SYBASE iAnywhere

SQL Remote

改訂 2007 年 3 月

版権と商標

目次

はじめに

このマニュアルの内容

このマニュアルでは、モバイル・コンピューティング用の SQL Remote データ・レプリケーション・システムについて説明します。このシステムによって、SQL Anywhere の統合データベースと複数の SQL Anywhere リモート・データベースの間で、電子メールやファイル転送などの間接的リンクを使用したデータ共有が可能になります。

対象読者

このマニュアルは、使用している情報システムに SQL Remote レプリケーションを追加したいと考えている SQL Anywhere のユーザを対象としています。

始める前に

SQL Remote と他の SQL Anywhere レプリケーション・テクノロジとの比較については、「データ交換テクノロジの概要」『SQL Anywhere 10 - 紹介』を参照してください。

SQL Anywhere のマニュアル

このマニュアルは、SQL Anywhere のマニュアル・セットの一部です。この項では、マニュアル・セットに含まれる各マニュアルと使用法について説明します。

SQL Anywhere のマニュアル

SQL Anywhere の完全なマニュアルは、各マニュアルをまとめたオンライン形式とマニュアル別の PDF ファイルで提供されます。いずれの形式のマニュアルも、同じ情報が含まれ、次のマニュアルから構成されます。

- **『SQL Anywhere 10 - 紹介』** このマニュアルでは、データの管理および交換機能を提供する包括的なパッケージである SQL Anywhere 10 について説明します。SQL Anywhere を使用すると、サーバ環境、デスクトップ環境、モバイル環境、リモート・オフィス環境に適したデータベース・ベースのアプリケーションを迅速に開発できるようになります。
- **『SQL Anywhere 10 - 変更点とアップグレード』** このマニュアルでは、SQL Anywhere 10 とそれ以前のバージョンに含まれる新機能について説明します。
- **『SQL Anywhere サーバ - データベース管理』** このマニュアルでは、SQL Anywhere データベースの実行、管理、設定について説明します。管理ユーティリティとオプションのほか、データベース接続、データベース・サーバ、データベース・ファイル、バックアップ・プロシージャ、セキュリティ、高可用性、Replication Server を使用したレプリケーションについて説明します。
- **『SQL Anywhere サーバ - SQL の使用法』** このマニュアルでは、データベースの設計と作成の方法、データのインポート・エクスポート・変更の方法、データの検索方法、ストアド・プロシージャとトリガの構築方法について説明します。
- **『SQL Anywhere サーバ - SQL リファレンス』** このマニュアルは、SQL Anywhere で使用する SQL 言語の完全なリファレンスです。また、SQL Anywhere のシステム・ビューとシステム・プロシージャについても説明しています。
- **『SQL Anywhere サーバ - プログラミング』** このマニュアルでは、C、C++、Java プログラミング言語、Visual Studio .NET を使用してデータベース・アプリケーションを構築、配備する方法について説明します。Visual Basic や PowerBuilder などのツールのユーザは、それらのツールのプログラミング・インタフェースを使用できます。
- **『SQL Anywhere 10 - エラー・メッセージ』** このマニュアルでは、SQL Anywhere エラー・メッセージの完全なリストを、その診断情報とともに説明します。
- **『Mobile Link - クイック・スタート』** このマニュアルでは、セッションベースのリレーショナル・データベース同期システムである Mobile Link について説明します。Mobile Link テクノロジは、双方向レプリケーションを可能にし、モバイル・コンピューティング環境に非常に適しています。
- **『Mobile Link - サーバ管理』** このマニュアルでは、Mobile Link アプリケーションを設定して管理する方法について説明します。

- **『Mobile Link - クライアント管理』** このマニュアルでは、Mobile Link クライアントを設定、構成、同期する方法について説明します。Mobile Link クライアントには、SQL Anywhere または Ultra Light のいずれかのデータベースを使用できます。
- **『Mobile Link - サーバ起動同期』** このマニュアルでは、Mobile Link のサーバによって開始される同期について説明します。サーバによって開始される同期とは、統合データベースから同期またはその他のリモート・アクションの開始を可能にする Mobile Link の機能です。
- **『QAnywhere』** このマニュアルでは QAnywhere について説明します。QAnywhere は、従来のデスクトップ・クライアントやラップトップ・クライアント用のメッセージング・プラットフォームであるほか、モバイル・クライアントや無線クライアント用のメッセージング・プラットフォームでもあります。
- **『SQL Remote』** このマニュアルでは、モバイル・コンピューティング用の SQL Remote データ・レプリケーション・システムについて説明します。このシステムによって、SQL Anywhere の統合データベースと複数の SQL Anywhere リモート・データベースの間で、電子メールやファイル転送などの間接的リンクを使用したデータ共有が可能になります。
- **『SQL Anywhere 10 - コンテキスト別ヘルプ』** このマニュアルには、[接続] ダイアログ、クエリ・エディタ、Mobile Link モニタ、SQL Anywhere コンソール・ユーティリティ、インデックス・コンサルタント、Interactive SQL のコンテキスト別のヘルプが収録されています。
- **『Ultra Light - データベース管理とリファレンス』** このマニュアルでは、小型デバイス用 Ultra Light データベース・システムの概要を説明します。
- **『Ultra Light - AppForge プログラミング』** このマニュアルでは、Ultra Light for AppForge について説明します。Ultra Light for AppForge を使用すると、Palm OS、Symbian OS、または Windows CE を搭載しているハンドヘルド、モバイル、または埋め込みデバイスに対してデータベース・アプリケーションを開発、配備できます。
- **『Ultra Light - .NET プログラミング』** このマニュアルでは、Ultra Light.NET について説明します。Ultra Light.NET を使用すると、PC、ハンドヘルド、モバイル、埋め込みデバイスのデータベース・アプリケーションを開発し、これらのデバイスに配備できます。
- **『Ultra Light - M-Business Anywhere プログラミング』** このマニュアルは、Ultra Light for M-Business Anywhere について説明します。Ultra Light for M-Business Anywhere を使用すると、Palm OS、Windows CE、または Windows XP を搭載しているハンドヘルド、モバイル、または埋め込みデバイスに対して Web ベースのデータベース・アプリケーションを開発、配備できます。
- **『Ultra Light - C/C++ プログラミング』** このマニュアルでは、Ultra Light C および Ultra Light C++ のプログラミング・インタフェースについて説明します。Ultra Light を使用すると、ハンドヘルド、モバイル、埋め込みデバイスに対してデータベース・アプリケーションを開発、配備できます。

マニュアルの形式

SQL Anywhere のマニュアルは、次の形式で提供されています。

- **オンライン・マニュアル** オンライン・マニュアルには、SQL Anywhere の完全なマニュアルがあり、SQL Anywhere ツールに関する印刷マニュアルとコンテキスト別のヘルプの両方が含

まれています。オンライン・マニュアルは、製品のメンテナンス・リリースごとに更新されます。これは、最新の情報を含む最も完全なマニュアルです。

Windows オペレーティング・システムでオンライン・マニュアルにアクセスするには、[スタート] − [プログラム] − [SQL Anywhere 10] − [オンライン・マニュアル] を選択します。オンライン・マニュアルをナビゲートするには、左ウィンドウ枠で HTML ヘルプの目次、索引、検索機能を使用し、右ウィンドウ枠でリンク情報とメニューを使用します。

UNIX オペレーティング・システムでオンライン・マニュアルにアクセスするには、SQL Anywhere のインストール・ディレクトリまたはインストール CD に保存されている HTML マニュアルを参照してください。

- **PDF ファイル** SQL Anywhere の完全なマニュアル・セットは、Adobe Reader で表示できる Adobe Portable Document Format (pdf) 形式のファイルとして提供されています。

  Windows では、PDF 形式のマニュアルはオンライン・マニュアルの各ページ上部にある PDF のリンクから、または Windows の [スタート] メニュー ([スタート] − [プログラム] − [SQL Anywhere 10] − [オンライン・マニュアル - PDF フォーマット]) からアクセスできます。

  UNIX では、PDF 形式のマニュアルはインストール CD にあります。

表記の規則

この項では、このマニュアルで使用されている書体およびグラフィック表現の規則について説明します。

SQL 構文の表記規則

SQL 構文の表記には、次の規則が適用されます。

- **キーワード**　SQL キーワードはすべて次の例に示す ALTER TABLE のように大文字で表記します。

  ALTER TABLE [*owner.*]*table-name*

- **プレースホルダ**　適切な識別子または式で置き換えられる項目は、次の例に示す *owner* や *table-name* のように表記します。

  ALTER TABLE [*owner.*]*table-name*

- **繰り返し項目**　繰り返し項目のリストは、次の例に示す *column-constraint* のように、リストの要素の後ろに省略記号 (ピリオド 3 つ …) を付けて表します。

  ADD *column-definition* [*column-constraint*, …]

  複数の要素を指定できます。複数の要素を指定する場合は、各要素間をカンマで区切る必要があります。

- **オプション部分**　文のオプション部分は角カッコで囲みます。

  RELEASE SAVEPOINT [*savepoint-name*]

  この例では、角カッコで囲まれた *savepoint-name* がオプション部分です。角カッコは入力しないでください。

- **オプション**　項目リストから 1 つだけ選択する場合や、何も選択しなくてもよい場合は、項目間を縦線で区切り、リスト全体を角カッコで囲みます。

  [**ASC** | **DESC**]

  この例では、ASC と DESC のどちらか 1 つを選択しても、選択しなくてもかまいません。角カッコは入力しないでください。

- **選択肢**　オプションの中の 1 つを必ず選択しなければならない場合は、選択肢を中カッコで囲み、縦棒で区切ります。

  [**QUOTES** { **ON** | **OFF** }]

  QUOTES オプションを使用する場合は、ON または OFF のどちらかを選択する必要があります。角カッコと中カッコは入力しないでください。

オペレーティング・システムの表記規則

- **Windows**　デスクトップおよびラップトップ・コンピュータ用の Microsoft Windows オペレーティング・システムのファミリのことです。Windows ファミリには Windows Vista や Windows XP も含まれます。
- **Windows CE**　Microsoft Windows CE モジュラ・オペレーティング・システムに基づいて構築されたプラットフォームです。Windows Mobile や Windows Embedded CE などのプラットフォームが含まれます。

  Windows Mobile は Windows CE 上に構築されています。これにより、Windows のユーザ・インタフェースや、Word や Excel といったアプリケーションの小規模バージョンなどの追加機能が実現されています。Windows Mobile は、モバイル・デバイスで最も広く使用されています。

  SQL Anywhere の制限事項や相違点は、基盤となっているオペレーティング・システム (Windows CE) に由来しており、使用しているプラットフォーム (Windows Mobile など) に依存していることはほとんどありません。
- **UNIX**　特に記述がないかぎり、UNIX は Linux プラットフォームと UNIX プラットフォームの両方のことです。

ファイルの命名規則

マニュアルでは、パス名やファイル名などのオペレーティング・システムに依存するタスクと機能を表すときは、通常 Windows の表記規則が使用されます。ほとんどの場合、他のオペレーティング・システムで使用される構文に簡単に変換できます。

- **ディレクトリ名とパス名**　マニュアルでは、ドライブを示すコロンや、ディレクトリの区切り文字として使用する円記号など、Windows の表記規則を使用して、ディレクトリ・パスのリストを示します。次に例を示します。

  ```
  MobiLink¥redirector
  ```

  UNIX、Linux、Mac OS X では、代わりにスラッシュを使用してください。次に例を示します。

  ```
  MobiLink/redirector
  ```

  SQL Anywhere がマルチプラットフォーム環境で使用されている場合、プラットフォーム間でのパス名の違いに注意する必要があります。
- **実行ファイル**　マニュアルでは、実行ファイルの名前は、Windows の表記規則が使用され、拡張子 *.exe* が付きます。UNIX、Linux、Mac OS X では、実行ファイルの名前には拡張子は付きません。NetWare では、実行ファイルの名前には、拡張子 *.nlm* が付きます。

  たとえば、Windows では、ネットワーク・データベース・サーバは *dbsrv10.exe* です。UNIX、Linux、Mac OS X では、*dbsrv10* になります。NetWare では、*dbsrv10.nlm* になります。
- **install-dir**　インストール・プロセスでは、SQL Anywhere をインストールするロケーションを選択できます。マニュアルでは、このロケーションは *install-dir* という表記で示されます。

インストールが完了すると、環境変数 SQLANY10 によって SQL Anywhere コンポーネントがあるインストール・ディレクトリのロケーション (*install-dir*) が指定されます。SQLANYSH10 は、SQL Anywhere が他の Sybase アプリケーションと共有しているコンポーネントがあるディレクトリのロケーションを指定します。

オペレーティング・システム別の *install-dir* のデフォルト・ロケーションの詳細については、「SQLANY10 環境変数」『SQL Anywhere サーバ - データベース管理』を参照してください。

- **samples-dir** インストール・プロセスでは、SQL Anywhere に含まれるサンプルをインストールするロケーションを選択できます。マニュアルでは、このロケーションは *samples-dir* という表記で示されます。

  インストールが完了すると、環境変数 SQLANYSAMP10 によってサンプルがあるディレクトリのロケーション (*samples-dir*) が指定されます。Windows の [スタート] メニューから、[プログラム] – [SQL Anywhere 10] – [サンプル・アプリケーションおよびプロジェクト] を選択すると、このディレクトリで [Windows エクスプローラ] ウィンドウが表示されます。

  オペレーティング・システム別の *samples-dir* のデフォルト・ロケーションの詳細については、「サンプル・ディレクトリ」『SQL Anywhere サーバ - データベース管理』を参照してください。

- **環境変数** マニュアルでは、環境変数設定が引用されます。Windows では、環境変数を参照するのに、構文 *%envvar%* が使用されます。UNIX、Linux、Mac OS X では、環境変数を参照するのに、構文 *$envvar* または *${envvar}* が使用されます。

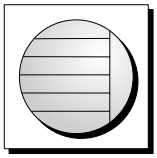
  UNIX、Linux、Mac OS X 環境変数は、*.cshrc* や *.tcshrc* などのシェルとログイン・スタートアップ・ファイルに格納されます。

グラフィック・アイコン

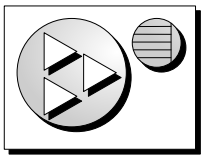
このマニュアルでは、次のアイコンを使用します。

- クライアント・アプリケーション

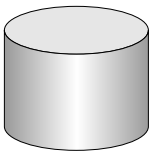
- SQL Anywhere などのデータベース・サーバ



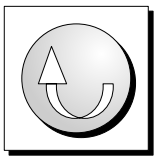
- Ultra Light アプリケーション



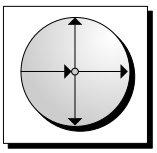
- データベース。高度な図では、データベースとデータベースを管理するデータ・サーバの両方をこのアイコンで表します。



- レプリケーションまたは同期のミドルウェア。ソフトウェアのこれらの部分は、データベース間のデータ共有を支援します。たとえば、Mobile Link サーバ、SQL Remote Message Agent などが挙げられます。



- Sybase Replication Server



- プログラミング・インタフェース

詳細情報の検索／フィードバックの提供

詳細情報の検索

詳しい情報やリソース (コード交換など) については、iAnywhere Developer Network (http://www.ianywhere.com/developer/) を参照してください。

ご質問がある場合や支援が必要な場合は、次に示す Sybase iAnywhere ニュースグループのいずれかにメッセージをお寄せください。

ニュースグループにメッセージをお送りいただく際には、ご使用の SQL Anywhere バージョンのビルド番号を明記し、現在発生している問題について詳しくお知らせくださいますようお願いいたします。バージョン情報は、コマンド・プロンプトで **dbeng10 -v** と入力して確認できます。

ニュースグループは、ニュース・サーバ *forums.sybase.com* にあります (ニュースグループにおけるサービスは英語でのみの提供となります)。以下のニュースグループがあります。

- sybase.public.sqlanywhere.general
- sybase.public.sqlanywhere.linux
- sybase.public.sqlanywhere.mobilink
- sybase.public.sqlanywhere.product_futures_discussion
- sybase.public.sqlanywhere.replication
- sybase.public.sqlanywhere.ultralite
- ianywhere.public.sqlanywhere.qanywhere

ニュースグループに関するお断り
iAnywhere Solutions は、ニュースグループ上に解決策、情報、または意見を提供する義務を負うものではありません。また、システム・オペレータ以外のスタッフにこのサービスを監視させて、操作状況や可用性を保証する義務もありません。
iAnywhere のテクニカル・アドバイザとその他のスタッフは、時間のある場合にかぎりニュースグループでの支援を行います。こうした支援は基本的にボランティアで行われるため、解決策や情報を定期的に提供できるとはかぎりません。支援できるかどうかは、スタッフの仕事量に左右されます。

フィードバック

このマニュアルに関するご意見、ご提案、フィードバックをお寄せください。

マニュアルに関するご意見、ご提案は、SQL Anywhere ドキュメンテーション・チームの iasdoc@ianywhere.com 宛てに電子メールでお寄せください。このアドレスに送信された電子メールに返信はいたしませんが、お寄せいただいたご意見、ご提案は必ず読ませていただきます。

マニュアルまたはソフトウェアについてのフィードバックは、上記のニュースグループを通してお寄せいただいてもかまいません。

パート I. SQL Remote の概要

パート I では、SQL Remote の概念、アーキテクチャ、機能について説明します。

第 1 章

SQLRemote の概念

目次

SQL Remote の概要

SQL Remote は、統合データ・サーバと、一般的には数多くのモバイル・データベースを含む多数のリモート・データベースとの間で双方向レプリケーションを行うためのデータレプリケーション・テクノロジです。

SQL Remote はメッセージを通じてレプリケートするので、サーバ間で直接接続する必要はありません。必要に応じて、ダイヤルアップ接続や電子メールを使用します。

リモート・サイトで必要な管理とリソースは最小限で済みます。統合データベースとリモート・データベース間のタイム・ラグは、分単位、時間単位、または日数単位で設定できます。

SQL Remote を使用するには、適切なライセンスを取得した SQL Remote ソフトウェアを、使用する個々のデータベースにインストールする必要があります。

SQL Remote の概念と機能の詳細については、「SQLRemote の概念」 3 ページを参照してください。

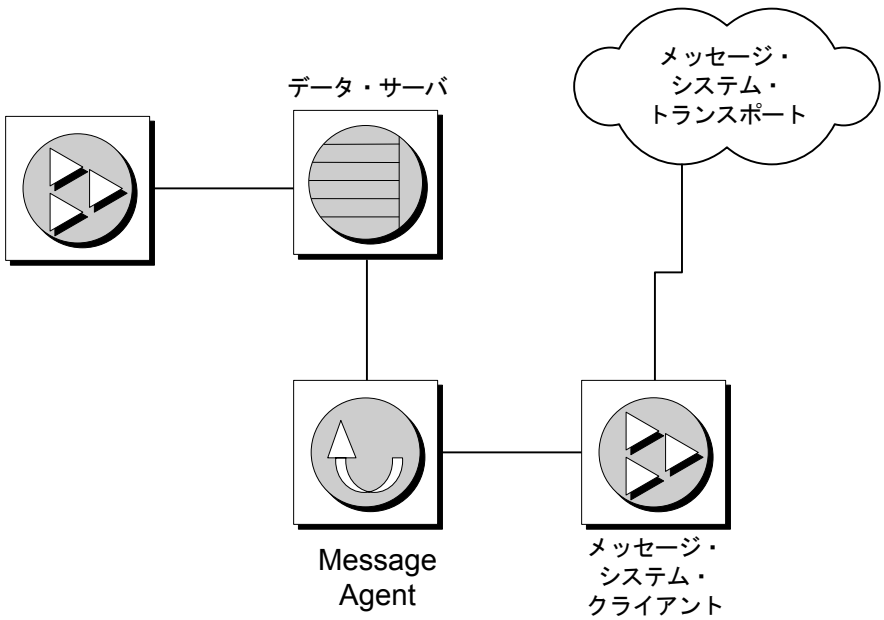
サポートされているオペレーティング・システムとメッセージ・リンクの一覧については、「プラットフォーム別 SQL Anywhere 10.0.1 コンポーネント」にある SQL Remote for SQL Anywhere の表を参照してください。

SQL Remote のコンポーネント

SQL Remote には、以下のコンポーネントが必要です。

- **データ・サーバ** データを管理するために、各リモート・サイトと統合データベースに SQL Anywhere データベースが必要です。
- **Message Agent** SQL Remote メッセージを送受信するために、統合サイトと各リモート・サイトに SQL Remote Message Agent が必要です。

  Message Agent は、クライアント／サーバ接続でデータ・サーバに接続します。Message Agent は、データ・サーバと同じマシンでも別のマシンでも使用できます。
- **データベース抽出ユーティリティ** 抽出ユーティリティを使用して、開発、テスト、および配備の際に、統合データベースからリモート・データベースを作成できます。
- **メッセージ・システム・クライアント・ソフトウェア** SQL Remote は、既存のメッセージ・システムを使用して、レプリケーション・メッセージを転送します。SQL Remote に備わっているファイル共有「メッセージ・システム」を使用する場合、クライアント・ソフトウェアは不要です。ファイル共有メッセージ・システム以外を使用して SQL Remote レプリケーションを行う場合は、そのメッセージ・システムを各コンピュータにインストールする必要があります。
- **クライアント・アプリケーション** SQL Remote データベースで使用するのは、標準のクライアント／サーバ・データベース・アプリケーションです。



データ・サーバ

データ・サーバは、SQL Anywhere サーバである必要があります。

クライアント・アプリケーション

クライアント・アプリケーションでは、ODBC や Embedded SQL だけでなく、その他のさまざまなプログラミング・インタフェースを使用できます。

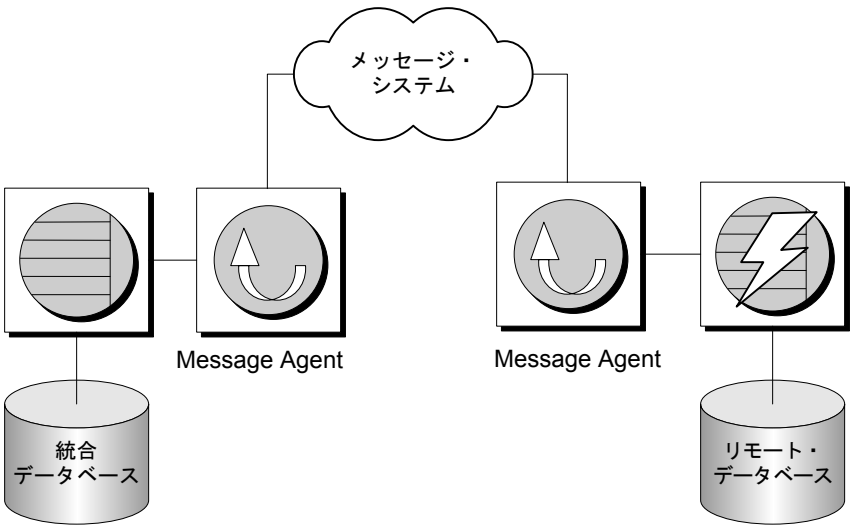
「SQL Anywhere データ・アクセス・プログラミング・インタフェース」『SQL Anywhere サーバ - プログラミング』を参照してください。

統合データベースとリモート・データベースのどちらを使用しているかを、クライアント・アプリケーション側で認識する必要はありません。クライアント・アプリケーション側から見ると、どちらでも同じだからです。

Message Agent

SQL Remote の「**Message Agent**」は、レプリケーション・メッセージの送受信を行います。これはデータベース間のメッセージの送受信を行うクライアント・アプリケーションです。統合サイトとリモート・サイトの両方に、Message Agent をインストールする必要があります。

Windows オペレーティング・システムでは *dbremote.exe*、UNIX では *dbremote* というプログラムが Message Agent です。



メッセージ・システム・クライアント

共有ファイル・メッセージ・システムを使用している場合、メッセージ・システム・クライアントは必要ありません。

電子メールやその他のメッセージ・システムを使用している場合、メッセージを送受信するためには、そのクライアント用のメッセージ・システムが必要です。

SQL Remote のパブリケーションとサブスクリプション

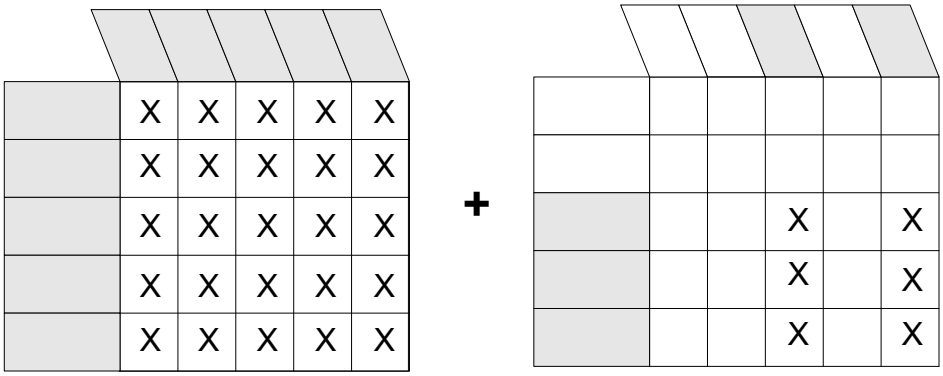
SQL Remote がレプリケートしたデータは、「**パブリケーション**」に記述されます。パブリケーション内の情報を共有する各データベースには、そのパブリケーションに対する「**サブスクリプション**」が必要です。

データはパブリケーションに記述される

パブリケーションは、レプリケートするデータが記述されたデータベース・オブジェクトです。データベースのリモート・ユーザがパブリケーションを受信するには、パブリケーションに対してサブスクリプションを作成する必要があります。

1 つのパブリケーションは、複数のデータベース・テーブルから取得したデータを含むことができます。各テーブルからパブリケーションに提供されるデータを、アーティクルと呼びます。各アーティクルは、1 つのテーブル全体か、1 つのテーブル内のローとカラムのサブセットで構成されます。

2 つのテーブルの同期定義



アーティクル 1 :
テーブル A 全体

アーティクル 2 :
テーブル B の一部のローとカラム

データベース内の各パブリケーションに対して変更を加えると、その変更内容はパブリケーションのサブスクライバに定期的にレプリケートされます。このようなレプリケーションを、パブリケーションの更新と呼びます。

メッセージは常に双方向に送信される

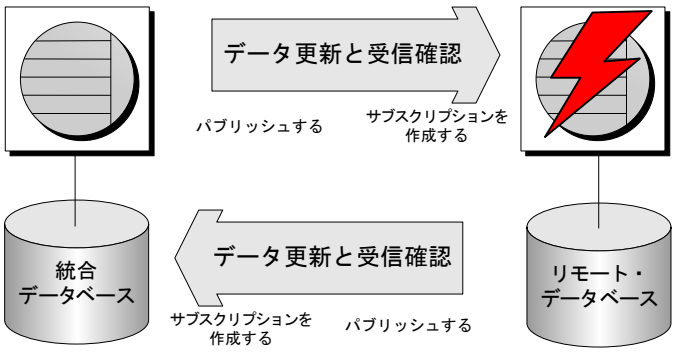
リモート・データベースは、統合データベースからデータを受信できるように、統合データベース上のパブリケーションにサブスクライブします。そのためには、統合データベースでサブスクリプションを作成し、名前と受信するパブリケーションでサブスクライバを識別します。

SQL Remote では、常に双方向にメッセージが送信されます。統合データベースは、パブリケーション更新を含んだメッセージをリモート・データベースに送信します。リモート・データベースから統合データベースにも、メッセージが送信されます。

たとえば、統合データベースのパブリケーション内でデータが更新されると、更新内容がリモート・データベースに送信されます。また、リモート・データベースでデータが更新されていない場合でも、レプリケーションのステータスを把握するために、リモート・データベースから統合データベースに確認メッセージを送り返す必要があります。

両方のデータベースがサブスクリプションを作成する

メッセージは双方向に送信する必要があります。そのため、統合データベースに作成されたパブリケーションに対してリモート・データベースがサブスクリプションを作成するだけでなく、リモート・データベースに作成された対応するパブリケーションに対しても、統合データベースがサブスクリプションを作成する必要があります。



リモート・データベース・ユーザが、自分のマシンにコピーされたデータに変更を加えると、変更内容が統合データベースにレプリケートされます。変更を含んだメッセージを統合データベースで適用すると、その変更が統合データベースのパブリケーションに取り込まれ、次回にすべてのリモート・サイト (元の変更を加えたサイトを除く) を更新するときに一緒に変更されます。このように、リモート・サイト間でのレプリケーションは、統合データベースを経由して行われます。

リモート・データベースの設定

サブスクリプションを最初に設定したときは、2 つのデータベースに同じ情報が含まれ、レプリケーションを開始できる状態にする必要があります。この設定は手動でも行えますが、データベース抽出ユーティリティで自動的に行うこともできます。抽出ユーティリティはコマンド・ライン・ユーティリティとして、または Sybase Central から実行できます。

SQL Remote データベース抽出ユーティリティでリモート・データベースを作成すると、適切なパブリケーションとサブスクリプションがリモート・データベースに自動的に作成されます。

SQL Remote の機能

SQL Remote の主な機能は以下のとおりです。

多数のサブスクライバのサポート　SQL Remote は、1 つのパブリケーションに対して多数のサブスクライバがある状態でのレプリケーションをサポートしています。

これは、モバイル環境用のアプリケーションにとっては特に重要な機能です。モバイル環境では、オフィスにある 1 つのデータベースから、膨大な数の営業担当者のラップトップ・コンピュータにレプリケートする必要があるからです。

トランザクション・ログ・ベースのレプリケーション　SQL Remote でのレプリケーションは、トランザクション・ログをベースとして行われます。この方法では、更新時に毎回すべてのデータをレプリケートするのではなく、変更のみをレプリケートすることができます。ログ・ベースのレプリケーションは、パフォーマンス面でも他のレプリケーション・システムより優れています。

トランザクション・ログは、データベースに加えた変更がすべて記録されたレポジトリです。SQL Remote は、トランザクション・ログの記録に従って、データベースへの変更をレプリケートします。何らかのパブリケーションに含まれる統合データベースでは、トランザクション・ログに記録されているコミットされた全トランザクションが、定期的にリモート・データベースに送信されます。リモート・サイトでは、トランザクション・ログに記録されているコミットされた全トランザクションが、定期的に統合データベースに送信されます。

SQL Remote では、データのレプリケート中に多少のタイムラグが生じますが、コミットされたトランザクションのみをレプリケートすることによって、レプリケーション設定内でのトランザクションのアトミック性を適正に保ち、レプリケーションに関わるデータベース間で一貫性を維持します。

集中管理　SQL Remote は、統合データベースでの集中管理を考えて設計されています。これは、モバイル環境用アプリケーションでは特に重要なことです。ラップトップ・コンピュータのユーザが、データベース管理タスクを行う必要があってはならないからです。また、小規模なオフィスがいくつかあり、それぞれにサーバはあっても管理タスクを行うほどのリソースがないという状況でのレプリケーションにも、集中管理が必要です。

管理タスクには、パブリケーション、リモート・ユーザ、サブスクリプションの設定や管理のほか、エラーや競合が起きた場合の解決も含まれます。

経済的なリソース要件　SQL Remote の実行に必要なソフトウェアは、お使いの SQL Anywhere DBMS 以外には、Message Agent とメッセージ・システムのみです。共有ファイル・リンクを使用していて、メッセージ・ファイルの保管先ディレクトリに各リモート・ユーザ ID でアクセスできる場合は、メッセージ・システム・ソフトウェアも不要です。

レプリケーション・システムのコンポーネントは、どれもそれほど多くのメモリとディスク容量を必要としないので、SQL Remote を実行するためにハードウェアを増設する必要はありません。

マルチプラットフォーム・サポート　SQL Remote は、さまざまなオペレーティング・システムとメッセージ・リンクに対応しています。

「プラットフォーム別 SQL Anywhere 10.0.1 コンポーネント」にある SQL Remote for SQL Anywhere の表を参照してください。

典型的な SQL Remote 設定

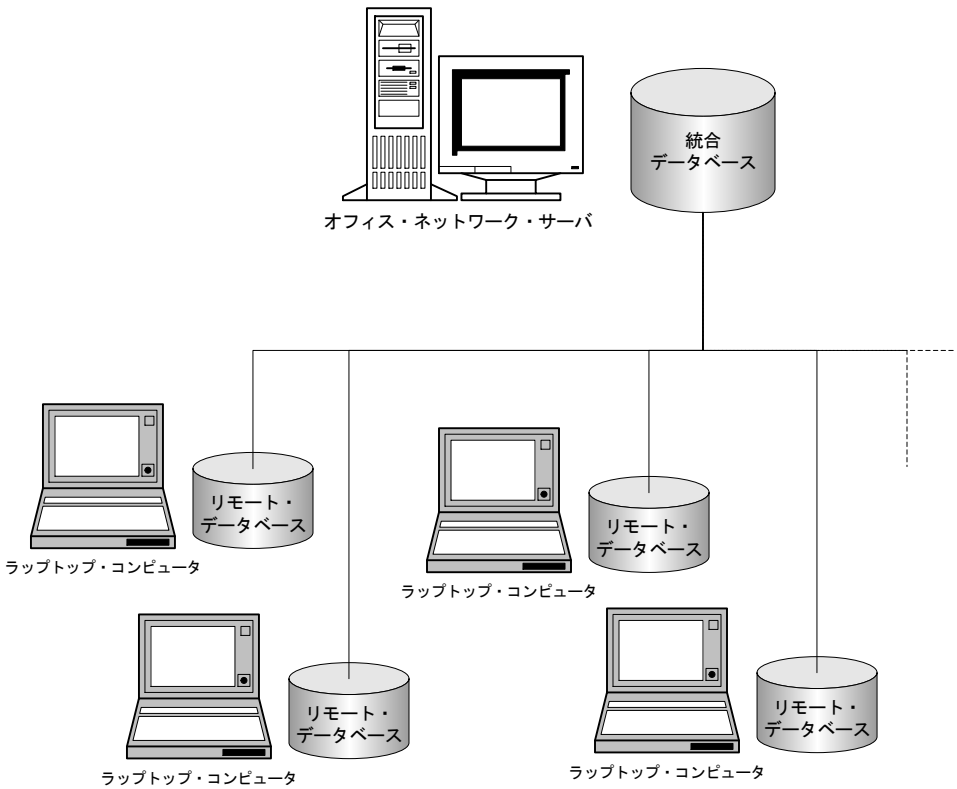
SQL Remote ではさまざまな環境下でレプリケーション・サービスを利用でき、その機能は以下の特徴を考慮して設計されています。

- モバイル環境用アプリケーションの場合と同様に、リモート・データベースに管理の負荷を課すことができない場合にも、SQL Remote がソリューションとなること。
- サイト間のデータ通信は、散発的かつ間接的であること。永続的で直接的である必要はない。
- リモート・サイトでのメモリとリソースの要件を重要視する。

以下に、SQL Remote の典型的な設定を示します。

モバイル環境でのサーバ／ラップトップ間レプリケーション

SQL Remote では、オフィス・ネットワーク上のデータベースと、営業担当者のラップトップ・コンピュータ上にあるパーソナル・データベースとの間で、双方向のレプリケーションが可能です。このような設定では、メッセージの伝送手段として電子メールシステムを使用します。



オフィス・サーバが、会社のデータベースを管理するためのサーバを稼働させている場合、会社のデータベースにある Message Agent は、そのサーバのクライアントとして稼働します。

ラップトップ・コンピュータでは、各営業担当者は SQL Anywhere パーソナル・サーバを使用して、自分のデータを管理します。

営業担当者は、ラップトップ・コンピュータから 1 回電話をかけるだけで、外出先から以下の機能を実行できます。

- 新着メールを受信する。
- 作成済みの電子メールがある場合は送信する。
- オフィス・サーバからパブリケーションの更新を受信する。
- ローカルで行った更新の内容 (新しい注文など) を、オフィス・サーバに送信する。

更新の例としては、その営業担当者が取り扱っている製品の新しい特別割引や、新価格、在庫情報などがあります。これらの情報はラップトップの Message Agent によって読み込まれ、担当者のデータベースに自動的に適用されます。担当者側での操作は一切不要です。

営業担当者が登録した新しい注文も、特別な操作をしなくても自動的にオフィスに送信されます。

複数のオフィスにわたるサーバ間レプリケーション

SQL Remote では、営業所／販売店と本社のデータベース・サーバ間で、双方向レプリケーションが可能です。各営業所では、最初の設定とサーバの管理さえ行うことができれば、それ以上のデータベース管理の経験は必要ありません。

SQL Remote は、各サイトのデータを常に最新に保つような設計にはなっていません。たとえば、電子メール・システムを使用して、レプリケーションを実行することがあります。あるいは、ダイアルアップ・システムとファイル転送ソフトウェアを随時使用して、FILE メッセージ・システムを実行することもできます。

SQL Remote では、簡単な設定を行うだけで、各オフィスでそれぞれに適したデータ・セットを受信できます。オフィス専用のテーブル (フランチャイズ・オフィスの場合はスタッフ・レコードなど) は、レプリケーション・データと同じデータベース内にあっても機密性を保つことができます。

SQL Remote の階層には、レイヤを追加できます。たとえば、各営業所のサーバを統合データベースとして利用して、その営業所からのリモート・サブスクライバをサポートすることができます。

パート II. SQL Remote のレプリケーション設計

パート II では、SQL Remote のレプリケーション設計の問題について説明します。

第 2 章

SQL Remote 設計の原則

目次

SQL Remote パブリケーション設計の概要

この章では、SQL Remote アプリケーションの設計時に考慮すべきパブリケーションの設計に関する一般的な問題について説明します。また、SQL Remote がデータをレプリケートする方法についても説明します。

システム固有の問題の詳細については、「SQL Remote インストール環境の設計」 27 ページを参照してください。

統合データベースの設計

すべての SQL Remote 管理タスクと同様に、設計はデータベース管理者またはシステム管理者が統合データベースで実行します。

データベース管理者が、すべての SQL Remote 設定タスクを実行するようにしてください。

互換性のあるソート順と文字セットの使用

SQL Remote Message Agent は、文字セットを変換しません。

SQL Anywhere 統合データベースが使用する文字セットと照合順を、リモート・データベースと同じ設定にする必要があります。サポートされている文字セットについては、「国際言語と文字セット」『SQL Anywhere サーバ - データベース管理』を参照してください。

文のレプリケーション方法

SQL Remote のレプリケーションは、トランザクション・ログに基づいて行われ、個々のデータを更新するときに、すべてのデータを更新する代わりに変更部分だけをレプリケートします。SQL Remote は、データをレプリケートする場合、実際はデータを変更する SQL 文をレプリケートします。

コミットされたトランザクションのみがレプリケートされる

SQL Remote は、コミットされたトランザクションの文のみをレプリケートします。これによって、データのレプリケート中に多少のタイムラグが生じますが、レプリケーション設定内でのトランザクションのアトミック性が適正に保たれ、レプリケーションに関わるデータベース間で一貫性が維持されます。

プライマリ・キー

UPDATE 文または DELETE 文をレプリケートする場合、SQL Remote はプライマリ・キー・カラムを使用して、更新または削除を行うローをユニークに識別します。レプリケートされるすべてのテーブルは、宣言されたプライマリ・キーを持つか、一意性制約を守るものでなければなりません。ユニーク・インデックスでは十分ではありません。プライマリ・キーのカラムは、レプリケートされた更新や削除の WHERE 句で使用されます。テーブルにプライマリ・キーがない場合、WHERE 句はテーブルのすべてのカラムを参照します。

UPDATE 文は常に UPDATE 文とは限らない

簡単な INSERT 文をあるデータベースに入力すると、その文は、SQL Remote 設定内の他のデータベースに INSERT 文として送信されます。しかし、すべての文がクライアント・アプリケーションで入力されたとおりにレプリケートされるわけではありません。ここでは、SQL Remote が SQL 文をレプリケートする方法について説明します。堅牢な SQL Remote インストール環境を設計するには、このしくみを理解することが重要です。

Message Agent は、文のレプリケーションを実行するコンポーネントです。

挿入と削除のレプリケーション

INSERT 文と DELETE 文は、レプリケーションで最も簡単な例です。SQL Remote は、トランザクション・ログから個々の INSERT または DELETE オペレーションを取り出し、その挿入または削除されたローをサブスクライブするすべてのサイトに送信します。

テーブルのカラムのサブセットだけがサブスクライブされている場合、サブスクライバに送信される INSERT 文にはそのサブセットのみが含まれます。

Message Agent は、文を最初に入力したユーザにそれらの文がレプリケートされないようにします。

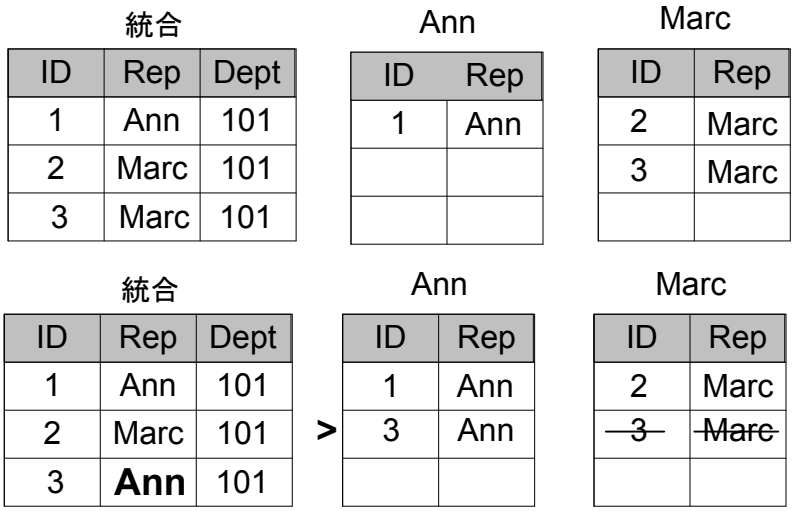
更新のレプリケーション

UPDATE 文は、クライアント・アプリケーションが入力したとおりにはレプリケートされません。ここでは、レプリケートされる UPDATE 文が、入力した UPDATE 文と異なる 2 つの場合を説明します。

UPDATE 文が INSERT 文または DELETE 文としてレプリケートされる場合

UPDATE 文は、いずれかのリモート・ユーザのサブスクリプションからローを削除する場合、DELETE 文としてそのユーザに送信されます。UPDATE 文は、いずれかのリモート・ユーザのサブスクリプションにローを追加する場合、INSERT 文としてそのユーザに送信されます。

