション終了後も維持されるロックをかけたりするための正確な構文など、詳しくは、「*solidDB SQL ガイド*」の SQL コマンド LOCK TABLE および UNLOCK TABLE に関する説明を参照してください。

SYNC_APP_SCHEMA_VERSION を使用したバージョン確認

マスターとレプリカのスキーマを変更する場合、通常は同時に変更することはできません。一般的には、マスターでスキーマを変更し、レプリカでは変更しなかった場合、レプリカでもスキーマが更新されるまで、レプリカをリフレッシュする必要はありません。SYNC_APP_SCHEMA_VERSION という名前の掲示板パラメーターを使用して、マスターとレプリカに同じスキーマがない場合は同期を禁止することができます。

マスターまたはレプリカのいずれかにこのパラメーターが設定されている場合、マスターとレプリカは同期を行おうとするときに、このパラメーターの値を比較します。マスターの値がレプリカの値と一致しない場合は、マスターとレプリカは同期を拒否します。その代わりに、レプリカは同期メッセージの格納だけを行います。レプリカのスキーマを更新後、MESSAGE FORWARD コマンドを使用してこのメッセージを再送できます。

例えば、SYNC_APP_SCHEMA_VERSION をマスターでは「Version2」に設定し、レプリカでは「Version1」に設定してある場合、マスターとレプリカは同期を拒否します。

マスター・サーバーとレプリカ・サーバーはこのパラメーターの値を比較するだけで、実際のスキーマの比較は行わない点に注意してください。両方のサーバーに実際には同じスキーマがないのに、マスターとレプリカの SYNC_APP_SCHEMA_VERSION を同じ値に間違って設定してしまった場合、マスターとレプリカは同期を試みます。

サーバーでは、この掲示板パラメーターは、設定されている場合に限り使用されます。この掲示板パラメーターはオプションで、自動的には設定されません。デフォルト値はなく、サーバーはスキーマが更新されるたびに値を自動的には「インクリメント」しません。実際の値には意味がありません。マスターとレプリカが同じ値を持っているか違う値を持っているかだけが重要です。
「Version1」、「VersionA」、「XYZ」やその他の任意のストリングなどの任意の値を (当然ながら、掲示板パラメーター値の最大長まで) 使用できます。

分散スキーマを更新する機能の使用

最初の 3 つの機能 (SYNC MODE MAINTENANCE、CREATE OR REPLACE PUBLICATION での REPLACE オプション、および表レベル・ロック方式) を組み

合わせることで、パブリケーションを使用する各レプリカのフル・リフレッシュを必要とせずに、そのパブリケーション内の表の構造を更新できます。

パブリケーション内の表を更新するために可能な 1 つのプロセスの概要を以下に示します。最初にマスターをアップグレードし、次にレプリカをアップグレードします。

1. カタログの同期モードを Maintenance に設定して、同期履歴機能を一時的にオフにします。
2. パブリケーション内の表または表集合をロックします。
3. 必要な表またはスキーマの変更を行います。例えば、表の変更、新しい表の追加、またはパブリケーションの変更を行います。
4. 掲示板パラメーター SYNC_APP_SCHEMA_VERSION の値を更新します。
5. アプリケーション・プログラムをアップグレードします (必要な場合)。
6. 表のロックを解除します。
7. カタログの同期モードを Maintenance から Normal に戻します。

マスターのアップグレード後、ほとんど同じプロセスを使用してレプリカを更新します。

以下では、分散システムのスキーマの作成と更新のための標準的な手順を説明します。最初に (Maintenance モード機能を使用しないで) スキーマを作成する方法を説明し、次に (Maintenance モード機能を使用して) スキーマをアップグレードする方法を説明します。

分散スキーマのアップグレードの例

初期スキーマの作成

分散システムを最初に構築する場合は、システム全体の管理者は、データベースのタイプ (マスター、レプリカ) ごとに、データベース・カタログのスキーマを作成するスクリプト (ストアード・プロシージャーの場合もあります) のセットを定義します。これらのスクリプトは、プロシージャー、トリガー、イベント、パブリケーションなど、スキーマの全データベース・オブジェクトの作成に使用されます。レプリカでは、スクリプトの多くがマスターと同じになることがあります。例えば、CREATE TABLE コマンドの一部、またはすべてがレプリカでもマスターでも同じになることがあります。マスターとレプリカで異なるスクリプトもあります。例え

ば、パブリケーションを作成するスクリプトはマスター上でのみで実行しますが、レプリカを登録するスクリプトはレプリカ上でのみ実行します。

同期可能なカタログごとに、そのプロパティーの 1 つとして、スキーマ・バージョン名を持つことができます。マスター・カタログとレプリカ・カタログのスキーマ・バージョン名が違う場合には、レプリカからマスターへの同期メッセージの送信は失敗します。このプロパティーの名前は、SYNC_APP_SCHEMA_VERSION です。このバージョン名は、SET SYNC PARAMETER ステートメントを使用して設定できます。

初期マスター・スキーマの作成

初期マスター・データベースを作成するスクリプトは、以下のとおりです。

```
CREATE TABLE MYTABLE (
   ID INTEGER NOT NULL PRIMARY KEY,
   STATUS INTEGER NOT NULL,
   TEXTDATA VARCHAR NOT NULL);

ALTER TABLE MYTABLE SET SYNCHISTORY ;
COMMIT WORK ;

"CREATE PUBLICATION
MYPUBLICATION
BEGIN
   RESULT SET FOR MYTABLE
   BEGIN
      SELECT * FROM MYTABLE ;
   END
END";
COMMIT WORK ;

SET SYNC PARAMETER SYNC_APP_SCHEMA_VERSION 'VER1';
COMMIT WORK ;
```

レプリカ・スキーマの作成

各レプリカを最初に作成する場合、レプリカの管理者は、レプリカ・サーバーに接続し、現在のカタログを設定し、スクリプトを実行します。スキーマ作成プロセスの一環として、レプリカをマスターに登録します。

このプロセスが完了すると、マスター・カタログとレプリカ・カタログのバージョン名は同じになります。つまり、両方のスキーマは相互に互換性があることになります。マスターとレプリカの両方で同期パラメーター SYNC_APP_SCHEMA_VERSION を同じ値に設定して、両方のスキーマに互換性があることを相互に認識できるようにしてください。

レプリカの場合には、初期スキーマを作成するスクリプトは以下のようになります。

```
CREATE TABLE MYTABLE (
   ID INTEGER NOT NULL,
   STATUS INTEGER NOT NULL,
   TEXTDATA VARCHAR NOT NULL,
   PRIMARY KEY (ID, STATUS));

ALTER TABLE MYTABLE SET SYNCHISTORY ;
COMMIT WORK ;

CALL SYNC_REGISTER_PUBLICATION (NULL, 'MYPUBLICATION');
```

```
COMMIT WORK ;

SET SYNC PARAMETER SYNC_APP_SCHEMA_VERSION 'VER1';
COMMIT WORK ;
```

上記のスクリプトでは、SYNCHISTORY をオンに設定して、インクリメンタル・パブリケーションをサポートするように表 MYTABLE を設定しています。また、SYNC_REGISTER_PUBLICATION システム・プロシージャーを呼び出して、レプリカ・データベースを MYPUBLICATION パブリケーションに登録しています。マスターおよびレプリカ両方のデータベース・カタログのバージョン名を 'VER1' に設定しています。

スキーマ・アップグレードの指定と分散

システム全体の管理者は、各データベース・タイプ (マスターとレプリカ) 用に、スキーマを現行バージョンからアップグレード・バージョンに変更する一連のスクリプトを作成します。スクリプトの完了時に、バージョン名が新しいものにアップグレードされます。

新しいスクリプトは、拡張レプリケーション・データベース階層全体に渡る同期など、任意のデータ分散メカニズムを使用してレプリカに分散できます。

スキーマ・アップグレードの定義時には、以下のルールが適用されます。

- 任意の新しいデータベース・オブジェクトを、スキーマに追加できます。また、任意のデータベース・オブジェクトをスキーマからドロップできます。スキーマから表をドロップする場合、まずその表をパブリケーション定義から削除する必要があります。
- ストアード・プロシージャーの呼び出しインターフェース (つまり、パラメーター・リスト) は変更すべきでありません。このような変更が必要な場合、新しいプロシージャーに別の名前を付ける必要があります。レプリカに保存されているが、まだマスターに伝搬されていないトランザクションが正しく実行されるよう保証するため、古いプロシージャーをしばらくシステムに残す必要があります。
- パブリケーションは、結果セットの追加と削除、または結果セットの列の追加と削除により、変更できます。また、結果セットの検索基準も変更できます。ただし、この変更を行うと、結果セットの次回のリフレッシュは、インクリメンタル・リフレッシュではなく、フル・リフレッシュになります。
- パブリケーションのパラメーター・リストを変更してはいけません。パラメーター・リストの変更が必要な場合には、古いパブリケーションをドロップし、新しいものを作成する必要があります。新しいパブリケーションからのリフレッシュは、インクリメンタル・リフレッシュではなく、フル・リフレッシュになります。

スキーマを変更するスクリプトを作成する場合、コマンドの大部分は、レプリカとマスターで同じになります。例えば、マスター上の表に新しい列を追加する場合、その新しい列を各レプリカ上の対応する表にも追加しなければならないことがあります。しかし、コマンドによっては、マスターとレプリカで同じでないものも存在します。例えば、パブリケーションを変更するコマンドは、レプリカには適用されません。同様に、変更を行うときにパブリケーションを完全にドロップし、再作成する必要があり、したがってマスターにレプリカを再登録する必要がある場合、再登録コマンドはレプリカによってのみ実行され、マスターでは実行されません。スクリプトを作成する場合、共通要素 (ALTER TABLE ステートメントなど) を最大限に再利用するとともに、誤ったサーバー上で他のステートメント (パブリケーションを作成するステートメントやレプリカを登録するステートメントなど) を実行しないような方法で、そのスクリプトを編成しなければならないことがあります。

以下に、solidDB のスキーマ・アップグレード機能を使用して、分散データベース・システムのスキーマをアップグレードする単純な例を示します。

一般に、マスター・データベースのデータベース・スキーマを最初にアップグレードします。マスターをアップグレードした後、レプリカのデータベースをアップグレードします。層が 2 つより多いシステム (したがって、中間レベルのノードがマスターとレプリカの両方であるシステム) では、この処理は最上層で開始され、この層をアップグレードします。データベース内でスクリプトが完了すると、影響を受けるデータベース・カタログのバージョン名が新しい名前にアップグレードされます。これにより、データベースの次の層に対して、同様にアップグレードが必要なことが示されます。

分散スキーマのアップグレード

マスター・スキーマのアップグレード

マスター・スキーマはレプリカ・スキーマのアップグレード前にアップグレードする必要があります。そのために、マスター・データベース・サーバーの管理者がアップグレード・スクリプトを実行します。アップグレード中は、管理者は通常、パブリケーション内の表への同時書き込みアクセスを拒否し (LOCK TABLE ステートメントを使用)、さらにカタログへのすべての同期アクセスも拒否する (MAINTENANCE モードを使用) 必要があります。

以下のスクリプトは、新しい表をスキーマに追加し、既存のパブリケーションに含む方法を示しています。

```
-- 同期モードを MAINTENANCE に設定して、パブリケーションが参照する
-- 表への変更を許可します。同期モードを MAINTENANCE にすることで、
-- マスター・データベースへの同期アクセスもブロックされます。
SET SYNC MODE MAINTENANCE ;
COMMIT WORK ;

-- 新しい列を追加して既存の表を変更します。
LOCK TABLE MYTABLE IN LONG EXCLUSIVE MODE ;
COMMIT WORK ;
ALTER TABLE MYTABLE ADD COLUMN NEWCOL INTEGER ;
COMMIT WORK ;

-- 新しい列にデフォルト値を設定します。
-- 同期モードが MAINTENANCE に設定されている間は、更新はレプリカには
-- 送信されません。したがって、同期モードが MAINTENANCE に
-- 設定されている間にマスター上で更新が行われた場合は、
-- 同じ更新を各レプリカでローカルに実行する必要があります。
```

```
UPDATE MYTABLE SET NEWCOL = 1 ;
COMMIT WORK ;

-- MYTABLE 表でロックを解除します。
UNLOCK TABLE MYTABLE ;
COMMIT WORK ;

-- スキーマ内で新しい表を作成します。
CREATE TABLE MYSECONDTABLE (
   ID INTEGER NOT NULL,
   MYTABLEID INTEGER NOT NULL,
   STATUS INTEGER NOT NULL,
   TEXTDATA VARCHAR NOT NULL,
   UPDATETIME TIMESTAMP NOT NULL,
   PRIMARY KEY (ID, MYTABLEID, STATUS)) ;
ALTER TABLE MYSECONDTABLE SET SYNCHISTORY ;
COMMIT WORK ;

-- パブリケーションの新しいバージョンを作成します。
"CREATE OR REPLACE PUBLICATION MYPUBLICATION
BEGIN
RESULT SET FOR MYTABLE
BEGIN
   SELECT * FROM MYTABLE ;
   RESULT SET FOR MYSECONDTABLE
   BEGIN
      SELECT * FROM MYSECONDTABLE
      WHERE MYTABLEID = MYTABLE.ID ;
   END
END
END";
COMMIT WORK ;

-- マスター・カタログのバージョン情報を変更します。
SET SYNC PARAMETER SYNC_APP_SCHEMA_VERSION 'VER2';
COMMIT WORK ;
-- 同期モードを MAINTENANCE から NORMAL に戻します。
SET SYNC MODE NORMAL ;
COMMIT WORK ;
```

