SYBASE iAnywhere

SQL Remote®

2009 年 2 月

バージョン 11.0.1

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。

SQL Anywhere のマニュアルについて

SQL Anywhere の完全なマニュアルは 4 つの形式で提供されており、いずれも同じ情報が含まれています。

- **HTML ヘルプ**　オンライン・ヘルプには、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。

  Microsoft Windows オペレーティング・システムを使用している場合は、オンライン・ヘルプは HTML ヘルプ (CHM) 形式で提供されます。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 11] - [マニュアル] - [オンライン・マニュアル]** を選択します。

  管理ツールのヘルプ機能でも、同じオンライン・マニュアルが使用されます。

- **Eclipse**　UNIX プラットフォームでは、完全なオンライン・ヘルプは Eclipse 形式で提供されます。マニュアルにアクセスするには、SQL Anywhere 11 インストール環境の *bin32* または *bin64* ディレクトリから *sadoc* を実行します。

- **DocCommentXchange**　DocCommentXchange は、SQL Anywhere マニュアルにアクセスし、マニュアルについて議論するためのコミュニティです。

  DocCommentXchange は次の目的に使用できます (現在のところ、日本語はサポートされておりません)。

  ○ マニュアルを表示する

  ○ マニュアルの項目について明確化するために、ユーザによって追加された内容を確認する

  ○ すべてのユーザのために、今後のリリースでマニュアルを改善するための提案や修正を行う

  http://dcx.sybase.com を参照してください。

- **PDF**　SQL Anywhere の完全なマニュアル・セットは、Portable Document Format (PDF) 形式のファイルとして提供されます。内容を表示するには、PDF リーダが必要です。Adobe Reader をダウンロードするには、http://get.adobe.com/reader/ にアクセスしてください。

  Microsoft Windows オペレーティング・システムで PDF 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 11] - [マニュアル] - [オンライン・マニュアル - PDF]** を選択します。

  UNIX オペレーティング・システムで PDF マニュアルにアクセスするには、Web ブラウザを使用して *install-dir/documentation/ja/pdf/index.html* を開きます。

マニュアル・セットに含まれる各マニュアルについて

SQL Anywhere のマニュアルは次の構成になっています。

- **『SQL Anywhere 11 - 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 11 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 11 - 変更点とアップグレード』** このマニュアルでは、SQL Anywhere 11 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 データベースを実行、管理、構成する方法について説明します。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、管理ユーティリティとオプションについて説明します。
- **『SQL Anywhere サーバ - プログラミング』** このマニュアルでは、C、C++、Java、PHP、Perl、Python、および Visual Basic や Visual C# などの .NET プログラミング言語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。ADO.NET や ODBC などのさまざまなプログラミング・インタフェースについても説明します。
- **『SQL Anywhere サーバ - SQL リファレンス』** このマニュアルでは、システム・プロシージャとカタログ (システム・テーブルとビュー) に関する情報について説明します。また、SQL Anywhere での SQL 言語の実装 (探索条件、構文、データ型、関数) についても説明します。
- **『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。
- **『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 - 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。また、dbmlsync API についても説明します。dbmlsync API を使用すると、同期を C++ または .NET のクライアント・アプリケーションにシームレスに統合できます。
- **『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 - サーバ起動同期』** このマニュアルでは、Mobile Link サーバ起動同期について説明します。この機能により、Mobile Link サーバは同期を開始したり、リモート・デバイス上でアクションを実行することができます。
- **『QAnywhere』** このマニュアルでは、モバイル・クライアント、ワイヤレス・クライアント、デスクトップ・クライアント、およびラップトップ・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、QAnywhere について説明します。
- **『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。