次の図は、各サブスクライバが自分の名前を使用してサブスクライブするパブリケーションを示します。



Marc のローの **Rep** 値を Ann に変更する UPDATE 文は、Marc には DELETE 文として、Ann には INSERT 文としてレプリケートされます。

サブスクライバ間のローの再編成は、営業担当者間に顧客を定期的に再割り当てする営業地域自動割り当てアプリケーションに共通の機能で、「**領域の再編成**」とも呼ばれます。

UPDATE 文の競合の検出

UPDATE 文は、既存の 1 つまたは複数のローの値を新しい値に変更します。変更されるローは、UPDATE 文の WHERE 句によって異なります。

SQL Remote が UPDATE 文をレプリケートする場合、複数のシングル・ロー更新をセットにまとめて行います。これらのシングル・ロー文は、次のいずれかの理由で失敗する場合があります。

- **更新するローが存在しない**　個々のローはプライマリ・キーの値によって識別されるため、プライマリ・キーが他のユーザによって変更された場合は、更新すべきローが見つかりません。

  この場合、UPDATE 文は何も更新しません。

- **更新するローが 1 つまたは複数のカラムで異なる**　存在するはずの値の 1 つが他のユーザによって変更された場合、「**更新の競合**」が発生します。

  リモート・データベースでは、ローの値に関わらず更新が実行されます。

  統合データベースでは、SQL Remote によって「**競合解決**」オペレーションが実行されます。競合解決オペレーションは、トリガまたはストアド・プロシージャ内に格納され、競合が検出されると自動的に実行されます。競合解決トリガは更新の前に実行され、このトリガが終了すると更新が実行されます。

- **主キー制約または一意性制約のないテーブルはレプリケートされた更新の WHERE 句のカラムをすべて参照する**　2 人のユーザが同じローを更新する場合、レプリケートされた更新は更新作業を行わず、データベースの整合性が失われます。したがって、レプリケートされたすべてのテーブルに主キー制約または一意性制約を持たせ、制約対象のカラムが更新されないようにする必要があります。

プロシージャのレプリケーション

どのようなレプリケーション・システムでも、ストアド・プロシージャ・コールをレプリケートするときに、次の 2 つのオプションのどちらかを選択します。

- **プロシージャ・コールをレプリケートする**　対応するプロシージャがレプリケート・サイトで実行されます。
- **プロシージャの動作をレプリケートする**　プロシージャの個別の動作 (INSERT、UPDATE、DELETE など) がレプリケートされます。

SQL Remote は、プロシージャをレプリケートするときに、プロシージャの動作をレプリケートします。プロシージャ・コールはレプリケートされません。

トリガのレプリケーション

デフォルトでは、Message Agent は、トリガが実行した動作をレプリケートしません。トリガはリモートで定義するように想定されています。これによって、パーミッションに関する問題と各動作が 2 回発生する可能性を回避します。この原則には次のような例外があります。

- **競合解析トリガの動作**　競合解析 (RESOLVE UPDATE) トリガが実行する動作は、統合データベースから、競合を発生させたメッセージを送信したデータベースを含む、すべてのデータベースにレプリケート*されます*。
- **BEFORE トリガのレプリケーション**　BEFORE トリガの中には、SQL Remote を使用しているときに好ましくない結果を発生させるものがあるため、更新中のローを変更する BEFORE トリガの動作は、UPDATE 動作の前にレプリケート*されます*。

  インストール環境を設計する場合は、この動作に注意してください。たとえば、ローの更新回数を追跡するカウンタ・カラムをロー内で増加させる BEFORE UPDATE は、レプリケートされると 2 回カウントが行われます。これは、UPDATE がレプリケートされると、BEFORE

UPDATE トリガが起動されるためです。この問題を回避するには、サブスクライバ・データベースにトリガが存在しないこと、またはトリガがレプリケートされた動作を実行しないことを確認してください。また、カラムを最終更新時間に設定する BEFORE UPDATE は、UPDATE がレプリケートされた時間を取得します。

トリガの動作をレプリケートするオプション

Message Agent には、メッセージを送信するときにすべてのトリガをレプリケートさせるオプションがあります。それは、dbremote -t オプションです。

このオプションを使用する場合は、トリガの動作がリモート・データベースで 2 回 (リモート・サイトでトリガを起動するとき、および統合データベースからレプリケートされた動作を明示的に適用するとき) 実行されないようにする必要があります。

トリガの動作が 2 回実行されないようにするには、トリガの本文を IF CURRENT REMOTE USER IS NULL … END IF 文で囲むか、Message Agent のユーザ ID に対して、SQL Anywhere の fire_triggers オプションを Off に設定します。

データ定義文のレプリケーション

データ定義文 (CREATE、ALTER、DROP、およびデータベース・オブジェクトを変更するその他の文) は、パススルー・モードで入力した場合を除き、SQL Remote にレプリケートされません。

「パススルー・モードの使用」 148 ページを参照してください。

データ型がレプリケートされる方法

long binary データや character データ、および datetime データの扱いには特別な注意が必要です。

BLOB のレプリケーション

BLOB は LONG VARCHAR、LONG BINARY、TEXT、および IMAGE データ型で、256 文字より長い値です。

SQL Remote には、データベース間で BLOB をレプリケートする特殊なメソッドがあります。

Message Agent は、レプリケートされている INSERT 文または UPDATE 文の値の代わりに変数を使用します。変数の値は、次のように文を結合して作成します。

```
SET vble = vble || 'more_stuff'
```

この結果、長い値を含んでいる SQL 文が短くなり、1 つのメッセージ内に収まります。SET 文は独立した SQL 文であるため、BLOB は複数の SQL Remote メッセージに効果的に分割されます。

verify_threshold オプションを使用してメッセージ・サイズを最小にする

verify_threshold データベース・オプションを使用すると、(レプリケートされた UPDATE 文の VERIFY 句による) 確認の対象から長い値を除外できます。このオプションのデフォルト値は 1000 です。カラムのデータ型が閾値より長いと、カラムの古い値は UPDATE がレプリケートされるときに確認されません。このようにして、SQL Remote メッセージのサイズが大きくならないようにしますが、競合する長い値の更新が検出されないという短所もあります。

メッセージのサイズを小さくするために verify_threshold を使用しているときでも、競合を検出する方法があります。"BLOB" を更新するときに、同じテーブル内の last_modified カラムも必ず更新するようにします。これにより、last_modified カラムの古い値が確認されるため、競合を検出できます。

ワーク・テーブルを使用して重複する更新を防ぐ

BLOB が繰り返し更新されるときは、「ワーク」テーブルで更新し、最終バージョンをレプリケートされたテーブルに割り当てるようにします。たとえば、作業中の資料が 1 日に 20 回更新される場合、1 日の終りに Message Agent を 1 回実行すると、20 回の更新すべてがレプリケートされます。資料のサイズが 200 KB の場合は、4 MB のメッセージが送信されます。

これよりも良い方法は、**document_in_progress** テーブルを作成することです。ユーザが資料の変更を終了したときに、アプリケーションがその資料を **document_in_progress** テーブルからレプリケートされたテーブルに移動します。この結果、1 回の更新 (200 KB のメッセージ) で済みます。

BLOB のレプリケーションの制御

SQL Anywhere の blob_threshold オプションを使用すると、長い値のレプリケーションを詳細に制御できます。blob_threshold オプションより長い値は、BLOB としてレプリケートされます。つまり、細かく分割して各部分をレプリケートし、SQL 変数を使用して、受信者サイトで分割された部分を連結して再構成します。

日付と時刻のレプリケーション

日付カラムまたは時刻カラムをレプリケートする場合、Message Agent は、データベース・オプションの sr_date_format、sr_time_format、sr_timestamp_format の設定を使用して、日付をフォーマットします。

たとえば、次のオプション設定は、Message Agent に 1987 年 5 月 2 日という日付を 1987-05-02 として送信するように命令します。

```
SET OPTION sr_date_format = 'yyyy-mm-dd'
```

「SQL Remote オプション」 170 ページを参照してください。

日付と時刻をレプリケートするときに、次の点を考慮します。

- 時刻、日付、およびタイムスタンプには、インストール環境全体で一貫したフォーマットを使用します。
- 日付とタイムスタンプのフォーマットに使われている年、月、日の順序が、データベース・オプションの date_order の設定と同じになるようにします。

  個々の接続の間、date_order オプションを変更できます。

どのサブスクライバにどの文が送信されるか

テーブルのローに挿入、削除、または更新が行われるたびに、ローにサブスクリプションを作成したユーザにメッセージを送信する必要があります。また、更新の場合はサブスクリプション式が変更されることがあるため、あるサブスクライバには削除文、別のサブスクライバには更新文、また別のサブスクライバには挿入文が送信されます。

送信される文と宛先のサブスクライバの詳細については、「文のレプリケーション方法」 17 ページを参照してください。

サブスクリプションの詳細については、次の項を参照してください。

- 「SQL Remote インストール環境の設計」 27 ページ
- 「SQL Remote を使用したデータベースの配備と同期」 75 ページ

ここでは、SQL Remote が適切な受信者に適切なオペレーションを送信する方法について説明します。

どのサブスクライバにどのオペレーションを送信するかを決定するタスクは、データベース・サーバと Message Agent が分担します。データベース・サーバはパブリケーションに関係のある作業を処理し、Message Agent はサブスクリプションに関係のある作業を処理します。

SQL Anywhere の動作

SQL Anywhere は、パブリケーションの一部であるテーブルを更新する個々のサブスクリプション式を評価します。更新前と更新後に、式の値をログに追加します。

複数のパブリケーションの一部であるテーブルでは、個々のパブリケーションに対して更新前と更新後に、サブスクリプション式が評価されます。

ログに情報を追加すると、以下の場合にパフォーマンスが低下することがあります。

- **高負荷の式** サブスクリプション式の評価が高い負荷を生む場合は、パフォーマンスが低下することがあります。
- **多数のパブリケーション** テーブルが多数のパブリケーションに属している場合は、多くの式を評価する必要があります。これに対し、サブスクリプションの数はパフォーマンスに影響しません。
- **複数の値が含まれる式** 一部の式には複数の値が含まれています。このような式では、トランザクション・ログの情報が大幅に増加し、パフォーマンスが低下する場合があります。

Message Agent の動作

Message Agent は、トランザクション・ログから評価済みのサブスクリプション式またはサブスクリプション・カラムへのエントリを読み込み、更新前の値と更新後の値をパブリケーションの個々のサブスクライバのサブスクリプション値と照合します。Message Agent は、このようにして適切なオペレーションを個々のサブスクライバに送信します。

サブスクライバの数が非常に多くてもサーバのパフォーマンスは低下しませんが、Message Agent のパフォーマンスは低下する場合があります。サブスクリプション値を大量のサブスクリプション値と照合する作業と、メッセージを送信する作業は、大きな負荷となる場合があります。

レプリケーション・エラーと競合

SQL Remote を使用すると、さまざまなサイトにあるデータベースを更新できます。しかし、特にデータベースの構造が複雑な場合は、レプリケーション・エラーを回避するように注意深くパブリケーションを設計する必要があります。ここでは、レプリケーションの設定で発生する可能性のあるエラーと競合について説明し、次にエラーを回避して競合を処理するようにパブリケーションを設計する方法について説明します。

配信エラーについては説明しません
この項では、メッセージ配信障害に関する問題については説明しません。配信エラーとその解決方法については、「メッセージ・トラッキング・システム」 123 ページを参照してください。

レプリケーション・エラー

次のようなレプリケーション・エラーがあります。

- **重複プライマリ・キー・エラー** 2 人のユーザが同じプライマリ・キー値を使用してローを挿入する場合や、あるユーザがプライマリ・キーを更新し、別のユーザが新しい値のプライマリ・キーを挿入する場合に発生します。レプリケーション・システムのデータベースに 2 度目にアクセスしようとすると、プライマリ・キーが 2 つ作成されるため、エラーが発生します。
- **ローが見つからないエラー** あるユーザがロー (特定のプライマリ・キー値を持つロー) を削除し、別のユーザが異なるサイトで同じローを更新または削除すると発生します。

  この場合は、後者のユーザが UPDATE 文または DELETE 文を送信したときに、ローが見つからないという理由でエラーが発生します。
- **参照整合性エラー** 外部キーを持つカラムがパブリケーションに含まれているにもかかわらず、それと関連付けされているプライマリ・キーが含まれていない場合、抽出ユーティリティはリモート・データベースに外部キーの定義を渡さず、そのリモート・データベースで INSERT 文が失敗しないようにします。

  この問題は、適切なデフォルトをテーブル定義に入れることによって解決できます。

  また、プライマリ・テーブルに SUBSCRIBE BY 式が含まれているが、それと関連付けされている外部テーブルには含まれていない場合に、参照整合性エラーが発生することがあります。外部テーブルのローはレプリケートされる可能性がありますが、プライマリ・テーブルのローはパブリケーションから除外されることがあるためです。

レプリケーションの競合

レプリケーションの競合は、エラーとは異なります。競合は適切に処理すれば、SQL Remote では問題を起こしません。

- **競合**　あるユーザがローを更新するとします。この同じローを別のユーザが他のサイトで更新します。2 番目のユーザのオペレーションが成功すると、SQL Remote によってトリガが起動され、データが適切に変更されるような方法で競合が解決されます。

  競合は多くのインストール環境で発生します。SQL Remote では、SQL Remote 設定の通常のオペレーションの一部として、トリガとプロシージャを使用して競合を適切に解決できます。

SQL エラーのトラッキング

レプリケーションにおける SQL エラーは、セットアップ時にトラッキングできるように設計する必要があります。SQL Remote のオプションを使用して SQL 文のエラーをトラックできますが、このオプションではエラーを解決できません。

replication_error オプションを設定すると、SQL エラーが発生したときに Message Agent で呼び出すストアド・プロシージャを指定できます。デフォルトではプロシージャを呼び出しません。

♦ replication_error オプションを設定するには、次の手順に従います。

1. 次の文を発行します。

```
SET OPTION
remote-user.replication_error
= 'procedure-name'
```

remote-user は Message Agent コマンド・ラインのユーザ ID、*procedure-name* は SQL エラーが検出されたときに呼び出されるプロシージャです。

レプリケーション・エラー・プロシージャの稼働条件

レプリケーション・エラー・プロシージャでは、データ型が CHAR、VARCHAR、または LONG VARCHAR の引数を 1 つ指定してください。プロシージャは、SQL エラー・メッセージで 1 回、エラーの原因となった SQL 文でもう 1 回呼び出されます。

第 3 章

SQL Remote インストール環境の設計

目次

設計の概要

SQL Remote インストール環境を設計するには、次の作業を行います。

- **パブリケーションの設計**　パブリケーションは、データベース間で共有する情報を決定します。
- **サブスクリプションの設計**　サブスクリプションは、各ユーザが受信する情報を決定します。
- **設計の実装**　システムの全ユーザ用に、パブリケーションとサブスクリプションを作成します。

すべての管理タスクは統合データベースで実行する

すべての SQL Remote 管理タスクと同様に、設計はデータベース管理者またはシステム管理者が統合データベースで実行します。

SQL Anywhere データベースの管理者は、すべての SQL Remote 設定タスクを実行します。

データのパブリッシュ

この項では、テーブル全体またはテーブルのカラムワイズ・サブセットで構成される、簡単なパブリケーションの作成方法について説明します。これらのテーブルはアーティクルとも呼ばれます。Sybase Central を使用するか、Interactive SQL の CREATE PUBLICATION 文によって、これらのタスクを実行できます。

Sybase Central のパブリケーションはすべて、パブリケーション・フォルダに表示されます。パブリケーション用に作成するアーティクルはすべて、パブリケーションを選択すると右ウィンドウ枠の [アーティクル] タブに表示されます。

各パブリケーションには 1 つまたは複数の完全テーブルが入りますが、部分テーブルも許可されます。テーブルは、カラム、ローまたはその両方によってさらに分割できます。

テーブル全体のパブリッシュ

作成できる最も簡単なパブリケーションは、単一のアーティクルで構成されます。このアーティクルは、1 つまたは複数のテーブルのすべてのローとカラムで構成されます。これらのテーブルをあらかじめ用意しておく必要があります。

♦ 完全テーブルを 1 つ以上パブリッシュするには、次の手順に従います (Sybase Central の場合)。

1. DBA 権限のあるユーザとしてデータベースに接続します。
2. 左ウィンドウ枠の [パブリケーション] フォルダを選択します。
3. [ファイル] メニューから、[新規] – [パブリケーション] を選択します。

   [パブリケーション作成] ウィザードが表示されます。
4. パブリケーションの名前を入力します。[次へ] をクリックします。
5. [テーブル] タブで、[使用可能なテーブル] のリストからテーブルを 1 つ選択します。[追加] をクリックします。選択したテーブルが、右側の [選択したテーブル] リストに表示されます。
6. オプションで、さらにテーブルを追加することもできます。テーブルの順序は重要ではありません。
7. [終了] をクリックします。

♦ 1 つまたは複数の完全テーブルをパブリッシュするには、次の手順に従います (SQL の場合)。

1. DBA 権限のあるユーザとしてデータベースに接続します。
2. CREATE PUBLICATION 文を実行して、新しく作成するパブリケーションの名前とパブリッシュするテーブルの名前を指定します。

例

- 次の文は、Customers テーブル全体をパブリッシュするパブリケーションを作成します。

```
CREATE PUBLICATION PubCustomers (
  TABLE Customers
)
```

- 次の文は、テーブル・セットの各テーブルのすべてのカラムとローを含むパブリケーションを作成します。

```
CREATE PUBLICATION PubSales (
  TABLE Customers,
  TABLE SalesOrders,
  TABLE SalesOrderItems,
  TABLE Products
)
```

「CREATE PUBLICATION 文 [Mobile Link] [SQL Remote]」『SQL Anywhere サーバ - SQL リファレンス』を参照してください。

テーブル内の一部のカラムだけをパブリッシュする

Sybase Central では、テーブルのすべてのローと、一部のカラムだけを含むパブリケーションを作成できます。また、CREATE PUBLICATION 文でカラムのリストを指定しても、同様に作成できます。

♦ テーブル内の一部のカラムだけをパブリッシュするには、次の手順に従います (Sybase Central の場合)。

1. DBA 権限のあるユーザとしてデータベースに接続します。
2. 左ウィンドウ枠の [パブリケーション] フォルダを選択します。
3. [ファイル] メニューから、[新規] – [パブリケーション] を選択します。

   [パブリケーション作成] ウィザードが表示されます。
4. 新しく作成するパブリケーションの名前を入力します。[次へ] をクリックします。
5. [テーブル] タブで、[使用可能なテーブル] のリストからテーブルを 1 つ選択します。[追加] をクリックします。選択したテーブルが右側の [選択したテーブル] のリストに追加されます。
6. [カラム] タブで、テーブルのアイコンをダブルクリックし、[使用可能なカラム] のリストを展開します。パブリッシュするカラムを 1 つずつ選択し、[追加] をクリックします。右側の [選択したカラム] リストに、選択したカラムが表示されます。
7. [終了] をクリックします。

♦ テーブル内の一部のカラムだけをパブリッシュするには、次の手順に従います (SQL の場合)。

1. DBA 権限のあるユーザとしてデータベースに接続します。

2. CREATE PUBLICATION 文を実行して、パブリケーション名とテーブル名を指定します。テーブル名の後ろにあるカッコの中に、パブリッシュするカラムをリストします。

例

- 次の文は、Customers テーブルのカラムである ID、CompanyName、City のすべてのローをパブリッシュするパブリケーションを作成します。

```
CREATE PUBLICATION PubCustomers (
  TABLE Customers (
    ID,
    CompanyName,
    City )
)
```

「CREATE PUBLICATION 文 [Mobile Link] [SQL Remote]」『SQL Anywhere サーバ - SQL リファレンス』を参照してください。

テーブル内の一部のローだけをパブリッシュする

Sybase Central では、テーブルのすべてのカラムと、一部のローだけを含むパブリケーションを作成できます。Sybase Central と SQL のどちらの場合も、パブリッシュするローにのみ一致する検索条件を書き込んで処理します。

テーブル内の一部のローだけをパブリッシュする方法は、Sybase Central と SQL 言語から 2 とおり提供されますが、Mobile Link との互換性があるのは 1 つだけです。

- **WHERE 句**　WHERE 句を使用すると、アーティクルにローのサブセットを入れることができます。このアーティクルを含むパブリケーションのすべてのサブスクライバは、WHERE 句を満たすローを受信します。
- **サブスクリプション式**　サブスクリプション式を使用すると、アーティクルを含むパブリケーションに対する別のサブスクリプションに、異なるロー・セットを入れることができます。

アーティクルでは WHERE 句とサブスクリプション式を結合できます。Sybase Central または CREATE PUBLICATION 文で、それらを指定します。

パブリケーションのサブスクライバがそれぞれ、1 つのテーブルから異なるローを受け取る場合には、サブスクリプション式を使用します。サブスクリプション式は、テーブルを分割するための最も強力な手段です。

あるパブリケーションのすべてのサブスクリプションから同じロー・セットを除外する場合には、WHERE 句を使用します。

WHERE 句を使用して一部のローだけをパブリッシュする

WHERE 句を指定して、WHERE 条件を満たすローだけをパブリケーションに入れることができます。

♦ WHERE 句を使用してパブリケーションを作成するには、次の手順に従います (Sybase Central の場合)。

1. DBA 権限のあるユーザとしてデータベースに接続します。
2. 左ウィンドウ枠の [パブリケーション] フォルダを選択します。
3. [ファイル] メニューから、[新規] – [パブリケーション] を選択します。

   [パブリケーション作成] ウィザードが表示されます。
4. 新しく作成するパブリケーションの名前を入力します。[次へ] をクリックします。
5. [テーブル] タブで、[使用可能なテーブル] のリストからテーブルを 1 つ選択します。[追加] をクリックします。選択したテーブルが右側の [選択したテーブル] のリストに追加されます。
6. [WHERE 句] タブでテーブルを選択し、下段のボックスに検索条件を入力します。
7. [終了] をクリックします。

♦ WHERE 句を使用してパブリケーションを作成するには、次の手順に従います (SQL の場合)。

1. DBA 権限のあるユーザとしてデータベースに接続します。
2. CREATE PUBLICATION 文を実行して、パブリケーションと WHERE 条件に対象のローを入れます。

例

- 次の文は、status カラムの中でアクティブとマーク付けされた顧客に、Customers テーブルの ID、CompanyName、City、State、Country カラムをパブリッシュするパブリケーションを作成します。

  ```
  CREATE PUBLICATION PubCustomers (
    TABLE Customers (
      ID,
      CompanyName,
      City,
      State,
      Country )
    WHERE Status = 'active'
  )
  ```

  この場合、status カラムはパブリッシュされません。パブリッシュ対象外のすべてのローにはデフォルト値を設定しておきます。設定しないと、統合データベースから挿入用にダウンロードするときに、エラーが発生します。

- 次の文は、関連する受注情報を営業担当者の Samuel Singer に送信する、単一のアーティクルで構成されたパブリケーションを作成します。

  ```
  CREATE PUBLICATION PubOrdersSamuelSinger (
    TABLE SalesOrders WHERE SalesRepresentative = 856
  )
  ```

「CREATE PUBLICATION 文 [Mobile Link] [SQL Remote]」『SQL Anywhere サーバ - SQL リファレンス』を参照してください。

SUBSCRIBE BY
create publication 文でも SUBSCRIBE BY 句は有効です。SQL Remote を使って、選択したローだけをパブリッシュするときにも、この句を使用できます。ただし、Mobile Link 同期では無視されます。

サブスクリプション式を使用して一部のローだけをパブリッシュする

サブスクリプション式を指定して、アーティクルを含むパブリケーションの異なるサブスクリプションに、異なるロー・セットを指定できます。

たとえば、モバイル環境において、各営業担当者が自分専用の受注用サブスクリプションを作成する sales パブリケーションがあるとします。これによって、ローカルで受注を更新したり、統合データベースに受注をレプリケートしたりできます。

WHERE 句のモデルを使用すると、営業担当者ごとにパブリケーションが必要です。営業担当者 Samuel Singer 用のパブリケーションは次のとおりです。その他の営業担当者ごとに、同様のパブリケーションが必要です。

```
CREATE PUBLICATION PubOrdersSamuelSinger (
  TABLE SalesOrders
    WHERE SalesRepresentative = 856
)
```

複数の異なるサブスクリプションを必要とする設定に対応するため、SQL Remote では、「**サブスクリプション式**」をアーティクルと関連付けできます。サブスクリプションが受信するローは、指定された式の値によって異なります。

サブスクリプション式の利点

サブスクリプション式を使用すると、より簡潔で理解しやすいパブリケーションを作成できます。また、WHERE 句を使用した複数のパブリケーションを管理するよりも、優れたパフォーマンスが得られます。データベース・サーバは、パブリケーションの数に正比例して、トランザクション・ログに情報を追加し、そのログをスキャンしてメッセージを送信する必要があります。サブスクリプション式を使用すると、複数の異なるサブスクリプションを単一のパブリケーションに関連付けできます。WHERE 句ではこの関連付けができません。

♦ サブスクリプション式を使用してアーティクルを作成するには、次の手順に従います (Sybase Central の場合)。

1. DBA 権限のあるユーザとしてデータベースに接続します。
2. 左ウィンドウ枠の [パブリケーション] フォルダを選択します。
3. [ファイル] メニューから、[新規] – [パブリケーション] を選択します。

   [パブリケーション作成] ウィザードが表示されます。

4. パブリケーションの名前を入力し、[次へ] をクリックします。
5. [SUBSCRIBE BY 制限の指定] ページで、サブスクリプション式を入力します。
6. ウィザードの残りの指示に従います。

♦ サブスクリプション式を使用してアーティクルを作成するには、次の手順に従います (SQL の場合)。

1. DBA 権限のあるユーザとしてデータベースに接続します。
2. サブスクリプション式でマッチ式として使用する式を含む CREATE PUBLICATION 文を実行します。

例

- 次の文は、Customers テーブルのカラムである ID、CompanyName、City、State、Country をパブリッシュするパブリケーションを作成します。このパブリケーションは、State カラムの値に従ってローとサブスクライバを一致させます。

```
CREATE PUBLICATION PubCustomers (
  TABLE Customers (
    ID,
    CompanyName,
    City,
    State,
    Country )
  SUBSCRIBE BY State
)
```

- 次の文は、パブリケーションに対して 2 人の従業員のサブスクリプションを作成します。Ann Taylor はジョージア州 (GA) の顧客情報を受信し、Sam Singer はマサチューセッツ州 (MA) の顧客情報を受信します。

```
CREATE SUBSCRIPTION
TO PubCustomers ('GA')
FOR Ann_Taylor ;
CREATE SUBSCRIPTION
TO PubCustomers ('MA')
FOR Sam_Singer
```

ユーザは、複数のパブリケーションに対するサブスクリプションを作成できます。また、単一のパブリケーションに対する複数のサブスクリプションも作成できます。

参照

- 「CREATE PUBLICATION 文 [Mobile Link] [SQL Remote]」 『SQL Anywhere サーバ - SQL リファレンス』
- 「サブスクリプション式を含まないテーブルの分割」 41 ページ
- 「サブスクリプションの作成」 71 ページ
- 「WHERE 句を使用して一部のローだけをパブリッシュする」 31 ページ
- 「既存のパブリケーションの変更」 35 ページ

既存のパブリケーションの変更

パブリケーションを作成してから、アーティクルの追加、修正、削除などの変更を加えたり、パブリケーションの名前を変更したりできます。アーティクルを修正する場合、そのアーティクル全体の仕様を入力する必要があります。

Sybase Central を使用するか、Interactive SQL の ALTER PUBLICATION 文によって、これらのタスクを実行できます。

♦ 既存のパブリケーションまたはアーティクルのプロパティを修正するには、次の手順に従います (Sybase Central の場合)。

1. パブリケーションを所有するユーザ、または DBA 権限を持つユーザとしてデータベースに接続します。
2. パブリケーションまたはアーティクルを右クリックし、ポップアップ・メニューで [プロパティ] を選択します。
3. 必要なプロパティを設定します。

♦ アーティクルを追加するには、次の手順に従います (Sybase Central の場合)。

1. パブリケーションを所有するユーザ、または DBA 権限を持つユーザとしてデータベースに接続します。
2. 左ウィンドウ枠で、[パブリケーション] フォルダを開きます。
3. アーティクルを追加するパブリケーションを選択します。
4. [ファイル] メニューから [新規] – [アーティクル] を選択します。

   [アーティクル作成] ウィザードが表示されます。
5. [アーティクル作成] ウィザードで、次の作業を実行します。
   - テーブルを選択して [次へ] をクリックします。
   - アーティクルのカラムを選択します。[次へ] をクリックします。
   - 必要に応じて、WHERE 句を入力します。[次へ] をクリックします。
   - 必要に応じて、SUBSCRIBE BY 制限を作成します。
6. [完了] をクリックすると、アーティクルが作成されます。

♦ アーティクルを削除するには、次の手順に従います (Sybase Central の場合)。

1. パブリケーションを所有するユーザ、または DBA 権限を持つユーザとしてデータベースに接続します。
2. [パブリケーション] フォルダを開きます。
3. アーティクルを削除するパブリケーションを選択します。

4. 削除するアーティクルを右クリックし、ポップアップ・メニューで [削除] を選択します。

♦ **既存のパブリケーションを修正するには、次の手順に従います (SQL の場合)。**

1. パブリケーションを所有するユーザ、または DBA 権限を持つユーザとしてデータベースに接続します。
2. ALTER PUBLICATION 文を実行します。

例

♦ 次の文は、Customers テーブルを PubContacts パブリケーションに追加します。

```
ALTER PUBLICATION PubContacts (
  ADD TABLE Customers
)
```

参照

♦ 「ALTER PUBLICATION 文 [Mobile Link] [SQL Remote]」 『SQL Anywhere サーバ - SQL リファレンス』
♦ 「WHERE 句を使用して一部のローだけをパブリッシュする」 31 ページ
♦ 「サブスクリプション式を使用して一部のローだけをパブリッシュする」 33 ページ

パブリケーションの削除

Sybase Central または DROP PUBLICATION 文のいずれかを使用して、パブリケーションを削除できます。パブリケーションを削除すると、そのパブリケーションに対するサブスクリプションもすべて自動的に削除されます。

パブリケーションを削除するには、DBA 権限が必要です。

♦ **パブリケーションを削除するには、次の手順に従います (Sybase Central の場合)。**

1. DBA 権限のあるユーザとしてデータベースに接続します。
2. [パブリケーション] フォルダを開きます。
3. 対象のパブリケーションを右クリックし、ポップアップ・メニューで [削除] を選択します。

♦ **パブリケーションを削除するには、次の手順に従います (SQL の場合)。**

1. DBA 権限のあるユーザとしてデータベースに接続します。
2. DROP PUBLICATION 文を実行します。

例

次の文は、パブリケーション PubOrders を削除します。

```
DROP PUBLICATION PubOrders
```


「DROP PUBLICATION 文 [Mobile Link] [SQL Remote]」『SQL Anywhere サーバ - SQL リファレンス』を参照してください。

パブリケーションについての注意

- これまでに説明した異なるタイプのパブリケーションを、組み合わせることができます。1つのパブリケーションで、テーブル・セットのカラム・サブセットをパブリッシュし、WHERE句を使用して、レプリケートするロー・セットを選択できます。
- パブリケーションの作成と削除には DBA 権限が必要です。
- DBA 権限を持つユーザか、パブリケーションの所有者だけがパブリケーションを変更できます。
- SQL Remote 設定の実行中にパブリケーションを変更すると、レプリケーション・エラーが発生しやすくなります。レプリケーション・システムのデータが失われる可能性があるので、慎重に行ってください。
- パブリケーションにビューを入れることはできません。
- パブリケーションにストアド・プロシージャを入れることはできません。SQL Remote でプロシージャとトリガをレプリケートする方法については、「プロシージャのレプリケーション」 19 ページを参照してください。

パブリケーションの設計

簡単なパブリケーションの作成方法について理解できたら、適切なパブリケーション設計について考える必要があります。最適化設計は、SQL Remote のインストールを成功させる重要な要素です。この項では、SQL Anywhere の SQL Remote に対する最適な設計方法について説明します。

設計に関する問題の概要

各サブスクリプションは、完全なリレーショナル・データベース

リモート・データベースは、統合データベースとサブスクリプションに関する情報を共有します。サブスクリプションは、統合サイトで保持されるリレーショナル・データベースのサブセットであると同時に、リモート・サイトの完全なリレーショナル・データベースでもあります。したがって、サブスクリプション内の情報は、他のリレーショナル・データベースと同様に次の規則に従います。

- **外部キーの関係を有効にする** データベースには、外部キーの各エントリに対応するプライマリ・キーのエントリが必要です。

  データベース抽出ユーティリティを使用すると、リモート・データベースに対する CREATE TABLE 文が、リモートに存在しないテーブルに定義された外部キーを持たないようになります。

- **プライマリ・キーの一意性を管理する** 別のサイトで入力され、まだレプリケートされていない新しいローを検査する方法はありません。異なるサイトのユーザが同じプライマリ・キー値を使用してローを追加することがないように、設計する必要があります。そうでない場合は、統合データベースにローをレプリケートするときに競合が発生します。

ロックがない状態でトランザクションの整合性を管理する

特定のローに対してシステム全体をロックするメカニズムがなくても、すべてのサイトで更新が行われた場合に、分散データベース (統合データベースとすべてのリモート・データベースで構成される) のデータは、整合性を保つ必要があります。

- **ロック競合を回避または解決する** SQL Remote のインストール環境には、各ユーザがローを同時に変更しないように、すべてのデータベースに渡ってローをロックする方法はありません。このような競合は、競合が起こらないようにシステムを設計して回避するか、統合データベースで適切な方法で解決する必要があります。

これらリレーショナル・データベースの主要な機能を、パブリケーションとサブスクリプションの設計に組み込む必要があります。この項では、設計方法と技術について説明します。

有効なアーティクルの条件

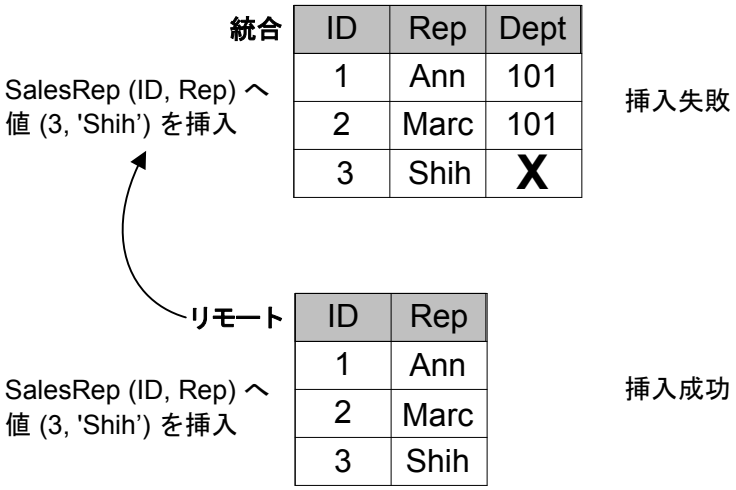
プライマリ・キーの全カラムが、アーティクルに含まれる必要があります。

リモート・データベースでの INSERT のサポート

リモート・データベースの INSERT 文が統合データベースへ正しくレプリケートされるために、有効な INSERT 文にならないカラムだけをアーティクルから除外できます。基本テーブルは次のとおりです。

- NULL を使用できるカラム
- デフォルトのカラム

これらの条件を満たさないカラムを除外すると、リモート・データベースで INSERT 文を実行して統合データベースにレプリケートすると失敗します。



代わりに BEFORE トリガを使用する
統合データベースが SQL Anywhere データベースである場合、BEFORE トリガを書き込んでおき、INSERT 文に含まれていないカラムを管理します。この方法で、この問題を回避します。

パフォーマンスのための設計ヒント

この項では、ハイパフォーマンスな SQL Remote のインストール環境を設計するための、チェック・リストを示します。

- **パブリケーションの数を小さくする**　特に、1 つの同じテーブルをさまざまなパブリケーションで参照しないようにします。

  データベース・サーバが必要とする作業は、パブリケーションの数に比例します。パブリケーションの数を小さく抑えて、サブスクリプションを有効に利用すれば、データベース・サーバの負荷を軽減できます。

  テーブル上で操作すると、データベース・サーバと Message Agent は、そのテーブルを含むパブリケーションごとにいくつかの作業を行います。リモート・ユーザごとにパブリケーショ

ンが 1 つあると、データベース・サーバの負荷が急激に増加します。SUBSCRIBE BY を使用するパブリケーションをいくつか持ち、リモート・ユーザごとにサブスクリプションを持つ方が、負荷は減ります。パブリケーションに追加するサブスクリプションを増やしても、データベース・サーバの作業は増えません。Message Agent は、大量のサブスクリプションを使用して効率的に作業するように設計されています。

- **パブリケーションを論理的にグループ分けする** たとえば、価格表などのすべてのリモート・ユーザが必要とするテーブルがある場合、そのテーブル用に別のパブリケーションを作成します。カラム値によって分割できるデータが格納されたテーブルごとに、パブリケーションを 1 つ作成します。
- **サブスクリプションを有効に利用する** リモート・ユーザが同じような統合データベースのサブセットを受信するたびに、SUBSCRIBE BY 式を組み込むパブリケーションが使用されます。リモート・ユーザごとに別のパブリケーションを作成しないでください。
- **パブリケーション・トリガの更新に注意する** 特に、次の点に注意してください。
  - NEW / OLD SUBSCRIBE BY 構文を使用します。
  - 効率的にデータベースにアクセスするように SELECT 文を調整します。
- **トランザクション・ログ・サイズを監視する** トランザクション・ログのサイズが大きいければ大きいほど、Message Agent のスキャン時間は長くなります。ログ名は定期的に変更し、delete_old_logs オプションを使用してください。

サブスクリプション式を含まないテーブルの分割

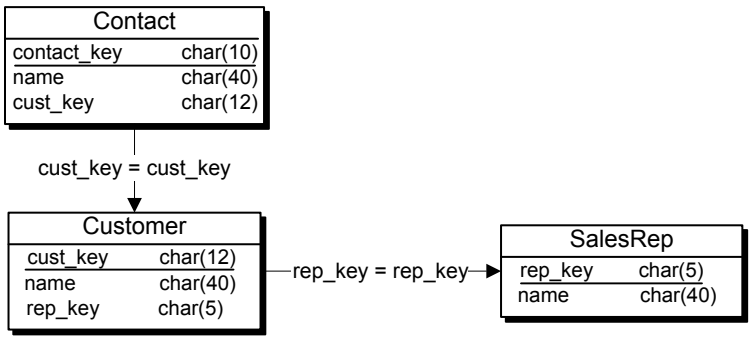
多くの場合、テーブルにサブスクリプション式がなくても、テーブルのローを分割する必要があります。

Contacts データベースの例

Contacts データベースの例では、サブスクリプション式を含まないテーブルの分割の必要性とその方法について説明します。

例

次に、この問題について説明する簡単なデータベースを示します。



各営業担当者は、複数の顧客に対して販売活動を行います。連絡先が 1 箇所だけの顧客もいれば、複数ある顧客もいます。

データベース内のテーブル

3 つのテーブルの詳細は次のとおりです。

テーブル	説明

SalesReps	社内のすべての営業担当者。**SalesReps** テーブルには、次のカラムがあります。
	◆ **rep_key** 各営業担当の識別子。これがプライマリ・キーです。
	◆ **name** 各営業担当の名前
	このテーブルを作成する SQL 文は次のとおりです。
	`CREATE TABLE SalesReps (`
	`  Rep_key CHAR(12) NOT NULL,`
	`  Name CHAR(40) NOT NULL,`
	`  PRIMARY KEY (rep_key)`
	`)`

テーブル	説明

Customers	会社と取引があるすべての顧客。**Customers** テーブルには、次のカラムがあります。
	◆ **cust_key** 各顧客の識別子。これがプライマリ・キーです。
	◆ **name** 各顧客の名前。
	◆ **rep_key** 取引を行う営業担当者の識別子。これが SalesReps テーブルへの外部キーです。
	このテーブルを作成する SQL 文は次のとおりです。
	`CREATE TABLE Customers (`
	`Cust_key CHAR(12) NOT NULL,`
	`Name CHAR(40) NOT NULL,`
	`Rep_key CHAR(12) NOT NULL,`
	`FOREIGN KEY REFERENCES SalesReps,`
	`PRIMARY KEY (cust_key)`
	`)`

Contacts	会社と取引があるすべての個別の連絡先。連絡先はそれぞれ 1 人の顧客に属します。**Contacts** テーブルには、次のカラムがあります。
	◆ **contact_key** 各連絡先の識別子。これがプライマリ・キーです。
	◆ **name** 各連絡先の名前。
	◆ **cust_key** 連絡先が関連する顧客の識別子。これが Customers テーブルへの外部キーです。
	このテーブルを作成する SQL 文は次のとおりです。
	`CREATE TABLE Contacts (`
	`Contacts_key CHAR(12) NOT NULL,`
	`Name CHAR(40) NOT NULL,`
	`Cust_key CHAR(12) NOT NULL,`
	`FOREIGN KEY REFERENCES Customers,`
	`PRIMARY KEY (contact_key)`
	`)`

レプリケーションの目標

この設計上の目標は、各営業担当者に次の情報を提供することです。

- 完全な **SalesReps** テーブル
- 営業担当者に割り当てられている顧客 (**Customers** テーブルの情報)
- 関連する顧客の連絡先 (**Contacts** テーブルの情報)

Contacts データベースの例における Customers テーブルの分割

サブスクリプション式に **rep_key** 値を使用すると、**Customers** テーブルを分割できます。**SalesReps** テーブルと **Customers** テーブルを含むパブリケーションは、次のとおりです。

```
CREATE PUBLICATION SalesRepData (
  TABLE SalesReps
  TABLE Customers SUBSCRIBE BY rep_key
)
```

Contacts データベースの例における Contacts テーブルの分割

Contacts テーブルも営業担当者の間で分割する必要がありますが、営業担当者の **rep_key** 値への参照は含みません。**rep_key** がテーブルに含まれない場合でも、Message Agent を使用すると、このテーブルのローとサブスクリプション値を同じものにできます。

このためには、**Customers** テーブルの **rep_key** カラムを評価する **Contacts** アーティクルで、サブクエリを使用します。このパブリケーションは、次のようになります。

```
CREATE PUBLICATION SalesRepData (
  TABLE SalesReps
  TABLE Customers
    SUBSCRIBE BY rep_key
  TABLE Contacts
    SUBSCRIBE BY (SELECT rep_key
      FROM Customers
      WHERE Contacts.cust_key = Customers.cust_key )
)
```