マスター・データベースでスクリプトが正常に実行されたら、マスター・データベースのスキーマがアップグレードされ、カタログのバージョン名が 'VER2' に変更されます。同期モードを NORMAL に設定すると、データベースが同期アクセス用に再度開きます。ただし、レプリカ・データベースはスキーマをマスターと同じレベルにアップグレードするまで、マスターと同期できません。

レプリカ・スキーマのアップグレードの要求の検出

掲示板パラメーター SYNC_APP_SCHEMA_VERSION を設定して、マスターとレプリカのバージョンをそれぞれ定義した場合、マスターとレプリカのバージョンが一致しないときは、レプリカが次回マスターとの同期を試みるとエラーが返されます。通常は、以下のような SQL スクリプトをレプリカ・データベースで実行することでデータが同期されます。

```
MESSAGE syncmsg BEGIN ;
MESSAGE syncmsg APPEND PROPAGATE TRANSACTIONS ;
MESSAGE syncmsg APPEND REFRESH MYPUBLICATION ;
MESSAGE syncmsg END ;
COMMIT WORK ;
MESSAGE syncmsg FORWARD TIMEOUT 10 ;
COMMIT WORK ;
```

マスター・データベースのバージョン名が、レプリカ・データベースのバージョン名と一致しない場合は、以下のステートメント

```
MESSAGE <msgname> FORWARD
```

が、以下のエラーで失敗します。

```
25092 - User version strings are not equal in master and replica, operation failed.
```

マスターへのメッセージ送信は失敗しましたが、メッセージはレプリカ・データベース内でそのまま維持されます。レプリカ・スキーマをマスター・スキーマに一致するようアップグレードしたら、以下のステートメントを使用して、失敗したメッセージをマスターに再送できます。

```
MESSAGE <msgname> FORWARD;
```

レプリカ・スキーマのアップグレードの要求を検出後、レプリカ・サーバーの管理者はそのレプリカ・データベース用に作成されたバージョンアップ・スクリプトを使用して、スキーマを新しいバージョンにアップグレードする必要があります。

レプリカ・スキーマのアップグレード

典型的なアップグレード処理では、マスター・データベース・サーバーの管理者は、スキーマを変更するための一連のスクリプトを作成し、該当するスクリプトをレプリカ・データベース・サーバーの管理者に送信します。

アップグレードが正常に実行されたら、スキーマのバージョン名が新しい名前に変更されます。ここで、ハングしている可能性のある同期メッセージをマスターに再送することができます。

この例では、レプリカを、マスター・スキーマの新しいバージョン名と一致させるためにアップグレードするためのスクリプトは、以下のようになります。

```
-- 同期モードを MAINTENANCE に設定して、パブリケーションが参照する
-- 表への変更を許可します。
-- レプリカ・データベースの同期機能は中断されます。
SET SYNC MODE MAINTENANCE ;
COMMIT WORK ;

-- 新しい列を追加して既存の表を変更します。
-- MAINTENANCE モードで行った更新は、次の同期でロールバックされません。
-- 該当する更新がマスター DB で既に実行されています。
LOCK TABLE MYTABLE IN LONG EXCLUSIVE MODE ;
ALTER TABLE MYTABLE ADD COLUMN NEWCOL INTEGER
;
COMMIT WORK ;
UPDATE MYTABLE SET NEWCOL = 1 ;
COMMIT WORK ;

-- MYTABLE 表のロックを解除します。UNLOCK TABLE MYTABLE ;
COMMIT WORK ;

-- レプリカ・スキーマに新しい表を作成します。
CREATE TABLE MYSECONDTABLE (
   ID INTEGER NOT NULL,
   MYTABLEID INTEGER NOT NULL,
   STATUS INTEGER NOT NULL,
   TEXTDATA VARCHAR NOT NULL,
   UPDATETIME TIMESTAMP NOT NULL,
```

```
   PRIMARY KEY (ID, MYTABLEID, STATUS)) ;
ALTER TABLE MYSECONDTABLE SET SYNCHISTORY ;
COMMIT WORK ;

-- パブリケーション定義を変更する際に、レプリカ側ではアクションが不要な点に
-- 注意してください。パブリケーションのメタデータの変更と
-- 追加した表のデータが自動的に
-- レプリカに送信されます。

-- レプリカ・データベース・カタログのバージョン情報を変更します。
SET SYNC PARAMETER SYNC_APP_SCHEMA_VERSION 'VER2';
COMMIT WORK ;
-- 同期モードを MAINTENANCE から NORMAL に戻します。
SET SYNC MODE NORMAL ;
COMMIT WORK ;
```

レプリカ・データベースでスクリプトが正常に実行されたら、以下のステートメントを実行して、停止した可能性のある同期メッセージをマスター・データベースに再送することができます。

```
MESSAGE <msgname> FORWARD
```

この例では、以下のステートメントを再実行します。

```
MESSAGE syncmsg FORWARD TIMEOUT 10 ;
COMMIT WORK ;
```

保守モードの注意

保守モードは強力な機能なので、使用には注意が必要です。

データベースで SET SYNC MODE MAINTENANCE を実行すると、サーバーは同期履歴情報を更新しなくなります。サーバーが変更を記録しないため、次のインクリメンタル・リフレッシュでは、必ずしもすべての変更がマスターからレプリカにコピーされません。保守モードでスキーマ・バージョンのアップグレード・プロセス中にマスター・データベースで行われた変更は、次の同期が行われる前に、レプリカ・データベースでも (同様に保守モードで) 対応するスキーマのアップグレード・プロセスで行われると見なされます。

これは、solidDB 同期システムの通常の動作との重要な違いです。solidDB 独自の同期テクノロジーの主要な利点の 1 つは、場合によってはシステムが「自己修復」できるという点です。レプリカの誤ったデータはマスターのデータに置き換えられ、その結果、時間の経過とともにレプリカのエラーはなくなっていきます。

ただし、保守モードでは、この自己修復プロパティーが失われます。マスターとレプリカは同期履歴データを保管しないため、行われた変更を認識せず、次にレプリカがリフレッシュするときに、マスターはレプリカにすべての更新を送信するとは限りません。

エラーの原因として、以下の 2 つが考えられます。

1.

   1 つ目のエラーの可能性として、マスターと異なる変更を誤ってレプリカで行ったことが考えられます。例えば、表に新しい列を追加するときに、マスターではデフォルト値を 1 に設定し、レプリカでは誤って 2 に設定する場合などです。

2.

2 つ目のエラーの可能性として、マスターとレプリカの両方で同じ操作をエラーなしに実行した場合でも、レプリカとマスターでデータの開始値が同じでないと、ある種のエラーが発生することが考えられます (そして、自動的には修復されません)。(マスターとレプリカのそれぞれが、請求書の消費税を計算するとします。マスターとレプリカで、請求書の合計価格の値が異なる場合、同じ公式を使用しても、計算される消費税は異なります。) 保守モードの操作中は、レプリカとマスターが独立して (非同期に) 更新されるため、このような状態が簡単に発生します。レプリカが更新を開始する時点で、そのレプリカは必ずしもマスターと「同期」していません。

そのため、保守モードでは、レプリカとマスターが完全に同期していることを前提とする操作を実行することは、安全ではありません。保守モードでは、レプリカのデータが完全に最新であるかどうかに影響されないタイプの操作を実行してください。例えば、新しい列の追加は、既存のデータ値に影響を与えません。しかし、保守モードで既存の列の値を変更する場合、この変更の整合性がとれない可能性があります。保守モードで DML 操作を実行する場合は、十分に注意してください。

サーバー・バージョンのアップグレード

solidDB サーバーを (例えばバージョン 4.20 からバージョン 4.50 へ) アップグレードするときには、レプリカ・サーバーのアップグレード前にマスター・サーバーをアップグレードしておく必要があります。古いフォーマットから新しいフォーマットにデータが確実に変換されるようにするため、新しいサーバーを始動し、コマンド行で -x convert または -x autoconvert オプションを使用する必要があります。これらのコマンド行オプションについて詳しくは、「*solidDB 管理者ガイド*」を参照してください。変換後は、古いサーバー・バージョンでデータベースを使用することはできない点に注意してください。

solidDB HotStandby を使用している場合は、最初に 1 次サーバーを PRIMARY ALONE 状態に設定して、2 次サーバーのアップグレードを許可する必要があります。1 次サーバーを PRIMARY ALONE 状態に切り替えた後で、2 次サーバーをシャットダウンして、アップグレードすることができます。2 次サーバーのアップグレード後、再始動します。正常にキャッチアップを行ってから、元の 1 次サーバーをシャットダウンして、2 次サーバーをすぐに PRIMARY ALONE 状態に切り替えます。元の 1 次サーバーは、以前の 2 次サーバーが PRIMARY ALONE 状態で稼働している間にアップグレードすることができます。最後に、古い 1 次サーバーを 2 次サーバーとして始動し、新しい 1 次サーバーのトランザクション・ログを使用してキャッチアップ・モードで実行します。(詳しくは、「*solidDB 高可用性ユーザー・ガイド*」を参照してください。)

リンク・ライブラリー・アクセスを使用する場合、通常は、スキーマが変更されると、アプリケーションが変更されます。これは、スキーマのアップグレード処理の一環として、アプリケーションとリンク・ライブラリー・アクセスを新たに作成する必要があることを意味します。

7 拡張レプリケーションによる solidDB の管理

この章では、拡張レプリケーション・テクノロジーを使用して solidDB を保守する方法について説明します。この章で取り上げる管理タスクには、同期エラーの管理、およびマスターとレプリカのバックアップに関するヒントが含まれます。

重要:

リンク・ライブラリー・アクセスを含む solidDB の場合、標準 solidDB とは管理が少し異なります。この章では、リンク・ライブラリー・アクセスと拡張レプリケーションを同時に使用する場合、「*solidDB リンク・ライブラリー・アクセス・ユーザー・ガイド*」を既に読んでいるものとします。

前提となる知識

このセクションでは、拡張レプリケーションの管理と保守を開始する前に必要となる、solidDB についての知識を説明します。

- solidDB のインストール

  solidDB をまだインストールしていない場合は、ソフトウェアと一緒に提供されている、または以下の IBM Corporation の Web サイトにあるリリース・ノート・ファイルを参照してください。

  http://www.ibm.com/software/data/soliddb

  リリース・ノート・ファイルには、インストールの詳細説明が記載されています。

- データベース管理のための特殊なロール

  同期関連操作によっては、実行するために、SYS_SYNC_ADMIN_ROLE というロールを付与してもらうことが必要になる場合があります。詳しくは、「*solidDB 管理者ガイド*」を参照してください。

solidDB 拡張レプリケーションのモニター

以下のセクションでは、solidDB データベースのデータ同期の状況を照会する方法について説明します。

同期メッセージの状況のモニター

同期のインプリメントには同期メッセージが使用されるため、既存の現行メッセージの状況を確認すれば、同期化処理の状況をモニターすることができます。

メッセージがアクティブであれば、システム内で何らかの状態で必ず永続的状態になっています。メッセージは正常に処理されるとデータベースから削除されます。マスター・データベースまたはレプリカ・データベースのいずれかでハングしているメッセージは、処理が完全には終了していません。ほとんどの場合、アイドルで永続的なメッセージは、同期エラーが発生したことを意味します。

図 12 は、ストア・アンド・フォワード・メッセージングと、メッセージング・エラーが発生しうるプロセスの時点を示しています。

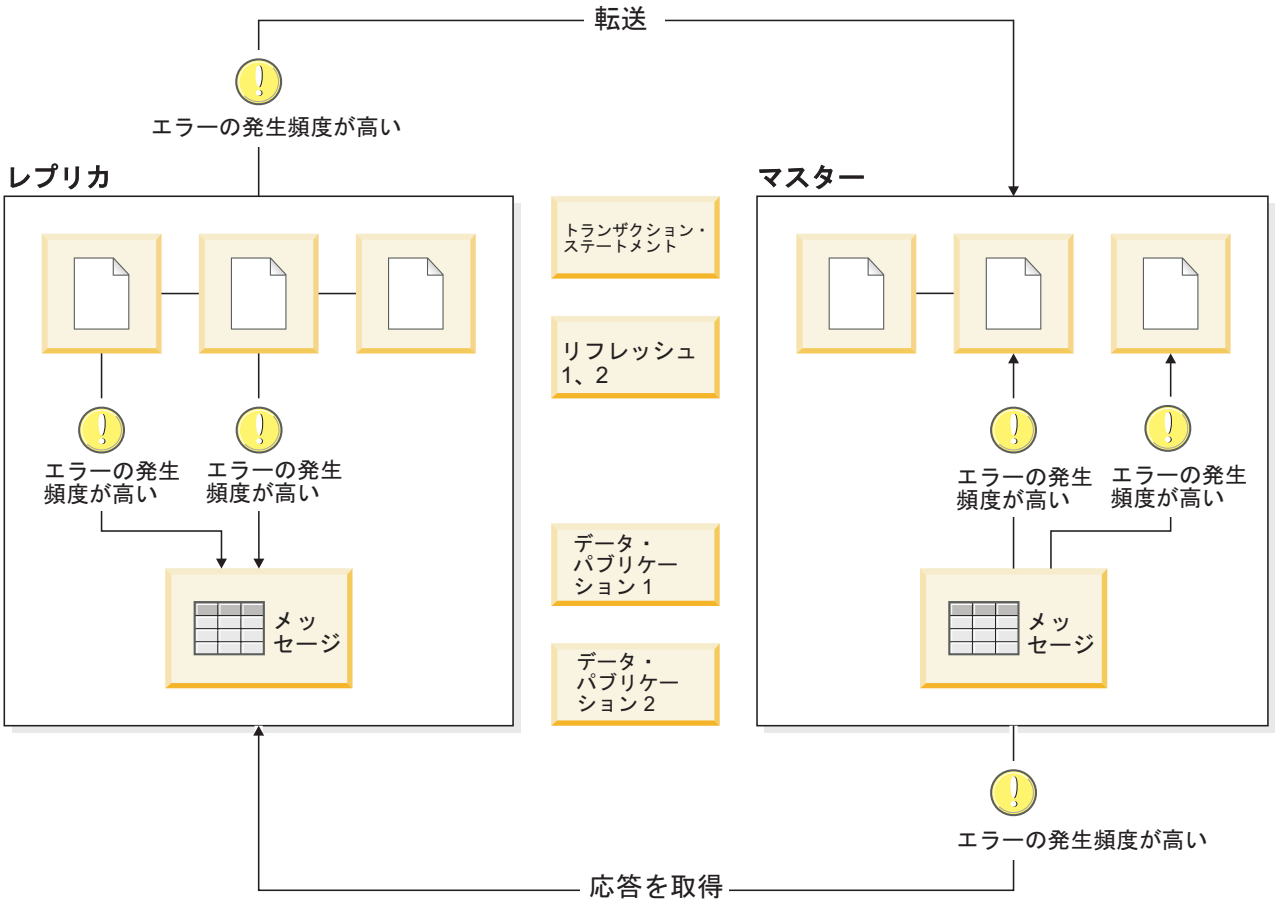


図 12. 同期メッセージングの中でエラーの発生頻度が高い領域

図 12 に示すように、エラーの発生頻度が高い領域が 4 つあります。

1. レプリカからマスターにメッセージを転送するとき
2. メッセージをマスターで実行するとき
3. 応答メッセージをマスターから受信するとき
4. 応答メッセージをレプリカで実行するとき