- **『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 - 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・デバイス、モバイル・デバイス、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 Mobile、または Windows を搭載しているハンドヘルド・デバイス、モバイル・デバイス、または埋め込みデバイスの Web ベースのデータベース・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 - .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・デバイス、モバイル・デバイス、または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 J』** このマニュアルでは、Ultra Light J について説明します。Ultra Light J を使用すると、Java をサポートしている環境用のデータベース・アプリケーションを開発し、配備することができます。Ultra Light J は、BlackBerry スマートフォンと Java SE 環境をサポートしており、iAnywhere Ultra Light データベース製品がベースになっています。
- **『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示し、その診断情報を説明します。

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表記規則について説明します。

オペレーティング・システム

SQL Anywhere はさまざまなプラットフォームで稼働します。ほとんどの場合、すべてのプラットフォームで同じように動作しますが、いくつかの相違点や制限事項があります。このような相違点や制限事項は、一般に、基盤となっているオペレーティング・システム (Windows、UNIX など) に由来しており、使用しているプラットフォームの種類 (AIX、Windows Mobile など) またはバージョンに依存していることはほとんどありません。

オペレーティング・システムへの言及を簡素化するために、このマニュアルではサポートされているオペレーティング・システムを次のようにグループ分けして表記します。

- **Windows** Microsoft Windows ファミリを指しています。これには、主にサーバ、デスクトップ・コンピュータ、ラップトップ・コンピュータで使用される Windows Vista や Windows XP、およびモバイル・デバイスで使用される Windows Mobile が含まれます。

  特に記述がないかぎり、マニュアル中に Windows という記述がある場合は、Windows Mobile を含むすべての Windows ベース・プラットフォームを指しています。

- **UNIX**　特に記述がないかぎり、マニュアル中に UNIX という記述がある場合は、Linux および Mac OS X を含むすべての UNIX ベース・プラットフォームを指しています。

ディレクトリとファイル名

ほとんどの場合、ディレクトリ名およびファイル名の参照形式はサポートされているすべてのプラットフォームで似通っており、それぞれの違いはごくわずかです。このような場合は、Windows の表記規則が使用されています。詳細がより複雑な場合は、マニュアルにすべての関連形式が記載されています。

ディレクトリ名とファイル名の表記を簡素化するために使用されている表記規則は次のとおりです。

- **大文字と小文字のディレクトリ名**　Windows と UNIX では、ディレクトリ名およびファイル名には大文字と小文字が含まれている場合があります。ディレクトリやファイルが作成されると、ファイル・システムでは大文字と小文字の区別が維持されます。

  Windows では、ディレクトリおよびファイルを参照するとき、大文字と小文字は**区別されません**。大文字と小文字を混ぜたディレクトリ名およびファイル名は一般的に使用されますが、参照するときはすべて小文字を使用するのが通常です。SQL Anywhere では、*Bin32* や *Documentation* などのディレクトリがインストールされます。

  UNIX では、ディレクトリおよびファイルを参照するとき、大文字と小文字は**区別されます**。大文字と小文字を混ぜたディレクトリ名およびファイル名は一般的に使用されません。ほとんどの場合は、すべて小文字の名前が使用されます。SQL Anywhere では、*bin32* や *documentation* などのディレクトリがインストールされます。

  このマニュアルでは、ディレクトリ名に Windows の形式を使用しています。ほとんどの場合、大文字と小文字が混ざったディレクトリ名をすべて小文字に変換すると、対応する UNIX 用のディレクトリ名になります。

- **各ディレクトリおよびファイル名を区切るスラッシュ**　マニュアルでは、ディレクトリの区切り文字に円記号を使用しています。たとえば、PDF 形式のマニュアルは *install-dir¥Documentation¥ja¥pdf* にあります。これは Windows の形式です。

  UNIX では、円記号をスラッシュに置き換えます。PDF マニュアルは *install-dir/documentation/ja/pdf* にあります。

- **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、*.exe* や *.bat* などの拡張子が付きます。UNIX では、実行ファイルの名前に拡張子は付きません。