サブスクリプション式の WHERE 句によって、サブクエリは値を 1 つだけ戻します。これは、**Customers** テーブル内にあるローの 1 つだけが、**Contacts** テーブルの現在のローと同じ **cust_key** 値を持つからです。

Contacts データベースの例における領域の再編成

「**領域の再編成**」を実行すると、サブスクライバ間でローが再割り当てされます。今回の例では、領域の再編成とは、営業担当者の間で **Customers** テーブル (つまり **Contacts** テーブルも含む) のローを再割り当てすることを意味します。

新しい営業担当者に顧客が再割り当てされると、**Customers** テーブルが更新されます。UPDATE 文が、以前の営業担当者へ DELETE 文としてレプリケートされて、新しい営業担当者へ INSERT 文としてレプリケートされます。こうして、Customer ローは正しく新しい営業担当者へ移動されます。

SQL Anywhere と SQL Remote を同時に使用してこのような状況を処理する方法については、「どのサブスクライバにどの文が送信されるか」 23 ページを参照してください。

顧客が再割り当てされても、**Contacts** テーブルは影響を受けません。**Contacts** テーブルには変更が加えられていないため、このテーブルに関するトランザクション・ログにはエントリがありません。この情報がないと、SQL Remote は、**Customers** テーブルだけでなく **Contacts** テーブルのローも再割り当てできません。

この失敗によって、参照整合性の問題が発生します。以前の営業担当のリモート・データベースで、**Customers** テーブルと対応しなくなった **cust_key** 値が、**Contacts** テーブルに含まれています。

トリガを使用して Contacts テーブルを管理する

この問題を解決するには、特殊な UPDATE 文を含むトリガを使用します。これによって、データベース・テーブルをまったく変更しないで、トランザクション・ログにエントリを作成します。このログ・エントリでは、サブスクリプション式に BEFORE、AFTER を指定します。これで、Message Agent を使用して正しくローをレプリケートできます。

ローにおいて、このトリガの BEFORE オペレーションを起動してください。これによって、BEFORE 値を評価してログに配置することができます。また、各文ではなく FOR EACH ROW を起動して、トリガの提供する情報を新しいサブスクリプション式としてください。Message Agent は、この情報を使用して、どのサブスクライバがどのローを受信するかを決定します。

トリガの定義

トリガの定義は次のとおりです。

```
CREATE TRIGGER UpdateCustomer
BEFORE UPDATE ON Customers
REFERENCING NEW AS NewRow
   OLD as OldRow
FOR EACH ROW
BEGIN
   // determine the new subscription expression
   // for the Customers table
   UPDATE Contacts
   PUBLICATION SalesRepData
   OLD SUBSCRIBE BY ( OldRow.rep_key )
   NEW SUBSCRIBE BY ( NewRow.rep_key )
   WHERE cust_key = NewRow.cust_key;
END;
```

パブリケーションのための特殊な UPDATE 文

このトリガの UPDATE 文は、次のような特殊なフォームになります。

UPDATE *table-name*
PUBLICATION *publication-name*
{ **SUBSCRIBE BY** *subscription-expression* |
    OLD SUBSCRIBE BY *old-subscription-expression*
    NEW SUBSCRIBE BY *new-subscription-expression* }
WHERE *search-condition*

- 各句の意味は、次のとおりです。
- *table-name* は、修正する必要がある、リモート・データベースのテーブルです。
- *publication-name* は、サブスクリプションを変更する必要のあるパブリケーションです。
- *subscription-expression* の値は、ローの新しい受信者と既存の受信者を決定するため、Message Agent によって使用されます。代わりに、サブスクリプション式 OLD と NEW を設定することもできます。
- WHERE 句は、サブスクリプションを作成したデータベース間で移動させるローを指定します。

トリガに関する注意事項

- トリガが次の構文を使用する場合は、注意してください。

  ```
  UPDATE table-name
  PUBLICATION pub-name
    SUBSCRIBE BY sub-expression
  WHERE search-condition
  ```

  トリガには、BEFORE トリガを使用する必要があります。この例では、BEFORE UPDATE トリガとなります。他のコンテキストでは、BEFORE DELETE トリガと BEFORE INSERT トリガが必要です。

- トリガが次の代替構文を使用する場合は、注意してください。

  ```
  UPDATE table-name
  PUBLICATION publication-name
    OLD SUBSCRIBE BY old-subscription-expression
    NEW SUBSCRIBE BY new-subscription-expression }
  WHERE search-condition
  ```

  トリガを BEFORE トリガまたは AFTER トリガにすることができます。

- UPDATE 文は、変更が適用されるパブリケーションとテーブルをリストします。文中の WHERE 句は、変更が適用されるローを示します。この UPDATE 文では、テーブルのデータは変更されませんが、トランザクション・ログにエントリが作成されます。

- この例におけるサブスクリプション式は、1 つの値を返します。複数の値を返すサブクエリも使用できます。サブスクリプション式の値は、UPDATE 文を実行したあとの値でなければなりません。

  この場合、ローへのサブスクライバだけが新しい営業担当者になります。新規サブスクライバと既存のサブスクライバが存在するケースについては、「複数のサブスクリプション間におけるローの共有」 47 ページを参照してください。

トランザクション・ログ内の情報

トランザクション・ログの情報を理解しておくと、有効なパブリケーションの設計に役立ちます。

- 次のようなデータを想定します。
  - SalesReps テーブル

rep_key	Name
rep1	Ann
rep2	Marc

  - Customers テーブル

cust_key	name	rep_key
cust1	Sybase	rep1
cust2	SQL Anywhere	rep2

- Contacts テーブル

contact_key	name	cust_key
contact1	David	cust1
contact2	Stefanie	cust2

- 次のような Update 文を適用して、領域を再編成します。

```
UPDATE Customers
SET rep_key = 'rep2'
WHERE cust_key = 'cust1'
```

この文を使用して、トランザクション・ログにエントリを 2 つ作成します。1 つは Contacts テーブルの BEFORE トリガに対するエントリ、もう 1 つは Customers テーブルへの実際の UPDATE 文に対するエントリです。

```
SalesRepData - Publication Name
rep1 - BEFORE list
rep2 - AFTER list
UPDATE Contacts
SET contact_key = 'contact1',
    name = 'David',
    cust_key = 'cust1'
WHERE contact_key = 'contact1'
SalesRepData - Publication Name
rep1 - BEFORE list
rep2 - AFTER list
UPDATE Customers
SET rep_key = 'rep2'
WHERE cust_key = 'cust1'
```

Message Agent は、これらのタグに対するログをスキャンします。この情報に基づいて、どのリモート・ユーザが INSERT、UPDATE、または DELETE を取得するかが決定されます。

この場合、BEFORE list は **rep1**、AFTER list は **rep2** となっています。BEFORE list と AFTER list の値が異なると、UPDATE 文によって影響を受けるローは、あるサブスクライバの値から別のサブスクライバの値へ「移動」します。つまり、Message Agent は、値 **rep1** によって Customers レコードの値 **cust1** に対するサブスクリプションを作成したすべてのリモート・ユーザに DELETE を送信し、値 **rep2** によってサブスクリプションを作成したすべてのリモート・ユーザに INSERT を送信します。

BEFORE list と AFTER list の値が同じ場合は、リモート・ユーザがすでにローを保有しているため、UPDATE が送信されます。

複数のサブスクリプション間におけるローの共有

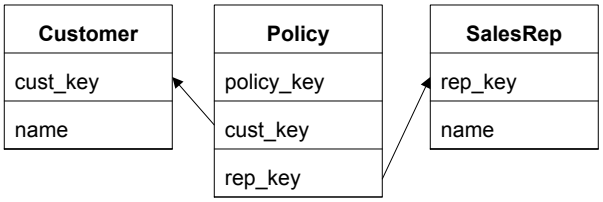
多対多関係のように、ある 1 つのローを複数のサブスクリプションに含めなければならない場合があります。この項では、このような状況での処理方法について説明します。

Policy データベースの例

Policy データベースでは、データベースに多対多関係がある場合、テーブルを分割する理由とその方法について説明します。

データベースの例

次に、この問題について説明する簡単なデータベースを示します。



各営業担当者は複数の顧客を担当していますが、複数の営業担当者と取り引きしている顧客もいます。この場合、**Customers** と **SalesReps** の関係は、多対多の関係になります。

データベース内のテーブル

3 つのテーブルの詳細は次のとおりです。

テーブル	説明

SalesReps	社内のすべての営業担当者。**SalesReps** テーブルには、次のカラムがあります。
	◆ **rep_key**　各営業担当の識別子。これがプライマリ・キーです。
	◆ **name**　各営業担当の名前
	このテーブルを作成する SQL 文は次のとおりです。
	`CREATE TABLE SalesReps (`
	`  Rep_key CHAR(12) NOT NULL,`
	`  Name CHAR(40) NOT NULL,`
	`  PRIMARY KEY (rep_key)`
	`);`

テーブル	説明

Customers	会社と取引があるすべての顧客。**Customers** テーブルには、次のカラムがあります。
	◆ **cust_key** 各顧客の識別子を含んだ、プライマリ・キーのカラム
	◆ **name** 各顧客の名前を含んだカラム
	このテーブルを作成する SQL 文は次のとおりです。
	`CREATE TABLE Customers (`
	`Cust_key CHAR(12) NOT NULL,`
	`Name CHAR(40) NOT NULL,`
	`PRIMARY KEY (cust_key)`
	`);`

Policy	顧客と営業担当者間の多対多の関係を管理する、3 つのカラムで構成されたテーブル。**Policy** テーブルには次のカラムがあります。
	◆ **policy_key** 取り引きの識別子を含んだ、プライマリ・キーのカラム
	◆ **cust_key** 取り引きを行う顧客の識別子を含んだカラム
	◆ **rep_key** 取り引きを行う営業担当者の識別子を含んだカラム
	このテーブルを作成する SQL 文は次のとおりです。
	`CREATE TABLE Policy (`
	`policy_key CHAR(12) NOT NULL,`
	`cust_key CHAR(12) NOT NULL,`
	`rep_key CHAR(12) NOT NULL,`
	`FOREIGN KEY ( cust_key )`
	`REFERENCES Customers ( cust_key )`
	`FOREIGN KEY ( rep_key )`
	`REFERENCES SalesReps (rep_key ),`
	`PRIMARY KEY ( policy_key )`
	`);`

レプリケーションの目的

このレプリケーションの設計上の目的は、各営業担当に次の情報を提供することです。

- **SalesReps** テーブル全体
- データに対するサブスクリプションが作成されている営業担当者を含む、取り引きを記録した **Policy** テーブルのロー
- データに対するサブスクリプションが作成されている営業担当者と取り引きする顧客をリストした、**Customers** テーブルのロー

新しい問題

顧客と営業担当者間の多対多の関係では、適切に情報を共有する上で新しい問題が発生します。

- データを分割するサブスクリプションで使用される、営業担当者の値を参照しないテーブル (この場合は Customers テーブル) があります。

再びパブリケーションでサブクエリを使用して、この問題に対処します。

- **Customers** テーブルの各ローが、**SalesReps** テーブルの複数のローと関連付けされ、複数の営業担当者のデータベースで共有されている場合があります。

  つまり、「サブスクリプション式を含まないテーブルの分割」 41 ページの項に示す **Contacts** テーブルのローが、パブリケーションによって別々のセットに分割されます。この項の例では、重複しているサブスクリプションがあります。

レプリケーションの目的を達成するには、1 つのパブリケーションと 1 つのサブスクリプション・セットが必要です。この場合、2 つのトリガを使用して、ある営業担当者から別の担当者に顧客を移動できます。

パブリケーション

1 つのパブリケーションによって、データ共有の基本を指定します。

```
CREATE PUBLICATION SalesRepData (
   TABLE SalesReps,
   TABLE Policy SUBSCRIBE BY rep_key,
   TABLE Customers SUBSCRIBE BY (
      SELECT rep_key FROM Policy
      WHERE Policy.cust_key =
         Customers.cust_key
   ),
);
```

サブスクリプション文は前述の例と同一です。

パブリケーションの仕組み

パブリケーションには、3 つのテーブルの一部または全部が含まれています。パブリケーションの仕組みを理解するには、次の各アーティクルを参照してください。

- **SalesReps テーブル**　このアーティクルに対応した修飾子はありません。このため、**SalesReps** テーブル全体がパブリケーションに含まれます。

  ```
  ...
     TABLE SalesReps,
  ...
  ```

- **Policy テーブル**　このアーティクルは、サブスクリプション式を使用して、営業担当者間でデータの分割に使用するカラムを指定します。

  ```
  ...
     TABLE Policy
     SUBSCRIBE BY rep_key,
  ...
  ```

  このサブスクリプション式によって、**rep_key** カラムの値がサブスクリプションで指定された値に一致するテーブルのローだけを、各営業担当者が受信します。

  Policy テーブルの分割は「**切断**」です。複数のサブスクライバが共有するローはありません。

Customers テーブル　サブクエリを持つサブスクリプション式は、分割を定義するときに使用します。このアーティクルは、次のように定義されます。

```
...
  TABLE Customers SUBSCRIBE BY (
    SELECT rep_key
    FROM Policy
    WHERE Policy.cust_key =
      Customers.cust_key
  ),
...
```

Customers テーブルの分割は「**非切断**」です。複数のサブスクライバが共有するローがいくつかあります。

複数の値を返すパブリケーションのサブクエリ

Customers アーティクルのサブクエリは、その結果セットに単一のカラム (**rep_key**) を返します。ただし、特定の顧客を担当する営業担当者全員に対応して、複数のローを返す場合があります。サブスクリプション式に複数の値がある場合は、この値のいずれかに一致するサブスクリプションを持つすべてのサブスクライバにローがレプリケートされます。複数の値を持つサブスクリプション式の場合、テーブルの分割を非切断にできます。

多対多の関係を使用した領域の再編成

「Contacts データベースの例における領域の再編成」 43 ページの項で示すように、領域の再編成 (サブスクライバ間におけるローの再割り当て) の問題には特に注意が必要です。

インストール環境全体を通して正しいデータを管理するため、領域の再編成 (サブスクライバ間におけるローの再割り当て) が実行可能な場合は、トリガを書き込む必要があります。

顧客の移動方法

この例では、**Policy** テーブルのローを挿入／削除して顧客を移動させます。

顧客と営業担当者間の取り引きを取り消すには、**Policy** テーブルのローを削除します。この場合、**Policy** テーブルの変更内容は正しくレプリケートされ、営業担当者のデータベースにこのローが表示されなくなります。ただし、**Customers** テーブルは変更されません。このため、**Customers** テーブルに対する変更は、サブスクライバにレプリケートされません。

トリガがないと、**Customers** テーブルに不正確なデータを持つサブスクライバが残されます。**Policy** テーブルに新しいローを追加する場合にも、同様の問題が発生します。

トリガを使用した問題の解決法

Policy テーブルを変更すると起動されるトリガを書き込んで、この問題を解決します。このトリガには、UPDATE 文の特別な構文を入れます。この特別な UPDATE 文によって、データベース・テーブルは変更されません。代わりに、サブスクライバ・データベースでのデータ管理用に SQL Remote が使用するトランザクション・ログに、エントリが作成されます。

BEFORE INSERT トリガ

次のようにトリガを作成し、**Policy** テーブルに対する INSERT を追跡します。これで、リモート・データベースに適切なデータが含まれるようにします。

```
CREATE TRIGGER InsPolicy
BEFORE INSERT ON Policy
REFERENCING NEW AS NewRow
FOR EACH ROW
BEGIN
  UPDATE Customers
  PUBLICATION SalesRepData
  SUBSCRIBE BY (
    SELECT rep_key
    FROM Policy
    WHERE cust_key = NewRow.cust_key
    UNION ALL
    SELECT NewRow.rep_key
  )
  WHERE cust_key = NewRow.cust_key;
END;
```

BEFORE DELETE トリガ

次に、**Policy** テーブルに対する DELETE を追跡するトリガを示します。

```
CREATE TRIGGER DelPolicy
BEFORE DELETE ON Policy
REFERENCING OLD AS OldRow
FOR EACH ROW
BEGIN
  UPDATE Customers
  PUBLICATION SalesRepData
  SUBSCRIBE BY (
    SELECT rep_key
    FROM Policy
    WHERE cust_key = OldRow.cust_key
    AND Policy_key <> OldRow.Policy_key
  )
  WHERE cust_key = OldRow.cust_key;
END;
```

トリガの機能の一部は、前述の項で示す内容と同じです。主な新機能は、INSERT トリガがサブクエリを含んでいること、およびこのサブクエリに複数の値を使用できることです。

複数の値を返すサブクエリ

BEFORE INSERT トリガのサブクエリは UNION 式であり、複数の値を返すことができます。

```
...
SELECT rep_key
FROM Policy
WHERE cust_key = NewRow.cust_key
UNION ALL
SELECT NewRow.rep_key
...
```

- UNION 式の 2 番目の部分は、INSERT 文から取得した、顧客と取り引きする新しい営業担当者の **rep_key** 値です。
- UNION 式の最初の部分は、Policy テーブルから取得した、顧客と取り引きする既存の営業担当者のセットです。

つまり、サブスクリプション・クエリの結果セットには、新しい営業担当者だけでなく、ローを受信する営業担当者全員が含まれる必要があるということです。

BEFORE DELETE トリガ内のサブクエリは、複数の値を返します。

```
...
SELECT rep_key
FROM Policy
WHERE cust_key = OldRow.cust_key
AND rep_key <> OldRow.rep_key
...
```

- サブクエリは、**Policy** テーブルから **rep_key** の値を取得します。削除される値 (AND **rep_key** <> **OldRow.rep_key**) を除き、移動される顧客 (WHERE **cust_key** = **OldRow.cust_key**) と取り引きする営業担当者全員のプライマリ・キー値が、その値に含まれます。

  ここでも、サブスクリプション・クエリの結果セットには、DELETE に続くローを受信する営業担当者と一致した値のすべてが含まれる必要があります。

注意

- **Customers** テーブルのデータは、(プライマリ・キーの値などによって) 個々のサブスクライバと関連付けられることはなく、複数のサブスクライバ間で共有されます。このため、レプリケーション・メッセージ間の複数のリモート・サイトでデータが更新される可能性があり、レプリケーションの競合が発生することがあります。(特定のユーザだけに Customers テーブルの更新権を付与するなどの方法で) パーミッションを使用するか、データベースに RESOLVE UPDATE トリガを追加してプログラム上で競合を処理することによって、この問題に対処できます。
- Policy テーブル上の UPDATE については、ここでは説明していません。そのような操作は避けるか、例で示したように、BEFORE UPDATE トリガで BEFORE INSERT と BEFORE DELETE の機能を組み合わせる必要があります。

多対多の関係における subscribe_by_remote オプションの使用

subscribe_by_remote オプションを On に設定すると、subscribe by 値が NULL または空の文字列であるローに対して、リモート・データベースから操作が行われた場合、リモート・ユーザがローに対するサブスクリプションを作成するとみなされます。デフォルトでは、subscribe_by_remote オプションは On に設定されています。多くの場合、この設定は必要に応じて変更されます。

subscribe_by_remote オプションは、これを使用しない場合にいくつかのパブリケーション (Policy データベースの例を含む) で発生する問題を解決します。この項では、問題の詳細を説明し、また、オプションを使用してその問題を自動的に回避する方法について説明します。

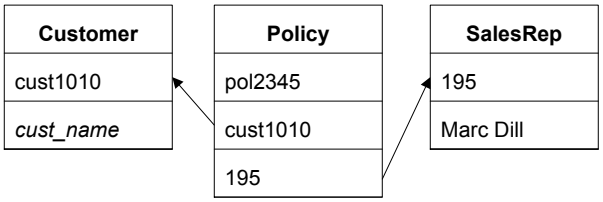
各顧客が複数の営業担当者に所属できるため、パブリケーションでは、**Customers** テーブルのサブスクリプション式にサブクエリを使用します。

```
CREATE PUBLICATION SalesRepData (
  TABLE SalesReps,
  TABLE Policy SUBSCRIBE BY rep_key,
  TABLE Customers SUBSCRIBE BY (
    SELECT rep_key FROM Policy
    WHERE Policy.cust_key =
```



```
        Customers.cust_key
    ),
);
```

営業担当者 Marc Dill が、新しい顧客との取り引きをデータに入力します。まず、Marc Dill はデータベースに新しい **Customers** ローを挿入し、**Policy** テーブルにもローを挿入して新規の顧客を自分自身に割り当てます。



Customers ローの INSERT は、Message Agent によって統合データベースで実行されるため、INSERT を実行するとき、SQL Anywhere はトランザクション・ログにサブスクリプション値を記録します。

あとで Message Agent がログをスキャンすると、サブスクリプション式からサブスクライバのリストが構築されます。顧客を割り当てた Policy テーブルでローがまだ適用されていないため、この場合 Marc Dill はリストされません。subscribe_by_remote オプションが Off に設定されていると、この新しい顧客は DELETE オペレーションとして Marc Dill に返されます。

subscribe_by_remote オプションが On に設定されているかぎり、Message Agent は、ローの所属先はローを挿入した営業担当者であるとみなし、INSERT は Marc Dill にレプリケートされません。また、レプリケーション・システムは影響を受けません。

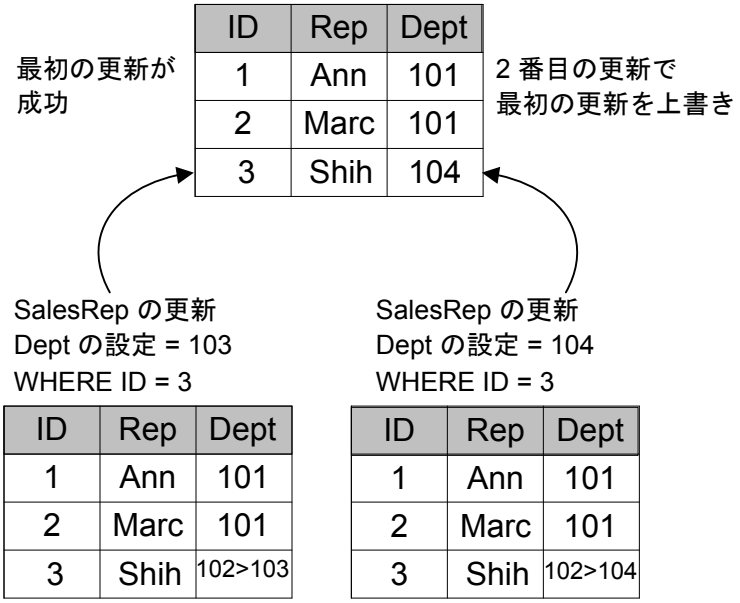
subscribe_by_remote オプションが Off に設定されている場合は、必ず Policy ローを挿入してから Customers ローを挿入し、トランザクションの最後までチェックを延期することによって参照整合性違反を回避してください。

競合の管理

UPDATE 競合は、次のような順序でイベントが実行されると発生します。

1. ユーザ 1 が、リモート・サイト 1 でローを更新します。
2. ユーザ 2 が、リモート・サイト 2 で同じローを更新します。
3. ユーザ 1 による更新内容が、統合データベースにレプリケートされます。
4. ユーザ 2 による更新内容が、統合データベースにレプリケートされます。

SQL Remote Message Agent が UPDATE 文をレプリケートするときは、ローごとに別の UPDATE 文をレプリケートします。またメッセージには、比較するために古いローの値が含まれています。ユーザ 2 による更新内容が統合データベースで受信される場合、ローの値はメッセージに記録されたものと異なります。



ID	Rep	Dept
1	Ann	101
2	Marc	101
3	Shih	104

ID	Rep	Dept
1	Ann	101
2	Marc	101
3	Shih	102>103

ID	Rep	Dept
1	Ann	101
2	Marc	101
3	Shih	102>104

デフォルトによる競合解決

デフォルトでは、UPDATE の処理は継続されるため、ユーザ 2 による更新内容 (最終的に統合データベースに到達する) が統合データベースの値になります。また、この値が、ローにサブスクリプションを作成するその他のデータベースすべてにレプリケートされます。

通常、デフォルトの競合解決方法では、最後に実行されたオペレーション (この場合はユーザ 2 によるオペレーション) を正常に処理して、競合の発生がレポートされません。ユーザ 1 による更新内容は失われます。SQL Remote では、変更されるデータにとって意味のある方法で競合を解決するためにトリガを使用して、カスタム競合解決を行うことができます。

プライマリ・キーの更新に競合解決は適用されない
UPDATE 競合は、プライマリ・キーの更新には適用*されません*。SQL Remote のインストール環境で、プライマリ・キーを更新しないでください。適切な設計をすることによって、プライマリ・キーの競合をインストール環境から取り除く必要があります。

この項では、統合データベースにおける、SQL Remote のインストール環境への競合解決方法の構築について説明します。

SQL Remote の競合処理方法

競合が検出された場合

SQL Remote のレプリケーション・メッセージには、単一ローの更新セットとして UPDATE 文が含まれ、各文には更新前の値を含む VERIFY 句が指定されています。

UPDATE 競合は、データベースのローと一致させるための VERIFY 句の値のエラーとして、データベース・サーバによって検出されます。

統合データベースにおける競合だけが Message Agent によって検出されて解決されます。リモート・データベースからのメッセージで UPDATE 競合が検出されると、そのデータベースで次の 2 つのアクションが実行されます。

1. 競合解決 (RESOLVE UPDATE) トリガが起動されます。
2. UPDATE が適用されます。

RESOLVE UPDATE トリガの有無にかかわらず、UPDATE 文は VERIFY 句の値が一致しなくても適用されます。

競合解決はいくつかの手法を取ることができます。次に例を示します。

- アプリケーションによっては、競合をテーブルにレポートします。
- リモート・サイトでの更新内容よりも、統合データベースでの更新内容を優先して保持します。
- 商品のデリバリや受注時に在庫数を分析するといった、より高度な競合解決をします。

競合解決の実装

この項では、SQL Remote でカスタム設定した競合解決を実装するために必要な作業について説明します。

SQL Remote では、「**競合解決トリガ**」を定義して UPDATE 競合を処理できます。競合解決トリガは、リモート・ユーザがメッセージを適用する場合に、統合データベースでのみ起動されます。統合データベースで UPDATE 競合が検出されると、次の順序でイベントが実行されます。

1. オペレーションに対して定義された、すべての競合解決トリガが起動されます。

2. UPDATE が実行されます。
3. UPDATE だけでなくそのトリガのすべてのアクションが、リモート・データベース全部にレプリケートされます。この中には、その競合をトリガするメッセージを送信した、リモート・データベースも含まれます。

   通常、SQL Anywhere の SQL Remote は、トリガ・アクションをレプリケートしません。トリガがリモート・データベースにあることを想定しているからです。競合解決トリガは統合データベースでのみ起動されるため、このトリガ・アクションはリモート・データベースにレプリケートされます。
4. リモート・データベースでは、統合データベースからのメッセージに UPDATE 競合が含まれていると、RESOLVE UPDATE トリガは起動されません。
5. UPDATE がリモート・データベースで実行されます。

プロセスの最後では、設定を通してデータの一貫性が保たれます。

データが読み取り用に共有されていても、各ロー (プライマリ・キーによって識別される) が 1 つのサイトだけで更新される場合は、UPDATE 競合は発生しません。複数のサイトでデータが更新された場合にのみ、この競合が発生します。

競合解決トリガの使用

この項では、RESOLVE UPDATE トリガまたは「**競合解決**」トリガの使用方法について説明します。

VERIFY 句を持つ UPDATE 文

UPDATE 文の VERIFY 句に含まれる値にエラーが発生すると、競合解決トリガが起動されて、更新前にデータベースの値を一致させます。次に、VERIFY 句を持つ UPDATE 文を示します。

```
UPDATE table-list
SET column-name = expression, …
[ VERIFY (column-name, …)
   VALUES ( expression, …) ]
[ WHERE search-condition ]
```

VERIFY 句が使用できるのは、*table-list* にテーブルが 1 つしか存在しない場合だけです。VERIFY 句は、指定したカラムの値と、予測される値のセットを比較します。この予測値は、UPDATE 文が適用されたときにパブリッシャ・データベースに存在した値です。VERIFY 句を指定すると、テーブルが一度に 1 つずつ更新されます。

VERIFY 句を使用できるのは、単一のローを更新する場合だけです。ただし、このことがクライアント・アプリケーションを記述する場合に制約になることはありません。これは、複数のローの UPDATE 文がデータベースに入力されても、Message Agent によって単一のローに対する UPDATE 文の繰り返しとして解釈されるためです。

競合解決トリガの構文

RESOLVE UPDATE トリガの構文は、次のとおりです。


```
CREATE TRIGGER trigger-name
RESOLVE UPDATE
OF column-name ON table-name
[ REFERENCING [ OLD AS old_val ]
  [ NEW AS new_val ]
    [ REMOTE AS remote_val ] ]
FOR EACH ROW
BEGIN
  …
END
```

RESOLVE UPDATE トリガの起動後に、各ローが更新されます。REFERENCING 句を使用すると、更新するテーブルのロー値 (OLD)、更新用のロー値 (NEW)、VERIFY 句によれば存在するはずのロー (REMOTE) にアクセスできます。REMOTE AS 句の中で参照できるのは、VERIFY 句内のカラムだけです。その他のカラムの場合、「カラムが見つかりません」というエラーが表示されます。

VERIFY_ALL_ COLUMNS オプションの使用

データベース・オプション verify_all_columns は、デフォルトでは Off に設定されています。このオプションを On に設定すると、レプリケート・データベースですべてのカラムが検証され、カラムが 1 つでも異なる場合は、常に RESOLVE UPDATE トリガが起動されます。設定が Off の場合は、更新されたカラムだけが検証されます。

このオプションを On に設定すると、各 UPDATE に対して送信される情報が多くなるため、メッセージが大きくなります。

このオプションは、統合データベースで設定してからリモート・データベースを抽出すると、リモート・データベースでも設定されます。

verify_all_columns オプションは、PUBLIC グループ用、または Message Agent の接続文字列に含まれるユーザ用のいずれかに設定できます。

CURRENT REMOTE USER 特殊定数の使用

CURRENT REMOTE USER 特殊定数は、メッセージを送信するリモート・ユーザの ID を保持します。競合のレポートをテーブルに配置する RESOLVE UPDATE トリガにこの特殊定数を使用して、競合を発生させたユーザを特定できます。

競合解決の例

この項では、RESOLVE UPDATE トリガを使用して競合を処理する方法について説明します。

日付の競合解決

交渉管理システムのテーブルの 1 つに、各顧客との最新の交渉日のデータを保持するカラムがあるとします。

ある担当者が金曜日に顧客と交渉を行いましたが、この担当は、変更を次の月曜日まで統合データベースにアップロードしません。一方、別の担当者が土曜日に同じ顧客と交渉し、変更をその日の夕方に更新しました。

土曜日に UPDATE が統合データベースにレプリケートされても競合は発生しませんが、月曜日に UPDATE が受信された時点で、ローがすでに変更されていることがわかります。

デフォルトでは月曜日の UPDATE が処理され、金曜日を最新の交渉日とする、間違った情報を持つカラムが残されます。

このカラムの UPDATE 競合は、最新の日付をローに挿入して解決してください。

解決の実装

次の RESOLVE UPDATE トリガは、新しい 2 つの値から最新のものを選択して、データベースにその値を入力します。

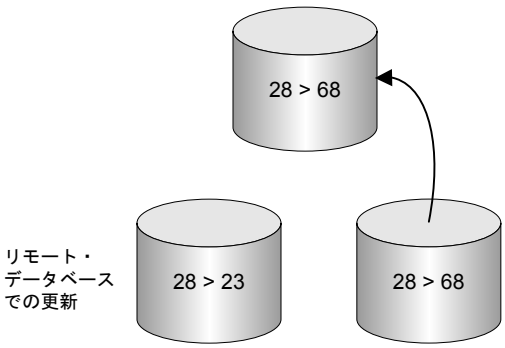
```
CREATE TRIGGER contact_date RESOLVE UPDATE
ON Contacts
REFERENCING OLD AS old_name
NEW AS new_name
FOR EACH ROW
BEGIN
  IF new_name.contact_date <
       old_name.contact_date THEN
    SET new_name.contact_date
        = old_name.contact_date
  END IF
END
```

更新される値が置き換える値よりも新しい場合は、新しい値はリセットされて、変更されないままエントリが残されます。

在庫数の競合解決

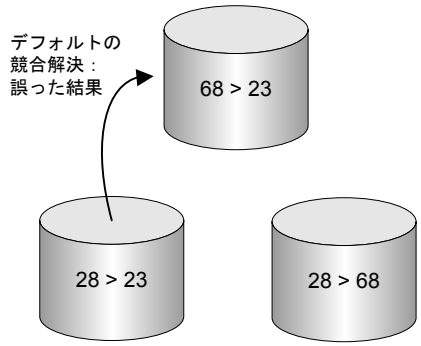
スポーツ用品メーカのための倉庫システムがあるとします。製品情報のテーブルには、各製品の在庫数を記録する **Quantity** カラムがあります。このカラムへの更新は、通常は在庫数を引いていくことであり、新しく製品が搬入される場合は、在庫数を追加します。

リモート・データベースで営業担当者が受注を入力し、S サイズのタンクトップ T シャツの在庫を 5 枚差し引いて 28 から 23 としました。この在庫数も担当者のデータベースに入力されます。一方、この更新内容が統合データベースにレプリケートされる前に、T シャツが新しく搬入されたため、倉庫では **Quantity** カラムの値を 40 追加して 68 になるように入力したとします。

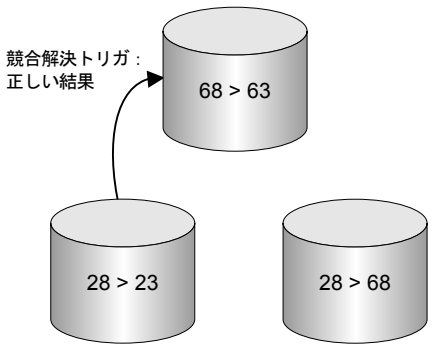


このエントリがデータベースに追加され、現在の **Quantity** カラムは、在庫に 68 枚の S サイズのタンクトップ T シャツがあることを示しています。ここで営業担当者からの更新が受信されると、28 から 23 に変更するという内容にもかかわらず、実際のカラム値は 68 であるという競合が SQL Anywhere で検出されます。

デフォルトでは、最新の UPDATE が処理され、在庫レベルが間違った値 23 に設定されます。



この場合の競合は、データベースの最終的な値が 63 になるように、在庫のカラムに対する変更を合計して最終結果を算出することによって解決してください。



解決の実装

このような状況に適した RESOLVE UPDATE トリガでは、2 つの更新内容による増加分を追加します。次に例を示します。

```
CREATE TRIGGER resolve_quantity
RESOLVE UPDATE OF Quantity
ON "DBA".Products
REFERENCING OLD AS old_name
NEW AS new_name
REMOTE AS remote_name
FOR EACH ROW
BEGIN
  SET new_name.Quantity = new_name.Quantity
                + old_name.Quantity
                - remote_name.Quantity
END
```

統合データベースの以前の値 (68) と、元の UPDATE が実行された時点のリモート・データベースの以前の値 (28) の差が、このトリガによって送信する新しい値に追加されてから、UPDATE が実行されます。したがって、**new_name.Quantity** は 63 (= 23 + 68 - 28) となり、この値が **Quantity** カラムに入力されます。

リモート・データベースでの一貫性は、次のように管理されます。

1. 元のリモート UPDATE によって、値が 28 から 23 に変更されました。
2. 倉庫システムのエントリがリモート・データベースにレプリケートされますが、以前の値が予測値と一致しないためにエラーが発生します。
3. RESOLVE UPDATE トリガによる変更内容が、リモート・データベースにレプリケートされます。

競合のレポート

場合によっては、SQL Remote による競合解決のデフォルトの方法を変更しないほうがよいときもあります。この場合は、テーブルに競合を格納し、これをレポートするようにできます。こうすると、競合が発生した場合にはその内容を調査でき、必要に応じて競合を解決できます。

参照整合性エラーを回避する設計

リレーショナル・データベースのテーブルは、外部キーの参照で関連付けられています。これらの参照の結果として適用される参照整合性制約により、データベースの一貫性が保たれます。データベースの一部のみをレプリケートする場合は、レプリケート・データベースの参照整合性に問題が発生する場合があります。

パブリケーションの設計中に参照整合性の問題に注意すれば、これらの問題を回避できます。この項では、よくある整合性の問題と、それを回避する方法について説明します。

参照テーブルがレプリケートされないエラー

「テーブル全体のパブリッシュ」 29 ページの項で示す **sales** パブリケーションには、**SalesOrders** テーブルが含まれます。

```
CREATE PUBLICATION PubSales (
   TABLE Customers,
   TABLE SalesOrders,
   TABLE SalesOrderItems,
   TABLE Products
)
```

SalesOrders テーブルには、**Employees** テーブルに対する外部キーがあります。営業担当者の ID は、**Employees** テーブルのプライマリ・キーを参照する、**SalesOrders** テーブルの外部キーです。ただし、**Employees** テーブルはパブリケーションに含まれていません。

この方法でパブリケーションを作成すると、リモート・データベースの **SalesOrders** テーブルから外部キー参照が削除されないかぎり、新規受注のレプリケートは失敗します。

抽出ユーティリティを使用してリモート・データベースを作成する場合、外部キー参照はリモート・データベースから自動的に取り除かれ、この問題は発生しません。ただし、データベースには、**SalesOrders** テーブルの **SalesRepresentative** カラムに無効な値が挿入されるのを防ぐ制約はありません。また、このような場合は、統合データベースで INSERT の実行に失敗します。パブリケーションに Employees テーブル (または少なくともプライマリ・キー) を含めることによって、この問題を回避できます。

エラーを回避するトリガの設計

トリガによって実行されるアクションはレプリケートされません。SQL Remote 設定内のあるデータベースに存在するトリガは、レプリケーション・プロシージャによって、設定内の別のデータベースにも存在しているとみなされます。統合データベースでトリガを起動するアクションが、レプリケート・サイトにレプリケートされると、そのトリガは自動的に起動します。デフォルトでは、データベース抽出ユーティリティがトリガ定義を抽出するため、リモート・データベースにもトリガ定義が配置されます。

パブリケーションにデータベースのサブセットだけが含まれる場合、統合データベースのトリガは、統合データベースでテーブルやローを参照できますが、リモート・データベースでは参照できません。IF 文を使用して条件付きのトリガ・アクションを作成し、このエラーを回避するようなトリガを設計できます。次に、統合データベースとリモート・データベースで機能するトリガの設計方法について示します。

- CURRENT PUBLISHER の値を使用して、条件付きのトリガ・アクションにします。この場合、トリガはリモート・データベース上で特定のアクションを実行しません。
- NULL 値を返さない **object_id** 関数を使用して、条件付きのトリガ・アクションにします。**object_id** 関数は、テーブルやその他のオブジェクトを引数として取得し、そのオブジェクトの ID 番号か NULL 値 (オブジェクトが存在しない場合) を返します。
- ローが存在するかどうか決定する SELECT 文を使用して、条件付きのトリガ・アクションにします。

RESOLVE UPDATE トリガは、UPDATE 競合の解決に使う特殊なトリガ・タイプです。RESOLVE UPDATE トリガについては、「競合解決の例」 57 ページの項で説明されています。

RESOLVE UPDATE トリガのアクションは、競合を引き起こしたデータベースなど、リモートのデータベースにレプリケートされます。

ユニークなプライマリ・キーの確保

プライマリ・キーの値はユニークである必要があります。すべてのユーザが同じデータベースに接続する場合、ユニークな値の維持に関する問題は発生しません。ユーザが値の再使用を試みる場合に、INSERT 文のエラーが発生します。

レプリケーション・システムにおいては状況が異なります。これは、ユーザが複数のデータベースに接続しているためです。異なるデータベースに接続している 2 人のユーザが、同じプライマリ・キーの値を使用してローを挿入すると問題が発生します。各データベースでは、値がユニークなので各文が正常に実行されます。

ただし、異なるデータベースに接続している 2 人のユーザが、同じプライマリ・キーの値を使用してローを挿入すると、レプリケーション・システムで問題が発生します。レプリケーション・システムの指定されたデータベースで受信される、2 番目の INSERT の実行は失敗します。SQL Remote は不定期で接続するユーザのためのレプリケーション・システムなので、インストール環境のすべてのデータベースに対するロック・メカニズムはありません。したがって、プライマリ・キーの重複エラーが発生しないように SQL Remote のインストール環境を設計する必要があります。

SQL Remote のインストール環境以外で設計されたプライマリ・キー・エラーの場合は、複数のサイトで修正するテーブルのプライマリ・キーがユニークであることを保証する必要があります。この目的を達成する方法はいくつかあります。この章では、一般的で効率的な、信頼性の高い 2 つの方法について説明します。

1. SQL Anywhere のデフォルトのグローバル・オートインクリメント機能を使用します。
2. プライマリ・キー・プールを使用して、各サイトで未使用のユニークなプライマリ・キー値のリストを管理します。

これらの方法のいずれか一方または両方を使用して、値の重複を回避できます。

グローバル・オートインクリメント・デフォルト・カラム値の使用

SQL Anywhere では、デフォルトのカラム値を GLOBAL AUTOINCREMENT に設定できます。このデフォルト設定は、ユニークな値を管理するカラムのすべてに適用できますが、特にプライマリ・キーの場合に役立ちます。この機能の目的は、通常 Mobile Link 同期によってデータが複数のデータベース間でレプリケートされるときに、ユニークな値を生成するタスクを簡素化することです。

このデフォルトのグローバル・オートインクリメントを指定すると、そのカラムの値のドメインが分割されます。各分割には同じ数の値が含まれます。たとえば、データベース内の整数カラムの分割サイズを 1000 に設定した場合、1 つの分割が 1001 から 2000 まで拡大します。また、2 つ目の分割は 2001 から 3000 まで拡大し、以降、同じように拡大していきます。

データベースの各コピーに、ユニークなグローバル・データベース ID 番号を割り当てます。SQL Anywhere では、データベースのデフォルト値は、そのデータベース番号でユニークに識別された分割からのみ設定されます。たとえば、上の例のデータベースに ID 番号 10 を割り当てた場合、このデータベースのデフォルト値は 10001 ー 11000 の範囲から選択されます。このデータ