これらのどのケースでも、障害が発生すると、同期が停止します。障害の理由は、以下のメッセージ情報システム表を照会すると得られます。

- レプリカ・データベースの SYS_SYNC_REPLICA_MSGINFO
- マスター・データベースの SYS_SYNC_MASTER_MSGINFO

これらの表について詳しくは、「*SQL ガイド*」の該当する付録を参照してください。表で確認したエラーの解決については、以下のセクションを参照してください。

同期エラーの管理

メッセージの配信または受信が失敗すると、同期メッセージング・エラーが発生します。このセクションでは、同期エラーの管理手順について説明します。同期エラーの発生場所に応じて、エラーからのリカバリー手段はさまざまです。

マスターへのメッセージの転送エラー

レプリカからマスターへのメッセージの送信が失敗した場合、メッセージはレプリカ・データベースに残され、マスターに再送信することができます。この場合は、SYS_SYNC_REPLICA_MSGINFO 表の行の STATE 列の値が 22 R_SAVED になります。以下の SQL ステートメントを使用して、マスターに正常に送信されなかったメッセージの名前を照会できます。

```
SELECT MSG_NAME
FROM SYS_SYNC_REPLICA_MSGINFO
WHERE STATE = 22;
```

失敗したメッセージは、以下のコマンドをレプリカ・データベースで使用して、マスター・データベースに再送信できます。

```
MESSAGE message_name FORWARD;
```

SYS_SYNC_REPLICA_MSGINFO 表の STATE 列で可能な値については、「*solidDB SQL ガイド*」の『付録 D システム表』を参照してください。

マスターでの同期メッセージの実行エラー

マスター・データベースでのメッセージの実行は、以下の場合に失敗することがあります。

- トランザクションの SQL ステートメントが失敗した
- パブリケーションからのデータのリフレッシュが失敗した
- レプリカへの応答メッセージの送信が失敗した

このセクションでは、メッセージの実行が失敗する上記の理由のそれぞれについて、対処方法を説明します。

solidDB インテリジェント・トランザクションでのエラー処理

致命的エラーによってインテリジェント・トランザクションが失敗した場合、マスター・データベースでのメッセージの実行が停止され、トランザクションがロールバックされます。失敗した操作のエラー・コードが、レプリカ・データベースへの応答メッセージを戻すことになっていた同期メッセージ・コマンドのエラー・コードとしてレプリカに戻されます。

以下のいずれかのステートメントから、エラー・コードが戻りコードとしてレプリカに戻されます。

```
MESSAGE message_name FORWARD TIMEOUT timeout_in_seconds
```

または

```
MESSAGE message_name GET REPLY TIMEOUT timeout_in_seconds
```

マスター・データベースでは、システム表 SYS_SYNC_MASTER_MSGINFO に、マスター・データベース内に現在存在しているすべてのメッセージに関する情報が含まれています。エラーによってメッセージの実行が失敗した場合、そのメッセージは STATE 列に値 1 を持ちます。ERROR_CODE および ERROR_TEXT の各列には、メッセージの実行を停止させたエラーに関する情報が含まれています。SYNC_FAILED_MESSAGES と呼ばれるビューを照会することで、これらのハングしたメッセージ、その送信元のレプリカ・データベース、およびマスター・データベースからメッセージの実行を停止させたステートメントを照会できます。このビューはマスター上にあり、レプリカ名、メッセージ名、ステートメント・ストリング、エラー情報、およびその他の情報を表示します。

マスターでハングしたメッセージからリカバリーするための正しい方法は、マスター・データベースでエラーを修正することです。例えば、エラーの理由がユニーク制約違反の場合があります。このエラーを修正するには、マスター・データベースの既存のデータを変更して、新しい行を挿入できるようにしなければなりません。(トランザクション・キューにあるトランザクションの内容を変更することはできません。) あるいは、ストアード・プロシージャー内にプログラミング・エラーがあり、マスター・データベース内でそのストアード・プロシージャーを再作成することで訂正する必要がある場合もあります。

エラーを訂正したら、以下のコマンドを使用して、マスター・データベース内でメッセージを再開します。

```
MESSAGE message_name FROM REPLICA replica_name EXECUTE
```

メッセージがマスター・データベースで正常に実行された後、以下のコマンドを使用して、レプリカ・データベースへの応答メッセージを要求することができます。

```
MESSAGE message_name GET REPLY TIMEOUT timeout_in_seconds
```

あるいは、以下のコマンドを使用して、停止したメッセージ全体をマスター・データベースから削除することもできます。

```
MESSAGE message_name [FROM REPLICA replica_name] DELETE
```

または、以下のコマンドを使用して、メッセージ内の現行トランザクションだけをマスター・データベースから削除することもできます。

```
MESSAGE message_name FROM REPLICA replica_name DELETE CURRENT TRANSACTION
```

ただし、この代替手段を使用するとデータが失われるため、エラーを解決するために他の方法がない場合に、最後の手段としてのみ使用してください。

MESSAGE DELETE CURRENT TRANSACTION はトランザクション操作なので、メッセージの実行を継続する前にコミットする必要があることに注意してください。削除のコミット後、(中断した場所から) メッセージを再開するには、以下のステートメントを使用します。

```
MESSAGE msgname FROM REPLICA replicaname EXECUTE
```

一般に、トランザクションは並行性競合やデッドロックを回避するように作成する必要があります。トランザクションによって行が更新または削除された場合も、競合が発生することがあるので、SYS_TRAN_MAXRETRY 掲示板パラメーターをマスター・データベース内で SET SYNC PARAMETER コマンドを使用して指定することを推奨します。SYS_TRAN_MAXRETRY パラメーターは、ユーザーが構成可能な最大試行回数に基づき、並行性競合またはデッドロックによって失敗したトランザクションを再試行します。詳しくは、169 ページの『付録 A. 掲示板パラメーター』の SYS_TRAN_MAXRETRY の説明を参照してください。

リフレッシュ操作中のエラー処理

トランザクションとは異なり、マスター上での REFRESH 要求実行中のエラーは、メッセージ全体を停止させることはありません。代わりに、レプリカ・データベースに、メッセージング・コマンドの結果セットでエラーが折り返し報告されます。エラー・コードは結果セットの ERRCODE 列で戻されます。同様に、エラー・テキストは結果セットの ERRSTR 列から検索することができます。

メッセージング・コマンドの結果セットは必ずフェッチして、ERRCODE を検査する必要があります。ゼロ以外のすべての値は、マスター・データベースでのメッセージ実行中にエラーが発生したことを意味します。

エラーの 1 つの原因として考えられるのは、パブリケーションのバージョンがマスター・データベースで変更されているということです。パブリケーションがドロップされ、再作成された場合、パブリケーションの旧バージョンへのサブスクリプションは、新しいバージョンへのサブスクライブの前に、レプリカ・データベースでドロップする必要があります。以下のコマンドを使用して、サブスクリプションをドロップします。

```
DROP SUBSCRIPTION publication_name [{(parameter_list) | ALL}]
   [COMMITBLOCK number_of_rows] [OPTIMISTIC | PESSIMISTIC];
```

サブスクリプションをドロップすると、そのサブスクリプションのすべてのデータがレプリカ・データベースから削除されます。新しいバージョンにサブスクライブし、その後リフレッシュを要求することで、常にレプリカにフル・パブリケーションが提供されます。

ほとんどの場合、パブリケーションの更新が必要なときに、CREATE PUBLICATION コマンドの「OR REPLACE」オプションを使用することで、この問題を回避することができます。パブリケーションのドロップと再作成によるのではなく、CREATE OR REPLACE PUBLICATION の使用によってそのパブリケーションを更新した場合、必ずしもサブスクリプションをドロップして再作成する必要はありません。サブスクリプションを再作成する必要がなければ、マスターはレプリカにフル・リフレッシュではなく、インクリメンタル・リフレッシュを送信できる場合もあります。インクリメンタル・リフレッシュによってネットワーク・トラフィックが軽減されます。

レプリカへの応答メッセージの受信エラー

マスターからレプリカへの応答メッセージの転送は、ネットワーキング・エラーのために失敗することがあります。この場合、メッセージがマスター・データベースに残されます。

以下の SQL ステートメントを使用して、レプリカで応答を正常に受信できなかったメッセージをリストすることができます。

```
SELECT MSG_NAME
FROM SYS_SYNC_REPLICA_MSGINFO
WHERE STATE = 23
```

以下のコマンドを使用して、マスター・データベースにメッセージを再度要求できます。

```
MESSAGE message_name GET REPLY TIMEOUT timeout_in_seconds
```

SYS_SYNC_REPLICA_MSGINFO 表の STATE 列で可能な値については、「*solidDB SQL ガイド*」の『付録 D システム表』を参照してください。

レプリカでの応答メッセージの実行エラー

レプリカ・データベースでの応答メッセージの実行は、並行性競合によって失敗する場合があります。トランザクションに適用されるレコードがロックされ、レプリカで操作が実行できなくなる場合があります。この場合、メッセージがレプリカ・データベースに残ります。例えば、他のトランザクションが表を更新している場合、並行リフレッシュ操作が失敗する場合があります。この場合は、REFRESH コマンドは、レプリカ・データベースに残るので、再実行しなければなりません。

以下のコマンドを使用して、レプリカ・データベースからメッセージを再実行することができます。

```
MESSAGE message_name EXECUTE
```

例:

```
MESSAGE MyMsg0002 EXECUTE;
```

場合によっては、ペシミスティック・ロック方式を使用して、メッセージを実行する必要があります。これにより、特に REFRESH 操作中は、並行性競合の処理を回避することができます。詳しくは、71 ページの『レプリカの同期表における並行性競合の処理』を参照してください。

エラー・リカバリーのためのメッセージの削除

レプリカ・データベースから明示的にメッセージを削除して、エラーからリカバリーすることができます。メッセージを削除するときに、永続的に削除する対象をメッセージの内容全体にするのか、メッセージのうち、マスター・データベースに伝搬される現行トランザクションのみにするのかを指定することができます。メッセージ全体を削除するコマンドは、以下のとおりです。

```
MESSAGE message_name [FROM REPLICA replica_name] DELETE
```

現行トランザクションを削除するコマンドは、以下のとおりです。

```
MESSAGE message_name FROM REPLICA replica_name DELETE CURRENT TRANSACTION
```

上記のステートメントは、マスター・データベース内でのみ使用可能なことに注意してください。

マスター・データベースからメッセージを削除するときは、FROM REPLICA *replica_name* 節に必ずレプリカ名を指定するようにしてください。

例えば、以下のように指定します。

```
Message MyMsg0001 FROM REPLICA bills_laptop DELETE;
```

バックアップおよびリカバリーの実行

このセクションでは、データベースをバックアップする方法と、システム障害からリカバリーする方法について説明します。

「*solidDB 管理者ガイド*」も参照してください。

バックアップの作成

バックアップを作成する目的は、データベース・ファイルに保管された情報を保護することです。システム障害が発生してデータベース・ファイルが失われたとしても、バックアップ・ファイルからデータベースをリストアすることができます。

データベース・サーバー独自の「バックアップ」コマンドを使用する場合には、以下のファイルがバックアップされます。

- データベース自体を含むファイル。これらのファイルは、solid.ini 構成ファイルの [IndexFile] セクションに指定されている「FileSpec*」ファイルです。
- 障害発生後の「ロールフォワード」に使用する情報を含むログ・ファイル。これらログ・ファイルには sol#####.log という名前が付いています。これらのファイルは、solid.ini 構成ファイルの [Logging] セクションに以下のように指定されています。