  たとえば、Windows でのネットワーク・データベース・サーバは *dbsrv11.exe* です。UNIX では *dbsrv11* です。

- ***install-dir***　インストール・プロセス中に、SQL Anywhere をインストールするロケーションを選択します。このロケーションを参照する環境変数 SQLANY11 が作成されます。このマニュアルでは、そのロケーションを *install-dir* と表します。

  たとえば、マニュアルではファイルを install-dir¥readme.txt のように参照します。これは、Windows では、%SQLANY11%¥readme.txt に対応します。UNIX では、$SQLANY11/readme.txt または ${SQLANY11}/readme.txt に対応します。

install-dir のデフォルト・ロケーションの詳細については、「SQLANY11 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・プロセス中に、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します。このロケーションを参照する環境変数 SQLANYSAMP11 が作成されます。このマニュアルではそのロケーションを *samples-dir* と表します。

  Windows エクスプローラ・ウィンドウで *samples-dir* を開くには、**[スタート] - [プログラム] - [SQL Anywhere 11] - [サンプル・アプリケーションとプロジェクト]** を選択します。

  samples-dir のデフォルト・ロケーションの詳細については、「SQLANYSAMP11 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。

コマンド・プロンプトとコマンド・シェル構文

ほとんどのオペレーティング・システムには、コマンド・シェルまたはコマンド・プロンプトを使用してコマンドおよびパラメータを入力する方法が、1 つ以上あります。Windows のコマンド・プロンプトには、コマンド・プロンプト (DOS プロンプト) および 4NT があります。UNIX のコマンド・シェルには、Korn シェルおよび bash があります。各シェルには、単純コマンドからの拡張機能が含まれています。拡張機能は、特殊文字を指定することで起動されます。特殊文字および機能は、シェルによって異なります。これらの特殊文字を誤って使用すると、多くの場合、構文エラーや予期しない動作が発生します。

このマニュアルでは、一般的な形式のコマンド・ラインの例を示します。これらの例に、シェルにとって特別な意味を持つ文字が含まれている場合、その特定のシェル用にコマンドを変更することが必要な場合があります。このマニュアルではコマンドの変更について説明しませんが、通常、その文字を含むパラメータを引用符で囲むか、特殊文字の前にエスケープ文字を記述します。

次に、プラットフォームによって異なるコマンド・ライン構文の例を示します。

- **カッコと中カッコ** 一部のコマンド・ライン・オプションは、詳細な値を含むリストを指定できるパラメータを要求します。リストは通常、カッコまたは中カッコで囲まれています。このマニュアルでは、カッコを使用します。次に例を示します。

  ```
  -x tcpip(host=127.0.0.1)
  ```

  カッコによって構文エラーになる場合は、代わりに中カッコを使用します。

  ```
  -x tcpip{host=127.0.0.1}
  ```

  どちらの形式でも構文エラーになる場合は、シェルの要求に従ってパラメータ全体を引用符で囲む必要があります。

  ```
  -x "tcpip(host=127.0.0.1)"
  ```

- **引用符** パラメータの値として引用符を指定する必要がある場合、その引用符はパラメータを囲むために使用される通常の引用符と競合する可能性があります。たとえば、値に二重引用符を含む暗号化キーを指定するには、キーを引用符で囲み、パラメータ内の引用符をエスケープします。

  ```
  -ek "my ¥"secret¥" key"
  ```


多くのシェルでは、キーの値は my "secret" key のようになります。

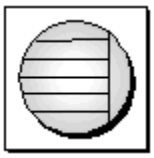
- **環境変数**　マニュアルでは、環境変数設定が引用されます。Windows のシェルでは、環境変数は構文 *%ENVVAR%* を使用して指定されます。UNIX のシェルでは、環境変数は構文 *$ENVVAR* または *${ENVVAR}* を使用して指定されます。