ベースの別のコピーで、ID 番号 11 が割り当てられたデータベースからは、11001 ー 12000 の範囲にある同一カラムのデフォルト値が指定されます。

デフォルトのグローバル・オートインクリメントを宣言する

Sybase Central でカラムのプロパティを選択するか、TABLE 文または ALTER TABLE 文に DEFAULT GLOBAL AUTOINCREMENT 句を組みこむことで、作業データベースにデフォルト値を設定できます。

オプションで、AUTOINCREMENT キーワードの直後にカッコで分割サイズを指定できます。この分割サイズには任意の正の整数を設定できますが、通常、分割サイズは、サイズの値がすべての分割で不足しないように選択されます。

INT または UNSIGNED INT 型のカラムの場合、デフォルトの分割サイズは $2^{16} = 65536$ です。他の型のカラムの場合、デフォルトの分割サイズは $2^{32} = 4294967296$ です。特にカラムが INT または BIGINT 型以外のときは、これらのデフォルトが適切ではない場合があるため、分割サイズを明示的に指定してください。

たとえば、次の文では 2 つのカラム (顧客 ID 番号を保持する整数カラム、顧客名を保持する文字列カラム) を持つ簡単なテーブルが作成されます。

```
CREATE TABLE Customers (
  ID   INT         DEFAULT GLOBAL AUTOINCREMENT (5000),
  name VARCHAR(128) NOT NULL,
  PRIMARY KEY (ID)
)
```

上の例では、選択された分割サイズは 5000 です。

GLOBAL AUTOINCREMENT の詳細については、「CREATE TABLE 文」 『SQL Anywhere サーバ - SQL リファレンス』を参照してください。

global_database_id 値の設定

アプリケーションを配備するときには、各データベースに対して必ず異なる ID 番号を割り当てる必要があります。ID 番号の作成と配布はさまざまな方法で実行できます。テーブルに値を設定し、ユーザ名など、ユニークなプロパティに基づいて、各データベースに適切なローをダウンロードするのも 1 つの方法です。

♦ **グローバル・データベース ID 番号を設定するには、次の手順に従います。**

- パブリック・オプション **global_database_id** の値を設定して、データベースの ID 番号を設定します。ID 番号は正の整数にします。

たとえば、次の文ではデータベースの ID 番号が 20 に設定されます。

```
SET OPTION PUBLIC.global_database_id = 20
```

特定カラムの分割サイズが 5000 の場合、このデータベースのデフォルト値は 100001 ー 105000 の範囲から選択されます。

データベース抽出時のユニークなデータベース ID 番号の設定

抽出ユーティリティを使用してリモート・データベースを作成する場合は、ストアド・プロシージャを作成してタスクを自動化できます。sp_hook_dbxtract_begin という名前のストアド・プロシージャを作成すると、抽出ユーティリティによって自動的に呼び出されます。プロシージャが呼び出される前に、次の内容の入った #hook_dict という名前のテンポラリ・テーブルを、抽出ユーティリティが作成します。

name	value
extracted_db_global_id	抽出されるユーザ ID

sp_hook_dbxtract_begin プロシージャを書き込んでローの value カラムを修正する場合、その値は抽出されたデータベースの global_database_id オプションとして使用され、GLOBAL DEFAULT AUTOINCREMENT 値のプライマリ・キー値範囲の開始位置にマークを付けます。

例

101 という **user_id** を持つリモート・ユーザ **user2** のために、データベースを抽出するとします。sp_hook_dbxtract_begin プロシージャを定義しない場合、抽出されるデータベースは、**global_database_id** が **101** に設定されます。

sp_hook_dbxtract_begin プロシージャを定義しても、このプロシージャが #hook_dict のいずれのローも修正しない場合、オプションは **101** に設定されたままになります。

データベースを次のように設定するとします。

```
set option "PUBLIC"."global_database_id" = '1';
create table extract_id ( next_id integer not null) ;
insert into extract_id values( 1 );
create procedure sp_hook_dbxtract_begin
as
   declare @next_id  integer
   update extract_id set next_id = next_id + 1000
   select @next_id = (next_id )
   from extract_id
   commit
   update #hook_dict
   set value = @next_id
   where name = 'extracted_db_global_id'
```

この設定を使用すると、データベースが抽出されるたびに、**global_database_id** の値が毎回異なります。1 回目は 1001、2 回目は 2001 という具合です。

プロシージャ・フックのデバッグに役立つように、*dbxtract* を冗長モードで動作するよう設定すると、次のものが出力されます。

- 検出されたプロシージャ・フック
- プロシージャ・フックが呼び出される前の #hook_dict の内容
- プロシージャ・フックが呼び出された後の #hook_dict の内容

デフォルト値の選択方法

各データベース内のパブリック・オプション **global_database_id** は、ユニークな正の整数に設定してください。特定のデータベースのデフォルト値の範囲は、*pn* + 1 から *p*(*n* + 1) です。ここで、*p* は分割サイズ、*n* はパブリック・オプション **global_database_id** の値を表します。たとえば、分割サイズを 1000、**global_database_id** を 3 に設定すると、範囲は 3001 ー 4000 になります。

global_database_id が正の整数に設定されている場合、SQL Anywhere は次のルールを適用してデフォルト値を選択します。

- カラムに現在の分割の値が含まれていない場合、最初のデフォルト値は *pn* + 1 である。
- カラムに現在の分割の値が含まれていても、そのすべてが *p*(*n* + 1) 未満であれば、この範囲内でこれまで使用した最大値より 1 大きい値が次のデフォルト値になる。
- デフォルトのカラム値は、現在の分割以外のカラムの値の影響を受けない。つまり、*pn* + 1 より小さいか *p*(*n* + 1) より大きい数には影響されない。Mobile Link 同期を介して別のデータベースからレプリケートされた場合に、このような値が存在する可能性があります。

public オプション **global_database_id** がデフォルト値の 2147483647 に設定されると、NULL 値がカラムに挿入されます。NULL 値が許可されていない場合に、ローの挿入を試みるとエラーが発生します。たとえば、テーブルのプライマリ・キーにカラムが含まれている場合に、この状況が発生します。

public オプション **global_database_id** は、負の値に設定できないため、選択された値は常に正になります。ID 番号の最大値を制限するのは、カラムのデータ型と分割サイズだけです。

デフォルトの NULL 値は、分割で値が不足したときにも生成されます。この場合には、別のパーティションからデフォルト値を選択できるように、データベースに **global_database_id** の新しい値を割り当ててください。カラムで NULL が許可されていない場合、NULL 値を挿入しようとするとエラーが発生します。未使用の値が残り少ないことを検出し、このような状態を処理するには、**GlobalAutoincrement** タイプのイベントを作成します。

特定の分割で値が不足する場合は、新しいデータベース ID をそのデータベースに割り当てることができます。方法が適切なものであれば、新しいデータベース ID 番号を割り当てることができます。未使用のデータベース ID 値のプールを管理する方法も、その 1 つです。このプールは、プライマリ・キー・プールと同じ方法で管理されます。

分割で値が不足しそうな場合に、自動的にデータベース管理者へ通知する (またはその他のアクションを実行する) ようにイベント・ハンドラを設定できます。「イベントのトリガ条件の定義」『SQL Anywhere サーバ - データベース管理』を参照してください。

「global_database_id オプション [データベース]」『SQL Anywhere サーバ - データベース管理』を参照してください。

プールの詳細については、「プライマリ・キー・プールの使用」 66 ページを参照してください。

プライマリ・キー・プールの使用

「**プライマリ・キー・プール**」は、SQL Remote のインストール環境で、各データベースのプライマリ・キー値のセットを保持するテーブルです。リモート・ユーザは、各自のプライマリ・キー値のセットを受信します。リモート・ユーザは、新しいローをテーブルに挿入する場合、ストアド・プロシージャを使用してプールから有効なプライマリ・キーを選択します。プールは、使用できる値を補充するプロシージャを、統合データベースで定期的に実行することによって管理されます。

営業担当者とその顧客からなる簡単なデータベースの例を用いて、この方法について説明します。これらのテーブルは実際のデータベースで使用するものよりずっと単純なので、レプリケーションに関する重要な問題だけに注目できます。

プライマリ・キー・プールを使用するには、次のコンポーネントが必要です。

- **キー・プール・テーブル**　インストール環境の各データベースに対する、有効なプライマリ・キー値を保持するテーブル。
- **補充プロシージャ**　値を補充したキー・プール・テーブルを維持するストアド・プロシージャ。
- **キー・プールの共有**　インストール環境の各データベースは、キー・プール・テーブルから取得した、そのデーターベース自体の有効な値のセットに対してサブスクリプションを作成する必要があります。
- **データ入力プロシージャ**　次に有効なプライマリ・キー値をプールから選択し、キー・プールからその値を削除するストアド・プロシージャを使用して、新しいローを入力します。

プライマリ・キー・プールのテーブル

プライマリ・キー・プールは別のテーブルに記録されます。次の CREATE TABLE 文は、プライマリ・キー・プールのテーブルを作成します。

```
CREATE TABLE KeyPool (
  table_name VARCHAR(40) NOT NULL,
  value INTEGER NOT NULL,
  location CHAR(12) NOT NULL,
  PRIMARY KEY (table_name, value),
);
```

このテーブルの各カラムの意味は、次のとおりです。

カラム	説明
table_name	プライマリ・キー・プールで管理するテーブルの名前を保持する。ここに示す簡単なデータベースの例では、統合データベースだけに営業担当者が新しく追加されると、**Customers** テーブルだけがプライマリ・キー・プールを必要とし、このカラムが重複する。これは、一般的な解決方法を示すために含まれている。
value	プライマリ・キー値のリストを保持する。それぞれの値は、**table_name** にリストされた各テーブルに対してユニークである。

カラム	説明
location	受信者の識別子。設定によっては、この値が **SalesReps** テーブルの **rep_key** 値と同じになる場合がある。また、営業担当者以外のユーザがいて、この 2 つの識別子が別々でなければならない設定もある。

パフォーマンスを向上させるために、次のようにテーブルにインデックスを作成できます。

```
CREATE INDEX KeyPoolLocation
ON KeyPool (table_name, location, value);
```

プライマリ・キー・プールのレプリケーション

キー・プールは、既存のパブリケーションに組み込むか、または別のパブリケーションとして共有することができます。この例では、プライマリ・キー・プール用に別のパブリケーションを作成します。

- **プライマリ・キー・プールをレプリケートするには、次の手順に従います (SQL の場合)。**

1. プライマリ・キー・プールのデータ用に、パブリケーションを作成します。

   ```
   CREATE PUBLICATION KeyPoolData (
     TABLE KeyPool SUBSCRIBE BY location
   );
   ```

2. 各リモート・データベースのサブスクリプションを、KeyPoolData パブリケーションに作成します。

   ```
   CREATE SUBSCRIPTION
   TO KeyPoolData( 'user1' )
   FOR user1;
   CREATE SUBSCRIPTION
   TO KeyPoolData( 'user2' )
   FOR user2;
   ...
   ```

   サブスクリプションの引数は、location の識別子です。

状況によっては、既存のパブリケーションに KeyPool テーブルを追加し、同じ引数を使用して各パブリケーションにサブスクリプションを作成する方法は有効です。この例では、location の値と rep_key の値を別にして、より一般的な解決方法について説明しています。

参照

- 「CREATE PUBLICATION 文 [Mobile Link] [SQL Remote]」『SQL Anywhere サーバ - SQL リファレンス』
- 「CREATE SUBSCRIPTION 文 [SQL Remote]」『SQL Anywhere サーバ - SQL リファレンス』

キー・プールの補充

ユーザが新しい顧客を追加するたびに、使用できるプライマリ・キーのプールは 1 つずつ減少します。次のようなプロシージャを使用して、統合データベースでプライマリ・キー・プールのテーブルに定期的に補充する必要があります。

```
CREATE PROCEDURE ReplenishPool()
BEGIN
  FOR EachTable AS TableCursor
  CURSOR FOR
    SELECT table_name
    AS CurrTable, max(value) as MaxValue
    FROM KeyPool
    GROUP BY table_name
  DO
    FOR EachRep AS RepCursor
    CURSOR FOR
      SELECT location
      AS CurrRep, count(*) as NumValues
      FROM KeyPool
      WHERE table_name = CurrTable
      GROUP BY location
    DO
      // make sure there are 100 values.
      // Fit the top-up value to your
      // requirements
      WHILE NumValues < 100 LOOP
        SET MaxValue = MaxValue + 1;
        SET NumValues = NumValues + 1;
        INSERT INTO KeyPool
        (table_name, location, value)
        VALUES
        (CurrTable, CurrRep, MaxValue);
      END LOOP;
    END FOR;
  END FOR;
END;
```

このプロシージャによって、各ユーザのプールの値が 100 個まで増えます。指定する値は、ユーザがデータベースのテーブルにローを挿入する頻度によって異なります。

KeyPool テーブルのプライマリ・キー値のプールを再補充するため、**ReplenishPool** プロシージャを統合データベースで定期的に実行する必要があります。

ReplenishPool プロシージャには、各サブスクライバに対して少なくとも 1 つ以上のプライマリ・キー値が必要です。これによって、最大値を検索し、値を追加して次のセットを生成できます。はじめにプールを値で埋めるには、各ユーザに対して値を 1 つ挿入してから、**ReplenishPool** を呼び出して残りを埋めます。次の例では、3 人のリモート・ユーザと単一の統合ユーザ **Office** について示します。

```
INSERT INTO KeyPool VALUES( 'Customers', 40, 'user1' );
INSERT INTO KeyPool VALUES( 'Customers', 41, 'user2' );
INSERT INTO KeyPool VALUES( 'Customers', 42, 'user3' );
INSERT INTO KeyPool VALUES( 'Customers', 43, 'Office');
CALL ReplenishPool();
```

トリガを使用してキー・プールを補充できない
トリガ・アクションはレプリケートされないため、キー・プールの補充にはトリガを使用できません。

キー・プールのプライマリ・キーの使用

営業担当者が Customers テーブルに新しく顧客を追加する場合、挿入するプライマリ・キー値は、ストアド・プロシージャを使用して取得します。次の例では、プライマリ・キー値を提供するストアド・プロシージャと、INSERT を実行するストアド・プロシージャについて説明します。

このプロシージャは、営業担当者の識別子がリモート・データベースの CURRENT PUBLISHER であることを利用します。

- **NewKey プロシージャ** **NewKey** プロシージャは、キー・プールにある整数値を提供し、プールからその値を削除します。

```
CREATE PROCEDURE NewKey(
    IN @table_name VARCHAR(40),
    OUT @value INTEGER )
BEGIN
  DECLARE NumValues INTEGER;

  SELECT count(*), min(value)
  INTO NumValues, @value
      FROM KeyPool
      WHERE table_name = @table_name
      AND location = CURRENT PUBLISHER;
  IF NumValues > 1 THEN
    DELETE FROM KeyPool
    WHERE table_name = @table_name
    AND value = @value;
  ELSE
  // Never take the last value, because
  // ReplenishPool will not work.
  // The key pool should be kept large enough
  // that this never happens.
    SET @value = NULL;
  END IF;
END;
```

- **NewCustomers プロシージャ** **NewCustomers** プロシージャは、プライマリ・キーを構成する **NewKey** が取得した値を使用して、テーブルに新しい顧客を追加します。

```
CREATE PROCEDURE NewCustomers(
    IN customer_name CHAR( 40 ) )
BEGIN
  DECLARE new_cust_key INTEGER ;
  CALL NewKey( 'Customers', new_cust_key );
  INSERT
  INTO Customers (
    cust_key,
    name,
    location
    )
  VALUES (
    'Customers ' ||
```



```
        CONVERT (CHAR(3), new_cust_key),
        customer_name,
        CURRENT PUBLISHER
        );
);
END
```

NewKey から取得される **new_cust_key** 値をテストしてこの値が NULL でないことを確認し、値が NULL の場合には挿入しないように、プロシージャを強化できます。

サブスクリプションの作成

パブリケーションに対するサブスクリプションを作成するには、各サブスクライバに REMOTE パーミッションが付与されている必要があります。また、そのユーザ用にサブスクリプションを作成する必要があります。サブスクリプションの詳細は、パブリケーションにサブスクリプション式を使用するかどうかによって異なります。

Sybase Central におけるサブスクリプションの活用

- **Sybase Central でサブスクリプションを作成／管理するには、次の手順に従います。**

1. 左ウィンドウ枠で、[パブリケーション] フォルダを開きます。
2. 必要なパブリケーションを選択します。Sybase Central では次のタスクを実行できます。
3. 右ウィンドウ枠で、[SQL Remote サブスクリプション] タブをクリックします。適切な設定を次のように設定できます。

   - パブリケーションにリモート・ユーザのサブスクリプションを作成するには、[ファイル] メニューで [新規] – [SQL Remote サブスクリプション] を選択し、[SQL Remote サブスクリプションの作成] ウィザードの指示に従います。
   - リモート・ユーザのサブスクリプションを削除するには、[サブスクライバ] リストでユーザを右クリックし、ポップアップ・メニューで [削除] をクリックします。
   - サブスクリプションを手動で起動、停止、または同期するには、[サブスクライバ] リスト内のユーザを選択し、ポップアップ・メニューで [プロパティ] を選択します。

     [詳細] タブをクリックします。このタブで、サブスクリプションを起動、停止、同期するために、[すぐに起動] [すぐに停止] [すぐに同期化] をそれぞれクリックします。

     ボタンをクリックすると、すぐにサブスクリプションが処理されます。プロパティ・シートで [キャンセル] を続けてクリックしても、開始／停止／同期のアクションはキャンセル*されません*。

サブスクリプション式を使用しないサブスクリプション

パブリケーションにサブスクリプション式がない場合に、パブリケーションに対するユーザのサブスクリプションを作成するには、次の情報が必要です。

- **ユーザ ID**　このユーザのサブスクリプションがパブリケーションに対して作成されます。このユーザには、REMOTE パーミッションが付与されている必要があります。
- **パブリケーション名**　このパブリケーション名でユーザのサブスクリプションが作成されます。

次の文は、ユーザ ID **SamS** のサブスクリプションを **PubOrdersSamuelSinger** パブリケーションに作成します。このパブリケーションは WHERE 句を使用して作成されます。

```
CREATE SUBSCRIPTION
TO PubOrdersSamuelSinger
FOR SamS
```


サブスクリプション式を使用するサブスクリプション

パブリケーションにサブスクリプション式がある場合に、パブリケーションへユーザのサブスクリプションを作成するには、次の情報が必要です。

- **ユーザ ID**　このユーザのサブスクリプションがパブリケーションに対して作成されます。このユーザには、REMOTE パーミッションが付与されている必要があります。
- **パブリケーション名**　ユーザのサブスクリプションを作成するパブリケーション名
- **サブスクリプション値**　パブリケーションのサブスクリプション式に対するテスト用の値です。たとえば、従業員 ID を含むカラム名をパブリケーションがサブスクリプション式として保持する場合、サブスクリプションを作成するユーザの従業員 ID の値が、サブスクリプションに提供されなければなりません。サブスクリプション値は常に文字列です。

次の文は、Samuel Singer (ユーザ ID **SamS**、従業員 ID 856) のサブスクリプションを PubOrders パブリケーションに作成します。このパブリケーションには、サブスクリプション式 **SalesRepresentative** が定義され、Samuel Singer 自身の受注に対してローを要求します。

```
CREATE SUBSCRIPTION
TO PubOrders ( '856' )
FOR SamS
```

サブスクリプションの開始

正しく更新を受信して適用するため、各サブスクライバにはデータの初期コピーが必要です。同期の処理については、「データベースの同期」 79 ページで説明しています。

「CREATE SUBSCRIPTION 文 [SQL Remote]」『SQL Anywhere サーバ - SQL リファレンス』を参照してください。

パート III. SQL Remote の管理

パート III では、SQL Remote の配備と管理について説明します。

第 4 章

SQL Remote を使用したデータベースの配備と同期

目次

配備の概要

SQL Remote システムの設計段階が完了したら、次にリモート・データベースとアプリケーションを作成して配備します。

配備作業

場合によっては、配備作業は主要な作業となります。たとえば、営業支援システムに多数のリモート・ユーザがいる場合、配備作業は次のようになります。

1. 各リモート・ユーザ用の SQL Anywhere データベースとそのデータの最初のコピーを構築します。
2. 各ユーザのマシンに、SQL Anywhere データベース・サーバ、SQL Remote Message Agent、クライアント・アプリケーションとともにデータベースをインストールします。
3. ユーザ名、Message Agent 接続文字列、パーミッションなどが正しく、システムが適切に設定されていることを確認します。

大規模な配備の場合、リモート・サイトには一般的に SQL Anywhere データベースが使用されます。この章では、このことを前提に説明を進めます。

この章のトピック

この章のトピックは、次のとおりです。

- **リモート・データベースの作成**　各リモート・サイト用のリモート・データベースを作成してから、SQL Remote システムを配備します。

  主に、リモートの SQL Anywhere データベースの作成について説明します。
- **データの同期**　データベースを同期させるには、データの最初のコピーをリモート・データベースに設定します。

配備前のテスト

SQL Remote システムを配備する前に、特に多数のリモート・サイトがある場合には、そのシステムを十分にテストしてください。

設計および設定段階にある場合は、SQL Remote 設定のあらゆる部分を変更できます。パブリケーションやメッセージ・タイプを変更したり、トリガを作成して更新の競合を解決することが簡単にできます。

SQL Remote アプリケーションを配備した後は、状況が異なります。SQL Remote 設定は、単一の「**分散データベース**」として認識できます。データベースは多数のサイトに分散しており、緩やかな一貫性が保持されます。設定内のすべてのデータベースのデータは、常にまったく同じ状態にはなりません。しかし時間が経つと、すべてのデータ変更は完全なトランザクションとしてシステム全体にレプリケートされます。SQL Remote 設定の一貫性は、入念なパブリケーション設計と、更新の競合が発生したときにそれが解決されることによって成立します。

アップグレードと再同期

SQL Remote 設定を配備して実行した後に修正することは容易ではありません。SQL Remote インストールに対するアップグレードは、最初の配備と同じくらい注意して行う必要があります。SQL Anywhere データベース・ソフトウェアのメンテナンス・リリースをアップグレードする場合も同様です。これらのソフトウェアのアップグレードについては、配備前に互換性をテストする必要があります。

システム内の 1 つのデータベースでデータベース・スキーマを変更すると、データベース・オブジェクトに互換性がないために障害が発生する恐れがあります。パススルー・モードによって、スキーマの変更を SQL Remote 設定の一部またはすべてのデータベースに送信することが可能です。しかし、パススルー・モードの使用には注意と計画が必要です。

分散データベースの緩やかな一貫性とは、更新が常に進行中であるという意味です。通常、すべてのデータベースに実行中の変更を停止して、データベース・スキーマの一部を変更し、再起動することはできません。

計画を入念に行わないと、データベース・スキーマの変更によってインストールでエラーが発生し、すべてのサブスクリプションを停止して再同期させなければならない場合があります。再同期させるには、各リモート・サーバにデータの新しいコピーをロードする必要があります。サブスクライバが多数ある場合は、この処理に時間がかかり、作業の中断やデータの消失が発生する恐れがあります。

実行中のシステムに行ってはならない変更

次に、配備済みで実行中の SQL Remote 設定に行ってはならない変更の例を示します。「**許容できる**」と記述されている変更は、通常は許容できる変更です。「**制限のある**」と記述されている変更は、行うべきでない変更です。

以下に条件が記述されている場合を除き、次の変更は行わないでください。

- 統合データベースのパブリッシャの変更。

- カラムの削除や NULL 値を許容しないようにカラムを変更することなど、テーブルに対して行う制限のある変更。カラムや NULL 値に関連する変更は、メッセージによって SQL Remote 設定全体に送信済みである可能性があります。変更は失敗します。
- パブリケーションの変更。パブリケーション定義は、ローカル・サイトとリモート・サイトの両方で管理する必要があります。古いパブリケーション定義に従って行われた変更は、すでにメッセージによって SQL Remote 設定全体に送信済みである可能性があります。

  新規テーブルや新規カラムの追加など許容できる変更を行うことは可能です。ただし、パススルーによって、新規テーブルや新規カラムがリモート・データベース内およびリモート・データベースのパブリケーション内に存在するようにしている場合にかぎります。
- サブスクリプションの削除。パススルーの削除機能を使用してリモート・サイトのデータを削除する場合のみ、これを行うことは可能です。
- SQL Anywhere データベースのアンロードと再ロード。

  SQL Anywhere データベースがレプリケーションに関係している場合は、データベースを同期し直さなければ、そのデータベースのアンロードと再ロードはできません。レプリケーションはトランザクション・ログに基づいていますが、データベースをアンロードして再ロードすると、古いトランザクション・ログが使えなくなります。このため、レプリケーションが関係するときは、バックアップをきちんと実行することが特に重要です。

データベースの同期

同期とは何か

SQL Remote レプリケーションは、トランザクション・ログの情報を使用して実行されます。しかし、リモート・データベースと統合データベースという 2 つの状況があるので、SQL Remote は、パブリケーションの一部を成すリモート・データベースのテーブルにあるすべての既存のローを削除して、パブリケーションの全内容を統合データベースからリモート・サイトにコピーします。この、リモート・データベースと統合データベースを同じにする処理を、「**同期**」といいます。

同期の時期

同期は、次の場合に使用されます。

- 統合データベースでサブスクリプションが作成されると、リモート・データベースが統合データベースと同じ状態で開始できるように、同期が実行されます。
- リモート・データベースが破損したり、リモート・データベースと統合データベースの処理ステップが揃わなくなり、SQL パススルー・モードを使用して修復できない場合は、同期によってリモート・サイト・データベースと統合データベースの処理ステップが揃うようにします。

同期の方法

リモート・データベースの同期は、次の方法で実行できます。

- **データベース抽出ユーティリティの使用**　このユーティリティによって、リモートの SQL Anywhere データベースのスキーマが作成され、リモート・データベースが同期されます。通常は、この方法をおすすめします。
- **手動による同期**　PowerBuilder パイプラインや他のツールを使用してファイルからロードすることによって、リモート・データベースを手動で同期させます。
- **メッセージ・システムを介した同期**　SYNCHRONIZE SUBSCRIPTION 文を使用して、メッセージ・システムを介してリモート・データベースを同期させます。

> **警告**
> SYNCHRONIZE SUBSCRIPTION をリモート・データベースで実行しないでください。

複数のオペレーティング・システムとデータベース抽出

多くの場合、統合サーバのオペレーティング・システムとリモート・データベースのオペレーティング・システムは異なります。

SQL Anywhere データベースは、1 つのファイル／オペレーティング・システムから、別のファイル／オペレーティング・システムにコピーできます。これにより、データベースを最初に同期させる作業が柔軟に行えます。

同期と抽出に関する注意事項

- 多数のサブスクリプションを抽出したり、サブスクリプションを大規模で使用頻度の高いテーブルに同期させたりすると、他のユーザがデータベースにアクセスする際の処理速度が低下します。データベースへのアクセスが少ないときにサブスクリプションを抽出することをおすすめします。SEND AT 句を使用してアクセスが少ない時刻を指定すると、この処理は自動的に実行されます。
- 同期は、サブスクリプション全体に適用されます。現時点では、1 つのテーブルを同期させる簡単な方法はありません。

抽出ユーティリティ (dbxtract) の使用

抽出ユーティリティは、リモートの SQL Anywhere データベースを作成するのに役立ちます。

抽出ユーティリティの実行

抽出ユーティリティには次の方法でアクセスできます。

- Sybase Central からアクセスする方法 (統合データベースが SQL Anywhere の場合)。「Sybase Central でのリモート・データベースの抽出」 162 ページを参照してください。
- コマンド・ライン・ユーティリティとしてアクセスする方法。これは、*dbxtract* ユーティリティです。「抽出ユーティリティ」 163 ページを参照してください。

警告
抽出ユーティリティの実行中は、Message Agent を実行しないでください。予測できない結果を生じる可能性があります。

再ロード・ファイルからのデータベースの作成

コマンド・ライン・ユーティリティによって、名前を付けたサブスクライバ用のリモートの SQL Anywhere データベースを構築するのに適したデータベース・スキーマとデータをアンロードします。これにより、*reload.sql* というデフォルト名が付いた SQL コマンド・ファイルと、データ・ファイルのセットが生成されます。これらのファイルを使用して、リモートの SQL Anywhere データベースを作成できます。

reload.sql の編集が必要な場合があります。
データベース抽出ユーティリティはリモート・データベースの準備を支援するためのものですが、すべての場合においてブラック・ボックス・ソリューションとはなりません。リモート・データベースを作成するときは、必要に応じて *reload.sql* コマンド・ファイルを編集してください。

♦ 再ロード・ファイルからリモート・データベースを作成するには、次の手順に従います。

1. 次のいずれかの方法で、SQL Anywhere データベースを作成します。
   - Sybase Central の [データベースの作成] ウィザード ([ツール] – [SQL Anywhere 10] – [データベースの作成] を選択)
   - *dbinit* ユーティリティ
2. Interactive SQL からデータベースに接続して、*reload.sql* コマンド・ファイルを実行します。[SQL 文] ウィンドウ枠に次の文を入力して、*reload.sql* コマンド・ファイルを実行します。

   ```
   read path¥reload.sql
   ```

   path には、再ロード・コマンド・ファイルのパスを指定します。

Sybase Central から使用する場合、抽出ユーティリティでは、*dbxtract* と同様にデータベースをアンロードするタスクが実行され、さらに新規データベースの作成処理が実行されます。

抽出ユーティリティはメッセージ・システムを使用しません。再ロード・ファイル (*dbxtract*) またはデータベース (Sybase Central の場合) が、現在のマシンからアクセス可能なディレクトリに作成されます。メッセージ・リンクを介して多数のサブスクリプションを同期させると、メッセージ・トラフィックの量が増えます。また、メッセージ・システムが信頼性に欠ける場合は、すべてのメッセージがリモート・サイトで正常に受信されるのに多少時間がかかることがあります。

データベースを抽出する前に

統合データベースで抽出ユーティリティを使用する前に、次の作業を完了してください。

- レプリケーションのメッセージ・タイプを作成する。
- データベースにパブリッシャ・ユーザ ID を追加する。
- データベースにリモート・ユーザを追加する。
- データベースにパブリケーションを追加する。
- リモート・ユーザのサブスクリプションを作成する。
- メッセージ・リンク・パラメータを指定する必要がある場合は、それを指定する。「メッセージ・タイプ制御パラメータの設定」 100 ページを参照してください。

抽出ユーティリティを使用してリモート・データベースを作成すると、そのデータベースを使用するユーザには、統合データベースで所有しているパーミッションと同じパーミッションが与えられます。さらに、そのユーザが統合データベースのグループのメンバである場合は、そのグループ ID が、ユーザが統合データベースで所有しているパーミッションが存在するリモート・データベースに作成されます。

Sybase Central からの抽出ユーティリティの使用

この項では、現在の統合データベースからリモート・ユーザ用のデータベースを抽出する方法を説明します。この項は、SQL Anywhere 統合データベースだけに適用されます。

[データベースの抽出] ウィザードによって、マシンでは次のことが実行されます。

- リモート・データベースの作成
- 統合データベースからファイルへの、関連する構造やデータの抽出 (アンロード)
- 新規に作成されたリモート・データベースへのファイルのロード

♦ リモート・ユーザ用のデータベースを抽出するには、次の手順に従います (Sybase Central)。

1. [ツール] メニューから、[SQL Anywhere 10] – [データベースの抽出] の順に選択します。

2. ウィザードの指示に従います。

注意

- このウィザードには、[ツール] – [SQL Anywhere 10] – [データベースの抽出] の順にクリックしてもアクセスできます。
- 特定のデータベースまたは特定のリモート・ユーザ用に抽出ウィザードを起動することもできます。Sybase Central によって、ウィザードへの適切な入力が自動的に行われます。
- 抽出ウィザードは、必ず WITH SYNCHRONIZATION オプションを使用してリモート・サーバを抽出 (同期) します。このオプションを使用したくない場合は、代わりに dbxtract ユーティリティを使用してください。
- [データベースの抽出] ウィザードには、[所有者別にオブジェクトをフィルタ] ダイアログで選択したユーザのテーブルだけが表示されます。特定のデータベース・ユーザに属するテーブルを表示させたい場合、アンロードするデータベースを右クリックし、ポップアップ・メニューから [所有者別にオブジェクトをフィルタ] を選択して、表示されるダイアログから対象ユーザを選択します。

参照

- 抽出ユーティリティのオプション (コマンド・ライン・オプションとして、または抽出ウィザードの選択肢として使用可能) については、「抽出ユーティリティ」 163 ページを参照してください。

効率的な抽出プロシージャの設計

リモート・データベースごとに抽出ユーティリティを実行して、多数のデータベースを作成する処理は、非常に非効率的です。この処理を、より効率的にすることができます。この項では、処理を効率的にする方法を説明します。

大規模な抽出処理が非効率的になる原因には、次のものがあります。

- 抽出ユーティリティは、各ユーザ用のスキーマとデータを含め、一度にデータベースを 1 つだけ処理します。通常、多数のユーザが共通のスキーマを共有し、データだけが異なります。各ユーザについて抽出ユーティリティを実行する単調な方法では、大量の処理が不必要に繰り返されます。スキーマとデータを別々に抽出すると、この問題を解決できます。
- Sybase Central から抽出ユーティリティを実行すると、各ユーザ用の新しいデータベースが作成されます。サブスクライバがスキーマを共有している場合は、スキーマはあるがデータのないデータベースを 1 つ作成し、ファイルをコピーできます。
- デフォルトでは、抽出ユーティリティは独立性レベル 0 で実行されます。アクティブなサーバからデータベースを抽出する場合は、独立性レベル 3 で実行し、抽出されたデータベース内のデータがサーバ上のデータと一致するようにします。「抽出ユーティリティ」 163 ページを参照してください。

  独立性レベル 3 で実行すると、多数のロックが必要になるため、サーバ上の他のユーザのターンアラウンド・タイムに影響が出ることがあります。サーバへのアクセスが少ないときに抽

出ユーティリティを実行するか、データベースのコピーに対して抽出ユーティリティを実行することをおすすめします。

多数のデータベースを抽出する効率的な方法

上記の問題を回避する方法の 1 つは、次のとおりです。

1. 統合データベースのコピーを作成して、同時にライブ・データベースからサブスクリプションを開始します。サブスクライバへのメッセージの送信が開始されます。ただし、サブスクライバにはまだデータベースがなく、メッセージは受信されません。

   1 つのトランザクション内でいくつかのサブスクリプションを開始するには、REMOTE RESET 文を使用します。

2. データベースのコピーからリモート・データベースを抽出します。データベースはコピーであるため、ロックと同時性の問題は発生しません。データベースが多数ある場合は、この処理は数日かかることがあります。

3. リモート・データベースが作成されると、その情報は古いものになります。しかし、ユーザはライブ統合データベースから送信されたメッセージを受信して適用し、そのリモート・データベースを最新の情報に更新できます。

この解決法によって運用データベースに影響が出るのは、最初の手順においてのみです。データベースが使用中で多数のロックを使用している場合は、コピーを独立性レベル 3 で作成する必要があります。また、コピー作成と同時にサブスクリプションを開始する必要があります。コピー作成とサブスクリプション開始の間に発生したオペレーションはすべて失われ、これが原因でリモート・データベースでのエラーが発生する恐れがあります。

グループの抽出

リモート・ユーザがグループ・ユーザ ID である場合、抽出ユーティリティはグループのメンバのユーザ ID すべてを抽出します。異なるユーザ ID を使用して、カスタム抽出処理なしで、この機能を各リモート・データベースの複数のユーザすべてに使用できます。

ユーザ用にデータベースが抽出されるときには、ユーザおよびそのユーザがメンバであるグループの、すべてのメッセージ・リンク・パラメータが抽出されます。

抽出ユーティリティ使用の制限

統合データベースからリモート・データベースを作成して同期させる適切な方法は、抽出ユーティリティを使用することですが、場合によっては抽出ユーティリティを使用できず、リモート・データベースを手動で同期させる必要があります。この項では、そのような場合をいくつか説明します。

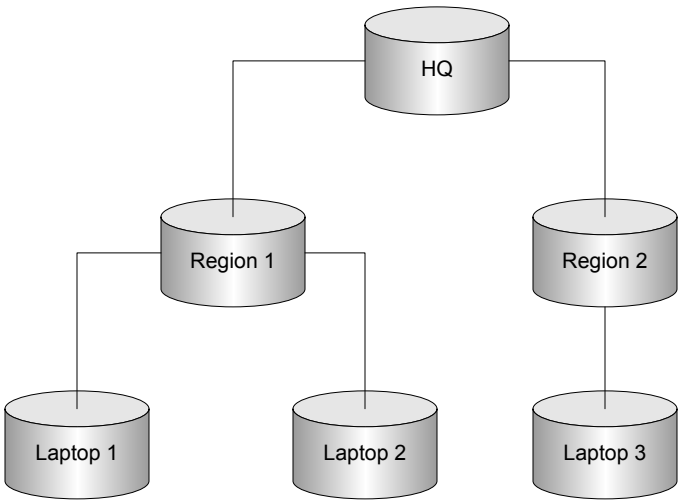
- **リモート・データベースの追加テーブル**　レプリケーションに関係しないテーブルであれば、統合データベースにないテーブルをリモート・データベースに追加できます。当然、抽出ユーティリティはこのようなテーブルを統合データベースから抽出できません。

- **プロシージャとビューの抽出**　デフォルトでは、抽出ユーティリティはすべてのストアド・プロシージャとビューをデータベースから抽出します。ビューとプロシージャには、リモート・サイトで必要なものと必要でないものがあります。必要でないビューとプロシージャは、データベースの、リモート・サイトに含まれない部分のみを参照するものです。

  抽出ユーティリティを実行後に再ロード・スクリプトを実行して、不必要なビューとプロシージャを削除してください。

- **多層の設定での抽出ユーティリティの使用**　多層配置での抽出ユーティリティの役割について理解するために、3 層の SQL Remote 設定について検討してみます。

  次の図は、この設定を表わしたものです。



最上位レベルの統合データベースから抽出ユーティリティを使用して、第 2 レベルのデータベースを作成できます。次に、第 2 レベルのデータベースにリモート・ユーザを追加して、第 2 レベルのデータベースから抽出ユーティリティを使用してリモート・データベースを作成します。ただし、最上位レベルの統合データベースから第 2 レベルのデータベースを再抽出する必要がある場合は、作成したリモート・ユーザをそのサブスクリプションとパーミッションとともに削除して、ユーザを再構築する必要があります。例外はデータのみを再同期させる場合で、抽出ユーティリティを使用して、データベースのスキーマを置き換えないで、データを置き換えることができます。

メッセージ・システムを介したデータの同期

サブスクリプションの作成

サブスクリプションの作成によって、受信されるデータが定義されます。サブスクリプションの同期 (データの最初のコピーの提供) または開始 (メッセージの交換) は行われません。

サブスクリプションの同期

サブスクリプションの同期により、Message Agent はサブスクリプション内のすべてのローのコピーをサブスクライバに送信します。ただし、適切なデータベース・スキーマが設定されていることが必要です。SYNCHRONIZE SUBSCRIPTION 文を使用して、サブスクリプションが同期されます。

同期メッセージがサブスクライバ・データベースで受信されると、Message Agent はデータベースの現在の内容を新しいコピーと置き換えます。サブスクリプションの一部を成し、統合データベースにレプリケートされなかったサブスクライバのデータは、失われます。同期が完了すると、Message Agent は START SUBSCRIPTION 文を使用して、サブスクリプションを開始します。

大量のメッセージの生成
メッセージ・システムを介してデータベースを同期させると、大量のメッセージが生成される可能性があります。多くの場合、メッセージ・システム内に大量のメッセージを保存せずに、抽出処理を使用してローカルにおいてデータベースを同期させることが望ましい方法です。

オペレーション中のサブスクリプションの同期

リモート・データベースと統合データベースの処理ステップが揃わなくなり、SQL Remote の SQL パススルー機能を使用して元に戻せない場合は、サブスクリプションを同期させると、リモート・データベースの内容を介してサブスクリプションのローが統合データベースからコピーされ、リモート・データベースと統合データベースの処理ステップが揃うようになります。

同期でのデータの消失
サブスクリプションの一部であっても、統合データベースにレプリケートされなかったリモート・データベースのデータは、サブスクリプションが同期されると消失してしまいます。データベースを同期させる前には、Sybase Central、または SQL Anywhere の場合は *dbunload* ユーティリティを使用して、リモート・データベースをアンロードまたはバックアップできます。

第 5 章

SQL Remote の管理

目次

管理の概要

この章では、稼働中の SQL Remote のインストール環境を管理する上での一般的な問題、および原則について説明します。

システム固有の問題の詳細については、「SQL Remote の管理」 127 ページを参照してください。

配備され稼働している SQL Remote 設定は、統合データベースで管理します。

- **パーミッション** SQL Remote のインストール環境にはさまざまな物理データベースが多数あるので、リモート・データベースと統合データベースのパーミッションを持っているユーザに対して一貫したスキームが必要になります。この章には、ユーザにパーミッションを割り当てるときに考慮すべき点を説明している項があります。
- **メッセージ・システムの設定** SQL Remote のインストール環境で使用される各メッセージ・システムでは、制御パラメータやその他の設定についてセット・アップします。この章ではこれらの設定について説明します。
- **Message Agent** Message Agent の役割は、メッセージの送信と受信です。
- **メッセージ・トラッキング** SQL Remote のインストール環境を管理することは、数多くのデータベース間でやりとりされる大量のメッセージを処理することを意味します。メッセージの内容、送信されるタイミング、適用方法などを理解するため、この章には SQL Remote のメッセージ・トラッキング・システムについて説明している項があります。
- **ログ管理** SQL Remote は、トランザクション・ログから送信するデータを取得します。したがって、SQL Remote のインストール環境をスムーズに稼働させるには、トランザクション・ログの適切な管理とバックアップが不可欠です。手順の詳細については稼働中のサーバによって異なりますが、一般的な問題はこの章で説明します。
- **パススルー・モード** 統合データベースからリモート・サイトに直接介入するための方法です。この方法については、この章で説明します。