  FileNameTemplate=<log_file_path>sol#####.log

バックアップ操作を実行すると、以下のファイルもコピーされますが、それらのファイルは、「リストア」処理の一部として必要なファイルではありません。

- デフォルトでは、solmsg.out がその他のファイルと一緒にコピーされます。これは、問題の診断時に役立つファイルです。このファイルは、「リストア」中に必要なものではありません。
- ディスクの破損後に、元の solid.ini が壊れる可能性があるため、solid.ini 構成ファイルもデフォルトでコピーされます。

solid.lic ファイルは自動的にはコピーされないことに注意してください。

以下の方法で、バックアップを開始することができます。

- 事前定義スケジュールに従ってバックアップを開始するようなタイミング・コマンドを使用して、バックアップを自動化します。「*solidDB 管理者ガイド*」の『タイミング・コマンドの入力』セクションを参照してください。
- solidDB SQL エディター (テレタイプ) で以下のコマンドを発行します。

  ADMIN COMMAND 'backup'

注:

データベース・ファイルとログ・ファイル用のバックアップ・ディレクトリーに十分なディスク・スペースがあることを確認してください。

バックアップ・ディレクトリーの solidDB メッセージの表示

システムは solidDB のメッセージ・ファイル (solmsg.out ファイル) をバックアップ・ディレクトリーにコピーします (solid.ini の [General] セクション内のパラメーター BackupCopySolmsgout は、デフォルトで yes に設定されます)。これによって、バックアップの実行前に、solidDB サーバーで実行された操作を簡単に表示することができます。また、対応するバックアップ・ファイルからデータベースをリストアする前に、表示用に solidDB のメッセージ・ファイルをバックアップ・ディレクトリーに置くことができます。

マスター・データベースおよびレプリカ・データベースのバックアップおよびリストア

非同期 solidDB データベースで使用する通常の災害防止およびリカバリー・タスクが、同期データベースにも適用されます。利用率が高くない時間のバックアップの自動化を推奨します。バックアップの完了後、ディスク破損から保護するために、バックアップ・ソフトウェアを使用してバックアップ・ファイルをテープにコピーします。

データベースを再開した後、進行中の可能性がある同期メッセージングが正常に完了したことを確認します。同期エラーの確認については、 149 ページの『同期メッセージの状況のモニター』を参照してください。停止した同期メッセージの訂正については、 151 ページの『同期エラーの管理』を参照してください。

マスター・データベースまたはレプリカ・データベースのデータベース・ファイルが破損した場合は、 バックアップ・ファイルからデータベースをリストアする必要があります。solidDB システムのすべてのデータベースは、オンライン・バックアップの作成が可能です。リカバリー時、solidDB はトランザクション・ログ・ファイルを使用して、バックアップ・データベースをバックアップの状態から最後にコミットされたトランザクションの状態にロールフォワードします。同期の最終永続状態も、その時点でリストアされます。

注:

-

直近に開始されたバックアップの状況は、solidDB SQL エディター (テレタイプ) で、コマンド ADMIN COMMAND 'status backup' を使用してプログラムで照会できます。すべての完了したバックアップと、その正常終了状況のリストを照会するには、コマンド ADMIN COMMAND 'backuplist' を使用します。

- 入力するバックアップ・ディレクトリーは、サーバー・オペレーティング・システムの有効なパス名にする必要があります。例えば、サーバーを UNIX オペレーティング・システムで実行する場合、パス区切り記号はバックスラッシュではなく、スラッシュにする必要があります。
- バックアップの作成に必要な時間は、メッセージ `Backup started` とメッセージ `Backup completed successfully` の間で経過した時間で、これは solmsg.out ログ・ファイルに書き込まれます。

バックアップ処理を開始する前に、チェックポイントが自動的に作成されます。これによって、バックアップ・データベースの状態が、バックアップ処理が開始された時点からのものになります。次に、以下のファイルがバックアップ・ディレクトリーにコピーされます。

- データベース・ファイル (複数可)
- 構成ファイル (solid.ini)
- 前のバックアップの後で変更または作成されたログ・ファイル (複数可) (solid.ini の [General] セクションのパラメーター BackupCopyLog は、デフォルトで yes に設定されます)
- solidDB メッセージ・ファイル solmsg.out のバックアップ (solid.ini の [General] セクションのパラメーター BackupCopySolmsgout は、デフォルトで yes に設定されます)

不要なログ・ファイルは、正常にバックアップが行われた後、元のディレクトリーから削除されます (solid.ini の

`General`

セクションのパラメーター BackupDeleteLog は、デフォルトで yes に設定されます)。

バックアップのガイドライン

拡張レプリケーション・システムでは、バックアップを使用することが、データの可用性とセキュリティーを高いレベルで保証する直接的で効果的な方法です。そのため、すべての重要なデータベースで、定期的にバックアップを行ってください。

システム障害が発生したときにデータが保護されるように、マスター・データベースは常に、レプリカ・データベースもできるだけ定期的にバックアップを行います。拡張レプリケーション・システムのバックアップでは、以下のガイドラインに注意してください。

- マスター・データベースにデータの正式なバージョンがあり、レプリカのデータは一時的なデータなので、レプリカ・データベースをマスター・データベースのバックアップとして使用しないでください。マスター・データベースの最終永続状態をリストアできるのは、マスター・データベース自身のバックアップ・ファイルからだけです。
- データ損失なしにマスター・データベースがバックアップからリカバリーされた場合 (つまり、ロールフォワード・リカバリーでログ・ファイルからすべてのトランザクションをリカバリーできる場合)、同期は正常に進行します。データ損失がある場合 (例えば、トランザクション・ログの欠落が原因)、次のリフレッシュでレプリカにも反映されます。マスターとレプリカで同期データが同期していない場合、次のリフレッシュで、フル・パブリケーション (パブリケーションのすべてのデータ) がレプリカ・データベースに送信されます。これは、パブリケーションの表がインクリメンタル・パブリケーション 用に設定されているかどうかにかかわらず行われます。
- レプリカ・データベースは、そのレプリカ・データベースのバックアップをリストアして再構成するか、マスター・データベースからデータをリフレッシュして最初から再構築することができます。前者のアプローチは、データベースが大きく、バックアップ用に使用可能なディスク・スペースがある場合に適しています。後者のアプローチは、レプリカ・データベースが小さく、レプリカ・データベースにローカル専用データがない場合に使用できます。バックアップのリストアについて詳しくは、「*solidDB 管理者ガイド*」のバックアップのリストアのセクションを参照してください。
- solidDB データベースのバックアップをリストアするときは、リストアで、すべてのトランザクションが最後にコミットされたトランザクションにロールフォワードされるようにしてください。これによって、データベース障害の時点以降の同期が継続するようになります。
- リストアが完了したら、マスター・データベースにもレプリカ・データベースにも未完了の同期メッセージがないことを確認します。

  同期エラーの確認については、 149 ページの『同期メッセージの状況のモニター』を参照してください。停止した同期メッセージの訂正については、 151 ページの『同期エラーの管理』を参照してください。

8 パフォーマンスのモニターおよびチューニング

この章では、solidDB の拡張レプリケーション・コンポーネントのパフォーマンスを向上させるために使用できる技法について説明します。solidDB の別の側面におけるパフォーマンスのチューニングについては、「*solidDB 管理者ガイド*」を参照してください。

メッセージの進行のモニター

solidDB のさまざまなイベントを使用すると、マスターおよびレプリカ間のデータの伝搬とリフレッシュの進行をモニターすることができます。それらのうち 2 つのイベントは、具体的には、それまでに送受信したメッセージのバイト数を追跡するためのものです。これらのイベントは主に、BLOB を含むメッセージのように大容量のメッセージを送信する場合、または非常に低速の通信チャネル経由でメッセージを送信する場合に役に立ちます。例えば、BLOB を送信する場合に、これを利用して、データを 20 K 送信するごとに通知し、次に画面表示を更新してダウンロードしたデータの容量を表示することが考えられます。

2 つのイベントは、以下のようになっています。

```
SYNC_MSGBYTES_SENT(
   sender_nodename              WVARCHAR,
   receiver_nodename            WVARCHAR,
   message_name                 WVARCHAR,
   cumulative_bytes_sent        INTEGER,
   total_bytes                  INTEGER);

SYNC_MSGBYTES_RECEIVED(
   sender_nodename              WVARCHAR,
   receiver_nodename            WVARCHAR,
   message_name                 WVARCHAR,
   cumulative_bytes_received    INTEGER,
   total_bytes                  INTEGER);
```

どちらのイベントも、基本的にパラメーターは同じです。

両方のメッセージは、マスターとレプリカ間の同期メッセージの転送中に 0 回以上通知されます。SYNC_MSGBYTES_SENT イベントは送信ノードで通知され、SYNC_MSGBYTES_RECEIVED イベントは受信ノードで通知されます。

このイベントは、それまでの送受信バイト数の累計と、対応する同期メッセージの送受信バイト数の合計をリストします。このイベントをキャッチし、それまでの送信バイト数の累計とメッセージのバイト数の合計を比較すれば、送受信処理の進行をモニターすることができます。

メッセージの送信頻度を制御するには、以下の solid.ini 構成パラメーターを設定します。

```
[Synchronizer]
RpcEventThresholdBytecount=<value>
```

値はバイト単位で指定し、0 以上にする必要があります。値として、キロバイトを K のように省略して指定することはできないことに注意してください。

値が 0 の場合は、SYNC_MSGBYTES_SENT イベントも SYNC_MSGBYTES_RECEIVED イベントも、対応するノードで通知されません。デフォルト値は 0 (つまり、イベントの通知なし) です。

このパラメーターは、最初のイベントを通知する前に、および連続する各イベント間で送信する必要がある最小バイト数を指定します。サーバーは、指定バイト数がほぼ送受信されると、その後で最初のイベントを通知し、追加で指定バイト数がほぼ送受信されるごとに別のイベントを通知します。例えば、**RpcEventThresholdByteCount** を 1000 に設定した場合、サーバーは、おおよそ以下のタイミングでイベントを通知します。

1000 バイト送信し終わったとき

2000 バイト送信し終わったとき

3000 バイト送信し終わったとき

...

RpcEventThresholdByteCount で設定されるのは、最初のイベントの前に、および以降の各イベント間で送信する必要がある最小バイト数です。イベントは必ずしも、RpcEventThresholdByteCount に指定した正確なバイト数の後で通知されるわけではありません。サーバーは、各バイトではなく、各通信パケットの送信が完了した後に、バイト数を検査し、イベントの通知を検討しているだけです。これらのパケットは、**RpcEventThresholdByteCount** と同じサイズであるとは限りません。

例えば、**RpcEventThresholdByteCount** を 1000 に設定して、3500 バイト送信するとします。また、各パケットが 1500 バイトであるとします。この場合、バイト数が正確に 1000、2000、および 3000 になった後にイベントを受け取るわけではありません。最初のパケット (1500 バイト) および 2 番目のパケット (3000 バイト) の後にイベントを受け取ります。

もう 1 つ例を挙げます。上の例と同様に、**RpcEventThresholdByteCount** を 1000 に設定して、3500 バイト送信するとします。しかし、今回の各パケットは 600 バイトです。この場合にも、バイト数が正確に 1000、2000、および 3000 になった後にイベントを受け取るわけではありません。2 番目のパケット (1200 バイト)、4 番目のパケット (2400 バイト)、および 5 番目のパケット (3000 バイト) の後にイベントを受け取ります。

サーバーは、最後のパケットを送信 (または受信) した後に、SYNC_MSGBYTES_SENT も SYNC_MSGBYTES_RECEIVED も通知しないことに注意してください。その代わり、各メッセージの送受信が完了した後に、サーバーは SYNC_MASTER_MESSAGE_RECEIVE_END などの別のイベントを通知します。

SYNC_MSGBYTES_SENT および SYNC_MSGBYTES_RECEIVED を使用する場合には、対応するメッセージの開始イベントと終了イベントもキャッチすることを推奨します。特定の同期メッセージの受信完了を通知するイベントを受け取ったら、そのイベントのための登録を抹消するか、または少なくとも、次のこのようなイベ

ントを待つ WAIT ステートメントの実行を停止する必要があります。同様に、新規メッセージの転送を通知するイベント (SYNC_MASTER_MESSAGE_RECEIVE_BEGIN) を受け取ったら、SYNC_MSGBYTES_RECEIVED のモニターを開始する必要があります。

以下に、メッセージ関連の一部のイベントについて簡単に説明します。

1. レプリカからメッセージを転送する場合。送信バイト数をモニターします。

   SYNC_REPLICA_MESSAGE_FORWARD_BEGIN

   ...

   0 回以上の SYNC_MSGBYTES_SENT

   ...

   SYNC_REPLICA_MESSAGE_FORWARD_END
2. マスターが転送メッセージを受信する場合。受信バイト数をモニターします。

   SYNC_MASTER_MESSAGE_RECEIVE_BEGIN

   ...

   0 回以上の SYNC_MSGBYTES_RECEIVED

   ...

   SYNC_MASTER_MESSAGE_RECEIVE_END
3. マスターがレプリカに応答メッセージを送信する場合。

   SYNC_MASTER_MESSAGE_SENDREPLY_BEGIN

   ...

   0 回以上の SYNC_MSGBYTES_SENT

   ...

   SYNC_MASTER_MESSAGE_SENDREPLY_END
4. レプリカがマスターから応答メッセージを受信する場合。

   SYNC_REPLICA_MESSAGE_REPLY_BEGIN

   ...

   0 回以上の SYNC_MSGBYTES_RECEIVED

   ...