グラフィック・アイコン

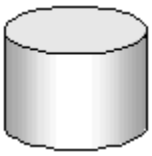
このマニュアルでは、次のアイコンを使用します。

- 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。



- データベース。ハイレベルの図では、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。

- プログラミング・インタフェース。

ドキュメンテーション・チームへのお問い合わせ

このヘルプに関するご意見、ご提案、フィードバックをお寄せください。

SQL Anywhere ドキュメンテーション・チームへのご意見やご提案は、弊社までご連絡ください。頂戴したご意見はマニュアルの向上に役立たせていただきます。ぜひとも、ご意見をお寄せください。

DocCommentXchange

DocCommentXchange を使用して、ヘルプ・トピックに関するご意見を直接お寄せいただくこともできます。DocCommentXchange (DCX) は、SQL Anywhere マニュアルにアクセスしたり、マニュアルについて議論するためのコミュニティです。DocCommentXchange は次の目的に使用できます (現在のところ、日本語はサポートされておりません)。

- マニュアルを表示する
- マニュアルの項目について明確化するために、ユーザによって追加された内容を確認する
- すべてのユーザのために、今後のリリースでマニュアルを改善するための提案や修正を行う

http://dcx.sybase.com を参照してください。

詳細情報の検索／テクニカル・サポートの依頼

詳しい情報やリソースについては、iAnywhere デベロッパー・コミュニティ (http://www.iAnywhere.jp/developers/index.html) 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。バージョンおよびビルド番号を調べるには、コマンド **dbeng11 -v** を実行します。

ニュースグループは、ニュース・サーバ *forums.sybase.com* にあります。

以下のニュースグループがあります。

- ianywhere.public.japanese.general

Web 開発に関する問題については、http://groups.google.com/group/sql-anywhere-web-development を参照してください。

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。

SQL Remote のレプリケーション設計

この項では、SQL Remote のレプリケーション設定の概要と手順について説明します。

SQL Remote の概要

目次

SQL Remote の機能

SQL Remote は、統合データベースと多数のリモート・データベース間のデータベース・トランザクションの双方向レプリケーション用に設計された、メッセージベースのテクノロジです。リモート・サイトにおける管理およびリソースの要件は最小限に抑えられているので、SQL Remote はモバイル・デバイスに最適です。

SQL Remote では、次のような機能を提供します。

- **複数サブスクライバのサポート**　SQL Remote を使用すると、不定期に接続するユーザが、SQL Anywhere 統合データベースと多数のリモート SQL Anywhere データベース (通常、多数のモバイル・データベースを含む) 間でデータをレプリケートできます。
- **トランザクション・ログ・ベースのレプリケーション**　SQL Remote では、トランザクション・ログを使用してレプリケーションを行います。このため、更新時にレプリケートされるのは、変更されたデータのみです。このことによって、レプリケーション・システム全体で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。
- **集中管理**　SQL Remote は、統合データベースで集中管理されます。企業では、数多くのユニークなデータベースが存在する多数のモバイル環境を使用できますが、各リモート・データベースを個別に管理することはありません。また、エンド・ユーザが SQL Remote の処理を意識することはありません。
- **効率的なメモリの使用**　効率的な実行のため、SQL Remote はメモリを効率よく使用します。このことによって、既存のリモート・コンピュータとデバイス上で SQL Remote を使用できるため、新しいハードウェアに投資する必要がありません。レプリケーションは、限られた領域を使用してリモート・コンピュータやデバイスと双方向に行うことができます。統合データベースからリモート・データベースにレプリケートされるのは、関連するデータのみです。
- **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・リンクでサポートされています。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。「サポートされるプラットフォーム」『SQL Anywhere 11 - 紹介』を参照してください。