SQL Remote パーミッションの管理

SQL Remote のレプリケーションに含まれるデータベースのユーザは、次のパーミッション・セットのいずれかで識別されます。

- **PUBLISH** データベース内にある 1 つのユーザ ID が、そのデータベースのパブリッシャとして識別されます。パブリケーションの更新と受信確認を含め、送信されるすべての SQL Remote メッセージは、パブリッシャ・ユーザ ID によって識別されます。SQL Remote 設定内のすべてのデータベースには、メッセージを送信するため、パブリッシャ・ユーザ ID が 1 つずつ必要です。
- **REMOTE** 現在のデータベースに対してメッセージを送受信するユーザのうち、SQL Remote の階層の中で現在のデータベースのすぐ下位に位置するユーザには、REMOTE パーミッションが付与されます。
- **CONSOLIDATE** 1 つのデータベース内に CONSOLIDATE パーミッションを付与されるユーザ ID は 1 つしかありません。CONSOLIDATE パーミッションは、SQL Remote 設定内の現在のデータベースの 1 段上位にあるデータベースを識別します。各データベースの 1 段上位には、統合データベースを 1 つしか配置できません。

これらのパーミッションについては SQL Remote のシステム・テーブルに定義されていて、他のデータベース・パーミッションからは独立しています。

PUBLISH パーミッションの付与と取り消し

データベースからメッセージが送信されると、そのデータベースを示すユーザ ID がメッセージに含まれるため、受信者側でメッセージの送信元を識別できます。このユーザ ID が、データベースの「**パブリッシャ・ユーザ ID**」です。1 つのデータベースに設定できるパブリッシャは 1 つだけです。SQL Anywhere データベースのパブリッシャとして割り当てられたユーザについては、Sybase Central で [ユーザとグループ] フォルダを開けばすぐにわかります。

レプリケーション・システム内にある読み込み専用のリモート・データベースにもパブリッシャは必要です。これは、レプリケーションのステータスについての情報を管理するため、これらのデータベースからも統合データベースに確認メッセージが送信されるためです。リモートの SQL Anywhere データベースの GRANT PUBLISH 文は、データベース抽出ユーティリティによって自動的に実行されます。

Sybase Central からの PUBLISH パーミッションの付与と取り消し

Sybase Central から、SQL Anywhere データベースの PUBLISH パーミッションを付与できます。それには、フル・システムまたはデータベース管理者のパーミッションを持つユーザとしてデータベースに接続する必要があります。

♦ 新規ユーザをパブリッシャとして作成するには、次の手順に従います (Sybase Central の場合)。

1. 左ウィンドウ枠で、[ユーザとグループ] フォルダを選択します。

2. [ファイル] メニューから [新規] – [ユーザ] を選択します。

   [ユーザの作成] ウィザードが表示されます。

3. ウィザードの指示に従います。そのユーザがパスワードを持ち、REMOTE DBA 権限が付与されていることを確認します。この権限によって、このユーザ ID で Message Agent を起動できます。

4. [完了] をクリックすると、ユーザが作成されます。

5. [ユーザとグループ] フォルダで、作成したユーザを右クリックしてポップアップ・メニューで [パブリッシャに変更] を選択します。

♦ **既存のユーザをパブリッシャにするには、次の手順に従います (Sybase Central の場合)。**

· [ユーザとグループ] フォルダで、ユーザを右クリックしてポップアップ・メニューで [パブリッシャに変更] を選択します。

Sybase Central から PUBLISH パーミッションを取り消すこともできます。

♦ **PUBLISH パーミッションを取り消すには、次の手順に従います (Sybase Central の場合)。**

· [ユーザとグループ] フォルダで PUBLISH パーミッションを付与されているユーザを右クリックし、ポップアップ・メニューで [パブリッシャの取り消し] を選択します。

PUBLISH パーミッションの付与と取り消し

SQL Anywhere では、次の GRANT PUBLISH 文を使用して PUBLISH パーミッションを付与します。

```
GRANT PUBLISH TO userid ;
```

userid は、現在のデータベースで CONNECT パーミッションを持つユーザです。たとえば、次の文では **S_Beaulieu** というユーザに PUBLISH パーミッションを付与します。

```
GRANT PUBLISH TO S_Beaulieu
```

次のように REVOKE PUBLISH 文を使用して、現在のパブリッシャの PUBLISH パーミッションを取り消します。

```
REVOKE PUBLISH FROM userid
```

PUBLISH パーミッションについての注意

♦ Sybase Central 以外でパブリッシャ・ユーザ ID を確認するには、特殊定数 CURRENT PUBLISHER を使用します。次の文で、「**パブリッシャ・ユーザ ID**」を取得します。

```
SELECT CURRENT PUBLISHER
```

♦ GROUP パーミッションを持つユーザ ID に PUBLISH パーミッションを付与しても、PUBLISH パーミッションはグループのメンバには継承されません。

♦ PUBLISH パーミッションには、送信メッセージのパブリッシャを識別する以外の権限はありません。

- 現在のデータベースから送信されたメッセージをユーザが受信して処理するには、パブリッシャ・ユーザ ID が受信するデータベースに対して REMOTE パーミッションまたは CONSOLIDATE パーミッションが必要です。
- データベースのパブリッシャ・ユーザ ID には、そのデータベースの REMOTE パーミッションまたは CONSOLIDATE パーミッションを付与できません。これにより、送信メッセージの送信者とメッセージの受信者を両方とも識別できます。
- リモート・データベースでパブリッシャのユーザ ID を変更すると、情報が失われるなど、そのデータベースを含むサブスクリプションのいずれかに深刻な問題が発生するおそれがあります。リモート・ユーザを最初から再同期するつもりでないかぎり、リモート・データベースのパブリッシャを絶対に変更しないでください。
- SQL Remote 設定の操作中に統合データベースのパブリッシャのユーザ ID を変更すると、データが失われるなどの深刻な問題が発生するおそれがあります。SQL Remote 設定を閉鎖してすべてのリモート・ユーザを再同期するつもりでないかぎり、統合データベースのパブリッシャのユーザ ID は絶対に変更しないでください。

REMOTE パーミッションと CONSOLIDATE パーミッションの付与と取り消し

REMOTE パーミッションと CONSOLIDATE パーミッションは非常に似ています。現在のデータベースからメッセージを受信するデータベースには、現在のデータベースに関連したユーザ ID が必要で、そのユーザ ID には REMOTE パーミッションまたは CONSOLIDATE パーミッションが付与されていなくてはなりません。パーミッションが付与されたユーザ ID は、そのデータベースが現在のデータベース内の受信データベースであることを示します。

統合ユーザはメッセージ・システムにより識別され、統合ユーザに対するメッセージの送受信方法が識別されます。統合ユーザは、各リモート・データベースに対して 1 人だけです。

リモート・ユーザはメッセージ・システムにより識別され、統合ユーザに対するメッセージの送受信方法が識別されます。

SQL Remote の階層内で、現在のデータベースの 1 段下位にあるデータベースには REMOTE パーミッションが付与され、上位にある多くても 1 つのデータベースには、CONSOLIDATE パーミッションが付与されます。

REMOTE パーミッションと CONSOLIDATE パーミッションの設定

GRANT REMOTE 文と GRANT CONSOLIDATE 文を使ってメッセージ・システムとレプリケーション・メッセージの送信先アドレスを特定します。

CONSOLIDATE パーミッションを読み込み専用リモート・データベースから統合データベースへも付与する必要があります。これは、リモート・データベースから統合データベースに受信確認メッセージが送信されるためです。リモートの SQL Anywhere データベースの GRANT CONSOLIDATE 文は、データベース抽出ユーティリティによって自動的に実行されます。

REMOTE パーミッションの付与

各リモート・データベースには、統合データベースにそれを表す単一のユーザ ID が必要です。ユーザ ID とアドレスがパブリケーションのサブスクライバとして識別されるために、ユーザ ID には REMOTE パーミッションを付与する必要があります。

REMOTE パーミッションを付与すると、次に挙げるいくつかのタスクが実行されます。

- ユーザ ID をリモート・ユーザとして識別する。
- このユーザ ID でメッセージを交換するときに使用するメッセージ・タイプを指定する。
- メッセージの送信先アドレスを指定する。
- メッセージがリモート・ユーザに送信される頻度を指示する。

REMOTE パーミッションの付与は、データベースへのリモート・ユーザの追加にも関連します。

Sybase Central の場合

Sybase Central を使用して、データベースにリモート・ユーザを追加することができます。リモート・ユーザとグループは、Sybase Central の [ユーザとグループ] フォルダと [SQL Remote ユーザ] フォルダの 2 ヵ所に表示されます。この項は、SQL Anywhere データベースだけに適用されます。

デフォルトでは、リモート・ユーザは作成時から REMOTE DBA 権限を持っています。リモート・データベースにアクセスするための Message Agent でこの権限が必要なので取り消さないでください。

データベース内で最低 1 つのメッセージ・タイプが定義されるまで、新規リモート・ユーザを作成することはできません。

グループには REMOTE パーミッションを付与できますが、テーブル・パーミッションなどとは異なり、REMOTE パーミッションがグループ内のユーザに自動的に適用されることはあり*ません*。グループ内のユーザにパーミッションを適用するには、各ユーザに REMOTE パーミッションを明示的に付与する必要があります。この操作を行わないと、リモート・グループはリモート・ユーザとまったく同じような動作をし、リモート・ユーザとして分類されます。

♦ 新規ユーザをリモート・ユーザとしてデータベースに追加するには、次の手順に従います (Sybase Central の場合)。

1. 左ウィンドウ枠で、[SQL Remote ユーザ] フォルダを選択します。
2. [ファイル] メニューから [新規] – [SQL Remote ユーザ] を選択します。

   [新しいリモート・ユーザの作成] ウィザードが表示されます。
3. ウィザードの指示に従います。

♦ 既存のユーザをリモート・ユーザまたはリモート・グループにするには、次の手順に従います (Sybase Central の場合)。

1. [ユーザとグループ] フォルダを開きます。

2. リモートに設定するユーザを右クリックして、ポップアップ・メニューで [リモート・ユーザに変更] を選択します。
3. 表示されるダイアログ・ボックスでリストからメッセージ・タイプを選択した後、アドレスを入力してメッセージを送信する頻度を選択し、[OK] をクリックします。すると、そのユーザがリモート・ユーザとして設定されます。

   このユーザが、[ユーザとグループ] フォルダと [SQL Remote ユーザ] フォルダの両方に表示されます。

例

次の文では、**S_Beaulieu** というユーザに、以下のオプションを持つ REMOTE パーミッションを付与します。

- SMTP 電子メール・システムの使用
- 電子メール・アドレス **s_beaulieu@acme.com** へのメッセージの送信
- 毎日午後 10 時にメッセージを送信

```
GRANT REMOTE TO S_Beaulieu
TYPE smtp
ADDRESS 's_beaulieu@acme.com'
SEND AT '22:00'
```

送信頻度の選択

メッセージを送信する頻度の設定には、選択肢が 3 つあります。選択肢は次のとおりです。

- **SEND EVERY**　頻度を時、分、秒 (HH:MM:SS フォーマット) で指定できます。

  SEND EVERY を設定したユーザにメッセージを送信すると、同じ頻度が設定されているすべてのユーザにもメッセージが送信されます。たとえば、12 時間ごとに更新内容を受信するリモート・ユーザ全員に、時間をずらすことなく同時に更新内容が送信されます。これにより、SQL Anywhere のトランザクション・ログを処理する回数を減らすことができます。あるユーザに固有な頻度を設定することは、できるだけ避けてください。

- **SEND AT**　1 日の特定の時刻 (時間と分)。

  毎日、指定した時刻に更新します。送信時刻をずらすよりも効率的に、できるだけ少ない回数で更新を行うことができます。また、データベースがビジーでない時刻を選択すると、他のユーザとの干渉を最小限にすることができます。

- **デフォルト設定 (SEND 句なし)**　SEND AT 句または SEND EVERY 句が設定されていないユーザには、Message Agent が起動されるたびにメッセージが送信され停止します。Message Agent はバッチ・モードで起動します。

Sybase Central での送信頻度の設定

Sybase Central では、次の方法で送信頻度を設定できます。

- 既存のユーザまたはグループをリモートに設定するときに指定する。「REMOTE パーミッションの付与」 92 ページを参照してください。
- リモート・ユーザまたはグループのプロパティ・シートの [SQL Remote] タブで指定する。プロパティ・シートは、リモート・ユーザまたはグループを右クリックして、ポップアップ・メニューで [プロパティ] を選択すると開きます。

CONSOLIDATE パーミッションの付与

リモート・データベースでは、パブリッシュ・ユーザ ID とサブスクライブ・ユーザ ID が統合データベースの場合と逆になります。統合データベースのサブスクライバ (リモート・ユーザ) は、リモート・データベースではパブリッシャになります。統合データベースのパブリッシャは、リモート・データベースからのパブリケーションのサブスクライバになり、CONSOLIDATE パーミッションが付与されます。

各リモート・データベースでは、統合データベースに CONSOLIDATE パーミッションを付与しなくてはなりません。データベース抽出ユーティリティを起動してリモート・データベースを生成すると、GRANT CONSOLIDATE 文がリモート・データベースで自動的に実行されます。

例

SQL Anywhere では、VIM 電子メール・システムを使用して、次の文でユーザ ID **hq_user** に CONSOLIDATE パーミッションを付与します。

```
GRANT CONSOLIDATE TO hq_user
TYPE vim
ADDRESS 'hq_address'
```

この文には SEND 句がないためデフォルト設定が適用され、Message Agent が起動されるたびにメッセージが統合データベースに送信されます。

REMOTE パーミッションと CONSOLIDATE パーミッションの取り消し

ユーザの REMOTE パーミッションを取り消すと、SQL Remote のインストール環境からそのユーザを削除できます。ユーザまたはグループから REMOTE パーミッションを取り消すと、通常のユーザまたはグループに戻ります。また、それらのサブスクリプションもすべてのパブリケーションから自動的に削除できます。

Sybase Central からのパーミッションの取り消し

Sybase Central から、SQL Anywhere データベースの REMOTE パーミッションを取り消せます。

- **REMOTE パーミッションを取り消すには、次の手順に従います (Sybase Central の場合)。**

1. [ユーザとグループ] フォルダまたは [SQL Remote ユーザ] フォルダを開きます。
2. リモート・ユーザまたはグループを右クリックし、ポップアップ・メニューで [リモートの取り消し] を選択します。

パーミッションの取り消し

REVOKE 文を使用して、ユーザから REMOTE パーミッションと CONSOLIDATE パーミッションを取り消すことができます。ユーザ **S_Beaulieu** から REMOTE パーミッションを取り消す文は次のとおりです。

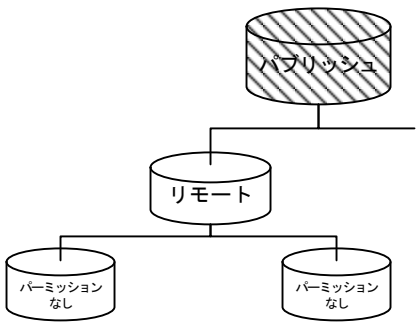
```
REVOKE REMOTE FROM S_Beaulieu
```

REMOTE アクセス権または CONSOLIDATE アクセス権を取り消すには、DBA 権限が必要です。

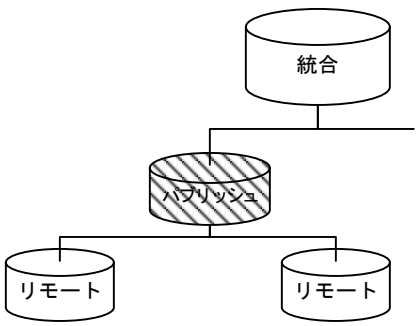
多層インストール環境でのパーミッションの割り当て

多層インストール環境でパーミッションを割り当てるには、特別に考慮すべきことがあります。次の各図は、3 層の SQL Remote 設定でのパーミッションの要約を示しています。各図のデータベースの 1 つにグレーの影を付けてあります。その他の各データベースには、そのグレーのデータベースで生成する必要がある、ユーザ ID に対するパーミッションを図示しています。「パーミッションなし」は、グレーのデータベースではそのデータベースに対し何のパーミッションも付与しないことを意味します。

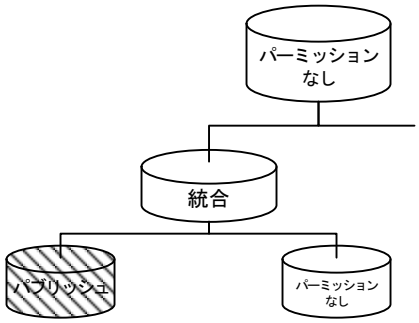
次の図は、3 層インストール環境の統合サイトで付与されている SQL Remote パーミッションを示しています。



次の図は、3 層インストール環境の内部サイトで付与されている SQL Remote パーミッションを示しています。



次の図は、3 層インストール環境の内部サイトで付与されている SQL Remote パーミッションを示しています。



リモート・データベースでの適切な PUBLISH パーミッションと CONSOLIDATE パーミッションの付与は、データベース抽出ユーティリティが自動的に行います。

メッセージ・タイプの使用

SQL Remote は、基本となるメッセージ・システムを使用して、データベース間のデータ交換を行います。SQL Remote では、次のメッセージ・システムをサポートしています。

- **ファイル共有**　他のソフトウェアを必要としない簡単なシステム
- **FTP**　インターネット・ファイル転送プロトコル
- **SMTP/POP**　インターネット電子メール転送プロトコル
- **MAPI**　Microsoft Messaging Application Programming Interface。Microsoft 製品と cc:Mail リリース 8 以降で使用されています。
- **VIM**　Vendor Independent Messaging。Lotus Notes と Lotus cc:Mail の一部のバージョンで使用されています。

> **注意**
> VIM および MAPI のサポートは廃止されました。

データベースでは、使用可能なメッセージ・システムを 1 つまたは複数使用してメッセージを交換できます。

すべてのシステムがすべてのオペレーティング・システムでサポートされるわけではありません。サポートされているターゲット・オペレーティング・システムの一覧については、「プラットフォーム別 SQL Anywhere 10.0.1 コンポーネント」にある SQL Remote for SQL Anywhere の表を参照してください。

オペレーティング・システムによって使用可能なメッセージ・システム

SQL Remote が使用可能なすべてのオペレーティング・システムが、すべてのメッセージ・システムをサポートしているわけではありません。Windows オペレーティング・システムでは、リンクは DLL として実装されています。

各オペレーティング・システムが対応しているメッセージ・システムの一覧については、「プラットフォーム別 SQL Anywhere 10.0.1 コンポーネント」にある SQL Remote for SQL Anywhere の表を参照してください。

参照

- 「FILE メッセージ・システム」 102 ページ
- 「FTP メッセージ・システム」 103 ページ
- 「SMTP メッセージ・システム」 105 ページ
- 「MAPI メッセージ・システム」 107 ページ
- 「VIM メッセージ・システム」 109 ページ

メッセージ・タイプの処理

各メッセージ・タイプの定義には、タイプ名 (**file**、**ftp**、**smtp**、**mapi**、**vim**) とそのメッセージ・タイプにおけるパブリッシャのアドレスも含まれます。リモート・データベースの作成時にデータベース抽出ユーティリティが、統合データベースでのパブリッシャ・アドレスを返信アドレスとして使用します。また Message Agent でも、**file** システムの受信メッセージを探す場所を識別するためにパブリッシャ・アドレスを使用します。

メッセージ・タイプの定義に入力されるアドレスには、データベースのパブリッシャ ID と密接なつながりがあります。有効なアドレスについては以降の項で説明します。

メッセージ・システムを使用する前に、必ずパブリッシャのアドレスを設定してください。

注意
VIM および MAPI のサポートは廃止されました。

Sybase Central を使用したメッセージ・タイプの処理

Sybase Central では、メッセージ・タイプを作成したり変更したりできます。メッセージ・タイプは、[SQL Remote ユーザ] フォルダが選択されているときに、右ウィンドウ枠の [メッセージ・タイプ] タブに表示されます。この項は、SQL Anywhere データベースだけに適用されます。

メッセージ・タイプの作成および変更には、DBA 権限が必要です。

♦ **メッセージ・タイプを追加するには、次の手順に従います (Sybase Central の場合)。**

1. データベースに接続します。
2. 左ウィンドウ枠で、そのデータベースの [SQL Remote ユーザ] フォルダを開きます。
3. 右ウィンドウ枠の [メッセージ・タイプ] タブをクリックします。
4. [ファイル] メニューから [新規] – [メッセージ・タイプ] を選択します。

   [SQL Remote メッセージ・タイプ作成] ウィザードが表示されます。
5. [SQL Remote メッセージ・タイプ作成] ウィザードで、メッセージ・タイプ名を入力します。入力する名前は、お使いの SQL Anywhere インストール・ディレクトリにすでにインストールされているメッセージ・タイプ DLL と一致させてください。[次へ] をクリックします。
6. パブリッシャ・アドレスを入力し、[完了] をクリックしてデータベースに定義を保存します。

パブリッシャのアドレスを変更するには、メッセージ・タイプを変更します。既存のメッセージ・タイプの名前を変更することはできないので、メッセージ・タイプを削除してから、新しい名前で新しいメッセージ・タイプを作成してください。

♦ **メッセージ・タイプを変更するには、次の手順に従います (Sybase Central の場合)。**

1. 左ウィンドウ枠で、データベースの [SQL Remote ユーザ] フォルダを開きます。
2. 右ウィンドウ枠の [メッセージ・タイプ] タブをクリックします。
3. 右ウィンドウ枠で、変更するメッセージ・タイプを右クリックし、ポップアップ・メニューで [プロパティ] を選択します。
4. プロパティ・シートで、さまざまなオプションを設定します。

インストール環境からメッセージ・タイプを削除することもできます。

♦ **メッセージ・タイプを削除するには、次の手順に従います (Sybase Central の場合)。**

1. 左ウィンドウ枠で、データベースの [SQL Remote ユーザ] フォルダを開きます。
2. 右ウィンドウ枠の [メッセージ・タイプ] タブをクリックします。
3. 右ウィンドウ枠で、変更するメッセージ・タイプを右クリックし、ポップアップ・メニューで [削除] を選択します。

Windows CE で使用するメッセージ・タイプの作成

Windows CE サービスがインストールされている場合は、Sybase Central から SQL Remote を ActiveSync 同期用に設定するオプションを選択できます。これにより、FILE メッセージ・リンクのメッセージのフォルダが、ActiveSync フォルダとして設定されます。Windows CE デバイスをデスクトップ・コンピュータにドッキングすると、ActiveSync は、デスクトップ・コンピュータの [ActiveSync] フォルダにあるファイルと、Windows CE の [ActiveSync] フォルダにあるファイルの同期をとります。[ツール] – [SQL Anywhere 10] – [Windows CE メッセージ・タイプの編集] を選択して、ユーティリティにアクセスし、SQL R e mote の ActiveSync 同期を設定できます。

コマンドを使用したメッセージ・タイプの処理

♦ **メッセージ・タイプを作成するには、次の手順に従います (SQL の場合)。**

1. 作成するメッセージ・タイプでのパブリッシャのアドレスが決定されていることを確認します。
2. CREATE REMOTE MESSAGE TYPE コマンドを実行します。

   CREATE REMOTE MESSAGE TYPE 文の構文は、次のとおりです。

   CREATE REMOTE MESSAGE TYPE *type-name*
   ADDRESS *address-string*

   type-name は SQL Remote が対応しているメッセージ・システムの 1 つで、*address-string* はそのメッセージ・システムでのパブリッシャのアドレスです。

パブリッシャのアドレスを変更するには、メッセージ・タイプを変更します。

♦ **メッセージ・タイプを変更するには、次の手順に従います (SQL の場合)。**

1. 変更するメッセージ・タイプでのパブリッシャの新しいアドレスが決定されていることを確認します。
2. ALTER REMOTE MESSAGE TYPE 文を実行します。

   ALTER REMOTE MESSAGE TYPE 文の構文は、次のとおりです。

   ALTER REMOTE MESSAGE TYPE *type-name*
   ADDRESS *address-string*

   type-name は SQL Remote がサポートしているメッセージ・システムの 1 つで、*address-string* はそのメッセージ・システムでのパブリッシャのアドレスです。

使用中のインストール環境ですでに使用されなくなったメッセージ・タイプを削除することもできます。メッセージ・タイプを削除すると、その定義からパブリッシャのアドレスが削除されます。

♦ **メッセージ・タイプを削除するには、次の手順に従います (SQL の場合)。**

· DROP REMOTE MESSAGE TYPE 文を実行します。

  DROP REMOTE MESSAGE TYPE 文の構文は、次のとおりです。

  DROP REMOTE MESSAGE TYPE *type-name*

  type-name は SQL Remote がサポートしているメッセージ・システムの 1 つです。

参照

- 「CREATE REMOTE MESSAGE TYPE 文 [SQL Remote]」『SQL Anywhere サーバ - SQL リファレンス』
- 「ALTER REMOTE MESSAGE TYPE 文 [SQL Remote]」『SQL Anywhere サーバ - SQL リファレンス』
- 「DROP REMOTE MESSAGE TYPE 文 [SQL Remote]」『SQL Anywhere サーバ - SQL リファレンス』

メッセージ・タイプ制御パラメータの設定

メッセージ・リンクごとにその動作を管理するパラメータがいくつかあります。パラメータはメッセージ・システムによって異なりますが、管理方法はすべて同じです。

特定のメッセージ・リンクで Message Agent を初めて使用する場合、リンクの動作を制御するパラメータを設定するダイアログ・ボックスが表示されます。これらのパラメータには、メッセージ・システムのユーザ ID、ftp メッセージが保管されるホスト名などがあります。入力したパラメータは、Message Agent によって保存されます。これらのパラメータを明示的に設定することもできます。

メッセージ・リンク・パラメータのデータベースへの保存

メッセージ制御パラメータは、データベース内に保存されます。オプションの設定方法は次のとおりです。

- **メッセージ制御パラメータを設定するには、次の手順に従います。**

・ 次の文を実行します。

SET REMOTE *link-name* **OPTION**
[*username*.]*option-name* = *option-value*

sys.sysremoteoptions ビューのクエリを実行すると、現在のメッセージ・リンク・パラメータが表示されます。

メッセージ・リンク・パラメータのディスクへの保存

このソフトウェアの古いバージョンでは、メッセージ・リンク・パラメータはデータベースの外に保存されていました。今でもこの方法を使用できますが、特別な理由があるとき以外は、データベース内にパラメータを保管することをおすすめします。

メッセージ・リンク制御パラメータは、次の場所に格納されます。

- **Windows** レジストリ内の次の場所に格納されます。

```
¥¥HKEY_CURRENT_USER
  ¥Software
    ¥Sybase
      ¥SQL Remote
```

各メッセージ・リンクのパラメータは、メッセージ・リンクの名前 (4、*smtp* など) を付けられて、SQL Remote キーの下のキーに入ります。

- **NetWare** FILE システム・ディレクトリ設定を保存するために、*sys:¥system* ディレクトリにファイル *dbremote.ini* を作成してください。このファイルは Windows 形式の INI ファイルではないので、ディレクトリ名のみが入った単一行にします。

たとえば、ディレクトリが *user:¥dbr43* の場合、*dbremote.ini* ファイルには次の行が入ります。

```
user:¥dbr43
```

- **UNIX** FILE システム・ディレクトリ設定は、SQLREMOTE 環境変数に保存されます。

sqlremote 環境変数には、ファイル共有システムの制御パラメータのうち、その代替の 1 つとして使用されるパスが保存されます。

各メッセージ・システムで使用できるパラメータについては、以降の項で説明します。各項ごとに 1 つのメッセージ・システムについて説明します。

Message Agent がメッセージ・リンクをロードすると、リンクは現在のパブリッシャの設定を使用します。設定が指定されていない場合は、そのパブリッシャが属するグループの設定を使用します。Windows では、メッセージ・リンク・パラメータのデータベースへの保存をサポートするバージョンの Message Agent を初めて起動すると、リンクのオプションがレジストリからデータベースにコピーされます。

FILE メッセージ・システム

file メッセージ・システムを使用すれば、メッセージ・システムが適切な場所になくても SQL Remote を使用できます。

FILE メッセージ・システムのアドレス

file メッセージ・システムは、単純なファイル共有システムです。リモート・ユーザの **file** アドレスは、すべてのメッセージが書き込まれるサブディレクトリです。「受信ボックス」からメッセージを取得するには、アプリケーションがユーザのファイルを格納しているディレクトリからメッセージを読み込みます。返信メッセージは、統合データベースのアドレスに送信 (ディレクトリに書き込み) されます。

NT サービスとして起動中の場合は、Message Agent が稼働しているアカウントに、必要なすべてのディレクトリに対する読み込みと書き込みのパーミッションがあることを確認してください。ネットワーク・ドライブにアクセスするときに、パーミッションの所持がよく問題になります。

アドレスのルート・ディレクトリ

file システムのアドレスは、一般的に共有ディレクトリのサブディレクトリです。このサブディレクトリは、モデムまたはローカル・エリア・ネットワークからアクセスする SQL Remote ユーザ全員が使用できます。各ユーザには、レジストリのエントリ、初期化ファイルのエントリ、または共有ディレクトリを示す SQLREMOTE 環境変数が必要です。

file システムを使用して、統合マシンまたはリモート・マシンのディレクトリにメッセージを入れることもできます。これにより、簡易ファイル転送メカニズムを使用してファイルを定期的に交換し、レプリケーションを実行できます。

FILE メッセージ制御パラメータ

FILE メッセージ・システムでは、次の制御パラメータを使用します。

- **Directory**　メッセージが保存されるディレクトリを設定します。この設定は、SQLREMOTE 環境変数の代替となります。
- **Debug**　これは YES か NO に設定されます (デフォルト設定では NO)。設定が YES の場合、FILE リンクによるすべてのファイル・システム呼び出しが表示されます。
- **Encode_dll**　カスタム・エンコード・スキームを実装している場合は、作成したカスタム・エンコード DLL のフル・パスをこの値に設定する必要があります。

  「メッセージのエンコードと圧縮」 121 ページを参照してください。
- **invalid_extensions**　メッセージング・システムでのファイルの生成時に dbremote で使用されないようにするファイル拡張子の、カンマで区切られたリストです。
- **Unlink_delay**　前回のファイル削除試行が失敗した場合、この秒数だけ待ってから次の削除を試行します。unlink_delay に対して値が定義されていない場合、デフォルト動作では、最初の試行失敗の後に 1 秒間、2 回目の試行失敗の後に 2 秒間、3 回目の試行失敗の後に 3 秒間、4 回目の試行失敗の後に 4 秒間一時停止します。

NetWare では、ディレクトリ設定を保存するために、*sys:¥system* ディレクトリにファイル *dbremote.ini* を作成してください。

Windows CE と ActiveSync

ファイル・リンクの場合、*dbremote* は *¥My Documents¥Synchronized Files* を調べます。デスクトップ・マシン上では、FILE リンク用の SQLREMOTE 環境変数または directory メッセージ・リンク・パラメータを次のように設定してください。

```
%SystemRoot%¥Profiles¥userid¥Personal¥ce-machine-name¥ Synchronized Files
```

ここで、*userid* と *ce-machine-name* に適切な値を設定してください。ActiveSync はこの設定を使用して、デスクトップと CE システム間でメッセージ・ファイルを自動的に同期します。

[モバイル・デバイス] – [ツール] – [ActiveSync] オプションをチェックして、ファイルの同期がアクティブ化されていることを確認します。

メッセージ・リンク・パラメータの設定方法については、「FILE メッセージ・システム」 102 ページを参照してください。

参照

- 「FTP メッセージ・システム」 103 ページ

FTP メッセージ・システム

FTP のアドレス

FTP メッセージ・システムでは、メッセージは FTP ホストのルート・ディレクトリの下位ディレクトリに保存されます。FTP ホストとルート・ディレクトリは、レジストリまたは初期化ファイルに保存されているメッセージ・システム制御パラメータによって指定されます。またメッセージが保存されるサブディレクトリが各ユーザのアドレスになります。

FTP をサポートしているオペレーティング・システムの一覧については、「プラットフォーム別 SQL Anywhere 10.0.1 コンポーネント」にある SQL Remote for SQL Anywhere の表を参照してください。

FTP メッセージ制御パラメータ

FTP メッセージ・システムでは、次の制御パラメータを使用します。

- **encode_dll**　カスタム・エンコード・スキームを実装している場合は、作成したカスタム・エンコード DLL のフル・パスをこの値に設定する必要があります。

  「メッセージのエンコードと圧縮」 121 ページを参照してください。

- **host**　メッセージが保存されるコンピュータのホスト名。ホスト名 (**ftp.ianywhere.com** など) または IP アドレス (192.138.151.66 など) を使用できます。
- **user**　ftp ホストにアクセスするためのユーザ名。
- **password**　ftp ホストにアクセスするためのパスワード。

- **root_directory**　メッセージが保存される、ftp ホスト・サイトのルート・ディレクトリ。
- **port**　通常は必要ありません。これは、FTP 接続に使用する IP ポート番号です。
- **debug**　YES または NO に設定されます。デフォルトでは NO です。YES を設定すると、デバッグ出力が表示されます。
- **active_mode**　YES または NO に設定されます。デフォルトは NO (受動モード) です。
- **reconnect_retries**　失敗になる前に、リンクがサーバでソケットを開こうと試行する回数。デフォルト値は 4 です。このパラメータの設定は、再接続のみに影響します。FTP リンクによる最初の接続には影響しません。
- **reconnect_pause**　接続の試行失敗後に次の接続まで待機する秒数。デフォルトは 30 秒です。このパラメータの設定は、再接続のみに影響します。FTP リンクによる最初の接続には影響しません。
- **suppress_dialogs**　true に設定されている場合は、FTP サーバへの接続の試行失敗後に、[接続] ダイアログは表示されません。代わりに、エラーが発生します。
- **invalid_extensions**　メッセージング・システムでのファイルの生成時に dbremote で使用されないようにするファイル拡張子の、カンマで区切られたリストです。

FTP 問題のトラブルシューティング

FTP メッセージ・リンクにおける問題のほとんどは、ネットワーク設定の問題です。この項では、問題に対処するためのテストについて説明します。

DEBUG メッセージ制御パラメータの設定　デバッグ出力を調査すると、FTP サーバに接続しているかどうかが分かります。接続している場合は、どの FTP コマンドに問題があるかが示されます。

FTP サーバの ping　FTP リンクが FTP サーバに接続できない場合は、システム・ネットワーク設定をテストしてみてください。システムに **ping** コマンドがある場合は、次のコマンドを入力してみます。

```
ping ftp-server-name
```

サーバの IP アドレスと、サーバへの ping (往復) 時間を示している出力を確認してください。サーバの ping が実行できない場合は、ネットワーク設定に問題があります。ネットワーク管理者に問い合わせてください。

受動モードの動作確認　FTP リンクが FTP サーバに接続しているのに、データ接続を開けない場合は、FTP クライアントが受動モードを使用してサーバからデータを転送できることを確認します。

受動モードは推奨の転送モードで、FTP メッセージ・リンクにデフォルトとして設定されています。受動モードでは、すべてのデータ転送接続はクライアント (この場合はメッセージ・リンク) から開始されます。アクティブ・モードでは、サーバがすべてのデータ接続を開始します。FTP サーバが正しく設定されていないファイアウォールの保護を受けていると、デフォルトの受動転送モードを使用できない場合があります。この場合、ファイアウォールは FTP 制御ポート以外のポート上の FTP サーバへのソケット接続をブロックします。

転送モードを **active** か **passive** に設定できる FTP ユーザ・プログラムを使用する場合は、転送モードを受動に設定して、ファイルのアップロードやダウンロードを行ってください。使用しているクライアントが、アクティブ・モードを使用しないとファイルを転送できない場合は、受動モードでの転送ができるようにファイアウォールと FTP サーバを設定し直すか、active_mode メッセージ制御パラメータに YES を設定してください。すべてのネットワーク設定でアクティブ・モード転送が動作するとはかぎりません。次に例を示します。クライアントが IP マスカレードが機能しているゲートウェイの後方に位置している場合、ゲートウェイ・ソフトウェアによっては受信接続で障害が発生する可能性があります。

パーミッションとディレクトリ構造の確認　FTP サーバが接続中で、ディレクトリのリストの取得やファイルの操作に問題がある場合は、パーミッションが正しく設定されていることと、必要なディレクトリが存在していることを確認します。

FTP プログラムを使用して FTP サーバにログインします。ディレクトリを、root_directory パラメータに保存されている場所に変更します。必要なディレクトリが表示されない場合は、root_directory 制御パラメータが間違っているか、ディレクトリが存在しない可能性があります。

メッセージ・ディレクトリのファイルを取り出してパーミッションをテストし、統合データベース・ディレクトリにファイルをアップロードします。FTP サーバでエラーが発生する場合は、パーミッションが正しく設定されていません。

SMTP メッセージ・システム

メール転送プロトコル (SMTP) は、インターネット電子メール製品で使用されています。

SQL Remote では、SMTP システムを使用してインターネット・メールでメッセージを送信します。メッセージはテキスト形式にエンコードされ、電子メール・メッセージとしてターゲット・データベースに送信されます。メッセージは SMTP サーバを介して送信され、POP サーバから受信されます。これは、多くの電子メール・プログラムがメッセージの送受信に使用する方法です。

SMTP をサポートしているオペレーティング・システムの一覧については、「プラットフォーム別 SQL Anywhere 10.0.1 コンポーネント」にある SQL Remote for SQL Anywhere の表を参照してください。

SMTP アドレスとユーザ ID

SQL Remote と SMTP メッセージ・システムを使用するには、設定に関係する各データベースで、SMTP アドレス、POP3 ユーザ ID、パスワードが必要です。これらは別々の識別子です。SMTP アドレスは各メッセージの送信先で、POP3 ユーザ ID とパスワードは、ユーザが自分のメールボックスに接続するときに入力する名前とパスワードです。

個別の電子メール・アカウントを推奨
SQL Remote メッセージには、POP 電子メール・アカウントを個別に設定することをおすすめします。

トラブルシューティング

SMTP リンクが動作しない場合は、Message Agent が稼働している同じマシンから、同じアカウントとパスワードを使用して SMTP/POP3 サーバに接続してみます。SMTP/POP3 をサポートするインターネット電子メール・プログラムを使用して、SMTP メッセージ・リンクが動作することがわかったら、必ずそのプログラムを無効にしてください。

SMTP メッセージ制御パラメータ

Message Agent がメッセージ・システムに接続してメッセージを送受信する前に、ユーザは制御パラメータのセットを自分のコンピュータにあらかじめ設定しておくか、ウィンドウで必要な情報を入力しなくてはなりません。この情報が必要となるのは、初回接続時のみです。この情報は保存され、以後の接続でデフォルトのエントリとして使用されます。

SMTP メッセージ・システムでは、次の制御パラメータを使用します。

- **encode_dll**　カスタム・エンコード・スキームを実装している場合は、作成したカスタム・エンコード DLL のフル・パスをこの値に設定する必要があります。

  「メッセージのエンコードと圧縮」 121 ページを参照してください。

- **local_host**　ローカル・コンピュータの名前。SQL Remote がローカル・ホスト名を決定できないコンピュータで設定すると便利です。ローカル・ホスト名は、任意の SMTP サーバとのセッションを開始するのに必要です。ほとんどのネットワーク環境では、ローカル・ホスト名が自動的に決定されるため、このエントリは不要です。

- **TOP_supported**　受信メッセージを列挙するときに、SQL Remote は TOP という POP3 コマンドを使用します。TOP コマンドは、すべての POP サーバでサポートされているわけではありません。このエントリを NO に設定すると、RETR コマンドを使用します。このコマンドは TOP よりも効率は落ちますが、すべての POP サーバで動作します。デフォルトは YES です。

- **smtp_authenticate**　SMTP リンクがユーザを認証するかどうかを決定します。デフォルト値は YES です。SMTP 認証を実行しない場合は、NO に設定します。

- **smtp_userid**　SMTP 認証のユーザ ID。デフォルトでは、このパラメータは **pop3_userid** パラメータと同じ値を取ります。**smtp_userid** は、ユーザ ID が POP サーバ上のユーザ ID と異なる場合にのみ設定する必要があります。

- **smtp_password**　SMTP 認証のパスワード。デフォルトでは、このパラメータは **pop3_password** パラメータと同じ値を取ります。**smtp_password** は、ユーザ ID が POP サーバ上のユーザ ID と異なる場合にのみ設定する必要があります。

- **smtp_host**　SMTP サーバが稼働しているコンピュータの名前。SMTP/POP3 ログイン・ダイアログの SMTP ホスト・フィールドに対応しています。

- **pop3_host**　POP ホストが稼働しているコンピュータの名前。一般的には SMTP ホストと同じです。SMTP/POP3 ログイン・ダイアログの POP3 ホスト・フィールドに対応しています。

- **pop3_userid**　メールの受信に使用します。POP ユーザ ID は、SMTP/POP3 ログイン・ダイアログのユーザ ID フィールドに対応しています。必ず POP ホスト管理者からユーザ ID を取得してください。
- **pop3_password**　メールの受信に使用します。SMTP/POP3 ログイン・ダイアログのパスワード・フィールドに対応しています。これら 5 つのフィールドがすべて設定されると、ログイン・ダイアログは表示されなくなります。
- **Debug**　YES に設定すると、SMTP と POP3 のすべてのコマンドと応答が表示されます。SMTP/POP のサポート問題のトラブルシューティングに便利です。デフォルトは NO です。
- **Suppress_dialogs**　true に設定されている場合は、メール・サーバへの接続の試行失敗後に、[接続] ダイアログは表示されません。代わりに、エラーが発生します。

SMTP/POP アドレスの共有

各データベースには、SQL Remote メッセージ用の電子メール・アカウントが必要です。このアカウントは、個人的な電子メール・メッセージを読み込むものとは区別します。これは、多くの電子メール読者が次の方法で電子メールを取得するためです。

1. POP ホストに接続してすべてのメッセージをダウンロードする。
2. POP ホストからメッセージをすべて削除する。
3. POP ホストとの接続を切断する。
4. ローカル・ファイルまたはメモリからメールを読む。

これは、電子メール・プログラムがすべての SQL Remote の電子メール・メッセージを個人のメッセージと同様にダウンロードしたり削除したりするため、問題が発生します。お使いの電子メール・プログラムが未読メッセージを POP ホストから削除しないことが確実な場合は、データベースのメッセージを削除したり変更したりしないかぎり、電子メール・アドレスをデータベースと共有できます。

SQL Remote メッセージはランダムなテキストの行から構成されているように見えるため、簡単に識別できます。

MAPI メッセージ・システム

> **注意**
> MAPI のサポートは廃止されました。

Message Application Programming Interface (MAPI) は、Microsoft Mail や最新の Lotus cc:Mail など、いくつかの有名な電子メール・システムで使用されています。