   SYNC_REPLICA_MESSAGE_REPLY_END

注:

- 送受信の間隔を制御する変数は 1 つのみです。1 つのノード内では、送受信の間隔 (バイト数) は同じです。さらに、「受信」イベントと「送信」イベントのオン、オフの切り替えも両方同時に行います。一方だけをオンにすることはできません。
- 通常、送信側と受信側のノードは異なるため、**RpcEventThresholdByteCount** の値が異なることがあります。実際に、一方のサーバーではイベントがオフ (**RpcEventThresholdByteCount=0**) になっており、もう一方のサーバーではイベントがオンになっていることがあります。
- 同期イベントの待機について詳しくは、「*IBM solidDB SQL ガイド*」の特に ADMIN EVENT コマンドと CREATE PROCEDURE コマンドに関するセクションを参照してください。

データ同期のためのチューニング

solidDB がデータ同期のために適切にチューニングされていることを確認するため、このような同期について、以下の solidDB ガイドラインを参照してください。

- パブリケーション定義のチューニング
- 同期履歴データ管理の最適化
- 同期メッセージの最適化

上記の各トピックについては、以下のセクションで説明します。

パブリケーション定義のチューニング

solidDB 拡張レプリケーションでは、パブリケーションを使用して、マスター・データベースからレプリカ・データベースにインクリメンタル・データをダウンロードします。パブリケーションでは、マスターからレプリカに伝搬するデータを定義します。CREATE PUBLICATION ステートメントを使用して、表と、レプリカへのデータを選択するための検索基準を指定します。パブリケーションについて詳しくは、100 ページの『パブリケーションの作成』を参照してください。

以下は、パブリケーション定義の簡単な例です。

```
CREATE PUBLICATION configuration_of_device (device_name VARCHAR)
BEGIN
   RESULT SET FOR device
   BEGIN
      SELECT * FROM device WHERE name = :device_name;
      RESULT SET FOR device_cfg_parameter
      BEGIN
         SELECT * FROM device_cfg_parameter WHERE device_id = device.id;
      END
   END END
```

内部では、RESULT SET FOR パラグラフの照会が通常の SELECT として実行されます。ネストされた結果セットは、常に外部の結果セットと内部の結果セットを結

合します。したがって、パフォーマンスの最適化では、他の照会と同様に、以下のような索引付けのルールが適用されます。

- 検索基準として使用する大きな表の列に索引を作成する
- ネストされた結果セットの結合に使用する列に索引を作成する

さらに、パブリケーション定義では結果セットをネストしないことを推奨します。ネストされた結果セットとネストされていない結果セットの例については、「*solidDB 拡張レプリケーション・ユーザー・ガイド*」の以下のセクションを参照してください。

- ネストされたパブリケーションのバージョン
- ネストされていないパブリケーションのバージョン

以下のコマンドを使用してマスター・データベース内で SQL トレースをオンに設定して、パブリケーション定義から実際に生成された SQL を抽出することができます。

ADMIN COMMAND 'trace on sql'

トレースの出力は、solidDB の標準トレース・ファイルに送信されます。ファイルのデフォルト名は、soltrace.out です。

同期履歴データ管理の最適化

初めてマスターのパブリケーションからリフレッシュするとき、レプリカは、パブリケーションのすべての情報のコピーをダウンロードする必要があります。つまり、レプリカは「フル・リフレッシュ」を取得する必要があります。初回のダウンロード以降は、毎回のリフレッシュ時に、レプリカは、前回のダウンロードから変更があったレコードをダウンロードするだけで済みます。つまり、「インクリメンタル・リフレッシュ」を行います。

マスターからレプリカに変更データを同期するとき、solidDB (マスターとレプリカの両方とも) は、前回の同期以降にこれらのデータベースに書き込まれたデータを把握している必要があります。データの更新が発生した場合、solidDB は、更新前の、以前の行バージョンを記録している必要があります。このような更新された行の古いバージョンは、同期履歴表に記録されています。

最適なパフォーマンスのために、できる限りインクリメンタル・パブリケーションを使用することを推奨します。インクリメンタル・パブリケーションのセットアップについては、100 ページの『インクリメンタル・パブリケーションの作成』を参照してください。

同期履歴表のチューニング

更新操作がレプリカによるリフレッシュ要求の頻度に比べて頻繁に行われるシステムでは、履歴表が大きくなることがあります。デフォルトでは、マスター・データベース内でメイン表の行が更新されるたびに、solidDB が履歴表に新しい行を作成します。ただし、これは必ずしも必要ではありません。データ更新中に履歴表への新たなエントリーが必要な同期表の列を指定することで、履歴表に保管するデータの量を減らすことができます。検索基準として使用するパブリケーション内の列 (WHERE 節または結合列) のみ、履歴列として指定する必要があります。(技術的な詳細については、『内部情報: SET HISTORY COLUMNS』を参照してください。) これらの列を指定するには、以下のコマンドを使用します。

```
ALTER TABLE tablename SET HISTORY COLUMNS (col1, col2, colN...)
```

パブリケーション定義に表のすべての行が含まれている場合は、その表の HISTORY COLUMNS として、主キー列 (複数可) を指定してください。

このような定義が行われていない場合は、対応する同期表が更新された場合に、マスター・データベース内のすべての更新操作で、履歴表内に新しいエントリーが作成されます。頻繁に更新する行がある場合は、履歴列を設定することで、マスター・データベースにおけるパフォーマンスとディスク・スペース使用量の面で、オーバーヘッドを大幅に減らすことができます。

注:

ALTER TABLE ... SET HISTORY COLUMNS を正常に実行するには、ステートメント ALTER TABLE ... SET SYNCHISTORY を最初に実行する必要があります。また、ALTER TABLE ... SET NOSYNCHISTORY を実行することで、ALTER TABLE ... SET HISTORY COLUMNS が無効になります。

例

同期履歴用に設定済みの以下のような表があるとします。

```
CREATE TABLE account
   (accountid VARCHAR NOT NULL PRIMARY KEY,
   balance numeric(12,2));
```

ここで以下のコマンドを使用して、accountid 列の値が更新操作によって変更された場合に限り、履歴エントリーが発生するように指定します。

```
ALTER TABLE account SET HISTORY COLUMNS (accountid);
```

ここで、balance 列の値を変更しても、履歴表の行は更新されません。

内部情報: SET HISTORY COLUMNS

履歴表がなぜ特定の列の変更だけ追跡する必要があるのか、疑問に思うかもしれません。結局は、各レプリカへは特定の列だけではなく、データへの変更のすべてを通知する必要があります。若干簡素化した例と説明を以下に示します。

レプリカ・データベースに、ロンドン支社の顧客のみの情報が含まれているとします。以下のような各タイプの変更について、レプリカに通知を行う必要があります。

1. ロンドン支社の顧客情報の変更 (顧客の電話番号の変更など)。

2. ロンドン支社への新規顧客の追加 (または既存顧客の削除)。例えば、リバプール支社を開設し、それまでロンドン支社に割り当てられていた顧客の一部をリバプール支社に割り当てた場合、更新データのリフレッシュを要求するときに、それらの顧客はロンドン支社では受け入れられなくなります。実際には、マスターはロンドンのレプリカに対して、このような顧客に関する情報のコピーを削除するように、明示的に通知する必要があります。

すなわち、セット (ロンドン・セット)*内の* レコードが変更され、*あるセットから別*のセット (ロンドン・セットからリバプール・セット) にレコードを移動する変更が行われます。行われる可能性があるこのような 2 つのタイプの変更は、異なるメカニズムで追跡されます。

セット内の変更は、サーバーの一般的なバージョン管理システムを使用して追跡されます。例えば、ロンドン支社が最後にデータのリフレッシュを行ったのが 2001 年 12 月 3 日の午前 12 時 1 分だった場合、ロンドン支社は、ロンドン・セット内にある、2001 年 12 月 3 日の午前 12 時 1 分以降に変更されたレコードの更新を必要とします。

レコードがどの セットに記録されるか (ロンドン、リバプールなど) に影響を及ぼす変更は、履歴表で追跡されます。履歴表は特定のセットでの顧客の追加と削除を追跡するだけでよいので、レコードがどのセットに記録されるのかを決めるフィールドへの変更のみを追跡するだけで済みます。これらのフィールドは「検索基準」— すなわち、パブリケーション定義の際に指定した WHERE 節と結合フィールドです。顧客レコードが突然ロンドン支社のパブリケーションに加わる、またはパブリケーションから消える場合として唯一考えられるのは、WHERE 節 (または結合節) で使用されるいずれかの列が変更された場合なので、履歴表はこれらの列の値への変更を記録するだけで済みます。

以下に例を示します。

```
CREATE TABLE customer (
   id VARCHAR NOT NULL PRIMARY KEY,
   name VARCHAR NOT NULL,
   salesman_id VARCHAR NOT NULL) ;

CREATE TABLE invoice (
   customer_id VARCHAR NOT NULL,
   invoice_number VARCHAR NOT NULL,
   invoice_date DATE NOT NULL,
   invoice_total NUMERIC (12,2),
   PRIMARY KEY (customer_id, invoice_number));

CREATE PUBLICATION customers_by_salesman (salesman_id VARCHAR)
BEGIN
   RESULT SET FOR customer
   BEGIN
      SELECT * FROM customer WHERE salesman_id = :salesman_id ;
      RESULT SET FOR invoice
      BEGIN
        SELECT * FROM invoice WHERE customer_id = customer.id ;
      END
   END
END
```

更新のパフォーマンス向上のためにこれを最適化するには、以下の ALTER TABLE が必要です。

```
ALTER TABLE customer SET HISTORY COLUMNS (salesman_id)
ALTER TABLE invoice SET HISTORY COLUMNS (customer_id)
```

注意:

ALTER TABLE tablename SET HISTORY COLUMNS コマンドを実行し、誤って該当するすべての列を指定しなかった場合、リフレッシュ・メカニズムが正しく機能せず、レコードは正しいレプリカに保管されなくなります。新しいパブリケーションを追加するたびに、...SET HISTORY COLUMNS コマンドを再実行して、新しいパブリケーションの検索基準で使用される追加の列を考慮するべきかどうかを検討する必要があります。

ALTER TABLE ... SET SYNC HISTORY コマンドは必須ではありません。表に対して HISTORY COLUMN プロパティーが設定されていなくても、同期は正しく機能しますが、パフォーマンス向上のための最適化は行われません。

履歴データの廃棄

同期後、バックグラウンド・プロセスが履歴表から廃止データ (最新の同期よりも古いデータ) をパージします。古い履歴データが累積されないようにするために、このデータを手動で削除する必要はありません。

読み取り専用レプリカ

レプリカ・データベースの性質として、複製されたデータが読み取り専用で使用される、より正確に言うとレプリカの変更がマスターに伝搬されない場合、履歴データの保守を以下の設定によって回避することができます。

```
set sync parameter SYS_SYNC_KEEPLOCALCHANGES 'Yes';
```

この場合、以下のステートメント

```
ALTER TABLE ... SET SYNCHISTORY
```

は、不要です。このステートメントが既に実行されている場合、以下のステートメント

```
ALTER TABLE ... SET NOSYNCHISTORY
```

によって、設定の効果を元に戻します。

また、この場合は、ALTER TABLE ... SET HISTORY COLUMNS を使用できないことに注意してください。

SYS_SYNC_KEEPLOCALCHANGES パラメーターを「Yes」に設定することで、すべてのローカルな変更と行の挿入は (行われた場合)、リフレッシュまたはサブスクライブの操作にもかかわらず保持されます。この設定によって、レプリカでの処理負荷および必要なストレージが削減されます。このパラメーターを使用するのは、通常、読み取り専用レプリカに限られます。ただし、レプリカが読み取り/書き込みデータベースで、アプリケーションがレプリカでのすべてのローカルの書き込みを正常にマスターに伝搬することを保証する場合もまた、このパラメーターを「Yes」に設定できます。

同期メッセージの最適化

solidDB 拡張レプリケーションのデータ同期では、拡張レプリケーション構成の 1 つのデータベースでコミットされたデータは、データベース間での同期中に失われることは決してありません。拡張レプリケーションのストア・アンド・フォワード・メッセージング機能では、データベース間で同期メッセージを送信する前に、メッセージは発信元のデータベースに格納されます。同様に、受信側データベースでメッセージが実行される前にも、メッセージは受信側データベースに格納されます。格納されたメッセージは、使用されなくなると削除されます。

ストア・アンド・フォワード・メッセージングでは、データをディスクに永続的に格納するため、同期化処理で多少のオーバーヘッドが発生します。メッセージの中の同期対象データが少ないほど、オーバーヘッドは非常に高くなります。例えば、データベース間のトランザクションを 1 件だけ含む同期メッセージを送信した場合、メッセージングの往復にかかる時間は長くて 1 秒程度ですが、1 つのメッセージに数 10 件のトランザクションを含む場合の同期でも、通常は 1 秒未満です。

ストア・アンド・フォワード・メッセージングに起因するオーバーヘッドを最小化するには、複数のトランザクションを含む同期メッセージを作成してください。 1 つのメッセージの中にトランザクションが 1 件しかない状態で同期することは禁止されてはいませんが、そのような同期は、特に、サイトのトランザクション量が多い場合に、パフォーマンスに相当の悪影響を与えます。1 つの同期メッセージに 1 件 (または数件) のトランザクションを入れる前に、サイトのデータベースに、パフォーマンスおよびスケーラビリティーの重大な要件がないか考慮してください。