参照

- 「同期テクノロジの比較」『SQL Anywhere 11 - 紹介』
- 「同期テクノロジの選択」『SQL Anywhere 11 - 紹介』

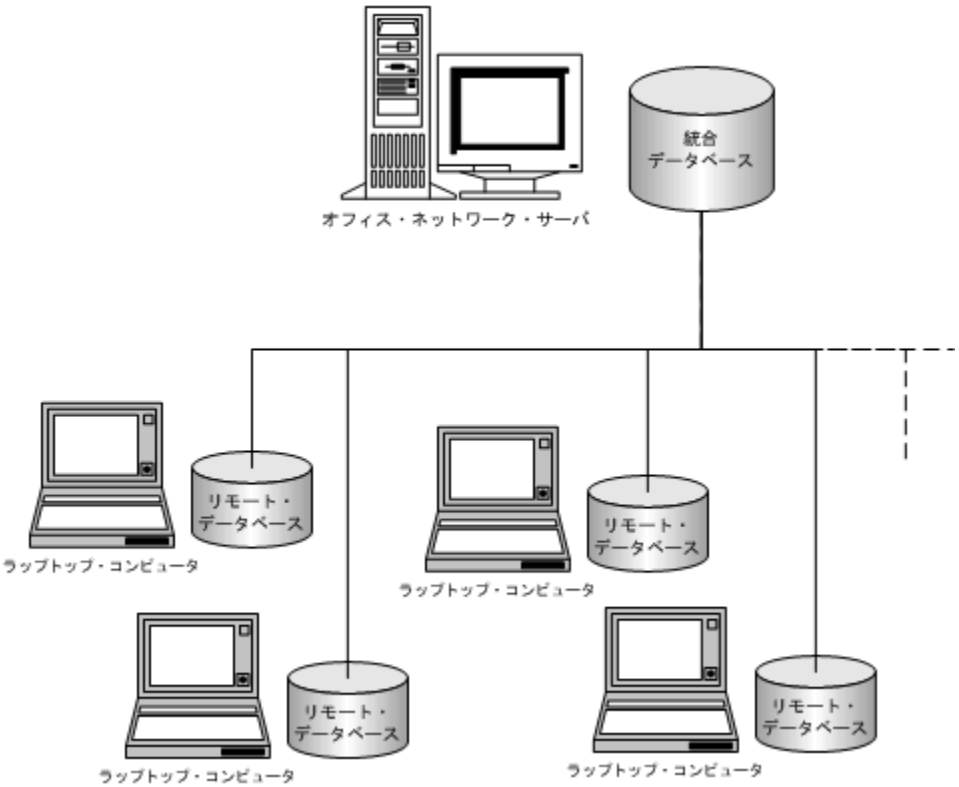
典型的な SQL Remote 設定

SQL Remote は、レプリケーション・システム用に設計されたもので、次の要件があります。

- **多数のリモート・データベース**　多数のリモート・データベースへのメッセージを同時に準備できるため、1 つのインストール環境で、数千のリモート・データベースをサポートできます。
- **随時接続**　SQL Remote は、ネットワークに時々または間接的に接続されるデータベースをサポートします。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、SMTP 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。
- **遅延時間：短～長**　遅延時間が長いというのは、システムにおいて、あるデータベースにデータが入力されてからそのデータが各データベースにレプリケートされるまでのタイムラグが長いということです。SQL Remote の場合、レプリケーション・メッセージは、秒、分、時間、または日単位の間隔で送信されます。
- **容量：低～中**　レプリケーション・メッセージは随時配信されるため、各リモート・データベースのトランザクションの容量が大きい場合は、メッセージの容量が大きくなる可能性があります。SQL Remote は、1 つのリモート・データベースについてのレプリケーション・データが比較的低容量であるシステムに最適です。統合データベースでは、SQL Remote は複数データベースへのメッセージを同時に準備できます。
- **同機種データベース**　システム内の各 SQL Anywhere データベースは、同様なスキーマを持つ必要があります。

モバイル環境でのサーバ／リモート・データベース間レプリケーション

次の例では、オフィス・ネットワーク上の統合データベースと、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、双方向にレプリケーションできます。SMTP 電子メール・システムがメッセージの伝送手段として使用されています。