MAPI をサポートしているオペレーティング・システムの一覧については、「プラットフォーム別 SQL Anywhere 10.0.1 コンポーネント」にある SQL Remote for SQL Anywhere の表を参照してください。

MAPI アドレスとユーザ ID

SQL Remote と MAPI メッセージ・システムを使用するには、その設定に関連する各データベースに MAPI ユーザ ID とアドレスが必要です。これらは別々の識別子です。MAPI アドレスは各メッセージの送信先で、MAPI ユーザ ID は、ユーザが自分のメールボックスに接続するために入力する名前とパスワードです。

MAPI メッセージと電子メール受信ボックス

SQL Remote のメッセージは読み込み用電子メールと同じメールボックスで受信しますが、一般的に電子メール受信ボックスには表示されません。

SQL Remote はアプリケーションで定義されたメッセージを送信しようとしますが、メールボックスを開くと、そうしたアプリケーション定義のメッセージを MAPI が識別して非表示にします。この方法では、ユーザは自分の個人的な電子メールとデータベースのアップデートを同じ電子メール・アドレスと同じ接続で受信することになりますが、SQL Remote のメッセージが読み込み用メールを干渉することはありません。

ただし、インターネット経由でメッセージをルーティングする場合、またはオペレーティング・システムに特定のパッチが適用されている場合は、特殊なメッセージ・タイプ情報が失われる可能性があります。そのようなメッセージは、受信者のメールボックスに表示されます。

MAPI メッセージ制御パラメータ

MAPI メッセージ・システムでは、次の制御パラメータを使用します。

- **Debug**　YES に設定すると、すべての MAPI 呼び出しとリターン・コードが表示されます。MAPI のサポート問題のトラブルシューティングに便利です。デフォルトは NO です。
- **Encode_dll**　カスタム・エンコード・スキームを実装している場合は、作成したカスタム・エンコード DLL のフル・パスをこの値に設定する必要があります。

  「メッセージのエンコードと圧縮」 121 ページを参照してください。
- **Force_Download**　(デフォルトは YES) MapiLogon を呼び出したときに MAPI_FORCE_DOWNLOAD オプションが設定されているかどうかを制御します。このオプションが設定されているとダイヤルするリモート・メール・ソフトウェアを使用するときに便利です。
- **IPM_Receive**　YES か NO に設定します (デフォルトは YES)。NO を設定すると、MAPI リンクは IPC メッセージを受信しますが、そのメッセージはメールボックスに表示されません。YES を設定すると、MAPI リンクは IPM メッセージと IPC メッセージを受信します。前者はメールボックスに表示されますが、後者は表示されません。IPM メッセージと IPC メッセージの両方が確実に SQL Remote によって受信されるように、この値はデフォルトである YES のままにしておくことをおすすめします。
- **IPM_Send**　YES か NO に設定します (デフォルトは NO)。YES を設定すると、MAPI リンクが IPM メッセージを送信し、メールボックスに表示します。NO を設定すると、MAPI リン

クは IPC メッセージを送信しようとますが、そのメッセージはメールボックスに表示されません。インターネット経由でメッセージをルーティングする場合、またはオペレーティング・システムに特定のパッチが適用されている場合、SQL Remote は IPC メッセージを送信できないことがあります。

- **Profile** 指定の Microsoft Exchange プロファイルを使用します。Message Agent をサービスとして実行している間は、これは使用しないでください。
- **Suppress_dialogs** true に設定されている場合は、メール・サーバへの接続の試行失敗後に、[接続] ダイアログは表示されません。代わりに、エラーが発生します。

VIM メッセージ・システム

> **注意**
> VIM のサポートは廃止されました。

Vendor Independent Messaging システム (VIM) は、Lotus Notes と一部のバージョンの Lotus cc:Mail で使用されています。

SQL Remote と VIM メッセージ・システムを使用するには、その設定に関連する各データベースに VIM ユーザ ID とアドレスが必要です。これらは別々の識別子です。VIM アドレスは各メッセージの送信先で、VIM ユーザ ID は、ユーザが自分のメールボックスに接続するときに入力する名前とパスワードです。

VIM をサポートしているオペレーティング・システムの一覧については、「プラットフォーム別 SQL Anywhere 10.0.1 コンポーネント」にある SQL Remote for SQL Anywhere の表を参照してください。

VIM メッセージ制御パラメータ

VIM メッセージ・システムでは、次の制御パラメータを使用します。

- **Path** cc:Mail のログイン・ダイアログの [パス] フィールドに対応します。Lotus Notes では適用されないため、無視されます。
- **Userid** cc:Mail のログイン・ダイアログの [ユーザ ID] フィールドに対応します。
- **Password** cc:Mail のログイン・ダイアログの [パスワード] フィールドに対応します。Path、Userid、Password をすべて設定すると、ログイン・ダイアログが表示されなくなります。
- **Debug** YES を設定すると、すべての VIM 呼び出しとリターン・コードが表示されます。VIM のサポート問題のトラブルシューティングに便利です。デフォルトは NO です。
- **Encode_dll** カスタム・エンコード・スキームを実装している場合は、作成したカスタム・エンコード DLL のフル・パスをこの値に設定する必要があります。

「メッセージのエンコードと圧縮」 121 ページを参照してください。

- **Receive_All**　YES を設定すると、Message Agent がすべてのメッセージをチェックして、SQL Remote のメッセージであるかどうか確認します。NO (デフォルト) を設定すると、Message Agent はアプリケーション定義型 **SQLRemoteData** のメッセージだけを探します。このため、Notes のパフォーマンスが向上します。

  Receive_All に YES を設定すると、メッセージ・タイプが失われたとき、リセットされたとき、または設定されていないときに便利です。ここには cc:Mail メッセージ、またはインターネットを介したメッセージを含む設定が入っています。

- **Suppress_dialogs**　true に設定されている場合は、メール・サーバへの接続の試行失敗後に、[接続] ダイアログは表示されません。代わりに、エラーが発生します。

- **Send_VIM_Mail**　YES を設定すると、Adaptive Server Anywhere の 5.5.01 より古いバージョンと cc:Mail に対して互換性のあるメッセージが Message Agent から送信されます。YES を設定している場合は、**Receive_All** にも YES を設定していることを確認してください。

Message Agent の実行

SQL Remote Message Agent は、SQL Remote レプリケーションの主要コンポーネントです。Message Agent では、メッセージの送受信処理を行います。Message Agent の機能は次のとおりです。

- 受信メッセージを処理して、データベースに適切な順序で適用する。
- 各パブリッシャ・データベースのトランザクション・ログまたはステーブル・キューをスキャンして、ログのエントリをサブスクライバへのメッセージに変換する。
- ログのエントリを、固定最大サイズ (デフォルトで 50,000 バイト) 以下のメッセージに分散し、サブスクライバに送信する。
- システム・テーブルのメッセージ・トラッキング情報を保持し、保証済みの転送メカニズムを管理する。

実行可能な名前

Windows オペレーティング・システムでは、Message Agent は *dbremote.exe*、UNIX オペレーティング・システムでは、dbremote です。Mac OS X では、SyncConsole を使用して Message Agent を起動することもできます。

「Mac OS X での Message Agent の起動」 128 ページを参照してください。

Message Agent のバッチ・モードおよび継続モード

Message Agent は次のいずれかのモードで動作します。

- **バッチ・モード** バッチ・モードでは、Message Agent は起動し、送受信できるメッセージをすべて処理し、そして停止します。

  バッチ・モードは、たまに接続するリモート・サイトにおいて有用です。接続したときにだけ、統合データベースとメッセージを交換できるためです。たとえば、リモート・サイトがメイン・ネットワークにダイヤルアップするようなときです。

- **継続モード** 継続モードでは、Message Agent は各リモート・ユーザのプロパティで指定された時刻に、定期的にメッセージを送信します。メッセージを送信していないときは、メッセージが到着すると受信します。

  継続モードは、メッセージが随時送受信される統合サイトにおいて有用です。負荷を分散させて迅速なレプリケーションを確実にするためです。

使用可能なオプションは、リモート・ユーザが選択した送信頻度オプションによって異なります。送信頻度オプションについては、「送信頻度の選択」 93 ページを参照してください。

- **継続モードで Message Agent を実行するには、次の手順に従います。**

1. すべてのユーザが送信頻度を指定していることを確認します。送信頻度は、GRANT REMOTE 文の SEND AT または SEND EVERY オプションで指定します。

2. -b オプションを使用しないで、Message Agent を起動します。

♦ **バッチ・モードで Message Agent を実行するには、次の手順に従います。**

· 次のいずれかを行います。

- ♦ リモートのプロパティに SEND AT オプションも SEND EVERY オプションも設定していないリモート・ユーザを最低 1 人設定する。
- ♦ -b オプションを使用して Message Agent を起動する。

Message Agent で使用される接続

Message Agent は、データベース・サーバへの接続をいくつか使用します。それらは次のとおりです。

- ♦ 汎用的な接続。Message Agent の起動中は常に接続されています。
- ♦ ログをスキャンするための接続。スキャンのフェーズ中のみ接続されています。
- ♦ ログスキャン・スレッドからコマンドを実行するための接続。スキャンのフェーズ中のみ接続されています。
- ♦ 同期サブスクリプション要求を処理するための接続。送信フェーズ中のみ接続されています。
- ♦ 各ワーカ・スレッドのための接続。受信フェーズ中のみ接続されています。

レプリケーション・システムのリカバリの手順

SQL Remote のレプリケーションは、統合データベース・サイトにおいてデータ・リカバリの実行を新しい稼働条件としています。標準的なバックアップとリカバリ処理により、システムまたはメディアの障害からデータをリカバリできます。レプリケーションのインストールでは、リカバリ処理をしていても、リカバリされたデータベースがリモート・データベースと同期していないことがあります。そこでリモート・データベースの完全な再同期が必要になりますが、インストール環境に大量のデータベースが含まれている場合、再同期は膨大なタスクになる可能性があります。

つまり、統合サイトでの障害から統合データベースをリカバリしても、レプリケーションのインストール全体をリカバリするタスクの一部に過ぎないのです。

レプリケーション・システムをメディア障害から保護する方法は、2 つあります。

- ♦ **バックアップとログ管理** 統合データベース・サーバの連続バックアップ・プロシージャとログ管理プロシージャは、リカバリ・プランの中で不可欠な部分です。バックアップ・プロシージャの実行は、データベース・デバイス上のメディア障害から保護します。トランザクション・ログ・ミラーの使用は、トランザクション・ログ・デバイス上のメディア障害から保護します。
「トランザクション・ログとバックアップの管理」 135 ページを参照してください。

- **Message Agent 設定**　Message Agent のコマンド・ライン・オプションによって、Message Agent の動作を各自のバックアップやリカバリ稼働条件に一致するようにチューニングできます。

  Message Agent の設定については、以降のページで説明します。

バックアップしたトランザクションのみレプリケート

デフォルトでは、Message Agent はコミットされたトランザクションをすべて処理します。Message Agent が -u オプションを使用して稼働しているときは、データベース・バックアップ・コマンドによってバックアップされたトランザクションのみが処理されます。

トランザクション・ログのバックアップを Sybase Central または *dbbackup* ユーティリティを使用して実行するか、ログ・ファイルのオフライン・コピーと名前の変更によって実行します。

バックアップしたトランザクションのみを送信することで、トランザクション・ログ上のメディア障害からレプリケーションのインストールを保護できます。ミラーされたトランザクション・ログを管理することでも、レプリケーションのインストールを保護できます。

さらに、他のサイトでもバックアップを実行すれば、-u オプションによってサイト全体の障害を防ぐこともできます。

一貫した Message Agent 設定の確認

Message Agent 設定には、インストール全体を通して同じでなくてはならないものがあります。Message Agent を配備する前に、これらを必ず同じになるように設定してください。この項では、そのような設定について説明します。

- **メッセージの最大長**　SQL Remote ではメッセージの最大長に、デフォルト値 50 K が設定されています。この値は、Message Agent の -l オプションを使用して変更できます。ただし、1 つのインストール環境内にある各 Message Agent のメッセージの最大長は、すべて同じ値にしてください。またメッセージ長は、オペレーティング・システムのメモリ割り当て制限によって制限されることがあります。

  制限より長い受信メッセージは、矛盾したメッセージとして削除されます。

  この設定の詳細については、「Message Agent」 154 ページを参照してください。

Message Agent とレプリケーション・セキュリティ

SQL Remote Message Agent で送信されるメッセージには、不意なスヌーピングから保護するために非常に簡単な暗号がついています。しかし、この暗号化スキームでは、強力な暗号解読方法からメッセージを完全に保護することはできません。

「Message Agent とレプリケーション・セキュリティ」 129 ページを参照してください。

リモート・サイトでのトラブルシューティング・エラー

管理者が、統合サイトにのみアクセスして、リモート・サイトで発生したエラーのトラブルシューティングを行うには、明らかな障害があります。この問題に関する支援として、リモート・サイトの出力ログの一部を統合サイトに送信してファイルに書き込むように、SQL Remote を設定できます。この 1 つのファイルに、システムの一部またはすべてのサイトのログ情報が含まれます。

ログ情報を収集するように SQL Remote を設定するには、リモート・サイトと統合サイトの両方で設定します。

♦ 統合データベースにログ情報を送信するようにリモート・データベースを設定するには、次の手順に従います。

1. エラーが発生したときにログ情報を送信するようにリンクのオプションを設定します。

   リモート・データベースに対して、次のコマンドを実行します。

   ```
   SET REMOTE link-name OPTION
   PUBLIC.output_log_send_on_error = 'Yes'
   ```

   このオプションを設定すると、エラーを示す "E" で始まるメッセージが発生した場合に SQL Remote が統合サイトにログ情報を送信します。

   「SET REMOTE OPTION 文 [SQL Remote]」 『SQL Anywhere サーバ - SQL リファレンス』を参照してください。

2. 統合サイトに送信される情報の量を制限するようにリンクのオプションを設定します。この手順はオプションです。

   リモート・データベースに対して、次のコマンドを実行します。

   ```
   SET REMOTE link-name OPTION
   PUBLIC.output_log_send_limit = 'nnn'
   ```

   このオプションの値は、統合データベースに送信されるバイト数で、出力ログの最後に表示されます (つまり、最後のエントリです)。*nnn*K としてキロバイトを指定できます。デフォルト設定値は "5 K" です。

   指定した値が大きすぎて最大メッセージ・サイズに対して適切でない場合、SQL Remote はこのオプションの値を上書きし、適切な大きさのメッセージだけを送信します。

output_log_send_now オプションを Yes に設定すると、エラーがなくてもログ情報を送信できます。この場合、SQL Remote は次のポーリング時に出力ログ情報を送信し、ログの送信後にオプションが "NO" にリセットされます。

♦ ログ情報を受信するように統合サイトを設定するには、次の手順に従います。

- Message Agent の -ro または -rt オプションを使用します。

  「Message Agent」 154 ページを参照してください。

Message Agent のパフォーマンスのチューニング

この項の対象読者
自分のサイトのパフォーマンスに問題がない場合は、この項を読む必要はありません。

Message Agent のパフォーマンスをチューニングするためのオプションが、いくつかあります。この項では、それらのオプションについて説明します。

メッセージの送信と受信は、独立した 2 つの処理です。パフォーマンスの主要な問題は、それぞれの処理により異なります。

- **レプリケーションのスループット** SQL Remote サイトの全体のスループットにおける主なボトルネックは、一般的に、多くのリモート・サイトからメッセージを受信して、それを統合サイトのデータベースに適用することです。この作業は、統合サイトにおける Message Agent の受信処理をチューニングすることにより制御できます。
- **レプリケーションのターンアラウンド** あるサイトにデータが入力されてから他のサイトに表示されるまでのタイム・ラグが、レプリケーションのターンアラウンド・タイムです。このタイムラグは制御できます。

Message Agent スレッドの制御によるスループットのチューニング

この項では、統合サイトにおいて継続モードで稼働している Message Agent のパフォーマンスをチューニングしていると仮定します。

Message Agent では、ワーカ・スレッドを使用してリモート・ユーザからの受信メッセージを適用します。この場合、メッセージを直列でなく並列に適用できるようにするとスループットが向上します。

ワーカ・スレッド数の設定

ワーカ・スレッド数は、Message Agent のコマンド・ラインで、-w オプションを使用して設定します。デフォルトではワーカ・スレッドを使用しない設定になっているので、すべてのメッセージは直列に適用されます。ワーカ・スレッドの最大数は 50 です。

ワーカ・スレッドからのパフォーマンス上の利点

分散されたドライブ・アレイがあるシステム上にサーバがあるときに、パフォーマンス上の利点が最も顕著に表れます。

並列で適用されるメッセージ

ワーカ・スレッドを使用中のときは、さまざまなリモート・ユーザからのメッセージが並列で適用されます。リモート・ユーザ 1 人からのメッセージは直列で適用されます。たとえば、リモート・ユーザ 1 人から 10 件のメッセージが送信されると、そのメッセージは 1 つのワーカ・スレッドで適切な順序で適用されます。

ロールバックされたトランザクションを後で再適用することで、デッドロックが処理されます。

メッセージ・システムからのメッセージの読み込みは、1 つのスレッドになります。メッセージを読み込んで、ヘッダ情報を検索 (リモート・ユーザと正しい適用順を決定するため) してから、適用するワーカ・スレッドにメッセージを渡します。

メッセージの構築と送信は、1 つのスレッドです。

メッセージのキャッシュによるスループットのチューニング

Message Agent は受信メッセージを読み込むと、メモリの設定可能エリアにそのメッセージをキャッシュします。

メッセージ・キャッシュ・サイズの指定

メッセージ・キャッシュのサイズは、-m オプションを使用して、Message Agent のコマンド・ラインで指定します。

-m オプションは、Message Agent がメッセージの構築に使用するメモリの最大容量を指定します。使用できるサイズは、*n* (バイト)、*n*K、*n*M で指定します。デフォルトは 2048 K (2 M) です。

例

次のコマンド・ラインは、12 メガバイトのメモリをメッセージ・キャッシュとして使用して Message Agent を起動します。

```
dbremote -c "eng=..." -m 12M
```

メッセージをキャッシュする方法

トランザクションが非常に大きいか、メッセージが正しく到着しない場合、メッセージは適用されるまで Message Agent がメモリに保存します。このようにメッセージをキャッシュすると、規模の大きいインストール環境においてパフォーマンスを低下させるおそれがある、メッセージ・システムからの不正なメッセージの再読み込みを防止できます。これは、メッセージを WAN (リモート・アクセス・サービスやモデム経由の POP3 など) 上で読み込む場合、特に重要になります。また、メッセージの内容がキャッシュされるため、メッセージを読み込む (単一スレッド・タスク) ワーカ・スレッド間の競合を防ぐこともできます。

-m オプションで指定したメモリ使用量を超過すると、最低使用頻度 (LRU) 方式でメッセージがフラッシュされます。

このオプションは、数千のリモート・データベースに対して単一統合データベースを検討している顧客に主に提供されます。

受信メッセージのポーリングのチューニング

Message Agent を継続モードで実行している場合は、一般的に、受信メッセージをポーリングする頻度や不正なメッセージが到着してから再送を要求するまでの待機時間を統合データベース・サイトで制御できます。これらの動作をチューニングすることで、状況によってはパフォーマンスの向上に大きな成果をもたらします。

注意点

メッセージ受信処理をチューニングするときに注意する点は、メッセージ送信処理をチューニングするときの注意点と類似しています。

- **通常のメッセージ** Message Agent がリモート・データベースから受信メッセージをポーリングする頻度を選択します。
- **再送要求** 不正なメッセージが到着してから再送を要求するまで待機するポーリング数を制御できます。
- **受信メッセージの処理** メッセージの受信頻度と比較して受信メッセージのポーリング間隔が長すぎる場合、キューに入っているメッセージを終了して、処理されるまで待機できます。ポーリング間隔が短すぎる場合は、キューにメッセージがないときもポーリングするため、リソースが無駄になります。

「メッセージ送信処理のチューニング」 119 ページを参照してください。

ポーリング間隔

デフォルトでは、継続モードで実行中の Message Agent は前回のポーリングが終了した 1 分後にポーリングを行って、新着メッセージの有無を確認します。ポーリング間隔は、-rd オプションを使用して設定できます。

ポーリングが終了してから次回のポーリングが開始されるまでのポーリング間隔のデフォルトは、1 分です。次のコマンド・ラインのように、秒単位の値を使用してポーリング頻度を上げることもできます。

```
dbremote -rd 30s
```

逆に、ポーリングの頻度を下げることもできます。次のコマンド・ラインでは、ポーリング間隔を 5 分に設定します。

```
dbremote -rd 5
```

間隔を非常に短く設定すると、システムのスループット全体に悪影響を及ぼすことがあります。それは次のような理由によります。

- 電子メールを使用している場合は、メール・サーバをポーリングするごとにメッセージ・システムに負荷がかかります。あまり頻繁にポーリングを行うと、メッセージ・システムに悪影響を及ぼし、利点はまったくありません。
- 不正なメッセージが失われたとみなされるまでの間、Message Agent の待機時間を変更せずに再送を要求すると、その要求でシステムがいっぱいになる場合があります。

一般的には、メッセージへの素早い応答が要求される特別な場合でないかぎり、ポーリング間隔はあまり短くしないでください。

長めの間隔を設定すると、各メッセージを適用するまで多少長く待機するので、システムのメッセージ・スループット全体が向上します。SQL Remote の多くのインストール環境では、ターンアラウンド・タイムの最適化はあまり重要ではありません。

再送要求

Message Agent で受信メッセージをポーリング中に、あるメッセージがシーケンスから欠落している場合、Message Agent はそのメッセージの再送を即座には要求しません。Message Agent には、1 回のポーリングにデフォルトの「**待機時間**」が設定されています。

6 番であるはずの次のメッセージが 7 番であっても、Message Agent は次のポーリングまで何もしません。その後、そのユーザ宛の新しいメッセージが検出されなければ、再送要求が発行されます。

要求を送信するまでに Message Agent が待機するポーリングの数は、-rp オプションを使用して変更できます。このオプションは、ポーリング間隔を設定する -rd オプションとともに使用されることが多くあります。

たとえば、ポーリング間隔が非常に短く、メッセージの到着順を維持しないメッセージ・システムを使用している場合、一般的には、非同期メッセージが到着するのは 2 ー 3 回のポーリングが完了してからになります。このような場合は、-rp の値を増やして、再送要求を送信するまでの待機時間が長くなるように Message Agent を設定してください。これを設定しないと、不要な再送要求が大量に送信されてしまいます。

例

user1 と **user2** という 2 人のリモート・ユーザがいて、Message Agent のコマンド・ラインが次のようであると仮定します。

```
dbremote -rd 30s -rp 3
```

次の一連の操作によってメッセージは *userX.n* とマーク付けされ、**user1.5** が user1 からの 6 番目のメッセージになります。Message Agent では、メッセージは両方のユーザについて 1 番から開始するものと考えます。

0 秒後

1. Message Agent が user1.1 と user2.4 を読み込みます。
2. Message Agent が user1.1 を適用します。
3. 順序不整合のメッセージが user2 から到着したため、この時点の Message Agent の待機時間は user1 が N/A、user2 が 3 です。

30 秒後

1. Message Agent でのメッセージの読み込み：新着メッセージなし
2. Message Agent での適用：なし
3. 現時点の Message Agent の待機時間：user1：N/A、user2：2

60 秒後

1. Message Agent でのメッセージの読み込み：user1.3
2. Message Agent での適用：新着メッセージなし

3. Message Agent の待機時間：user1：3、user2：1

90 秒後

1. Message Agent でのメッセージの読み込み：user1.4
2. Message Agent での適用：なし
3. Message Agent の待機時間：user1：3、user2：0
4. Message Agent が、user2 に再送を要求します。

ユーザが新着メッセージを受信すると、それが予期していたメッセージでなくても Message Agent の待機時間はリセットされます。

メッセージ送信処理のチューニング

レプリケーションのターンアラウンド・タイムは、各サイトからメッセージが送信される頻度と、各サイトが受信メッセージをポーリングする頻度によって制御されます。データ入力とデータ・レプリケーションの間のタイム・ラグを小さくするには、Message Agent の -sd オプションの値を小さく設定します。これにより、それ以上のデータの送信の必要性を確認するポーリングの頻度が制御されます。

注意点

メッセージ送信処理をチューニングするときに注意すべきことは、受信メッセージのポーリング頻度をチューニングするときの注意点に類似しています。

- **通常のメッセージ** リモート・データベースにアップデートを送信する頻度を選択します。
- **再送要求** リモート・ユーザがメッセージの再送要求を発行すると、Message Agent では特別な処理を実行する必要があるため、通常のメッセージ送信を中断してしまう可能性があります。再送要求の緊急度は制御できます。
- **メッセージの数とサイズ** 非常に頻繁にメッセージを送信すると、小さいメッセージが送信されることが多くなります。メッセージの送信頻度を下げると、より多くの命令を 1 つのメッセージにグループ化できます。お使いのメッセージ・システムで小さいメッセージを大量に送信しなければならない場合は、ポーリング間隔をあまり短くしないでください。

「受信メッセージのポーリングのチューニング」 116 ページを参照してください。

ポーリング間隔

トランザクション・ログからの追加データを送信するポーリングの待機間隔は、-sd オプションを使用して制御します。デフォルトは 1 分です。次の例では、ポーリング間隔を 30 秒に設定します。

```
dbremote -sd 30s ...
```


逆に、ポーリングの頻度を下げることもできます。次のコマンド・ラインでは、ポーリング間隔を 5 分に設定します。

```
dbremote -sd 5
```

間隔を非常に短く設定すると、システムのスループット全体に悪影響を及ぼすことがあります。それは次のような理由によります。

- あまり頻繁にポーリングを行うと、短いメッセージがたくさん作成されます。メッセージのロードによりメッセージ・システムに負荷がかかると、スループットに悪影響が及びます。

長めの間隔を設定すると、各メッセージを適用するまで多少長く待機するので、システムのメッセージ・スループット全体が向上します。SQL Remote の多くのインストール環境では、ターンアラウンド・タイムの最適化はあまり重要ではありません。

メッセージの再送

ユーザがメッセージの再送を要求すると、メッセージがトランザクション・ログの最初のほうから取り出されます。このメッセージを取り出し、送信するためにトランザクション・ログに戻ると、Message Agent によって通常の送信処理が中断されます。最適なパフォーマンスを得るために SQL Remote のインストール環境をチューニングする場合は、メッセージの再送要求の緊急度と通常のメッセージ処理の優先度のバランスを取ってください。

-ru オプションでは、再送要求の緊急度を制御します。パラメータの値は分単位 (数字の最後に s や h を付ければ他の単位も可能) で設定します。デフォルトはゼロです。

通常のメッセージ送信処理を中断する前に、他のメッセージが到着するまで Message Agent の再送要求の処理を遅らせるには、このオプションの設定時間を長くします。

次のコマンド・ラインでは、再送要求を処理するまで 1 時間待機します。

```
dbremote -ru 1h ...
```

-ru オプションを指定しないと、Message Agent は、データの再送を要求したユーザの送信間隔に基づいてデフォルト値を選択します。ユーザの再送要求を受信してからログを再スキャンするまでの経過時間が、そのユーザの送信間隔の半分を超えることはありません。

メッセージのエンコードと圧縮

メッセージは電子メールや他のメッセージ・システムを介して渡されるため、メッセージが壊れる危険性があります。たとえば、一部のメッセージ・システムでは、特定の文字や文字の組み合わせを制御文字として使用しています。

メッセージのサイズは、メッセージがシステムを介して渡されるときの効率に影響を与えます。圧縮したメッセージは、圧縮していないメッセージよりも効率的にメッセージ・システムで処理されます。一方、圧縮の実行自体にもかなりの時間がかかります。

SQL Remote でのエンコードと圧縮

SQL Remote では、メッセージのエンコードと圧縮スキームが Message Agent に構築されています。このスキームの機能は次のとおりです。

- **互換性** システムは、ソフトウェアの古いバージョンと互換性を持つように設定されています。
- **圧縮** メッセージの圧縮レベルを選択できます。
- **エンコード** SQL Remote ではメッセージをエンコードして、メッセージが破壊されずにメッセージ・システムを介して確実に渡されるようにします。エンコード・スキームをカスタマイズすると、特別な機能も使用できるようになります。

互換性の設定

ソフトウェアの古いバージョンと互換性を持たせるには、バージョン 6 のソフトウェアが動作しているそれぞれのデータベースで、データベース・オプション compression を -1 (マイナス 1) に設定します。すると、古いバージョンと互換性のあるフォーマットでメッセージが送信されるようになります。

SQL Remote のアップグレード

統合データベースの Message Agent を最初にアップグレードするには、データベース・オプション compression を -1 に設定します。レプリケーション・システムの各リモート・サイトをバージョン 6 にアップグレードすると、データベース・オプション compression の設定を 0 (圧縮なし) ー 9 (最大圧縮率) に変更できます。これにより、統合データベースに送信するメッセージに対して圧縮機能を利用できます。すべてのリモート・サイトをアップグレードしたら、統合データベースの Message Agent の compression オプションに -1 以外の値を設定できます。

また、このオプションに -1 以外の値を設定すると、バージョン 6 で改良されたメッセージのエンコード機能も利用できます。

エンコード・スキーム

SQL Remote のメッセージのエンコード機能は、デフォルトで次のように動作します。

- バイナリ形式のメッセージが使用できるメッセージ・システムでは、エンコードは実行されません。

- SMTP、VIM、MAPI などの一部のメッセージ・システムでは、テキストベースのメッセージ形式を使用してください。このようなシステムでは、エンコード DLL (*dbencod.dll*) によって、メッセージがテキスト・フォーマットに変換されてから送信されます。このメッセージ形式は、同じ DLL を使用する受信側ではエンコードされません。
- カスタム・エンコード・スキームの使用を SQL Remote に指示できます。カスタム・エンコード・スキームを構築するためのツールは、次の項で解説します。
- データベース・オプション compression に -1 を設定すると、すべてのメッセージ・システムでバージョン 5 と互換性のあるエンコードが実行されます。

注意
VIM および MAPI のサポートは廃止されました。

カスタム・エンコード・スキームの作成

カスタム・エンコード DLL を構築すると、カスタム・エンコード・スキームを実装することができます。この DLL を使用して、特定のメッセージ・システムで必要な特殊機能を適用したり、各ユーザに送信されたメッセージの数やバイト数などの統計を取ったりすることができます。

インストール・ディレクトリの *h* サブディレクトリにインストールされているヘッダ・ファイル *dbrmt.h* は、そのようなスキームを構築するためのアプリケーション・プログラミング・インタフェースを提供します。

特定のメッセージ・システムで自分の DLL を使用するように SQL Remote に指示するには、作成したカスタム DLL へのフル・パスの値をメッセージ制御パラメータ encode_dll に設定します。次に例を示します。

```
SET REMOTE ftp OPTION "Public"."encode_dll" = 'c:¥¥sany10¥¥win32¥¥custom.dll';
```

エンコードと復号化には互換性が必要
カスタム・エンコードを実装する場合、その DLL が受信側にもあり、自分のメッセージを正確に復号化できるようになっていることを必ず確認してください。

メッセージ・トラッキング・システム

SQL Remote にはメッセージ・トラッキング・システムが用意されていて、レプリケートされたすべてのオペレーションが正しい順序で適用され、欠落したオペレーションや二重に適用されているオペレーションがないか確認できます。

メッセージ・システムに障害があると、レプリケーション・メッセージが送信先に到着しなかったり、または壊れた状態で到着したりするおそれがあります。また、送信した順序と異なる順序で送信先に到着する可能性もあります。この項では、メッセージ・システム・エラーの検出と修正、およびメッセージの正常な適用を確認する SQL Remote のシステムについて説明します。

電子メール・メッセージ・システムを使用している場合、SQL Remote のメッセージが適切に送受信されないときは、2 台のコンピュータの間で電子メールが正しく動作していることを確認してください。

SQL Remote のメッセージ・トラッキング・システムは、SQL Remote の **remoteuser** システム・テーブルで管理されているステータス情報に基づいています。このテーブルは、Message Agent が管理しています。サブスクライバ・データベースの Message Agent はパブリッシャ・データベースに確認メッセージを送信し、サブスクリプションの最後で **remoteuser** が適切に管理されていることを確認します。

remoteuser テーブルは、**sys.isysremoteuser** システム・テーブルです。

remoteuser テーブルのステータス情報

SQL Remote の **remoteuser** システム・テーブルには各サブスクライバに対応したローがあり、そこにはそのサブスクライバが送受信するメッセージのステータス情報が示されています。統合データベースでは、**remoteuser** に各リモート・ユーザに対応したローがあります。各リモート・データベースでは、**remoteuser** に統合データベースの情報を管理するローが 1 つあります。(統合データベースは、リモート・データベースからパブリケーションにサブスクリプションを作成します)。

それぞれのサブスクリプションの最後にある SQL Remote の **remoteuser** システム・テーブルは、Message Agent が管理しています。

トランザクション・ログ・オフセットによるメッセージのトラッキング

メッセージ・トラッキングのステータス情報は、パブリッシャ・データベースとサブスクライバ・データベースのトランザクション・ログにあるオフセットの形式になっています。トランザクション・ログにある各 COMMIT は、十分に定義されたオフセットでマーク付けされています。トランザクションの順序は、オフセット値を比較して決定されます。

メッセージの順序

メッセージを送信すると、前回のメッセージの最後の COMMIT のオフセットによって、メッセージが順に並べられます。トランザクションがいくつかのメッセージにわたっている場合は、メッセージを正しい順序で並べるための通し番号がトランザクションにあります。デフォルトの

最大メッセージ・サイズは 50,000 バイトですが、Message Agent の -l オプションを使用すると、この設定を変更できます。

メッセージの送信

log_sent カラムには、サブスクライバに送信された最新メッセージのローカル・トランザクション・ログのオフセットが入ります。Message Agent はメッセージを送信すると、そのメッセージの最後の COMMIT のオフセットを **log_sent** 値に設定します。メッセージが受信されてサブスクリプションを作成したデータベースに適用されると、パブリッシャには確認メッセージが返送されます。パブリッシャの Message Agent が確認メッセージを受信すると、そのサブスクライバが **confirm_sent** カラムにローカル・トランザクション・ログのオフセットとともに設定されます。**log_sent** と **confirm_sent** は、両方ともローカル・データベースのトランザクション・ログにあるオフセットで、**confirm_sent** は **log_sent** より後のオフセットにはなりません。

メッセージの受信

サブスクライバ・データベースの Message Agent がレプリケーションのアップデートを受信して適用すると、そのメッセージの最後の COMMIT のオフセットによって **log_received** カラムが更新されます。したがって、すべてのサブスクライバ・データベースの **log_received** カラムには、パブリッシャ・データベースのトランザクション・ログにあるトランザクション・ログのオフセットがあります。オペレーションが受信され適用されると、Message Agent はパブリッシャ・データベースに確認メッセージを返送し、ローカルの SYSREMOTEUSER テーブルにある **confirm_received** 値も設定します。すべてのサブスクライバ・データベースの **confirm_received** カラムには、パブリッシャ・データベースのトランザクション・ログにあるトランザクション・ログのオフセットが入ります。

サブスクリプションの 2 通りの方法

SQL Remote のサブスクリプションには、2 通りの操作があります。各リモート・データベースは統合データベースのパブリケーションへのサブスクライバで、統合データベースは各リモート・データベースの一致するパブリケーションへサブスクライブします。したがって、統合データベースおよびリモート・データベースにある SQL Remote の **remoteuser** システム・テーブルには、補完的な情報が入っています。

Message Agent はトランザクションを適用し、**log_received** の値を自動的にアップデートします。メッセージに複数のトランザクションが含まれ、メッセージの適用中に障害が発生する場合は、**log_received** 値はすでに適用されコミットされている値と正確に一致します。

メッセージの再送

SQL Remote の **remoteuser** テーブルには、メッセージの再送を処理する 2 つのカラムがさらに含まれています。**resend_count** カラムと **rereceive_count** カラムは再試行の回数で、何らかの理由でメッセージが失われたり削除されたりすると増加します。

一般的に、**log_send** カラムには **log_sent** カラムと同じ値が入っています。しかし、**log_send** の値が **log_sent** より大きい場合、Message Agent は次の実行時にサブスクライバに直ちにメッセージを送信します。

消失または壊れたメッセージの取り扱い

サブスクライバ・データベースでメッセージが受信されると、Message Agent はそれを正しい順序 (ログのオフセットで決定される) で適用し、確認メッセージをパブリッシャに送信します。メッセージが欠落している場合、Message Agent は **rereceive_count** のローカル値を増加させ、再送を要求します。その他の現在あるメッセージや送信途中のメッセージは適用されません。

サブスクライバからのメッセージの再送要求によってパブリッシャ・データベースの **resend_count** 値が増加し、パブリッシャの **log_sent** 値も **confirm_sent** の値に設定されます。この方法で **log_sent** 値を設定し直すと、オペレーションが再送されます。

ユーザは log_sent をリセットできない
log_sent 値はシステム・テーブルにあるため、ユーザがリセットすることはできません。

メッセージの識別

各メッセージは次の 3 つの値で識別されます。

- メッセージの **resend_count**
- 前のメッセージの最後の COMMIT のトランザクション・ログ・オフセット
- 複数のメッセージにわたるトランザクションについては、トランザクションの通し番号

rereceive_count より **resend_count** の値が小さいメッセージは、適用されずに削除されます。これにより、オペレーションが 2 回以上適用されることが確実になくなります。

第 6 章

SQL Remote の管理

目次

この章では、SQL Remote 管理者を対象とした設定と管理上の問題について説明します。SQL Remote では、SQL Anywhere を統合データベースとして使用します。

Message Agent の実行

この項では、Message Agent の実行方法について説明します。

Message Agent の起動

Message Agent には、それ自体の動作を制御する一連のオプションがあります。Message Agent を実行するために最低限必要なオプションは、接続パラメータ・オプション (-c) です。「接続パラメータ」 『SQL Anywhere サーバ - データベース管理』を参照してください。

冗長キーワード	省略名	引数
DatabaseFile	DBF	string
DatabaseName	DBN	string
DatabaseSwitches	DBS	string
EngineName	ENG	string
Password	PWD	string
Start	Start	string
Userid	UID	string

Mac OS X での Message Agent の起動

SQL Anywhere には、Mac OS X で SQL Remote Message Agent を起動するのに使用する SyncConsole と呼ばれるアプリケーションが含まれています。また、dbremote ユーティリティを使用して、Mac OS X で Message Agent を起動することもできます。

♦ **SyncConsole を起動するには、次の手順に従います。**

1. [Finder] で、*/Applications/SQLAnywhere10* に移動します。
2. [SyncConsole] をダブルクリックします。
3. [ファイル] – [新規] – [SQL Remote] を選択します。

   クライアント・オプションのダイアログが表示されます。
4. SQL Remote Message Agent の接続情報を入力します。

   たとえば、次の接続パラメータでは、SQL Anywhere デモ・データベースの ODBC データソースが使用されます。

   ```
   DSN="SQL Anywhere 10 Demo"
   ```

[オプション] フィールドに入力できる dbremote オプションの完全なリストについては、「Message Agent」 154 ページを参照してください。

サービスとしての Message Agent の実行

Message Agent をバッチ・モードではなく連続モードで実行している場合、サーバの稼働中は常に Message Agent を実行させておく必要があります。

このためには、Message Agent を Windows 上で「**サービス**」として実行します。サービスは、現在使用しているユーザがログ・アウトしても実行し続け、オペレーティング・システムが起動され次第すぐに起動するように設定できます。

プログラムをサービスとして実行する方法については、「データベース・サーバの実行」 『SQL Anywhere サーバ - データベース管理』を参照してください。

Message Agent とレプリケーション・セキュリティ

前の章のチュートリアルでは、DBA パーミッションの付与されたユーザ ID を使用して Message Agent を実行しました。メッセージ内のオペレーションは、Message Agent の接続文字列で指定されたユーザ ID から実行されます。**DBA** というユーザ ID を使用すると、あらゆる変更を加えるパーミッションをそのユーザに持たせることができます。

DBA のユーザ ID とパスワードをすべてのリモート・データベース・ユーザに配布することは、多くの場合、セキュリティとデータのプライバシー上の問題からおすすめできません。SQL Remote によるソリューションでは、メッセージに含まれる変更を実行するために Message Agent がデータベースに完全にアクセスでき、かつセキュリティ上の問題を起こさずに済むようになります。

特別なパーミッションである REMOTE DBA には、以下のような性質があります。

- **Message Agent 以外からの接続では特別なパーミッションはなし**　REMOTE DBA 権限を付与されたユーザ ID は、Message Agent 以外からの接続に対して特別な権限を持ちません。したがって、REMOTE DBA ユーザのユーザ ID とパスワードを大勢に配布しても、セキュリティの問題は発生しません。そのデータベースで CONNECT より上のパーミッションが付与されたユーザ ID を使用しないかぎり、データベース内のデータにアクセスすることはできません。
- **Message Agent からの接続では完全な DBA パーミッション**　Message Agent から接続している場合、REMOTE DBA 権限を持つユーザ ID には、データベースに対する完全な DBA パーミッションが付与されます。

REMOTE DBA パーミッションの使用

おすすめしたいのは、統合データベースに対する REMOTE DBA 権限を、パブリッシャと各リモート・ユーザに付与するという方法です。リモート・データベースが抽出されると、リモート・ユーザはリモート・データベースのパブリッシャになり、統合データベースで付与されたのと同じパーミッションを付与されます。このパーミッションには REMOTE DBA 権限が含まれており、リモート・ユーザはこのユーザ ID を Message Agent 接続文字列で使用できます。このプロシージャを使用すると、管理するユーザ ID が余計に生じないので、各リモート・ユーザはデー

タベースに接続するためのユーザ ID を 1 つだけ知っていればいいということになります。このことは、Message Agent からの接続 (完全な DBA 権限が付与される) でも、他のクライアント・アプリケーションからの接続 (REMOTE DBA 権限から特に追加のパーミッションは付与されない) でも同様です。

REMOTE DBA パーミッションの付与

REMOTE DBA パーミッションを **dbremote** というユーザ ID に付与するには、次のように指定します。

```
GRANT REMOTE DBA
TO dbremote
IDENTIFIED BY dbremote
```

SQL Anywhere では、REMOTE DBA 権限をリモート・ユーザに付与するには、リモート・ユーザのプロパティ・シートの [権限] タブでそのオプションをオンにします。