同期における RPC メッセージ圧縮の使用

solidDB はクライアントとサーバー間のすべてのネットワーク・トラフィックのメッセージ圧縮をサポートしています。データ同期では、レプリカ・データベース・サーバーがクライアントとして機能し、マスター・データベース・サーバーがサーバーとして機能します。したがって、クライアント/サーバー間通信で使用可能なメッセージ圧縮ユーティリティーは、拡張レプリケーションのサーバー間通信でも使用可能です。

レプリカ・データベース・サーバーで、SET SYNC CONNECT コマンドで渡される接続ストリングに「-z」パラメーターを指定することで、メッセージ圧縮を設定できます。以下に例を示します。

```
SET SYNC CONNECT 'tcp -z masterserver 1315' TO MASTER myMaster
```

パフォーマンスへのデータ圧縮の影響は、データの圧縮可能率と使用可能な帯域幅に大きく依存します。非常に高速なネットワークでは、メッセージの圧縮および解凍による CPU 消費量の増加が、ネットワーク・メッセージを小さくすることで得られるパフォーマンス向上効果を上回ることがあります。一般に、ネットワークが低速であるほど、ネットワーク・トラフィック圧縮がパフォーマンス全体に及ぼすプラスの効果が大きくなります。

付録 A. 掲示板パラメーター

solidDB では、マスター・サーバーとレプリカ・サーバーの両方で、伝搬されたトランザクションを処理するときに使用できる情報を、掲示板パラメーターを使用して格納できます。この付録では、solidDB で定義されている掲示板パラメーターをリストします。

インテリジェント・トランザクションを作成および使用するときに、独自の追加の掲示板パラメーターを作成できることに留意してください。

掲示板パラメーターは、PUT_PARAM() 関数を使用して設定でき、 GET_PARAM() 関数を使用して読み取ることができます。

拡張レプリケーション・システム・パラメーターのカテゴリー

拡張レプリケーション・システム・パラメーターは、以下のカテゴリーに分類されます。

- *読み取り専用システム・パラメーター。* solidDB によって保守され、以下の構文を使用して読み取りだけが可能です。

  ```
  GET_PARAM(parameter_name)
  ```

  このカテゴリーのパラメーターのライフ・サイクルは、1 件のトランザクションです。つまり、これらのパラメーターの値は、トランザクションの開始時点で常に初期化されます。
- *更新可能なトランザクション・レベルのシステム・パラメーター。*ユーザーが、以下の構文を使用して、設定および更新できます。

  ```
  PUT_PARAM(parameter_name, value)
  ```

  これは、トランザクション内の関数呼び出しです。更新可能なシステム・パラメーターは、同期関連操作を構成するために solidDB 拡張レプリケーションが使用します。

  上記のカテゴリーと同様に、これらのパラメーターのライフ・サイクルも 1 件のトランザクションです。
- *データベース・カタログ・レベルのシステム・パラメーター。*以下の構文を使用して設定します。

  ```
  SET SYNC PARAMETER(parameter_name value)
  ```

  このカテゴリーのパラメーターは、変更または解除されるまで有効なデータベース・カタログ・レベルのパラメーターです。掲示板パラメーターとして指定されます。

GET_PARAM()、 PUT_PARAM()、および SET SYNC PARAMETER 関数の完全な構文および使用法の例が、「*solidDB SQL ガイド*」に記載されています。

以下の表では、「期間」列に、このパラメーターの値が現行のトランザクションでのみ有効 (T) か、または変更されるまで有効 (C) かが示されています。

「R/W」列は、値が読み取り専用 (R/O) か、読み取り/書き込み (更新) 可能か (R/W) を示しています。

レプリカのパラメーター

表 8. レプリカのパラメーター

名前	説明	ファクトリー値	期間	R/W
SYS_SYNC_ID	このパラメーターは、内部使用専用です。設定しないでください。	N/A	N/A	R/O

SYS_R_MAXBYTES_OUT	単一同期メッセージの最大サイズは、データベース・レベルのシステム・パラメーターで設定できます。SYS_R_MAXBYTES_OUT パラメーターは、レプリカ・データベースからマスターに送信されるメッセージの最大長を設定します。
	
	このパラメーターの値は、レプリカ・データベースでのみ設定できます。
	
	マスター・データベースが SYS_R_MAXBYTES_OUT の値よりも長いメッセージを受信した場合、solidDB は、以下のエラー・メッセージを発行します。
	
	`25042 - Message is too long (<number> bytes) to forward. Maximum is set to <number> bytes.`
	
	例:
	`SET SYNC PARAMETER SYS_R_MAXBYTES_OUT '1048576000';`
		2 GB
		
		有効値は 0 から 2 GB です。0 が指定された場合、 2 GB が使用されます。
			C
			
			(変更されるまで)
				R/W

SYS_R_MAXBYTES_IN	単一同期メッセージの最大サイズは、データベース・レベルのシステム・パラメーターで設定できます。SYS_R_MAXBYTES_IN パラメーターは、レプリカ・データベースが受信できるメッセージの最大長を設定します。
	
	マスター・データベースが SYS_R_MAXBYTES_IN の値よりも長いメッセージを送信した場合、solidDB は、以下のエラー・メッセージを発行します。
	
	`25043 - Reply message is too long (<number> bytes). Maximum is set to <number> bytes.`
	
	例:
	`SET SYNC PARAMETER SYS_R_MAXBYTES_IN '1048576000';`
		2 GB
		
		有効値は 0 から 2 GB です。0 が指定された場合、 2 GB が使用されます。
			C
			
			(変更されるまで)
				R/W

SYS_SYNC_KEEPLOCALCHANGES	レプリカが読み取り専用の場合、このパラメーターを「Yes」に設定すると、レプリカでの処理負荷および必要なストレージが削減されます。この場合、レプリカ表に対して ALTER TABLE SET SYNCHISTORY ステートメントが不要になるため、レプリカで履歴表が作成されません。この設定では、すべてのローカルな変更および行の挿入は (行われた場合)、リフレッシュ操作にもかかわらず保持されます。	No	変更されるまで有効	R/W

表 8. レプリカのパラメーター (続き)

名前	説明	ファクトリー値	期間	R/W
SYS_SYNC_OPERATION_TYPE	レプリカがマスターから REFRESH を受け取るときに、パラメーター SYS_SYNC_OPERATION_TYPE および SYS_SYNC_RESULTSET_TYPE は共に、このデータでマスターが実行していた元の操作に関する情報をレプリカに提供します。これらのパラメーターを使用する主な理由は、UPDATE 操作がマスターで行われ、レプリカで DELETE+INSERT のペアに「変換」されたかどうかを示すことです。これら 2 つのパラメーターについて詳しくは、65 ページの『UPDATE トリガーの処理』を参照してください。 DELETE トリガーの SYS_SYNC_OPERATION_TYPE パラメーターで可能な値は、以下のとおりです。 • CURRENT_TENTATIVE_DELETE (レプリカで応答メッセージを実行する前に、行のローカルに更新された現行値を削除する場合に設定) • OLD_OFFICIAL_DELETE (マスターで削除された行を削除する場合に設定) • OLD_OFFICIAL_UNIQUE_DELETE (追加する行をマスターがレプリカに送信したが、同様の行が既にレプリカに存在する場合に設定。このパラメーター値を指定すると、新しい行がレプリカに追加される前に、古い正式な行が削除されます。) • OLD_OFFICIAL_UPDATE (マスターの更新結果として作成された削除を実行する場合に設定) INSERT トリガーの SYS_SYNC_OPERATION_TYPE パラメーターで可能な値は、以下のとおりです。 • OLD_OFFICIAL_INSERT (レプリカで応答メッセージを実行する前に、古いマスターの値をリストアする場合に設定) • NEW_OFFICIAL_INSERT (マスターに挿入された行を挿入する場合に設定) • NEW_OFFICIAL_UPDATE (マスターの更新結果として作成された挿入を実行する場合に設定) トリガーを起動したのがローカル・トランザクションである場合 (つまり同期ロジックではない場合) には、このパラメーターの値は NULL です。	なし	T	R/W

表 8. レプリカのパラメーター (続き)

名前	説明	ファクトリー値	期間	R/W
SYS_SYNC_RESULTSET_TYPE	拡張レプリケーションの REFRESH 操作の結果セットがフルかインクリメンタルかを示します。 このパラメーターで可能な値は、以下のとおりです。 • FULL • INCREMENTAL SYS_SYNC_OPERATION_TYPE の説明も参照してください。 SYS_SYNC_RESULTSET_TYPE および SYS_SYNC_OPERATION_TYPE について詳しくは、65 ページの『UPDATE トリガーの処理』を参照してください。	なし	T	R/W

マスターのパラメーター

表 9. マスターのパラメーター

名前	説明	ファクトリー値	期間	R/W
SYNC_DEFAULT_PROPAGATE_ERRORMODE	このパラメーターは、メッセージの伝搬中にエラーが発生したときのサーバーの動作を制御します。可能な値は、IGNORE_ERRORS、LOG_ERRORS、または FAIL_ERRORS です。これらの値の意味は、SAVE コマンドと同じです。「*solidDB SQL ガイド*」の SQL コマンド SAVE の説明を参照してください。 エラー処理モードは、3 とおりの方法で設定できることに注意してください (掲示板パラメーターを設定する、SAVEコマンドでオプション・キーワードを指定する、または `MESSAGE APPEND PROPAGATE TRANSACTIONS` ステートメントを使用する)。詳しくは、117 ページの『致命的エラーからのリカバリーの指定』を参照してください。	デフォルトでは、システムは、エラー・モードが FAIL_ERRORS であるものとして動作します。	C	R/W
SYNC_DEFAULT_PROPAGATE_SAVEMODE	可能な値は、AUTOSAVE、AUTOSAVEONLY、および NULL です。NULL は、伝搬トランザクションが自動的には保存されないことを意味します。詳しくは、117 ページの『致命的エラーからのリカバリーの指定』を参照してください。	NULL	C	R/W

表 9. マスターのパラメーター (続き)

名前	説明	ファクトリー値	期間	R/W

SYS_ERROR_CODE	このパラメーターは、SYS_ROLLBACK パラメーターと一緒に使用できます。ユーザーは、トランザクションがロールバックされた理由を示す独自のエラー・コードをこのパラメーターに設定できます。このエラー・コードは、ロールバックの後に返されます。
	
	このパラメーターで指定されたエラー・コードは、MESSAGE FORWARD コマンドまたは MESSAGE GET REPLY コマンドの一部として、レプリカ・データベースにも返されます。
	
	例:
	`PUT_PARAM(`
	`'SYS_ERROR_CODE','99000'`
	`);`
		なし	T	R/W

SYS_ERROR_TEXT	このパラメーターは、SYS_ROLLBACK パラメーターと一緒に使用できます。ユーザーは、トランザクションがロールバックされた理由を示す独自のエラー・テキストをこのパラメーターに設定できます。このエラー・テキストは、ロールバックの後に返されます。
	
	例:
	`PUT_PARAM(`
	`'SYS_ERROR_TEXT',`
	`'User defined error text'`
	`);`
		なし	T	R/W

SYS_IS_PROPAGATE	このトランザクションが、マスターで実行される伝搬トランザクションの場合、このパラメーターの値は YES です。非拡張レプリケーション・トランザクションの場合、このパラメーターの値は NULL です。	なし	T	R/O

表 9. マスターのパラメーター (続き)

名前	説明	ファクトリー値	期間	R/W

SYS_NOSYNCESTIMATE	多くの場合、インクリメンタル・リフレッシュでもフル・リフレッシュでもレプリカを同期します。通常、インクリメンタル・リフレッシュが最も効率的な選択です。ただし、フル・リフレッシュの方が効率的な場合もあります。極端な例として、表の行の 99% が削除された場合、削除された個別の行 (行の 99%) を送信するよりも、フル・リフレッシュ (行の 1%) を送信する方が効率的です。
	
	デフォルト (SYS_NOSYNCESTIMATE=None) では、サーバーはインクリメンタル・リフレッシュがフル・リフレッシュよりも効率的かどうかを計算し、より効率的だと考えられる方を選択します。
	
	SYS_NOSYNCESTIMATE を Yes に設定すると、この計算をオフにして、サーバーが強制的にインクリメンタル・リフレッシュを使用するようにできます。
	
	このパラメーターを使用することはまれです。
	
	エスティメーターを無効にするには、マスターで、以下を実行します。
	`SET SYNC PARAMETER SYS_NOSYNCESTIMATE 'YES'; COMMIT WORK;`
	
	エスティメーターを有効にするには (デフォルトと同じ)、マスターで、以下を実行します。
	`SET SYNC PARAMETER SYS_NOSYNCESTIMATE NONE; COMMIT WORK;`
		「None」
		
		エスティメーターが無効ではないという意味です。ファクトリー値がないという意味ではありません。
				R/W

SYS_ROLLBACK	このパラメーターは、トランザクションの実行をロールバックする必要が生じたときに、トランザクションの内部で使用されます。このパラメーターの値が YES に設定されている場合に PUT_PARAM 関数を使用すると、solidDB はトランザクションの実行を停止し、実行済みのすべてのステートメントをロールバックします。トランザクションのロールバックによって、同期メッセージの実行が停止します。伝搬トランザクションでは、 COMMIT WORK コマンドおよび ROLLBACK WORK コマンドは使用できないことに注意してください。 SYS_ROLLBACK は、例えば、トランザクションがデータベースで参照整合性エラーのような致命的エラーを検出した場合に使用できます。 例: `PUT_PARAM('SYS_ROLLBACK','YES');`	ファクトリー値は「No」です。	T	R/W
SYS_TRAN_ID	このパラメーターは、伝搬トランザクションに対してのみ有効です。つまり、伝搬トランザクションがマスター・データベースで実行されると、SYS_TRAN_ID に、レプリカ・データベースからの元のトランザクション ID が含まれます。非拡張レプリケーション・トランザクションの場合、このパラメーターの値は NULL です。	なし	T	R/O