統合データベースを管理するには、オフィス・ネットワーク・サーバで SQL Anywhere データベース・サーバを実行します。SQL Remote は、他のクライアント・アプリケーションと同じ方法で統合データベースに接続します。

各営業担当者のラップトップ・コンピュータには、SQL Anywhere パーソナル・サーバ、SQL Anywhere リモート・データベース、SQL Remote がインストールされています。

営業担当者は、外出先からインターネットに接続して SQL Remote を実行できます。このことによって、次の機能が実行されます。

- オフィス・ネットワーク・サーバ上の統合データベースから、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内容 (新しい注文など) を、オフィス・ネットワーク・サーバ上の統合データベースに送信する。

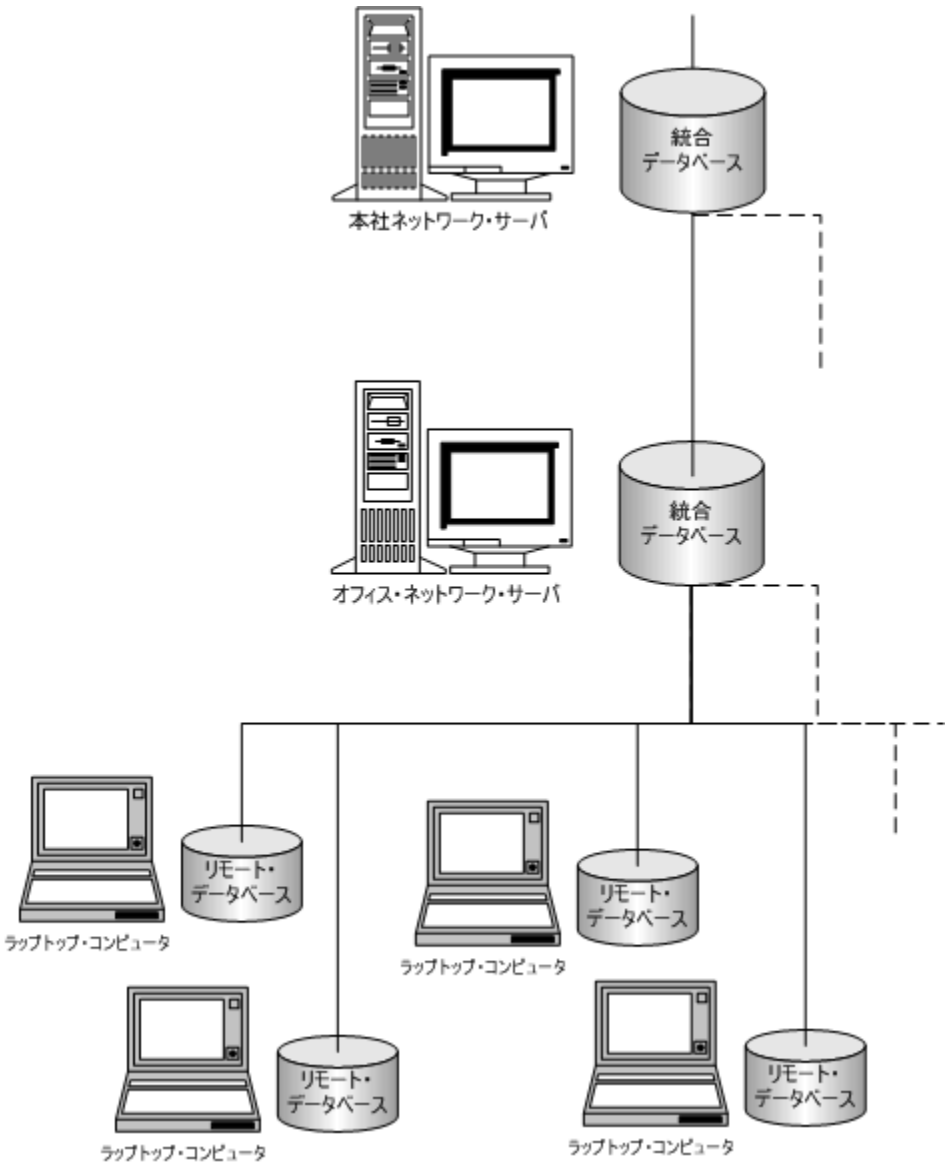
オフィス・ネットワーク・データベースのパブリケーションの更新には、その営業担当者が取り扱っている製品の新しい特別割引や、新価格、在庫情報などがあります。これらの更新はラップトップ上の SQL Remote によって読み込まれ、自動的に営業担当者のリモート・データベースに適用されます。営業担当者による追加操作は一切不要です。