エラーのレポートと処理

この項では、Message Agent がエラーをレポートし処理する方法について説明します。

デフォルトのエラー処理

デフォルトでは、エラーが発生すると、Message Agent はそのエラーをログ出力に記録します。Message Agent は、ログ出力をウィンドウまたは操作を記録するログ・ファイルに送信します。デフォルトでは、ログ出力はウィンドウにのみ送信されます。-o オプションを使用すると、出力がログ・ファイルにも送信されます。

Message Agent は、情報をウィンドウよりも出力ログに多く出力することがあります。Message Agent ログには、次のものが含まれます。

- 適用されたメッセージのリスト
- 失敗した SQL 文のリスト
- その他のエラーのリスト

エラーではない UPDATE 競合

UPDATE 競合はエラーではないので、Message Agent 出力にはレポートされません。

ログ・ファイルの詳細については、「Message Agent」 154 ページを参照してください。

エラーの無視

SQL 文の適用中に Message Agent がエラーを検出した場合、それをレポートしないようにしたいという例外的ケースもあります。これは、エラーが発生する状況を理解しており、データの不一致が発生しないことが確実で、その結果を無視しても大丈夫な場合です。

エラーがレポートされないようにするには、既知のエラーの原因となる動作に BEFORE トリガを作成します。このトリガは、REMOTE_STATEMENT_FAILED SQLSTATE (5RW09) または SQLCODE (-288) 値を通知する必要があります。

たとえば、参照先カラムが見つからないためテーブルの INSERT 文が失敗した場合に、エラーをレポートしないようにするには、参照先カラムが存在しない場合に REMOTE_STATEMENT_FAILED SQLSTATE を通知する BEFORE INSERT トリガを作成します。INSERT 文は失敗しますが、この失敗は Message Agent ログにはレポートされません。

エラー処理プロシージャの実装

SQL Remote では、エラーが発生した場合、メッセージのログを記録するだけでなく他の処理も実行できます。replication_error データベース・オプションを使用すると、エラーが発生したとき

に Message Agent で呼び出すストアド・プロシージャを指定できます。デフォルトではプロシージャを呼び出しません。

プロシージャには、データ型が CHAR、VARCHAR、または LONG VARCHAR の引数を 1 つ指定してください。プロシージャは、2 回呼び出されます。1 回はエラー・メッセージによる呼び出し、もう 1 回はエラーの原因となった SQL 文による呼び出しです。

このオプションを使用すると、レプリケーションでのエラーの追跡とモニタができますが、その場合でもやはり、エラーが起こらないように設計することが必要です。このオプションは、そのようなエラーの解決を意図したものではありません。

たとえば、プロシージャで、現在の時刻とリモート・ユーザ ID を付記してエラーをテーブルに挿入できます。その後この情報を、統合データベースにレプリケートして戻すことができます。エラーが表示されたとき、統合データベースのアプリケーションでレポートを作成したり、管理者に電子メールを送信したりできます。

replication_error オプションの設定については、「SQL Remote オプション」 170 ページを参照してください。

例：エラーの通知を電子メールで送信

Message Agent でエラーが発生したときは、統合データベースで通知を受信する必要があります。この項では、エラー発生時に管理者に電子メールでメッセージを送信する方法を説明します。

ストアド・プロシージャ

この例のストアド・プロシージャは、**sp_LogReplicationError** という名前で、ユーザ **cons** が所有しています。エラー発生時にこのプロシージャを呼び出すには、Interactive SQL または Sybase Central を使用して **replication_error** データベース・オプションを設定します。

```
SET OPTION PUBLIC.replication_error =
   'cons.sp_LogReplicationError'
```

この通知は、以下のストアド・プロシージャで実装できます。

```
CREATE PROCEDURE cons.sp_LogReplicationError
   (IN error_text LONG VARCHAR)
BEGIN
   DECLARE current_remote_user CHAR(255);
   SET current_remote_user = CURRENT REMOTE USER;

 // Log the error
   INSERT INTO cons.replication_audit
      ( remoteuser, errormsg)
   VALUES
      ( current_remote_user, error_text);
   COMMIT WORK;

 //Now notify the DBA by email that an error has occurred
 // on the consolidated database. The email should contain the error
 // strings that the Message Agent is passing to the procedure.
   IF CURRENT PUBLISHER = 'cons' THEN
      CALL sp_notify_DBA( error_text );
   END IF
END;
```


このストアド・プロシージャは、電子メールの送信を管理するために、以下のストアド・プロシージャを呼び出します。

```
CREATE PROCEDURE sp_notify_DBA( in msg long varchar)
BEGIN
   DECLARE rc INTEGER;
   rc=call xp_startmail( mail_user='davidf' );
   //If successful logon to mail
   IF rc=0 THEN
   rc=call xp_sendmail(
         recipient='Doe, John; John, Elton',
         subject='SQL Remote Error',
         "message"=msg);
   //If mail sent successfully, stop
      IF rc=0 THEN
         call xp_stopmail()
      END IF
   END IF
END;
```

監査テーブル

以下は、監査テーブルの定義の例です。

```
CREATE TABLE replication_audit (
   id         INTEGER DEFAULT AUTOINCREMENT,
   pub        CHAR(30) DEFAULT CURRENT PUBLISHER,
   remoteuser    CHAR(30),
   errormsg    LONG VARCHAR,
   timestamp     DATETIME DEFAULT CURRENT TIMESTAMP,
   PRIMARY KEY (id,pub)
);
```

各カラムの意味は、次のとおりです。

カラム	説明
pub	データベースの現在のパブリッシャ (どのデータベースで挿入されたのかを示す)
remoteuser	メッセージを適用しているリモート・ユーザ (どのデータベースから受け取ったのかを示す)
errormsg	replication_error プロシージャに渡されたエラー・メッセージ

以下は、上記のエラーでテーブルに挿入される内容の例です。

```
INSERT INTO cons.replication_audit
   (    id,
        pub,
        remoteuser,
        errormsg,
        "timestamp")
VALUES
   (    1,
        'cons',
        'sales',
        'primary key for table "reptable" is not unique (-193)',
        '1997/apr/21 16:03:13.836')
COMMIT WORK
```

SQL Anywhere はストアド・プロシージャからの外部 DLL の呼び出しをサポートするので、電子メールを使用する代わりにページング・システムを構築することもできます。

エラーの例

たとえば、リモートで挿入されたローと同じプライマリ・キーを使用して統合データベースでローを挿入すると、Message Agent に次のエラーが表示されます。

"cons" (0-0000000000-0) メッセージを受信しました。

SQL 文が失敗しました：(-193) テーブル 'reptable' のプライマリ・キーがユニークではありません。

INSERT INTO cons.reptable(id,text,last_contact)

VALUES (2,'dave','1997/apr/21 16:02:38.325')

COMMIT WORK

Doe, John と Elton, John に電子メールで届くメッセージは、それぞれ "SQL Remote Error" という件名になります。

テーブル 'reptable' のプライマリ・キーがユニークではありません (-193)。

INSERT INTO cons.reptable(id,text,last_contact) VALUES (2,'dave','1997/apr/21 16:02:52.605')

トランザクション・ログとバックアップの管理

バックアップをしっかり取る習慣の重要性

レプリケーションは、トランザクション・ログ内のオペレーションへのアクセスに依存して行われ、古いトランザクション・ログへのアクセスが必要になることがあります。この項では、統合データベースとリモート・データベースでバックアップ・プロシージャを設定して古いトランザクション・ログに正しくアクセスする方法を説明します。

SQL Remote の統合データベース・サイトでは、しっかりバックアップを取る習慣が大切です。トランザクション・ログを失ったときは、リモート・ユーザを再抽出しなくてはならないからです。そのため、統合データベース・サイトでは、トランザクション・ログ・ミラーリングをおすすめします。

トランザクション・ログ・ミラーとその他のバックアップ・プロシージャの詳細については、「バックアップとデータ・リカバリ」『SQL Anywhere サーバ - データベース管理』を参照してください。

古いトランザクションへのアクセスの確保

トランザクション・ログはすべて、レプリケーション・システムで必要とされなくなるまで保管しておく必要があります。

多くの場合、リモート・データベースのユーザは毎日のようにオフィス・サーバから更新を受信します。ただし、消失または削除したメッセージを、メッセージ・トラッキング・システムから再送しなければならない場合には、数日前に加えられた変更の内容が必要になることがあります。また、リモート・ユーザの休暇中にメッセージが失われた場合は、数週間も前の変更の内容が必要になります。トランザクション・ログのバックアップを毎日行っている場合、そのような以前の変更が含まれるログは、もうサーバで使用されていないことになります。

トランザクション・ログは増え続けるので、領域に関する問題が発生します。トランザクション・ログ・サイズのイベント・ハンドラを使用すると、指定サイズに達した場合にログ名を変更できます。変更したら、delete_old_logs オプションを使用して、不要になったログ・ファイルをクリーン・アップできます。

トランザクション・ログ・サイズの制御に関する詳細については、「BACKUP 文」『SQL Anywhere サーバ - SQL リファレンス』を参照してください。

トランザクション・ログ・ディレクトリの設定

現在のログ以外のトランザクション・ログをスキャンする必要がある場合、Message Agent は指定の「**トランザクション・ログ・ディレクトリ**」に保管されているすべてのトランザクション・ログ・ファイルを調べます。そのディレクトリは、Message Agent のコマンド・ラインで指定されています。

例

たとえば、次のコマンド・ラインは、ディレクトリ *e:¥archive* を調べて古いトランザクション・ログを見つけるように、Message Agent に指示します。このコマンドは、すべて 1 行で入力してください。


```
dbremote -c "eng=server_name;uid=DBA;pwd=sql" e:¥archive
```

ログ名は重要ではない

Message Agent は、トランザクション・ログ・ディレクトリにあるすべてのファイルを開いてログ・ファイルを見つけます。そのため、ログ・ファイルの実際の名前は重要ではありません。

この項では、バックアップ・プロシージャを設定して、このようなディレクトリを適切な形で維持する方法を説明します。

バックアップ・ユーティリティのオプション

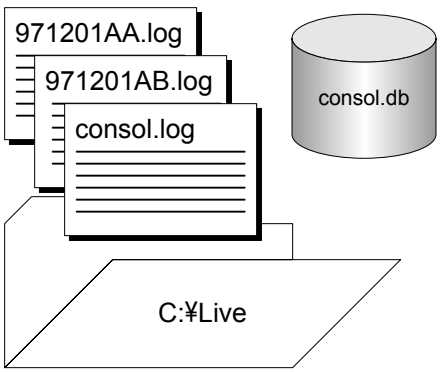
SQL Anywhere のバックアップ・ユーティリティには、動作を制御するためのオプションがいくつかあります。これらのオプションは、Sybase Central ウィザードで選択するか、dbbackup オプションを使って指定できます。

この項では、SQL Remote 統合データベース・バックアップでバックアップ・ユーティリティを使用する 2 つの方法について説明します。バックアップによって、Message Agent での使用に適した一連のトランザクション・ログを、常に保管しておく必要があります。

ライブ・ディレクトリをトランザクション・ログ・ディレクトリとして使用する

統合データベースとリモート・データベースのトランザクション・ログをバックアップするときは、トランザクション・ログを名前変更して再作成するオプションの使用をおすすめします。dbbackup ユーティリティでは、これが -r オプションです。

次の図は、データベース *consol.db* とトランザクション・ログ *consol.log* が同じディレクトリにある様子を表しています。これは通常、実際の運用環境では安全なこととは言えませんが、わかりやすくするために、この例ではログがデータベースと同じディレクトリにあると仮定しています。ディレクトリの名前は *c:¥live* です。



バックアップ・コマンド・ライン

以下は、名前変更と再作成のオプションを使用してデータベースをバックアップするコマンド・ラインです。

```
dbbackup -r -c "uid=DBA;pwd=sql" c:¥archive
```

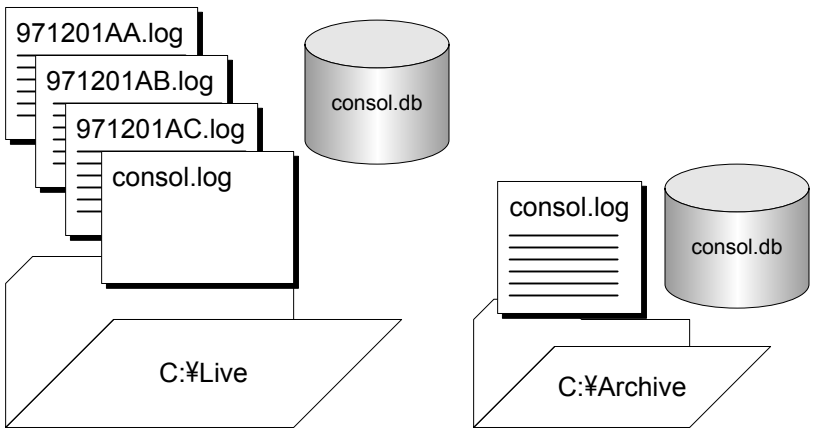
接続文字列のオプションは、データベースによって異なります。

バックアップの影響

名前の変更と再起動のオプションを使用してトランザクション・ログをディレクトリ *c:¥archive* にバックアップすると、バックアップ・ユーティリティが以下のタスクを実行します。

1. トランザクション・ログ・ファイルをバックアップし、バックアップ・ファイル *c:¥archive¥consol.log* を作成する。
2. 既存のトランザクション・ログ・ファイルの名前を *971201xx.log* (*xx* は AA ー ZZ のアルファベット順の英字) に変更する。
3. *consol.log* という名前で新しいトランザクション・ログを作成する。

バックアップを何回か行うと、ライブ・ディレクトリに連続した名前の一連のトランザクション・ログができます。



Message Agent コマンド・ライン

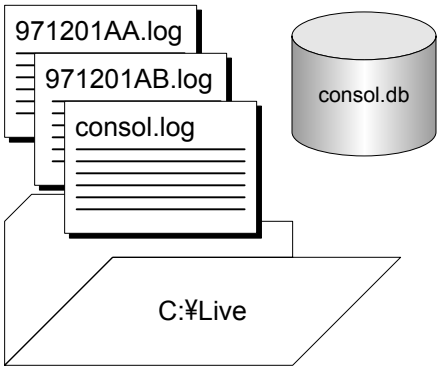
これらのログ・ファイルにアクセスしながら Message Agent を実行するには、次のコマンド・ラインを使用します。

```
dbremote -c "dbn=hq;..." c:¥live
```

バックアップ・ディレクトリをトランザクション・ログ・ディレクトリとして使用する

もう 1 つの方法では、バックアップ・ディレクトリをトランザクション・ログ・ディレクトリとして使用します。

次の図でも、データベース *consol.db* とトランザクション・ログ *consol.log* が同じディレクトリにある様子を表しています。これは通常、実際の運用環境では安全なこととは言えませんが、わかりやすくするために、この例ではログがデータベースと同じディレクトリにあると仮定しています。ディレクトリの名前は *c:¥live* です。



バックアップ・コマンド・ライン

次のコマンド・ラインは、名前変更と再作成のオプションを使用してデータベースをバックアップします。また、トランザクション・ログのバックアップ・ファイルの名前を変更するオプションも使用します。

```
dbbackup -r -n -c "uid=DBA;pwd=sql" c:¥archive
```

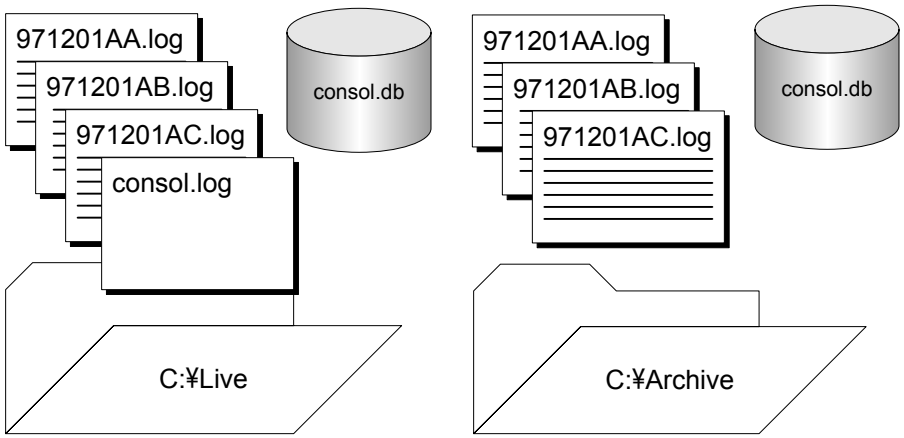
接続文字列のオプションは、データベースによって異なります。

バックアップの影響

名前の変更と再起動のオプションと、ログの名前変更オプションを使用して、ディレクトリ *c:¥archive* にトランザクション・ログをバックアップすると、バックアップ・ユーティリティが以下のタスクを実行します。

1. 既存のトランザクション・ログ・ファイルの名前を *971201xx.log* (*xx* は *AA* ー *ZZ* のアルファベット順の英字) に変更する。
2. トランザクション・ログ・ファイルをバックアップ・ディレクトリにバックアップし、バックアップ・ファイル *971201xx.log* を作成する。
3. *consol.log* という名前で新しいトランザクション・ログを作成する。

バックアップを何回か行うと、live ディレクトリと archive ディレクトリに連続した名前の一連のトランザクション・ログができます。



Message Agent コマンド・ライン

これらのログ・ファイルにアクセスしながら Message Agent を実行するには、次のコマンド・ラインを使用します。

```
dbremote -c "dbn=hq;..." c:¥archive
```

8.0.1 より前の古いログ・ファイル名は異なる
リリース 8.0.1 より前の Adaptive Server Anywhere では、古いログ・ファイルの名前が、*yymmdd01.log*、*yymmdd02.log* のようになっています。名前を変更したのは、古いログをより多く保存できるようにするためです。Message Agent は、指定されたディレクトリ内で、ファイル名に関係なく全ファイルをスキャンするので、ログ・ファイル名が変わっても既存のアプリケーションに影響はありません。

古いトランザクション・ログの管理

トランザクション・ログはすべて、レプリケーション・システムで必要とされなくなるまで保管しておく必要があります。不要になったら廃棄してかまいません。

レプリケーション・システムがログを必要としなくなるのは、ログ・ファイルに含まれるメッセージを、すべてのリモート・データベースが受信して正常に適用し終わったときです。統合データベースからのメッセージを正常に受信したことがリモート・データベースで確認されると、統合データベースの SQL Remote テーブルに値が設定されます。この受信確認をすべてのリモート・データベースから受け取った時点で、統合データベースの古いトランザクション・ログは SQL Remote で必要とされなくなります。

「メッセージ・トラッキング・システム」 123 ページを参照してください。

delete_old_logs オプションの使用

delete_old_logs データベース・オプションを統合データベースで使用すると、古いトランザクション・ログを自動的に管理できます。

delete_old_logs データベース・オプションは、デフォルトでは Off に設定されています。on に設定すると、不要になった古いトランザクション・ログが自動的に削除されます。古いログが不要になるのは、そのログ・ファイルに記録されている変更をすべて受信したという確認を、すべてのサブスクライバから受け取った時点です。

delete_old_logs オプションは、PUBLIC グループに対して、または Message Agent の接続文字列に含まれるユーザのみに対して設定できます。

例

次の文では、PUBLIC グループに delete_old_logs を設定して、作成されてから 10 日以上過ぎたログを削除します。

```
SET OPTION PUBLIC.delete_old_logs = '10 days'
```

統合データベースのデータベース・メディア障害からのリカバリ

この項では、統合データベースでデータベース・デバイスのメディア障害からリカバリする方法について説明します。

リカバリの手順が最も簡単なのは、トランザクション・ログ・ファイルが 1 つしかない場合です。統合データベースでは通常は複数のトランザクション・ログ・ファイルがあり、手順も複雑になりますが、ここではまず 1 つしかない場合について説明します。その後で、複数のトランザクション・ログ・ファイルがある場合について説明します。

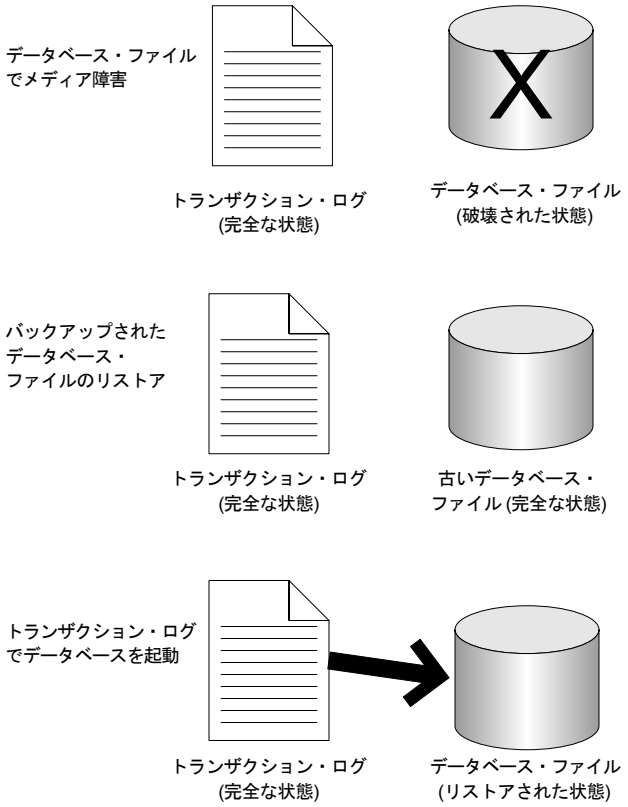
トランザクション・ログが 1 つの場合のリカバリ

ここでは、トランザクション・ログ・ファイルは 1 つしかないと仮定します。このトランザクション・ログ・ファイルは、データベースが作成されてからずっと存在しているものです。また、データベース・ファイルは前もってバックアップされており、テープなどに保存されていると仮定します。

♦ **データベースをリカバリするには、次の手順に従います。**

1. データベース・ファイルとログ・ファイルのコピーを作成します。
2. データベース (*.db*) ファイルをテープからテンポラリ・ディレクトリにリストアします。ログ・ファイルはリストア *しません*。
3. 既存のトランザクション・ログと -a オプションを使用してデータベースを起動し、トランザクションを適用してデータベース・ファイルを最新の状態に更新します。
4. データベースを通常どおり起動します。

   新しいアクティビティは、すべて現行のトランザクション・ログに追加されます。



例

この例では、ミラーリングされたトランザクション・ログを使用したリカバリについて説明します。

consol.db という統合データベース・ファイルがディレクトリ *c:¥dbdir* にあり、トランザクション・ログ・ファイル *c:¥logdir¥consol.log* が *d:¥mirdir¥consol.mlg* にミラーリングされるとします。

♦ C ドライブのメディア障害からリカバリするには、次の手順に従います。

1. ミラーリングされたトランザクション・ログ *d:¥mirdir¥consol.mlg* をバックアップします。
2. 障害の起きたハードウェアを交換し、影響を受けたすべてのソフトウェアを再インストールします。
3. リカバリを実行するテンポラリ・ディレクトリを作成します。ここでは *c:¥recover* とします。
4. データベース・ファイル *consol.db* の最新のバックアップを *c:¥recover¥consol.db* にリストアします。
5. ミラーリングされたトランザクション・ログ *d:¥mirdir¥consol.mlg* を、リカバリ・ディレクトリにコピーします。拡張子は *.log* に変更し、*c:¥recover¥consol.log* とします。

6. 次のコマンドを使用してデータベースを起動します。

   ```
   dbeng10 -a c:¥recover¥consol.log C:¥recover¥consol.db
   ```

7. データベース・サーバを停止します。
8. リカバリされたデータベースとトランザクション・ログを *c:¥recover* からバックアップします。
9. *c:¥recover* から実際に使用する適切なディレクトリに、ファイルをコピーします。
   - *c:¥recover¥consol.db* を *c:¥dbdir¥consol.db* にコピーします。
   - *c:¥recover¥consol.log* を *c:¥logdir¥consol.log* と *d:¥mirdir¥consol.mlg* にコピーします。
10. システムを通常どおり再起動します。

トランザクション・ログが複数の場合のリカバリ

複数のトランザクション・ログがある場合、オンライン・トランザクション・ログが 1 つある場合とはリカバリ手順が異なります。トランザクション・ログを個別に適用する場合は -a オプションを使用し、データベース・サーバでトランザクション・ログの正しい適用順序を特定してすべてのログを適用する場合は -ad オプションを使用します。

個別のトランザクション・ログの適用

次の手順では、データベースのリカバリ時に各トランザクション・ログを個別にデータベースに適用する方法を説明します。データベース・ファイルは前もってバックアップされており、テープなどに保存されていると仮定します。

♦ -a オプションを使用してデータベースをリカバリするには、次の手順に従います。

1. データベース・ファイルとログ・ファイルのコピーを作成します。
2. データベース (*.db*) ファイルをテープからテンポラリ・ディレクトリにリストアします。ログ・ファイルはリストア *しません*。
3. テンポラリ・ディレクトリで、データベースを起動します。このとき、-a オプションを使用して古いログを適用し、指定したトランザクション・ログを適切な順序で適用します。
4. 現在のトランザクション・ログと -a オプションを使用してデータベースを起動し、トランザクションを適用してデータベース・ファイルを最新の状態に更新します。
5. データベースを通常どおり起動します。

   新しいアクティビティは、すべて現行のトランザクション・ログに追加されます。

例

c:¥dbdir¥cons.db という統合データベース・ファイルがあると仮定します。トランザクション・ログ・ファイル *c:¥dbdir¥cons.log* は、*d:¥mirdir¥cons.mlg* にミラーリングされます。

毎週実行するフル・バックアップに加え、次のコマンドを使用してインクリメンタル・バックアップを毎日実行しているものとします。

```
dbbackup -c "uid=DBA;pwd=sql" -r -t e:¥backdir
```

このコマンドは、トランザクション・ログ *cons.log* をディレクトリ *e:¥backdir* にバックアップします。トランザクション・ログの名前は *datexx.log* (*date* はそのときの日付、*xx* はアルファベット順の次の英字) に変更され、新しいトランザクション・ログが作成されます。その後、ディレクトリ *e:¥backdir* がサード・パーティ・ユーティリティを使用してバックアップされます。

ここでは、名前を変更したトランザクション・ログ・ファイルを示すディレクトリを任意で指定して Message Agent を実行しているものとします。Message Agent のコマンド・ラインは次のとおりです。

```
dbremote -c "uid=DBA;pwd=sql" c:¥dbdir
```

毎週実行しているバックアップの 3 日後に、ディスク・ブロック破損のためにデータベース・ファイルが破壊されてしまったとします。

♦ C ドライブのメディア障害からリカバリするには、次の手順に従います。

1. ミラーリングされたトランザクション・ログ *d:¥mirdir¥cons.mlg* をバックアップします。
2. リカバリを実行するテンポラリ・ディレクトリを作成します。この例では *c:¥recover* というディレクトリを作成します。
3. データベース・ファイル *cons.db* の最新のバックアップを *c:¥recover¥cons.db* にリストアします。
4. 名前を変更したトランザクション・ログを、次の順序で適用します。

   ```
   dbeng10 -a c:¥dbdir¥date00.log c:¥recover¥cons.db
   dbeng10 -a c:¥dbdir¥date01.log c:¥recover¥cons.db
   ```

5. 現在のトランザクション・ログ *c:¥dbdir¥cons.log* を、リカバリ・ディレクトリにコピーし、*c:¥recover¥cons.log* を作成します。
6. 次のコマンドを使用してデータベースを起動します。

   ```
   dbeng10 c:¥recover¥cons.db
   ```

7. データベース・サーバを停止します。
8. リカバリされたデータベースとトランザクション・ログを *c:¥recover* からバックアップします。
9. *c:¥recover* から実際に使用する適切なディレクトリに、ファイルをコピーします。
   - *c:¥recover¥cons.db* を *c:¥dbdir¥cons.db* にコピーします。
   - *c:¥recover¥cons.log* を *c:¥dbdir¥cons.log* と *d:¥mirdir¥cons.mlg* にコピーします。
10. システムを通常どおり再起動します。

複数のトランザクション・ログの適用

次の手順では、データベース・サーバを起動して、すべてのトランザクション・ログを正しい適用順序でデータベースに自動的に適用する方法について説明します。-ad オプションを指定すると、データベース・サーバは指定されたディレクトリでデータベースのトランザクション・ログを検索します。次にトランザクション・ログの正しい順序を特定し、ログ・オフセットに基づいてログを適用します。

次の手順では、データベース・ファイルは前もってバックアップされており、テープなどに保存されていると仮定しています。

♦ **-ad オプションを使用してデータベースをリカバリするには、次の手順に従います。**

1. データベース・ファイルとログ・ファイルのコピーを作成します。
2. データベース (*.db*) ファイルをテープからテンポラリ・ディレクトリにリストアします。ログ・ファイルはリストア*しません*。
3. テンポラリ・ディレクトリでデータベースを起動し、-ad オプションを使用してトランザクション・ログを適用します。
4. データベースを通常どおり起動します。

   新しいアクティビティは、すべて現行のトランザクション・ログに追加されます。

例

c:¥dbdir¥cons.db という統合データベース・ファイルがあると仮定します。トランザクション・ログ・ファイル *c:¥dbdir¥cons.log* は、*d:¥mirdir¥cons.mlg* にミラーリングされます。

次のコマンドを使用して、フル・バックアップを毎週実行するとします。

```
dbbackup -c "uid=DBA;pwd=sql" -r e:¥backdir
```

また、次のコマンドを使用して、インクリメンタル・バックアップを毎日実行するとします。

```
dbbackup -c "uid=DBA;pwd=sql" -r -t e:¥backdir
```

このコマンドは、トランザクション・ログ *cons.log* をディレクトリ *e:¥backdir* にバックアップします。トランザクション・ログの名前は *datexx.log* (*date* はそのときの日付、*xx* はアルファベット順の次の英字) に変更され、新しいトランザクション・ログが作成されます。その後、ディレクトリ *e:¥backdir* がサード・パーティ・ユーティリティを使用してバックアップされます。

ここでは、名前を変更したトランザクション・ログ・ファイルを示すディレクトリを任意で指定して Message Agent を実行しているものとします。Message Agent のコマンド・ラインは次のとおりです。

```
dbremote -c "uid=DBA;pwd=sql" c:¥dbdir
```

毎週実行しているバックアップの 3 日後に、ディスク・ブロック破損のためにデータベース・ファイルが破壊されてしまったとします。

♦ **C ドライブのメディア障害からリカバリするには、次の手順に従います。**

1. *c:¥* ドライブを交換します。
2. ミラーリングされたトランザクション・ログ *d:¥mirdir¥cons.mlg* をバックアップします。
3. リカバリを実行するテンポラリ・ディレクトリを作成します。この例では *c:¥recover* というディレクトリを作成します。
4. データベース・ファイル *cons.db* の最新のバックアップを *c:¥recover¥cons.db* にリストアします。
5. バックアップ済みのトランザクション・ログを *c:¥dbdir* にコピーします。
6. 名前を変更したトランザクション・ログを適用します。

   ```
   dbeng10 c:¥recover¥cons.db -ad c:¥dbdir
   ```

7. 現在のトランザクション・ログ *c:¥dbdir¥cons.log* を、リカバリ・ディレクトリにコピーし、*c:¥recover¥cons.log* を作成します。
8. 次のコマンドを使用してデータベースを起動します。

   ```
   dbeng10 c:¥recover¥cons.db
   ```

9. データベース・サーバを停止します。
10. リカバリされたデータベースとトランザクション・ログを *c:¥recover* からバックアップします。
11. *c:¥recover* から実際に使用する適切なディレクトリに、ファイルをコピーします。
    - *c:¥recover¥cons.db* を *c:¥dbdir¥cons.db* にコピーします。
    - *c:¥recover¥cons.log* を *c:¥dbdir¥cons.log* と *d:¥mirdir¥cons.mlg* にコピーします。
12. システムを通常どおり再起動します。

リモート・データベースでのバックアップ・プロシージャ

リモート・データベースでは、バックアップ手順は統合データベースの場合ほど重要ではありません。データのバックアップを、統合データベースへのレプリケーションに頼る方法もあります。ただし、メディア障害が発生した場合は、統合データベースからリモート・データベースを再抽出しなければならず、レプリケートされていないオペレーションは失われます(ログ変換ユーティリティを使用して、失われたオペレーションのリカバリを実行することは可能です。「ログ変換ユーティリティ (dbtran)」『SQL Anywhere サーバ - データベース管理』を参照してください)。

レプリケーションを使ってリモート・データベースのデータを保護する場合でも、トランザクション・ログが大きくなりすぎるのを防ぐために、リモート・データベースで定期的にバックアップを実行する必要があります。統合データベースで使用するのと同じオプション (ログの名前変更と再起動) を使用して Message Agent を実行し、Message Agent が名前変更されたログ・ファイルにアクセスできるようにします。リモート・データベースで delete_old_logs オプション

を on に設定すると、不要になった古いログ・ファイルが Message Agent によって自動的に削除されます。

自動的にトランザクション・ログの名前を変更

-x Message Agent オプションを使用すると、データベース・サーバを停止した際に、リモート・コンピュータでトランザクション・ログの名前を変更する必要がなくなります。-x オプションを指定すると、トランザクション・ログで出力メッセージをスキャンした後に、名前が変更されます。

統合データベースのアップグレード

この項では、SQL Remote 環境で統合データベースをアップグレードするときの注意点について説明します。Sybase Replication Server でプライマリ・サイトとして使用されている SQL Anywhere データベースにも、同じ注意事項が当てはまります。

SQL Remote 統合データベースのバージョン 10 へのアップグレードの詳細については、「SQL Remote のアップグレード」 『SQL Anywhere 10 - 変更点とアップグレード』を参照してください。

新しいソフトウェアをインストールしても、必ずしも新しい機能が使用可能になるわけではありません。多くの場合、新しい機能を使用可能にするには、データベースでアップグレード・ユーティリティを実行する必要があります。アップグレード・ユーティリティは、新しい機能を使用可能にするのに必要な情報をシステム・カタログに書き込みます。アップグレード・ユーティリティを実行すると、トランザクション・ログをアーカイブすることを求めるメッセージが表示されます。これは、アップグレード・ユーティリティが新しいトランザクション・ログを新しいファイル・フォーマットで作成するからです。

SQL Remote または Replication Server を使用しているときは、Message Agent と Replication Agent のトランザクション・ログを別々に保管する必要があります。アップグレード・ユーティリティを実行した後は、エンジンを停止し、ログの名前を変更してください。ログは Message Agent によって自動的に削除されます。ログは、バックアップ用にもアーカイブしてください。

「アップグレード・ユーティリティ (dbupgrad)」 『SQL Anywhere サーバ - データベース管理』を参照してください。

レプリケーションに参加しているデータベースのアンロードと再ロード

レプリケーションされているデータベースがある場合、データベースのアンロードと再ロードには特に注意が必要です。

SQL Remote レプリケーションに関連するデータベースのアンロードと再ロードの手順については、「SQL Remote のアップグレード」 『SQL Anywhere 10 - 変更点とアップグレード』を参照してください。

レプリケーションは、トランザクション・ログに基いて行われます。データベースをアンロードして再ロードすると、古いトランザクション・ログは使用できなくなります。このため、レプリケーションが関係するときは、バックアップをきちんと実行することが特に重要です。

レプリケーションに関係するデータベースのアンロード手順については、「同期やレプリケーションに関連するデータベースの再構築」『SQL Anywhere サーバ - SQL の使用法』を参照してください。

パススルー・モードの使用

統合データベースのパブリッシャは、パススルー・モードを使用してリモート・サイトに直接介入できます。パススルー・モードでは、標準の SQL 文をリモート・サイトに渡すことができます。デフォルトでは、パススルー・モード文はローカル (統合) データベースでも実行されますが、オプションのキーワード指定によって、ローカルで実行しないようにすることもできます。

警告
パススルー・オペレーションは、リモート・データベースのサブスクリプションが作成されているテスト・データベース上で必ずテストしてください。テストを行っていないパススルー・スクリプトを、運用データベースに対して実行しないでください。

パススルーの開始と停止

パススルー・モードの開始と停止には、PASSTHROUGH 文を使用します。開始の PASSTHROUGH 文とパススルー・モードを終了させる PASSTHROUGH STOP 文の間に入力された文は、構文エラーのチェックを受け、現在のデータベースで実行されます。さらに、識別されたサブスクライバに渡され、サブスクライバのデータベースでも実行されます。開始のパススルー文と停止のパススルー文の間にある一連の文は、「**パススルー・セッション**」と呼ばれます。

次の文を使用すると、ローカルのデータベースで実行されずに、指定した 2 つのサブスクライバのリストにパススルー文を渡すパススルー・セッションが開始されます。

```
PASSTHROUGH ONLY
FOR userid_1, userid_2;
```

パススルー文の指示

次の文は、指定したパブリケーションのすべてのサブスクライバに文を渡すパススルー・セッションを開始します。

```
PASSTHROUGH ONLY
FOR SUBSCRIPTION TO [owner].pubname [ ( string ) ] ;
```

パススルー・モードは、加算的です。次の例では、**statement_1** は **user_1** に送られ、**statement_2** は **user_1** と **user_2** の両方に送られます。

```
PASSTHROUGH ONLY FOR user_1 ;
statement_1 ;
PASSTHROUGH ONLY FOR user_2 ;
statement_2 ;
```

次の文は、パススルー・モードを終了します。

```
PASSTHROUGH STOP ;
```

PASSTHROUGH STOP を実行すると、すべてのリモート・ユーザへのパススルー・モードが終了します。

パススルー文の適用順序

パススルー文は、通常のレプリケーション・メッセージと一緒に、文がログに記録された順序で連続してレプリケートされます。

パススルーは通常、データ定義言語の文を送信するために使用します。この場合、レプリケートされた DML 文は、パススルーの前に *before* スキーマを使用し、パススルーの後に *after* スキーマを使用します。

パススルー・モードの使用上の注意

- パススルー・オペレーションは常に、リモート・データベースのサブスクリプションが作成されているテスト・データベース上でテストしてください。テストを行っていないパススルー・スクリプトを、運用データベースに対して実行しないでください。
- オブジェクト名は必ず所有者名で修飾してください。PASSTHROUGH 文は、同じユーザ ID のリモート・データベースでは実行されません。したがって、所有者名の修飾子を持たないオブジェクト名は正しく解析されないことがあります。

パススルー・モードの使用法と制限事項

パススルー・モードは強力なツールなので、使用するときは注意が必要です。文 (特にデータ定義文) の中には、実行中の SQL Remote 設定が失敗する原因となるものがあります。SQL Remote は、1 つの設定内の各データベースが同じオブジェクトを持っているものとして動作します。あるテーブルが一部のサイトのみで変更されている場合、データの変更をレプリケートしようとしても失敗します。

また、デフォルトの設定では、パススルー・モードはローカル・データベースでも文を実行するということを忘れないでください。文をローカルで実行せずにリモート・データベースに送るには、ONLY キーワードが必要です。パススルー・セクションにストアド・プロシージャへの呼び出しが含まれている場合、そのストアド・プロシージャは、パススルー・コマンドを発行するサーバ上に存在していなければなりません (これらのストアド・プロシージャの中には、サーバ上では実行されないものもあります)。次の一連の文では、リモート・データベースだけでなく統合データベースでもテーブルを削除します。

```
-- Drop a table at the remote database
-- and at the local database
PASSTHROUGH TO Joe_Remote ;
DROP TABLE CrucialData ;
PASSTHROUGH STOP ;
```

リモート・データベースのみでテーブルを削除する構文は、次のとおりです。

```
-- Drop a table at the remote database only
PASSTHROUGH ONLY TO Joe_Remote ;
DROP TABLE CrucialData ;
PASSTHROUGH STOP ;
```

稼働中の SQL Remote 設定で行えるタスクは、以下のとおりです。

- 新しいユーザを追加する。
- ユーザを再同期する。
- インストール環境からユーザを削除する。
- リモート・ユーザのアドレス、メッセージ・タイプ、または頻度を変更する。

- テーブルにカラムを追加する。

その他のスキーマ変更の多くは、稼働中の SQL Remote 設定で行うと重大な問題を引き起こす可能性があります。

階層の 1 つのレベルのみで動作するパススルー

多層 SQL Remote インストール環境では、現在のレベルのすぐ下のレベルにあるデータベースでパススルー文が動作することが重要になります。多層インストール環境では、統合データベースで、その下のレベルを動作の対象としてパススルー文を入力してください。

パススルー・モードでレプリケートされないオペレーション

パススルー・モードで特に注意が必要な文があります。

プロシージャの呼び出し

パススルー・モードで CALL 文または EXEC 文を使用してストアド・プロシージャを呼び出すと、CALL 文自体がレプリケートされ、プロシージャ内の文はレプリケートされません。レプリケート側のプロシージャは適切な作用をしていると仮定しています。

フロー制御文とカーソル処理

IF や LOOP などのフロー制御文と、すべてのカーソル処理は、パススルー・モードではレプリケートされません。ループ構造または制御構造内の文は、レプリケートされます。

カーソルの処理はレプリケートされません。カーソルによるローの挿入、カーソル内でのローの更新、またはカーソルによるローの削除は、パススルー・モードではレプリケートされません。

静的な Embedded SQL の SET OPTION 文はレプリケートされません。つまり、次の文はパススルー・モードではレプリケートされません。

```
EXEC SQL SET OPTION . . .
```

しかし、次の動的 SQL 文はレプリケートされます。

```
EXEC SQL EXECUTE IMMEDIATE "SET OPTION . . . "
```

バッチ

バッチ文 (BEGIN と END に囲まれた一連の文) は、パススルー・モードではレプリケートされません。パススルー・モードでバッチ文を使おうとすると、エラー・メッセージが表示されます。

パート IV. リファレンス

パート IV では、SQL Remote のリファレンス情報を紹介します。

第 7 章

SQL Remote ユーティリティとオプションのリファレンス

目次

Message Agent

内容

SQL Remote メッセージの送信と適用、およびメッセージの配信を確認するメッセージ・トラッキング・システムの管理を行います。

構文

dbremote [*options*] [*directory*]

オプション

オプション	説明

@data	指定された環境変数または設定ファイルからオプションを読み込みます。指定された環境変数と設定ファイルが両方とも存在する場合は、環境変数が使用されます。
	
	環境変数には、あらゆるオプションのセットを格納できます。たとえば、次の文の組み合わせは、4 MB のキャッシュ・サイズで起動し、メッセージだけを受信し、**myserver** というサーバの **field** というデータベースに接続するデータベース・サーバ用に、オプションのセットを格納する環境変数を設定します。**set** 文は 1 行で入力してください。
	