表 9. マスターのパラメーター (続き)

名前	説明	ファクトリー値	期間	R/W
SYS_TRAN_USERID	このパラメーターは、伝搬トランザクションに対してのみ有効です。つまり、伝搬トランザクションがマスター・データベースで実行されると、SYS_TRAN_USERID に、レプリカ・データベースでそのトランザクションが実行されたときに使用された元のユーザー ID が含まれます。トランザクションは、このユーザー ID に定義されているアクセス権限を使用して、マスター・データベースで実行されます。ユーザー ID がレプリカからマスター・ユーザー ID にマップされた場合、マスター・データベースで実行時に使用されるアクセス権限は、マスター・ユーザー ID のアクセス権限になります (元のレプリカ・ユーザー ID ではありません)。 非拡張レプリケーション・トランザクションの場合、このパラメーターの値は NULL です。	なし	T	R/O

マスターとレプリカの両方のパラメーター

表 10. マスターとレプリカの両方のパラメーター

名前	説明	ファクトリー値	期間	R/W

SYNC_APP_SCHEMA_VERSION	同期可能なカタログごとに、そのプロパティーの 1 つとして、スキーマ・バージョン名を持つことができます。マスター・カタログとレプリカ・カタログのスキーマ・バージョン名が異なる場合、マスター・ノードとレプリカ・ノードの間の同期メッセージの送信は失敗します。
	
	このプロパティーの名前は、SYNC_APP_SCHEMA_VERSION です。このバージョン名は、 SET SYNC PARAMETER ステートメントを使用して設定できます。
	
	例:
	`SET SYNC PARAMETER SYNC_APP_SCHEMA_VERSION 'sputnik';`
	
	`SET SYNC PARAMETER SYNC_APP_SCHEMA_VERSION NONE;`
		なし	C	R/W

表 10. マスターとレプリカの両方のパラメーター (続き)

名前	説明	ファクトリー値	期間	R/W
SYNC_MODE	カタログごとに、SYNC_MODE という名前の読み取り専用パラメーターがあります。アプリケーションは、このパラメーターを使用して、カタログのモードを検査します。パラメーター値は、以下のとおりです。 • MAINTENANCE: カタログが保守同期モードの場合 • NORMAL: カタログが保守同期モードでない場合 • NULL: カタログがマスターでもレプリカでもない場合 カタログ・モードは、以下のコマンドで変更できます。 `SET SYNC MODE{NORMAL | MAINTENANCE}`	NORMAL	C	RO

表 10. マスターとレプリカの両方のパラメーター (続き)

名前	説明	ファクトリー値	期間	R/W
SYS_TRAN_MAXRETRY	このパラメーターは、トランザクションによる行の更新または削除時に発生する並行性競合およびデッドロックを処理できるようにします。このパラメーターが設定されている場合、サーバーは、並行性競合のためにマスターで実行に失敗したトランザクションを再試行します。値は、再試行の最大回数を指定します。 この値は、 SET SYNC PARAMETER ステートメントを使用して、マスターで設定できます。マスターで設定すると、この値が、そのマスターに伝搬されるすべてのトランザクションで使用されるデフォルト値になります。このパラメーターは、マスターがレプリカから受信した伝搬トランザクションを実行するときにだけ適用されることに注意してください。 この値は、レプリカでコマンド `SAVE PROPERTY ...` を使用して設定し、個別のトランザクションの掲示板に入れることができます。値は、同じトランザクションのマスターの掲示板に伝搬されます。このようにパラメーターを設定すると、値はそのトランザクションにだけ適用されます。 値が、レプリカでの SAVE PROPERTY コマンドとマスターでの SET SYNC PARAMETER コマンドの両方で設定された場合、レプリカでの SAVE PROPERTY コマンドが優先されます。特定のトランザクションに設定した値は、マスターの一般デフォルト値よりも優先されます。 SYS_TRAN_MAXRETRY が最大再試行回数に達した場合、問題のトランザクションには失敗のマークが付けられます。そのトランザクションをもう 1 度開始するには、別の MESSAGE EXECUTE コマンドを実行する必要があります。 SYS_TRAN_RETRYTIMEOUT は、SYS_TRAN_MAXRETRY パラメーターと密接に関連しています。詳しくは、下記の SYS_TRAN_RETRYTIMEOUT の説明を参照してください。	ファクトリー値はゼロ (0) で、トランザクションが失敗した場合に再試行しないことを意味します。パラメーターの有効な値は、0 から 2147483647 の整数です (両端を含む)。	C	R/W

表 10. マスターとレプリカの両方のパラメーター (続き)

名前	説明	ファクトリー値	期間	R/W
SYS_TRAN_RETRYTIMEOUT	このパラメーターは、SYS_TRAN_MAXRETRY パラメーターと一緒に使用します。SYS_TRAN_MAXRETRY パラメーターは、トランザクションが並行性競合のためにマスターで失敗した後、トランザクションを再試行する回数を指定します。SYS_TRAN_RETRYTIMEOUT パラメーターは、レプリカから受信した失敗したトランザクションを実際に再試行するまでのマスター・サーバーの待機時間を示すタイムアウト (秒単位) を設定します。このパラメーターの値は、SET SYNC PARAMETER ステートメントを使用して、マスター・データベースでのみ設定できます。SET SYNC PARAMETER について詳しくは、「*solidDB SQL ガイド*」の付録 B の『SET SYNC PARAMETER』の章を参照してください。	ファクトリー値はゼロ (0) で、マスター・サーバーが再試行間で待機しないことを意味します。パラメーターの有効な値は、0 から 2147483647 の整数です (両端を含む)。	C	R/W

付録 B. 同期イベント

この章では、拡張レプリケーション・イベントについて説明します。solidDB では、これらのイベントは、特定の拡張レプリケーション関連アクションが発生した場合にプログラムに通知を行うために提供されます。これらのイベントを使用して、マスター・データベースおよびレプリカ・データベース間の同期の進行状況をモニターすることができます。

これらのイベントは他のイベントとほとんど同じルールに従います。主な違いは、同期イベントに登録できないという点です。これは、

ADMIN EVENT 'wait'

コマンドが可変結果セットを返すことができないからです。代わりにストアード・プロシージャーを使用して、同期イベントを処理する必要があります。イベント全般については、「*solidDB SQL ガイド*」内の特に以下のセクションを参照してください。

- CREATE EVENT コマンド
- CREATE PROCEDURE コマンド (イベントの通知および待機方法についての説明)
- イベントについて詳細に説明した『ストアード・プロシージャー、イベント、トリガー、およびシーケンス』という表題の章

これらのイベントは事前定義されているので、ユーザーが作成する必要はありません。また、システム・イベントの通知は行わないでください。システム・イベントの登録と待機のみ行います。

すべてではありませんが、多くのシステム・イベントには、以下の共通する 5 つのパラメーターがあります。

- ename: イベント名
- postsrvtime: サーバーがイベントを通知した時刻
- uid: ユーザー ID (該当する場合)
- numdatainfo: 各種数値データ — 正確な意味はイベントごとに異なります。例えば、イベント SYS_EVENT_BACKUP は、バックアップの開始時と完了時の両方で通知されます。numdatainfo パラメーターの値は、このいずれの場合が該当するのかを示します。すなわち、バックアップを開始したばかりなのか、完了したばかりなのかを示します。数値データがない場合、このパラメーターは NULL になります。
- textdata: 各種テキスト・データ — 正確な意味はイベントごとに異なります。数値データがない場合、このパラメーターは NULL になります。

この付録には、以下の表が含まれています。

1. 同期関連イベント: レプリカがマスターにメッセージを送信するときに発生するイベントの順序。
2. 同期関連イベント: 同期関連イベントのタイプごとのパラメーター。

3. 現時点までの送信バイト数または受信バイト数など、メッセージの進行状況のモニターに役立つイベント。

レプリカからマスターへのメッセージ伝搬時のイベント

イベント・シーケンス

以下の表に、レプリカからマスターに送信されるメッセージを、レプリカとマスターが処理する際に通知されるイベントのシーケンスを示します。

イベントよっては、必ず発生するとは限らないものがあることに注意してください。例えば、このシーケンスは、SYNC_MASTER_MESSAGE_ERROR_OCCURRED イベントが発生する可能性のある位置を示しています。しかし、このイベントは、必ずしもそこで発生するとは限りません。

また、イベントの順序は、多少異なる場合があることに注意してください。例えば、マスターが古いメッセージを削除するタイミングには、レプリカのアクティビティーとは無関係な部分もあるので (その逆も同様)、ここに示す順序のままで実行されるとは限りません。

注:

管理者権限を持っていないアプリケーションは、同期イベントをモニターすることはできません。

後ほど、別の表で、これらの各同期関連イベントで使用するパラメーターの詳細を示します。

表 11. レプリカからマスターへのメッセージ伝搬時のイベント

レプリカ上のアクション、コマンド、または状態	レプリカに通知されるイベント	マスター上のアクションまたは状態	マスターに通知されるイベント
メッセージ開始			
メッセージ付加			
メッセージ終了/コミット	メッセージが永続的になった時点 (つまり、組み立てられ、コミットされた時点): SYNC_REPLICA_MESSAGE_ASSEMBLED		
メッセージ転送	レプリカがメッセージの送信を開始した時点: SYNC_REPLICA_MESSAGE_FORWARD_BEGIN レプリカが、マスターへのメッセージ送信を終了した時点: SYNC_REPLICA_MESSAGE_FORWARD_END		
		マスターがメッセージの受信を開始	マスターがメッセージの受信を開始した時点: SYNC_MASTER_MESSAGE_RECEIVE_BEGIN
		マスターがメッセージを受信	マスターがメッセージを受信し、そのメッセージを永続的にした時点: SYNC_MASTER_MESSAGE_RECEIVE_END

表 11. レプリカからマスターへのメッセージ伝搬時のイベント (続き)

レプリカ上のアクション、コマンド、または状態	レプリカに通知されるイベント	マスター上のアクションまたは状態	マスターに通知されるイベント
		マスターがメッセージの処理を開始	SYNC_MASTER_MESSAGE_REPLY_BEGIN
		マスターがメッセージを処理	SYNC_MASTER_MESSAGE_REPLY_END
		メッセージの処理中にエラーが発生した場合	SYNC_MASTER_MESSAGE_ERROR_OCCURRED
メッセージ応答取得	レプリカがマスターに応答取得要求を送信する時点: SYNC_REPLICA_MESSAGE_GETREPLY		
		マスターがレプリカから「応答取得」要求を受信	SYNC_MASTER_MESSAGE_GETREPLY_REQUEST
「応答取得」要求がタイムアウトになった場合	SYNC_REPLICA_MESSAGE_GETREPLY_TIMEDOUT		
		マスターが応答メッセージの送信を開始	SYNC_MASTER_MESSAGE_SENDREPLY_BEGIN
レプリカが、マスターからの応答の受信を開始	SYNC_REPLICA_MESSAGE_REPLY_BEGIN		
		マスターが応答の送信を終了	SYNC_MASTER_MESSAGE_SENDREPLY_END
応答を受信し、永続的にした	SYNC_REPLICA_MESSAGE_REPLY_END		
		メッセージを削除	SYNC_MASTER_MESSAGE_DELETED。メッセージは、MESSAGE DELETE ステートメントを使用して明示的に削除できることにも注意してください。
レプリカが応答の処理を開始	SYNC_REPLICA_MESSAGE_PROCESS_BEGIN		
処理中にエラーが発生した場合	SYNC_REPLICA_MESSAGE_ERROR_OCCURRED		
応答を処理	SYNC_REPLICA_MESSAGE_PROCESS_END		
メッセージを削除	SYNC_REPLICA_MESSAGE_DELETED。メッセージは、MESSAGE DELETE ステートメントを使用して明示的に削除できることにも注意してください。		

同期関連イベントのパラメーター

以下の表で、各同期関連イベントに関連付けられているパラメーターを示します。

表 12. 同期関連イベントに関連付けられているパラメーター

イベント名	目的	パラメーター
SYNC_MASTER_MESSAGE_DELETED	このイベントは、マスターがメッセージを削除したときに、マスターで通知されます。	master_name WVARCHAR、 replica_name WVARCHAR、 message_name WVARCHAR
SYNC_MASTER_MESSAGE_ERROR_OCCURRED	このイベントは、メッセージの処理中にエラーが発生したときに、マスターで通知されます。	master_name WVARCHAR、 replica_name WVARCHAR、 message_name WVARCHAR、 error_code INTEGER、 error_message WVARCHAR
SYNC_MASTER_MESSAGE_GETREPLY_REQUEST	このイベントは、マスターがレプリカから応答要求を受信するときに、マスターで通知されます。 パラメーター request_timeout は、要求された応答タイムアウトを秒単位で保持します。 パラメーター iftimedout は、以下の値のいずれかを保持します。 0: 要求がマスターでタイムアウトにならなかった場合 1: 要求がマスターでタイムアウトになった場合 マスターは、応答要求の処理をすぐに開始しない場合があります。マスターが、レプリカに要求されたタイムアウト時間内に応答の処理を開始できない場合、タイムアウトが 1 に設定されます。	master_name WVARCHAR、 replica_name WVARCHAR、 message_name WVARCHAR、 request_timeout INTEGER、 iftimedout INTEGER
SYNC_MASTER_MESSAGE_RECEIVE_BEGIN	このイベントは、マスターがレプリカからの新しいメッセージの受信を開始するときに、マスターで通知されます。	master_name WVARCHAR、 replica_name WVARCHAR、 message_name WVARCHAR
SYNC_MASTER_MESSAGE_RECEIVE_END	このイベントは、マスターがレプリカからの新しいメッセージの受信を終了するときに、マスターで通知されます。	master_name WVARCHAR、 replica_name WVARCHAR、 message_name WVARCHAR
SYNC_MASTER_MESSAGE_REPLY_BEGIN	このイベントは、マスターがレプリカへの応答メッセージの作成を開始するときに、マスターで通知されます。	master_name WVARCHAR、 replica_name WVARCHAR、 message_name WVARCHAR