営業担当者が登録した新しい注文も、この担当者が特別な操作をすることなく自動的にオフィス・ネットワーク・データベースに送信され、適用されます。

複数のオフィスにわたるサーバ間データベース・レプリケーション

この例では、営業所や販売店のデータベース・サーバと本社のデータベース・サーバ間で、双方向にレプリケーションできます。各営業所で必要となる作業は、サーバの初期設定と継続したメンテナンスのみです。

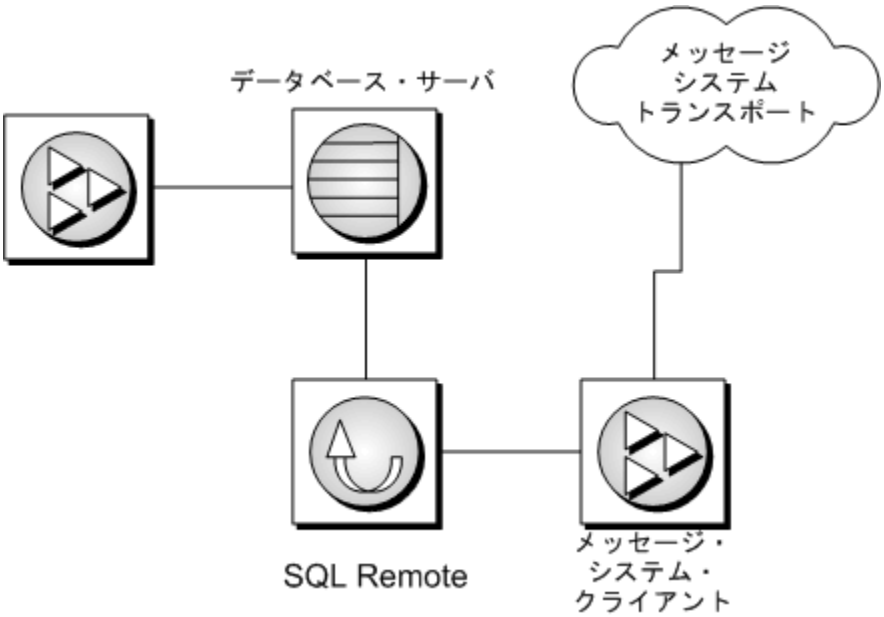
SQL Remote の階層には、レイヤを追加できます。たとえば、各営業所のサーバを統合データベースとして利用して、その営業所からのリモート・サブスクライバをサポートすることができます。



SQL Remote は、各オフィスでそれぞれに適したデータ・セットを受信するように設定できます。スタッフ・レコードなどのテーブルは、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。



- **データベース・サーバ**　統合サイトと各リモート・サイトには、SQL Anywhere データベースが必須です。
- **SQL Remote**　データベース間でレプリケーション・メッセージを送受信するには、統合サイトと各リモート・サイトに SQL Remote をインストールする必要があります。