	`set envvar=-m 4096 -r -c "eng=myserver;dbn=field;UID=sa;PWD=sysadmin"`
	`dbremote @envvar`
	
	設定ファイルには、改行や任意のオプションを指定できます。たとえば、次のコマンド・ファイルには、4 MB のキャッシュ・サイズで起動し、メッセージだけを送信し、**myserver** というサーバの **field** というデータベースに接続する Message Agent 用のオプションのセットが格納されています。
	
	`-m 4096`
	`-s`
	`-c "eng=myserver;dbn=field;UID=sa;PWD=sysadmin"`
	
	この設定ファイルを *c:¥config.txt* として保存すると、コマンドで次のように使用できます。
	
	`dbremote @c:¥config.txt`

-a	受信したメッセージ (受信ボックス内にあるもの) を、データベースに適用しないで処理します。このオプションを使用するときに、-v (冗長出力) と -p (メッセージがパージされない) の両方を指定すると、入力メッセージの問題を検出しやすくなります。このオプションを使用するときに -p を指定しないと、メッセージを適用しないで受信ボックスがパージされるため、サブスクリプションを再開する場合に便利です。
-b	バッチ・モードで実行します。このモードでは、Message Agent は入力メッセージを処理し、トランザクション・ログを 1 回スキャンし、出力メッセージを処理して停止します。

オプション	説明
-c *"keyword=value; ⋯"*	接続パラメータを指定します。SQL Anywhere では、このオプションを指定しない場合、環境変数 SQLCONNECT が使用されます。 たとえば、次の文ではデータベース・ファイル *c:¥mydata.db* で dbremote を実行します。接続にはユーザ ID **DBA**、パスワード **sql** を使用しています。 `dbremote -c "UID=DBA;PWD=sql;DBF=c:¥mydata.db"` Message Agent を実行するユーザには、REMOTE DBA 権限または DBA 権限が必要です。 REMOTE DBA 権限の詳細については、「Message Agent とレプリケーション・セキュリティ」 129 ページを参照してください。 Message Agent は、SQL Anywhere 接続パラメータの全種類をサポートします。
-dl	[Message Agent] ウィンドウまたはコマンド・プロンプトにメッセージを表示します。指定されている場合はログ・ファイルにも出力します。
-ek *key*	このオプションを使用すると、強力に暗号化されたデータベースの暗号化キーをコマンド・プロンプトで直接指定できます。強力に暗号化されたデータベースを扱う場合には、データベースやトランザクション・ログ (オフライン・トランザクション・ログなど) を使用するのに、常に暗号化キーを使用する必要があります。強力な暗号化が適用されたデータベースの場合、-ek または -ep のどちらかを指定します。両方同時には指定できません。強力に暗号化されたデータベースでは、キーを指定しないとコマンドが失敗します。
-ep	このオプションを使用すると、暗号化キーの入力を求めるプロンプトを表示するよう指定できます。このオプションでは、暗号化キーを入力するためのダイアログ・ボックスが表示されます。クリア・テキストでは暗号化キーを見ることができないようにすることで、高いセキュリティが得られます。強力な暗号化が適用されたデータベースの場合、-ek または -ep のどちらかを指定します。両方同時には指定できません。強力に暗号化されたデータベースでは、キーを指定しないとコマンドが失敗します。 `"language_name,charset_name[,sort_order]"` デフォルトでは、Message Agent はデフォルト・ロケールを使用します。このロケールは、ファイル *sybase¥locales ¥locales.dat* に定義されています。

オプション	説明
-g *n*	*n* 個未満のオペレーションのトランザクションを、後に続くトランザクションとまとめるように Message Agent に指示します。デフォルトのオペレーション数は 20 です。*n* の値を大きくするとコミットが少なくなるため、入力メッセージの処理速度が向上します。ただし、トランザクションのサイズが大きくなると、デッドロックやブロックの原因になることもあります。 -r と -s のいずれも指定しない場合、Message Agent は 3 つのフェーズすべてを実行します。それ以外の場合、指定されたフェーズのみ実行されます。
-l *length*	送信する各メッセージの最大長をバイトで指定します。長いトランザクションは複数のメッセージに分割されます。デフォルトは 50000 バイトで、最小長は 10000 バイトです。 **警告** メッセージの最大長は、同一のインストール環境内のすべてのサイトで必ず同じ長さにしてください。 メモリの割り付けに制限のあるプラットフォームの場合、この値はオペレーティング・システムのメモリ割り付けの最大値より小さい値にしてください。
-m *size*	メッセージ作成と入力メッセージのキャッシュに、Message Agent が使用する最大メモリ・サイズを指定します。使用できるサイズは、*n* (バイト)、*n* K、*n* M で指定します。デフォルトは 2048 KB (2 MB) です。 すべてのリモート・データベースが、レプリケートされるオペレーションのユニーク・サブセットを受信する場合、それぞれのリモート・データベースにメッセージが同時に作成されます。同じオペレーションを受信するリモート・ユーザのグループには、メッセージが 1 つだけ作成されます。使用されるメモリが -m 値を超えると、メッセージの送信後に最大サイズ (-l オプションで指定したサイズ) に達します。 メッセージが届くと、適用されるまで Message Agent によってメモリ内に格納されます。このようなメッセージのキャッシュによって、適切でないメッセージをメッセージ・システムが再読み込みしないようにできますが、大規模なインストール環境ではパフォーマンスが低下することもあります。-m オプションで指定したメモリ使用量を超過すると、最低使用頻度 (LRU) 方式でメッセージがフラッシュされます。

オプション	説明
-ml *directory*	オフライン・ミラー・ログのロケーションを指定する。このオプションを指定すると、次のどちらかの場合に、dbremote が古いミラー・ログ・ファイルを削除できます。 ◆ オフライン・ミラー・ログが、ミラー・トランザクション・ログと異なるディレクトリに存在する。 ◆ dbremote がリモート・データベース・サーバと異なるマシン上で実行されている。 通常の設定では、アクティブなミラー・ログと名前が変更されたミラー・トランザクション・ログは同じディレクトリ内に存在し、dbremote はリモート・データベースと同じマシン上で実行されるため、このオプションを指定しなくても、古いミラー・ログ・ファイルは自動的に削除されます。このディレクトリ内のトランザクション・ログが影響を受けるのは、delete_old_logs データベース・オプションが On または DELAY に設定されている場合だけです。
-o *file*	メッセージをファイルに出力する。出力をログ・ファイルに追加します。デフォルトでは、画面に出力が表示されます。
-os *size*	出力メッセージを記録するファイルの最大サイズ出力メッセージをロギングするファイルの最大サイズを指定します。使用できるサイズは、*n* (バイト)、*n***K** (KB)、または *n***M** (MB) で指定します。デフォルトによる制限はなく、最小制限は 10000 バイトです。 SQL Remote では、現在のファイル・サイズをチェックしてから、出力メッセージをファイルに記録します。ログ・メッセージによって、ファイル・サイズが指定したサイズより大きくなると、SQL Remote は出力ファイルの名前をそれぞれ *yymmddxx.dbr* に変更します。*xx* は AA ー ZZ のアルファベット順の英字で、*yymmdd* は現在の年月日を表します。 Message Agent を継続モードで長時間実行する場合、このオプションを使用して、手動で古いログ・ファイルを削除し、ディスク領域を解放できます。
-ot *file*	ログ・ファイルをトランケートし、このファイルに出力メッセージを追加します。デフォルトでは、画面に出力が表示されます。
-p	メッセージをパージしない。メッセージをパージしないで処理します。
-q	ウィンドウ操作のオペレーティング・システムの場合にのみ、Message Agent を最小化ウィンドウで起動します。
-qc	完了後、ウィンドウを閉じます。

オプション	説明
-r	メッセージを受信します。-r、-l、-s のいずれも指定しない場合、Message Agent は 3 つのフェーズすべてを実行します。それ以外の場合、指定されたフェーズのみ実行されます。 -r で呼び出された場合、Message Agent は継続モードで動作します。メッセージ受信後に Message Agent を停止するには、-r オプションと -b オプションを使用します。
-rd *minutes*	入力メッセージのポーリング頻度デフォルトでは、Message Agent は入力メッセージを 1 分ごとにポーリングします。このオプション (rd は「**受信遅延 (receive delay)**」の意味) によってポーリング頻度を設定できます。これは、ポーリングが高負荷である場合に便利です。 頻繁にポーリングする場合は、秒数を表す数の後にサフィックス **s** を付けると便利です。次に例を示します。 dbremote -rd 30s このように指定すると、30 秒ごとにポーリングされます。 「受信メッセージのポーリングのチューニング」 116 ページを参照してください。
-ro *filename*	ファイルにリモート出力を記録する。このオプションは統合サイトで使用します。統合データベースに出力ログ情報を送信するようにリモート・データベースを設定する場合、このオプションを使用すると情報がファイルに書き込まれます。このオプションを指定すると、管理者がリモート・サイトでエラーのトラブルシューティングを行う場合に役立ちます。 「リモート・サイトでのトラブルシューティング・エラー」 114 ページを参照してください。
-rp *number*	メッセージを消失したと判断するまでのポーリング受信回数 Message Agent を継続モードで実行すると、メッセージが一定の間隔でポーリングされます。設定回数 (デフォルトでは 1 回) のポーリングが行われた後にメッセージが見つからないと、Message Agent はそのメッセージが失われたと判断して、再送信を要求します。このため、メッセージ・システムの処理速度が遅い場合、必要のない再送信要求が多く出されることがあります。このオプションを使用して、再送信要求が出される前に行うポーリング回数を指定すると、再送信要求の数を少なくできます。 このオプションの設定方法については、「受信メッセージのポーリングのチューニング」 116 ページを参照してください。
-rt *filename*	トランケートし、ファイルにリモート出力を記録する。このオプションは統合サイトで使用します。ファイルが起動時にトランケートされることを除いて、-ro オプションと同じです。

オプション	説明
-ru *time*	再送信の宛先のログを再スキャンする待ち時間 「**再送信緊急度 (resend urgency)**」を制御します。再送信要求の検出から、Message Agent がこの要求の処理を開始するまでの時間を指定します。このオプションを使用すると、Message Agent が複数のユーザの再送信要求を集めてからログの再スキャンを行うことができます。指定できる時間の単位は {s = 秒、m = 分、h = 時間、d = 日数} のいずれかです。
-s	メッセージを送信します。-r、-l、-s のいずれも指定しない場合、Message Agent は 3 つのフェーズすべてを実行します。それ以外の場合、指定されたフェーズのみ実行されます。
-sd *time*	「**送信遅延 (send delay)**」を制御します。これは、ポーリングとポーリングの間に送信されるトランザクション・ログ・データを待つ時間です。
-t	すべてのトリガをレプリケートする。すべてのトリガ・アクションがレプリケートされます。このオプションを使用する場合は、リモート・データベースでトリガ・アクションが 2 回実行されないことを確認してください。この 2 回とは、リモート・サイトで起動されるトリガによる 1 回と、統合データベースからレプリケートされたアクションの明示的な適用による 1 回です。 トリガ・アクションが 2 回実行されないようにするには、トリガの本文を IF CURRENT REMOTE USER IS NULL … END IF 文で囲みます。このオプションは、SQL Anywhere でのみ使用できます。
-u	バックアップされたトランザクションだけを処理します。このオプションを指定すると、最後にバックアップされた以降のトランザクションは処理されません。これにより、出力トランザクションと入力トランザクションの確認とが、バックアップされてから送信されるようになります。 名前が変更されたログのトランザクションだけが処理されます。
-ud	UNIX プラットフォーム上では、-ud オプションを指定して、Message Agent をデーモンとして実行できます。Message Agent をデーモンとして実行する場合は、-o または -ot オプションも指定して、出力情報のログを取ってください。 Message Agent をデーモンとして実行し、FTP または SMTP メッセージ・リンクを使用する場合は、データベースにメッセージ・リンク・パラメータを格納してください。これは、Message Agent をデーモンとして実行していると、これらのオプションの入力をユーザに要求しないためです。 メッセージ・リンク・パラメータの詳細については、「メッセージ・タイプ制御パラメータの設定」 100 ページを参照してください。

オプション	説明
-ux	Solaris と Linux 上で、コンソール・ウィンドウを開く。 -ux が指定されている場合、dbremote は使用可能な表示を見つけます。たとえば、DISPLAY 環境変数が設定されていなかったり、X-Window サーバが実行されていなかったりしたために、使用可能な表示が見つからなかった場合、dbremote は起動できません。Windows では、コンソールが自動的に開きます。
-v	冗長出力です。このオプションによって、メッセージに含まれる SQL 文が画面に表示されます。-o または -ot オプションを指定するとログ・ファイルに出力されます。
-w *n*	入力メッセージを適用するワーカ・スレッドの数 (NetWare と Windows CE 以外) 受信メッセージの適用に使用されるワーカ・スレッドの数です。デフォルトは 0 です。この場合、すべてのメッセージがメインの (1 つだけの) スレッドによって適用されます。1 の値を指定すると、1 つのスレッドがメッセージ・システムからメッセージを受信し、1 つのスレッドがメッセージをデータベースに適用します。 -w オプションを指定すると、ハードウェアのアップグレードによって、受信メッセージのスループットを増加できます。多くのオペレーションを同時に実行できるデバイスに統合データベースを配置すると、入力メッセージのスループットが向上されます。このような統合データベースの配置方法をストライプ論理ドライブの RAID アレイといいます。また、Message Agent を実行するコンピュータにマルチ・プロセッサを搭載しても、入力メッセージのスループットは向上します。 多くのオペレーションを同時に実行できないハードウェアでは、-w オプションを使用してもパフォーマンスはそれほど向上しません。 単一のリモート・データベースからの入力メッセージを複数のスレッドに適用できません。単一のリモート・データベースからのメッセージは、常に適切な順序で逐次適用されます。
-x [*size*]	出力メッセージがスキャンされてから、トランザクション・ログの名前を変更し、再起動します。場合によっては、リモート・データベースのバックアップが実行されたり、データベース・サーバをシャットダウンするときにトランザクション・ログの名前を変更する代わりに、統合データベースにデータがレプリケートされます。このオプションは、SQL Anywhere でのみ使用できます。 オプションの *size* 修飾子が指定された場合、トランザクション・ログは、指定されたサイズよりも大きい場合にのみ名前を変更されます。使用できるサイズは、*n* (バイト)、*n*K、*n*M で指定します。デフォルトは 0 です。
directory	古いトランザクション・ログが保存されるディレクトリ

説明

Message Agent は、SQL Remote レプリケーションのメッセージの送信と適用、およびメッセージの配信を確認するメッセージ・トラッキング・システムの管理を行います。

Message Agent の実行プログラム名は dbremote です。

DBTools ライブラリに呼び出すと、自分のアプリケーションから Message Agent を実行することもできます。詳細については、SQL Remote インストール・ディレクトリの h サブディレクトリにある *dbrmt.h* ファイルを参照してください。

Message Agent コマンドのユーザ ID には REMOTE DBA か DBA 権限が必要です。

オプションの *directory* パラメータは、古いトランザクション・ログが保存されているディレクトリを指定します。これにより、Message Agent は現在のログが開始される前のイベントにアクセスできます。

Message Agent はいくつかのデータベース接続を使用します。「Message Agent で使用される接続」 112 ページを参照してください。

REMOTE DBA 権限の詳細については、「Message Agent とレプリケーション・セキュリティ」 129 ページを参照してください。

メッセージ・システム制御パラメータ

SQL Remote は、複数のレジストリ設定を使用してメッセージ・リンク動作を制御します。

メッセージ・リンク制御パラメータは、次の場所に格納されます。

- **Windows**　レジストリ内の次の場所に格納されます。

  ```
  ¥¥HKEY_CURRENT_USER
    ¥Software
      ¥Sybase
        ¥SQL Remote
  ```

- **NetWare**　FILE システム・ディレクトリ設定を保存するために、*sys:¥system* ディレクトリにファイル *dbremote.ini* を作成してください。

レジストリ設定のリストについては、「メッセージ・タイプの使用」 97 ページの各メッセージ・システムの項を参照してください。

データベース抽出ユーティリティ

次の方法で、リモート・データベース抽出ユーティリティにアクセスできます。

- 対話的に使うには、Sybase Central からアクセスする。
- dbxtract ユーティリティを使って、システム・コマンド・プロンプトからアクセスする。バッチまたはコマンド・ファイルへの組み込みには、このユーティリティが便利です。

デフォルトでは、抽出ユーティリティは独立性レベル 0 で実行されます。アクティブなサーバからデータベースを抽出する場合は、独立性レベル 3 で実行し、抽出されたデータベース内のデータがサーバ上のデータと一致するようにします。独立性レベル 3 で実行すると、多数のロックが必要になるため、サーバ上の他のユーザのターンアラウンド・タイムに影響が出ることがあります。サーバへのアクセスが少ないときに抽出ユーティリティを実行するか、データベースのコピーに対して抽出ユーティリティを実行することをおすすめします。

次の項を参照してください。

- 「抽出ユーティリティ」 163 ページ
- 「効率的な抽出プロシージャの設計」 83 ページ

dbo によって所有されるオブジェクト

抽出ユーティリティでは、データベース作成時に **dbo** ユーザ ID 用に作成されたオブジェクトをアンロードしません。データをアンロードするとき、システム・プロシージャの再定義など、これらのオブジェクトに加えられた変更は失われます。抽出ユーティリティでアンロードされるため、データベースの初期化以降に **dbo** ユーザ ID によって作成されたオブジェクトは保存されます。

Sybase Central でのリモート・データベースの抽出

Sybase Central から抽出ユーティリティを実行すると、SQL Remote サブスクリプションの作成と同期に関連する次のタスクを実行できます。

- 指定したパブリケーションのデータのコピーを含むリモート・データベースを構築するため、コマンド・ファイルを作成する。
- メッセージ・タイプ、パブリッシャ・ユーザ ID およびリモート・ユーザ ID、パブリケーションおよびサブスクリプションなどの SQL Remote オブジェクトを作成する。これらのオブジェクトは、統合データベースとの間でメッセージの送受信を行うため、リモート・データベースで必要なものです。
- 統合データベースとリモート・データベースの両方でサブスクリプションを開始する。

注意
[データベースの抽出] ウィザードには、[所有者別にオブジェクトをフィルタ] ダイアログで選択したユーザのテーブルだけが表示されます。特定のデータベース・ユーザに属するテーブルを表示させたい場合、アンロードするデータベースを右クリックし、ポップアップ・メニューから [所有者別にオブジェクトをフィルタ] を選択して、表示されるダイアログから対象ユーザを選択します。

♦ 実行中のデータベースからリモート・データベースを抽出する場合は、次の手順に従います (Sybase Central)。

1. データベースに接続します。
2. [ツール] メニューから、[SQL Anywhere 10] – [データベースの抽出] の順に選択します。
   [データベースの抽出] ウィザードが表示されます。
3. ウィザードの指示に従います。

抽出ユーティリティ

SQL Anywhere の統合データベースから SQL Anywhere のリモート・データベースを抽出します。

構文

dbxtract [*options*] [*directory*] *subscriber*

抽出ユーティリティのオプション

オプション	説明
@data	設定ファイルからオプションを読み出す。「@data サーバ・オプション」『SQL Anywhere サーバ - データベース管理』を参照してください。 このオプションを使用すると、指定された環境変数または設定ファイルからオプションを読み込むことができます。同じ名前の環境変数と設定ファイルが両方存在する場合は、環境変数が使用されます。 「設定ファイルの使用」『SQL Anywhere サーバ - データベース管理』を参照してください。 設定ファイル内のパスワードなどの情報を保護する場合は、ファイル非表示ユーティリティを使用して、設定ファイルの内容を難読化できます。 「ファイル非表示ユーティリティ (dbfhide)」『SQL Anywhere サーバ - データベース管理』を参照してください。

オプション	説明
-ac *"keyword=value; ⋯"*	接続文字列で指定したデータベースに接続して、再ロードする。 このオプションを使用すると、データベースのアンロード処理と、既存データベースへの結果の再ロード処理を組み合わせることができます。 たとえば、次のコマンド (1 行で入力) は、field_user サブスクライバのデータのコピーを既存のデータベース・ファイル *c:¥newdata.db* にロードします。 `dbxtract -c "UID=DBA;PWD=sql;DBF=c:¥olddata.db" -ac "UID=DBA;PWD=sql;DBF=c:¥newdata.db" field_user` このオプションを使用した場合、データのコピーはディスク上に作成されないため、コマンドでアンロード用ディレクトリを指定する必要はありません。これによりデータのセキュリティは高まりますが、パフォーマンスは多少悪くなります。
-al*filename*	新しいデータベースのログ・ファイル名を指定する。
-an *database*	アンロードするデータベースと同じ設定でデータベース・ファイルを作成し、それを自動的に再ロードする。 このオプションを使用すると、データベースのアンロード、新規データベースの作成、データのロードを組み合わせて実行できます。 たとえば、次のコマンド (1 行で入力) は、新規のデータベース・ファイル *c:¥mydatacopy.db* を作成し、そこに *c:¥mydata.db* の field_user サブスクライバのスキーマとデータをコピーします。 `dbxtract -c "UID=DBA;PWD=sql;DBF=c:¥mydata.db" -an c:¥mydatacopy.db field_user` このオプションを使用した場合、データのコピーはディスク上に作成されないため、コマンドでアンロード用ディレクトリを指定する必要はありません。これによりデータのセキュリティは高まりますが、パフォーマンスは多少低下します。
-b	サブスクリプションを開始しない。このオプションを指定した場合は、統合データベース (リモート・データベース用) とリモート・データベース (統合データベース用) で、レプリケーションを開始するために START SUBSCRIPTION 文を使用して、サブスクリプションを明示的に開始しなければいけません。「START SUBSCRIPTION 文 [SQL Remote]」『SQL Anywhere サーバ - SQL リファレンス』を参照してください。

オプション	説明
-c *"keyword=value; ⋯"*	文字列でデータベース接続パラメータを指定する。 ◆ **接続パラメータ** ユーザはデータベースの全テーブル上にパーミッションを持っている必要があるため、user ID には DBA 権限を持つユーザ ID を指定してください。 たとえば、次の文 (1 行で入力) は、ユーザ ID が DBA、パスワードが sql で接続している sample_server サーバ上で実行しているサンプル・データベースから、リモート・ユーザ ID joe_remote のデータベースを抽出します。データは *c:¥extract* ディレクトリにアンロードされます。 `dbxtract -c "ENG=sample_server;DBN=demo; UID=DBA;PWD=sql" c:¥extract joe_remote` 接続パラメータを指定しない場合、SQLCONNECT 環境変数が設定されていると、SQLCONNECT 環境変数からの接続パラメータを使用します。
-d	データのみをアンロードする。このオプションを指定すると、スキーマ定義はアンロードされません。また、リモート・データベースに対するパブリケーションとサブスクリプションも作成されません。このオプションは、適切なスキーマのあるリモート・データベースがすでに存在していて、データを格納するためだけに使用します。
-ea *alg*	新しいデータベースで使用する暗号化アルゴリズムを指定する。このオプションにより、新しいデータベースの暗号化に強力な暗号化アルゴリズムを選択できます。AES (デフォルト) または AES_FIPS (FIPS 承認のアルゴリズム) のどちらかを選択できます。AES_FIPS は個別のライブラリを使用するため、AES との互換性はありません。アルゴリズム名の大文字と小文字は区別されません。 -ea オプションを指定する場合は、-ep または -ek のどちらかのオプションも一緒に指定する必要があります。「強力な暗号化」『SQL Anywhere サーバ - データベース管理』を参照してください。 **別途ライセンスが必要な必須コンポーネント** ECC 暗号化と FIPS 承認の暗号化には、別途ライセンスが必要です。強力な暗号化テクノロジはすべて、輸出規制対象品目です。「別途ライセンスが必要なコンポーネント」『SQL Anywhere 10 - 紹介』を参照してください。

オプション	説明
-ek *key*	新しいデータベースで使用する暗号化キーを指定する。このオプションを使用すると、コマンドに暗号化キーを直接指定することで、強力に暗号化されたデータベースを作成できます。データベースの暗号化に使用されるアルゴリズムは、-ea オプションで指定した AES または AES_FIPS です。 -ek オプションを指定して、-ea オプションを指定しないと、AES アルゴリズムが使用されます。 **警告** キーは保護してください。キーのコピーは、安全な場所に保管してください。キーを紛失すると、データベースにまったくアクセスできなくなり、リカバリも不可能になります。
-ep	新しいデータベースで使用する暗号化キーの入力を求める。このオプションを使用すると、ダイアログ・ボックスに暗号化キーを入力することで、強力に暗号化されたデータベースを作成するように指定できます。クリア・テキストでは暗号化キーを見ることができないようにすることで、高いセキュリティが得られます。 暗号化キーは、正確に入力されたことを確認するために 2 回入力してください。キーが一致しない場合は、初期化は失敗します。「強力な暗号化」『SQL Anywhere サーバ - データベース管理』を参照してください。
-f	完全に修飾されたパブリケーションを抽出する。ほとんどの場合、完全に修飾されたパブリケーション定義をリモート・データベース用に抽出する必要はありません。通常、すべてのローはレプリケートされ、統合データベースに戻されます。 しかし、多層の設定や、統合データベースにないローがリモート・データベースにある設定では、完全に修飾されたパブリケーションが必要な場合もあります。
-ii	内部アンロードと内部再ロードを実行する。このオプションを使用すると、再ロード・スクリプトは、データのアンロードとロードそれぞれに対して、Interactive SQL の OUTPUT 文と INPUT 文ではなく、内部の UNLOAD 文と LOAD TABLE 文を強制的に使用します。 このオペレーションの組み合わせがデフォルトの動作です。 データ・ファイルのパスには、外部オペレーションでは dbxtract の現在の作業ディレクトリからの相対パスを使用し、内部の文ではサーバからの相対パスを使用します。

オプション	説明
-ix	内部アンロードと外部再ロードを実行する。このオプションを使用すると、再ロード・スクリプトは、データのアンロードに対して内部の UNLOAD 文を強制的に使用し、新しいデータベースへのデータのロードに対して Interactive SQL の INPUT 文を強制的に使用します。 データ・ファイルのパスには、外部オペレーションでは dbxtract の現在の作業ディレクトリからの相対パスを使用し、内部の文ではサーバからの相対パスを使用します。
-l *level*	指定した独立性レベルですべての抽出オペレーションを実行する。デフォルト設定では、独立性レベルは 0 です。アクティブなサーバからデータベースを抽出する場合は、独立性レベル 3 で実行し、抽出されたデータベース内のデータがサーバ上のデータと一致するようにします。独立性レベルを大きくすると、抽出ユーティリティが多数のロックを使用することになり、他のユーザによるデータベースの使用が制限される可能性があります。「抽出ユーティリティ」 163 ページを参照してください。
-n	スキーマ定義のみ抽出する。この定義を指定すると、データはアンロードされません。再ロード・ファイルには、データベース構造体だけを構築する SQL 文が記述されています。SYNCHRONIZE SUBSCRIPTION 文を使用すると、メッセージ・システム全体のデータをロードできます。パブリケーション、サブスクリプション、PUBLISH パーミッション、SUBSCRIBE パーミッションは、スキーマの一部です。
-o *file*	メッセージをファイルに出力する。
-p *character*	エスケープ文字を指定する。このオプションを使用して、デフォルトのエスケープ文字 (¥) を別の文字に置き換えることができます。
-q	クワイエット・モードで処理を実行し、メッセージまたはウィンドウの表示を行わない。このオプションは -y オプションと一緒に指定してください。そうしないと操作は失敗します。 このオプションは他の環境からは使用できません。コマンド・ライン・ユーティリティからのみ使用できます。
-r *file*	生成された再ロード Interactive SQL コマンド・ファイルの名前を指定する。 再ロード・コマンド・ファイルのデフォルト名は、現在のディレクトリの *reload.sql* です。このオプションを使用して異なるファイル名を指定できます。
-u	アンロード中にデータの順序を変更しない。デフォルトでは、各テーブルのデータはプライマリ・キーを基準に順序付けられます。-u オプションを使用するとアンロード処理は高速になりますが、リモート・データベースへのデータのロード処理は遅くなります。

オプション	説明
-v	冗長メッセージを表示する。アンロードされているテーブル名、アンロードされたロー数、使用された SELECT 文が表示されます。
-xf	外部キーを除外する。リモート・データベースに統合データベース・スキーマのサブセットがあり、いくつかの外部キー参照がない場合に、このオプションを使用できます。
-xh	プロシージャ・フックを除外する。
-xi	外部アンロードと内部再ロードを実行する。データベースのアンロードでは、デフォルトの動作として UNLOAD 文を使用します。これはデータベース・サーバが実行します。外部アンロードを選択すると、dbxtract は UNLOAD 文の代わりに OUTPUT 文を使用します。OUTPUT 文はクライアントで実行されます。 データ・ファイルのパスには、外部オペレーションでは dbxtract の現在の作業ディレクトリからの相対パスを使用し、内部の文ではサーバからの相対パスを使用します。
-xp	データベースからストアド・プロシージャを抽出しません。
-xt	データベースからトリガを抽出しません。
-xv	データベースからビューを抽出しません。
-xx	外部アンロードと外部ロードを実行する。データをアンロードする場合には、OUTPUT 文が使用されます。また、新規データベースにデータをロードする場合には、INPUT 文が使用されます。 アンロードのデフォルト動作では UNLOAD 文が使用され、ロードのデフォルト動作では LOAD TABLE 文が使用されます。内部の UNLOAD 文と LOAD TABLE 文を使用すると、OUTPUT 文と INPUT 文よりも高速で処理します。 データ・ファイルのパスには、外部オペレーションでは dbxtract の現在の作業ディレクトリからの相対パスを使用し、内部の文ではサーバからの相対パスを使用します。
-y	確認メッセージを表示せずにコマンド・ファイルを上書きする。このオプションを指定しないと、既存のコマンド・ファイルを置き換えるときに、確認メッセージが表示されます。
directory	ファイルが書き込まれたディレクトリを指定する。-an または -ac を指定する場合は不要です。
subscriber	データベースを抽出するサブスクライバを指定する。

戻り値

抽出ユーティリティは、コマンド・ファイルと、一連の関連データ・ファイルを作成します。新しく初期化したデータベースに対してコマンド・ファイルを実行し、データベース・オブジェクトを作成してリモート・データベース用のデータをロードできます。

デフォルトのコマンド・ファイル名は *reload.sql* です。

リモート・ユーザがグループの場合、グループのメンバのユーザ ID すべてを抽出します。これにより、リモート・データベースの複数のユーザに対して異なるユーザ ID を使用できます。カスタム抽出処理は必要ありません。

SQL Remote オプション

機能

レプリケーション・オプションは、レプリケーション動作を制御するためのデータベース・オプションです。

構文

SET [**TEMPORARY**] **OPTION**
[*userid*. | **PUBLIC**.]*option-name* = [*option-value*]

パラメータ

引数	説明
option-name	変更されるオプションの名前
option-value	オプションの設定が含まれる文字列

説明

これらのオプションは Message Agent が使用します。Message Agent のコマンドで指定されたユーザ ID 用に設定してください。一般的な public の使用にも設定できます。

次のオプションを使用できます。

オプション	値	デフォルト
「blob_threshold オプション [SQL Remote]」『SQL Anywhere サーバ - データベース管理』	整数 (バイト)	256
「compression オプション [SQL Remote]」『SQL Anywhere サーバ - データベース管理』	-1 ー 9 の整数	6
「delete_old_logs オプション [Mobile Link クライアント] [SQL Remote] [Replication Agent]」『SQL Anywhere サーバ - データベース管理』	On、Off、Delay、*n* 日 (日数)	Off
「external_remote_options [SQL Remote]」『SQL Anywhere サーバ - データベース管理』	On、Off	Off
「qualify_owners オプション [SQL Remote]」『SQL Anywhere サーバ - データベース管理』	On、Off	On

オプション	値	デフォルト
「quote_all_identifiers オプション [SQL Remote]」『SQL Anywhere サーバ - データベース管理』	On、Off	Off
「replication_error オプション [SQL Remote]」『SQL Anywhere サーバ - データベース管理』	ストアド・プロシージャ名	(プロシージャなし)
「replication_error_piece オプション [SQL Remote]」『SQL Anywhere サーバ - データベース管理』	ストアド・プロシージャ名	(プロシージャなし)
「save_remote_passwords オプション [SQL Remote]」『SQL Anywhere サーバ - データベース管理』	On、Off	On
「sr_date_format オプション [SQL Remote]」『SQL Anywhere サーバ - データベース管理』	*日付文字列*	*YYYY*/*MM*/*DD*
「sr_time_format オプション [SQL Remote]」『SQL Anywhere サーバ - データベース管理』	*時刻文字列*	*HH*:*NN*:*SS*.*SSSSSS*
「sr_timestamp_format [SQL Remote]」『SQL Anywhere サーバ - データベース管理』	*タイムスタンプ文字列*	*YYYY*/*MM*/*DD* *HH*:*NN*:*SS*.*SSSSSS*
「subscribe_by_remote オプション [SQL Remote]」『SQL Anywhere サーバ - データベース管理』	On、Off	On
「verify_all_columns オプション [SQL Remote]」『SQL Anywhere サーバ - データベース管理』	On、Off	Off
「verify_threshold オプション [SQL Remote]」『SQL Anywhere サーバ - データベース管理』	整数 (バイト)	1000

SQL Remote イベント・フック・プロシージャ

以下のストアド・プロシージャ名と引数は、SQL Remote データベースで同期をカスタマイズするためのインタフェースを提供します。

注意

特に明記しないかぎり、イベント・フック・プロシージャには次の条件が適用されます。

- ストアド・プロシージャには DBA 権限が必要です。
- プロシージャでは、オペレーションのコミットとロールバックはできません。また、暗黙的なコミットを実行するアクションも実行できません。プロシージャのアクションは、呼び出し側のアプリケーションによって自動的にコミットされます。
- Message Agent の冗長モードをオンにすると、フックのトラブルシューティングができます。

#Hook_dict テーブル

#hook_dict テーブルは、次の CREATE 文を使用して、フックが呼び出される直前に作成されます。

```
CREATE table #hook_dict(
name VARCHAR(128) NOT NULL UNIQUE,
value VARCHAR(255) NOT NULL )
```

Message Agent は #hook_dict テーブルを使用して、値をフック関数に渡します。フック関数は #hook_dict テーブルを使用して、値を Message Agent に返します。

sp_hook_dbremote_begin

機能

このストアド・プロシージャを使用すると、レプリケーション処理の開始時にカスタム・アクションを追加できます。

#hook_dict テーブルのロー

名前	値	説明
send	**true** または **false**	処理がレプリケーションの送信フェーズを実行中かどうかを示す
receive	**true** または **false**	処理がレプリケーションの受信フェーズを実行しているかどうかを示す

説明

この名前のプロシージャが存在する場合、プロシージャは、Message Agent が起動するときに呼び出されます。

sp_hook_dbremote_end

機能

このストアド・プロシージャを使用すると、Message Agent の終了直前にカスタム・アクションを追加できます。

#hook_dict テーブルのロー

名前	値	説明
send	**true** または **false**	処理がレプリケーションの送信フェーズを実行中かどうかを示す
receive	**true** または **false**	処理がレプリケーションの受信フェーズを実行しているかどうかを示す
exit code	**integer**	0 以外の終了コードはエラーを示す

説明

この名前のプロシージャが存在する場合、プロシージャは、Message Agent が停止する前の最後のイベントとして呼び出されます。

sp_hook_dbremote_shutdown

機能

このストアド・プロシージャを使用すると、Message Agent のシャットダウンを開始できます。

#hook_dict テーブルのロー

名前	値	説明
send	**true** または **false**	処理がレプリケーションの送信フェーズを実行中かどうかを示す
receive	**true** または **false**	処理がレプリケーションの受信フェーズを実行しているかどうかを示す
shutdown	**true** または **false**	プロシージャが呼び出されるときにはこのローは **false** です。プロシージャがこのローを **true** に更新すると、Message Agent がシャットダウンされます。

説明

この名前のプロシージャが存在する場合、プロシージャは、Message Agent がメッセージを送受信していないときに呼び出され、Message Agent のフックで開始されるシャットダウンを可能にします。

sp_hook_dbremote_receive_begin

機能

このストアド・プロシージャを使用すると、レプリケーションの受信フェーズの開始前にアクションを実行できます。

#hook_dict テーブルのロー

なし

sp_hook_dbremote_receive_end

機能

このストアド・プロシージャを使用すると、レプリケーションの受信フェーズの終了後にアクションを実行できます。

#hook_dict テーブルのロー

なし

sp_hook_dbremote_send_begin

機能

このストアド・プロシージャを使用すると、レプリケーションの送信フェーズの開始前にアクションを実行できます。

#hook_dict テーブルのロー

なし

sp_hook_dbremote_send_end

機能

このストアド・プロシージャを使用すると、レプリケーションの送信フェーズの終了後にアクションを実行できます。

#hook_dict テーブルのロー

なし

sp_hook_dbremote_message_sent

機能

このストアド・プロシージャを使用すると、任意のメッセージが送信された後にアクションを実行できます。

#hook_dict テーブルのロー

名前	値
remote user	メッセージの送信先

sp_hook_dbremote_message_missing

機能

このストアド・プロシージャを使用すると、Message Agent がリモート・ユーザからの 1 つ以上のメッセージが見つからないと判断した場合にアクションを実行できます。

#hook_dict テーブルのロー

名前	値
remote user	メッセージを再送する必要があるリモート・ユーザの名前

sp_hook_dbremote_message_apply_begin

機能

このストアド・プロシージャを使用すると、Message Agent がユーザからの一連のメッセージを適用する直前にアクションを実行できます。

#hook_dict テーブルのロー

名前	値
remote user	適用されるメッセージを送信したリモート・ユーザの名前

sp_hook_dbremote_message_apply_end

機能

このストアド・プロシージャを使用すると、Message Agent がユーザからの一連のメッセージを適用した直後にアクションを実行できます。

#hook_dict テーブルのロー

名前	値
remote user	適用されたメッセージを送信したリモート・ユーザの名前

第 8 章

SQL Remote システム・オブジェクト

目次

SQL Remote システム・テーブル

SQL Remote のシステム情報は、SQL Anywhere カタログに保持されます。この情報をより分かりやすい形にしたものが、一連のシステム・ビューに保持されます。次に、SQL Remote のデータにアクセスするのに使用できるビューを示します。

- 「SYSARTICLE システム・ビュー」『SQL Anywhere サーバ - SQL リファレンス』
- 「SYSARTICLECOL システム・ビュー」『SQL Anywhere サーバ - SQL リファレンス』
- 「SYSPUBLICATION システム・ビュー」『SQL Anywhere サーバ - SQL リファレンス』
- 「SYSREMOTEOPTION システム・ビュー」『SQL Anywhere サーバ - SQL リファレンス』
- 「SYSREMOTEOPTIONTYPE システム・ビュー」『SQL Anywhere サーバ - SQL リファレンス』
- 「SYSREMOTETYPE システム・ビュー」『SQL Anywhere サーバ - SQL リファレンス』
- 「SYSREMOTEUSER システム・ビュー」『SQL Anywhere サーバ - SQL リファレンス』
- 「SYSSUBSCRIPTION システム・ビュー」『SQL Anywhere サーバ - SQL リファレンス』

第 9 章

SQL Remote の SQL 文

目次

SQL Remote 文

以下に、SQL Remote のコマンドの実行に使用される SQL 文を示します。

- 「ALTER REMOTE MESSAGE TYPE 文 [SQL Remote]」『SQL Anywhere サーバ - SQL リファレンス』
- 「CREATE PUBLICATION 文 [Mobile Link] [SQL Remote]」『SQL Anywhere サーバ - SQL リファレンス』
- 「CREATE REMOTE MESSAGE TYPE 文 [SQL Remote]」『SQL Anywhere サーバ - SQL リファレンス』
- 「CREATE SUBSCRIPTION 文 [SQL Remote]」『SQL Anywhere サーバ - SQL リファレンス』
- 「CREATE TRIGGER 文」『SQL Anywhere サーバ - SQL リファレンス』
- 「DROP PUBLICATION 文 [Mobile Link] [SQL Remote]」『SQL Anywhere サーバ - SQL リファレンス』
- 「DROP REMOTE MESSAGE TYPE 文 [SQL Remote]」『SQL Anywhere サーバ - SQL リファレンス』
- 「DROP SUBSCRIPTION 文 [SQL Remote]」『SQL Anywhere サーバ - SQL リファレンス』
- 「GRANT CONSOLIDATE 文 [SQL Remote]」『SQL Anywhere サーバ - SQL リファレンス』
- 「GRANT PUBLISH 文 [SQL Remote]」『SQL Anywhere サーバ - SQL リファレンス』
- 「GRANT REMOTE 文 [SQL Remote]」『SQL Anywhere サーバ - SQL リファレンス』
- 「GRANT REMOTE DBA 文 [Mobile Link] [SQL Remote]」『SQL Anywhere サーバ - SQL リファレンス』
- 「PASSTHROUGH 文 [SQL Remote]」『SQL Anywhere サーバ - SQL リファレンス』
- 「REMOTE RESET 文 [SQL Remote]」『SQL Anywhere サーバ - SQL リファレンス』
- 「REVOKE CONSOLIDATE 文 [SQL Remote]」『SQL Anywhere サーバ - SQL リファレンス』
- 「REVOKE PUBLISH 文 [SQL Remote]」『SQL Anywhere サーバ - SQL リファレンス』
- 「REVOKE REMOTE 文 [SQL Remote]」『SQL Anywhere サーバ - SQL リファレンス』
- 「REVOKE REMOTE DBA 文 [SQL Remote]」『SQL Anywhere サーバ - SQL リファレンス』
- 「SET REMOTE OPTION 文 [SQL Remote]」『SQL Anywhere サーバ - SQL リファレンス』
- 「START SUBSCRIPTION 文 [SQL Remote]」『SQL Anywhere サーバ - SQL リファレンス』
- 「STOP SUBSCRIPTION 文 [SQL Remote]」『SQL Anywhere サーバ - SQL リファレンス』
- 「SYNCHRONIZE SUBSCRIPTION 文 [SQL Remote]」『SQL Anywhere サーバ - SQL リファレンス』
- 「UPDATE 文 [SQL Remote]」『SQL Anywhere サーバ - SQL リファレンス』

索引

記号

A

C

D

E

F

G

H

I

L

M

N

O

P

R

S

U

V

W

あ

い

す

せ

そ

た

ち

て

と

ゆ

り

れ

ろ