表 12. 同期関連イベントに関連付けられているパラメーター (続き)

イベント名	目的	パラメーター
SYNC_MASTER_MESSAGE_REPLY_END	このイベントは、マスターがレプリカへの応答メッセージの作成を終了するときに、マスターで通知されます。	master_name WVARCHAR、 replica_name WVARCHAR、 message_name WVARCHAR
SYNC_MASTER_MESSAGE_SENDREPLY_BEGIN	このイベントは、マスターがレプリカへの応答メッセージの送信を開始するときに、マスターで通知されます。	master_name WVARCHAR、 replica_name WVARCHAR、 message_name WVARCHAR
SYNC_MASTER_MESSAGE_SENDREPLY_END	このイベントは、マスターがレプリカへの応答メッセージの送信を終了したときに、マスターで通知されます。	master_name WVARCHAR、 replica_name WVARCHAR、 message_name WVARCHAR
SYNC_MASTER_REGISTER_REPLICA	このイベントは、新しいレプリカがマスターに登録されるときに、マスターで通知されます。	master_name WVARCHAR、 replica_name WVARCHAR、 message_name WVARCHAR
SYNC_MASTER_UNREGISTER_REPLICA	このイベントは、レプリカがマスターから登録抹消されるときに、マスターで通知されます。	master_name WVARCHAR、 replica_name WVARCHAR、 message_name WVARCHAR
SYNC_REPLICA_MESSAGE_ASSEMBLED	このイベントは、レプリカが新しいメッセージを作成するときに、レプリカによって通知されます。	replica_name WVARCHAR、 master_name WVARCHAR、 message_name WVARCHAR
SYNC_REPLICA_MESSAGE_DELETED	このイベントは、レプリカがメッセージを削除するときに、レプリカによって通知されます。	replica_name WVARCHAR、 master_name WVARCHAR、 message_name WVARCHAR
SYNC_REPLICA_MESSAGE_ERROR_OCCURRED	このイベントは、メッセージの処理中にエラーが発生したときに、レプリカで通知されます。	replica_name WVARCHAR、 master_name WVARCHAR、 message_name WVARCHAR、 error_code INTEGER、 error_message WVARCHAR
SYNC_REPLICA_MESSAGE_FORWARD_BEGIN	このイベントは、レプリカがメッセージの転送を開始するときに、レプリカによって通知されます。	replica_name WVARCHAR、 master_name WVARCHAR、 message_name WVARCHAR

表 12. 同期関連イベントに関連付けられているパラメーター (続き)

イベント名	目的	パラメーター
SYNC_REPLICA_MESSAGE_FORWARD_END	このイベントは、レプリカがメッセージの転送を終了するときに、レプリカによって通知されます。	replica_name WVARCHAR、 master_name WVARCHAR、 message_name WVARCHAR
SYNC_REPLICA_MESSAGE_GETREPLY	このイベントは、レプリカが応答メッセージを要求するときに、レプリカによって通知されます。	replica_name WVARCHAR、 master_name WVARCHAR、 message_name WVARCHAR
SYNC_REPLICA_MESSAGE_GETREPLY_TIMEDOUT	このイベントは、レプリカの「get reply」がタイムアウトになったときに、レプリカによって通知されます。	replica_name WVARCHAR、 master_name WVARCHAR、 message_name WVARCHAR
SYNC_REPLICA_MESSAGE_PROCESS_BEGIN	このイベントは、レプリカが応答メッセージの処理を開始するときに、レプリカによって通知されます。	replica_name WVARCHAR、 master_name WVARCHAR、 message_name WVARCHAR
SYNC_REPLICA_MESSAGE_PROCESS_END	このイベントは、レプリカが応答メッセージの処理を終了するときに、レプリカによって通知されます。	replica_name WVARCHAR、 master_name WVARCHAR、 message_name WVARCHAR
SYNC_REPLICA_MESSAGE_REPLY_BEGIN	このイベントは、レプリカが応答メッセージの受信を開始するときに、レプリカによって通知されます。	replica_name WVARCHAR、 master_name WVARCHAR、 message_name WVARCHAR
SYNC_REPLICA_MESSAGE_REPLY_END	このイベントは、レプリカが応答メッセージの受信を終了するときに、レプリカによって通知されます。	replica_name WVARCHAR、 master_name WVARCHAR、 message_name WVARCHAR

メッセージ進行イベントのパラメーター

この表に記載されているイベントを使用すると、それまでに送受信したメッセージのバイト数を追跡することができます。これらのイベントは主に、BLOB を含むメッセージのように大容量のメッセージを送信する場合、または非常に低速の通信チャネル経由でメッセージを送信する場合に役に立ちます。

表 13. メッセージ進行イベントのパラメーター

イベント名	目的	パラメーター
SYNC_MSGBYTES_SENT	このイベントは、それまでの送受信バイト数の累計と、対応する同期メッセージの送受信バイト数の合計をリストします。このイベントをキャッチし、それまでの送信バイト数の累計とメッセージのバイト数の合計を比較すれば、送受信処理の進行をモニターすることができます。詳しくは、 159 ページの『メッセージの進行のモニター』を参照してください。	sender_nodename WVARCHAR receiver_nodename WVARCHAR message_name WVARCHAR cumulative_bytes_sent INTEGER total_bytes INTEGER
SYNC_MSGBYTES_RECEIVED	このイベントは、それまでの送受信バイト数の累計と、対応する同期メッセージの送受信バイト数の合計をリストします。このイベントをキャッチし、それまでの送信バイト数の累計とメッセージのバイト数の合計を比較すれば、送受信処理の進行をモニターすることができます。詳しくは、 159 ページの『メッセージの進行のモニター』を参照してください。	sender_nodename WVARCHAR receiver_nodename WVARCHAR message_name WVARCHAR cumulative_bytes_received INTEGER total_bytes INTEGER

索引

日本語, 数字, 英字, 特殊文字の順に配列されています。なお, 濁音と半濁音は清音と同等に扱われています。

[ア行]

[カ行]

[サ行]

[タ行]

［ナ行］

［ハ行］

［マ行］

[ヤ行]

[ラ行]

[数字]

A

B

C

D

E

F

G

H

I

L

M

P

R

S

U

特記事項

Copyright © Solid Information Technology Ltd. 1993, 2008

All rights reserved.

Solid Information Technology Ltd. または International Business Machines Corporation の書面による明示的な許可がある場合を除き、本製品のいかなる部分も、いかなる方法においても使用することはできません。

本製品は、米国特許 6144941、7136912、6970876、7139775、6978396、および 7266702 により保護されています。

本製品は、米国輸出規制品目分類番号 ECCN=5D992b に指定されています。

本書は米国 IBM が提供する製品およびサービスについて作成したものです。

本書に記載の製品、サービス、または機能が日本においては提供されていない場合があります。日本で利用可能な製品、サービス、および機能については、日本 IBM の営業担当員にお尋ねください。本書で IBM 製品、プログラム、またはサービスに言及していても、その IBM 製品、プログラム、またはサービスのみが使用可能であることを意味するものではありません。これらに代えて、IBM の知的所有権を侵害することのない、機能的に同等の製品、プログラム、またはサービスを使用することができます。ただし、IBM 以外の製品とプログラムの操作またはサービスの評価および検証は、お客様の責任で行っていただきます。

IBM は、本書に記載されている内容に関して特許権 (特許出願中のものを含む) を保有している場合があります。本書の提供は、お客様にこれらの特許権について実施権を許諾することを意味するものではありません。実施権についてのお問い合わせは、書面にて下記宛先にお送りください。

〒106-8711
東京都港区六本木 3-2-12
日本アイ・ビー・エム株式会社
法務・知的財産
知的財産権ライセンス渉外

以下の保証は、国または地域の法律に沿わない場合は、適用されません。 IBM およびその直接または間接の子会社は、本書を特定物として現存するままの状態で提供し、商品性の保証、特定目的適合性の保証および法律上の瑕疵担保責任を含むすべての明示もしくは黙示の保証責任を負わないものとします。国または地域によっては、法律の強行規定により、保証責任の制限が禁じられる場合、強行規定の制限を受けるものとします。

この情報には、技術的に不適切な記述や誤植を含む場合があります。本書は定期的に見直され、必要な変更は本書の次版に組み込まれます。 IBM は予告なしに、随時、この文書に記載されている製品またはプログラムに対して、改良または変更を行うことがあります。

本書において IBM 以外の Web サイトに言及している場合がありますが、便宜のため記載しただけであり、決してそれらの Web サイトを推奨するものではありません。それらの Web サイトにある資料は、この IBM 製品の資料の一部ではありません。それらの Web サイトは、お客様の責任でご使用ください。

IBM は、お客様が提供するいかなる情報も、お客様に対してなんら義務も負うことのない、自ら適切と信ずる方法で、使用もしくは配布することができるものとします。

本プログラムのライセンス保持者で、(i) 独自に作成したプログラムとその他のプログラム (本プログラムを含む) との間での情報交換、および (ii) 交換された情報の相互利用を可能にすることを目的として、本プログラムに関する情報を必要とする方は、下記に連絡してください。

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

本プログラムに関する上記の情報は、適切な使用条件の下で使用することができますが、有償の場合もあります。

本書で説明されているライセンス・プログラムまたはその他のライセンス資料は、IBM 所定のプログラム契約の契約条項、IBM プログラムのご使用条件、またはそれと同等の条項に基づいて、IBM より提供されます。

この文書に含まれるいかなるパフォーマンス・データも、管理環境下で決定されたものです。そのため、他の操作環境で得られた結果は、異なる可能性があります。一部の測定が、開発レベルのシステムで行われた可能性がありますが、その測定値が、一般に利用可能なシステムのものと同じである保証はありません。さらに、一部の測定値が、推定値である可能性があります。実際の結果は、異なる可能性があります。お客様は、お客様の特定の環境に適したデータを確かめる必要があります。

IBM 以外の製品に関する情報は、その製品の供給者、出版物、もしくはその他の公に利用可能なソースから入手したものです。IBM は、それらの製品のテストは行っておりません。したがって、他社製品に関する実行性、互換性、またはその他の要求については確証できません。IBM 以外の製品の性能に関する質問は、それらの製品の供給者にお願いします。

IBM の将来の方向または意向に関する記述については、予告なしに変更または撤回される場合があり、単に目標を示しているものです。

本書には、日常の業務処理で用いられるデータや報告書の例が含まれています。より具体性を与えるために、それらの例には、個人、企業、ブランド、あるいは製品などの名前が含まれている場合があります。これらの名称はすべて架空のものであり、名称や住所が類似する企業が実在しているとしても、それは偶然にすぎません。

著作権使用許諾:

本書には、様々なオペレーティング・プラットフォームでのプログラミング手法を例示するサンプル・アプリケーション・プログラムがソース言語で掲載されています。お客様は、サンプル・プログラムが書かれているオペレーティング・プラットフォームのアプリケーション・プログラミング・インターフェースに準拠したアプリケーション・プログラムの開発、使用、販売、配布を目的として、いかなる形式においても、IBM に対価を支払うことなくこれを複製し、改変し、配布することができます。このサンプル・プログラムは、あらゆる条件下における完全なテストを経ていません。従って IBM は、これらのサンプル・プログラムについて信頼性、利便性もしくは機能性があることをほのめかしたり、保証することはできません。

それぞれの複製物、サンプル・プログラムのいかなる部分、またはすべての派生的創作物にも、次のように、著作権表示を入れていただく必要があります。

© (お客様の会社名) (西暦年)。このコードの一部は、IBM Corp. のサンプル・プログラムから取られています。

© Copyright IBM Corp. _年を入れる_. All rights reserved.

商標

IBM、IBM ロゴ、ibm.com®、Solid®、solidDB、InfoSphere、DB2®、Informix®、および WebSphere® は、International Business Machines Corporation の米国およびその他の国における商標です。これらおよび他の IBM 商標に、この情報の最初に現れる個所で商標表示 (® または ™) が付されている場合、これらの表示は、この情報が公開された時点で、米国において、IBM が所有する登録商標またはコモン・ロー上の商標であることを示しています。このような商標は、その他の国においても登録商標またはコモン・ロー上の商標である可能性があります。現時点での IBM の商標リストについては、「Copyright and trademark information」(www.ibm.com/legal/copytrade.shtml) をご覧下さい。

Java およびすべての Java 関連の商標およびロゴは Sun Microsystems, Inc.の米国およびその他の国における商標です。

Linux® は、Linus Torvalds の米国およびその他の国における商標です。

Microsoft および Windows は、Microsoft Corporation の米国およびその他の国における商標です。

UNIX は、The Open Group の米国およびその他の国における登録商標です。

他の会社名、製品名およびサービス名等はそれぞれ各社の商標です。

Printed in Japan

SC88-5815-00

日本アイ・ビー・エム株式会社
〒106-8711　東京都港区六本木3-2-12