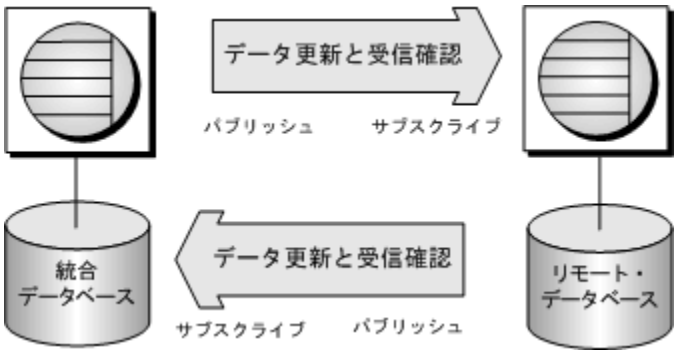
  SQL Remote Message Agent は、クライアント／サーバ接続経由でデータベース・サーバに接続します。Message Agent は、データベース・サーバと同じコンピュータでも異なるコンピュータでも実行できます。
- **メッセージ・システム・クライアント・ソフトウェア**　SQL Remote は、既存のメッセージ・システムを使用して、レプリケーション・メッセージを転送します。

  共有ファイルまたは FTP メッセージ・システムを使用している場合、メッセージ・システムはオペレーティング・システムに含まれています。

  SMTP 電子メール・システムを使用している場合は、統合サイトと各リモート・サイトに電子メール・クライアントをインストールしておく必要があります。
- **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、Embedded SQL、または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。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。「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。

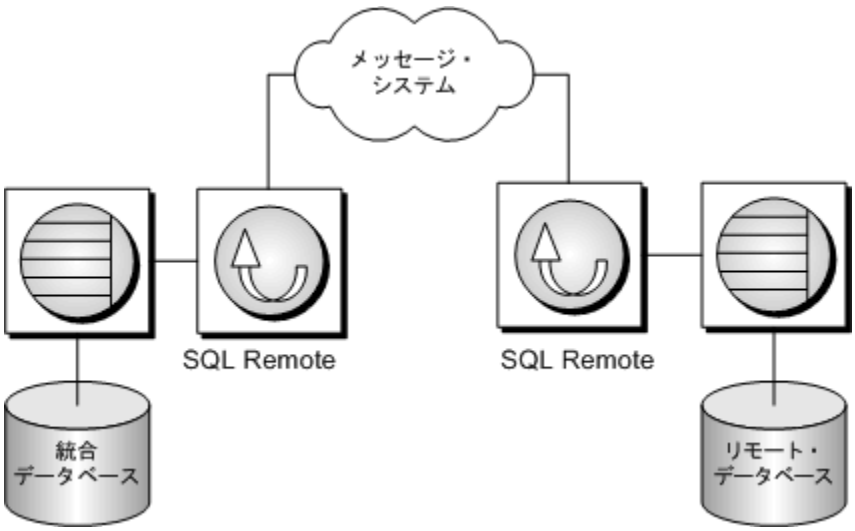
SQL Remote レプリケーション・プロセスの概要

SQL Remote では、メッセージは常に双方向に送信されます。統合データベースは、パブリケーションの更新を含む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・データベースは、更新されたデータと受信確認メッセージを統合データベースに送信します。



リモート・データベース・ユーザがデータを修正すると、その変更内容が統合データベースにレプリケートされます。この変更が統合データベースで適用されると、変更は統合データベースのパブリケーションに取り込まれ、(更新元のデータベースを除く) すべてのリモート・データベースに送信される更新内容に追加されます。このように、リモート・データベース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。

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。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、確認メッセージが統合データベースに送信されます。



♦ SQL Remote レプリケーション・プロセスに関連する手順

1. レプリケーションに関係する統合データベースとリモート・データベースごとに、Message Agent とレプリケーションを管理するトランザクション・ログが存在します。コミットされたすべての変更は、トランザクション・ログに記録されて保存されます。
2. 統合データベースの Message Agent では、トランザクション・ログを定期的にスキャンして、各パブリケーション (データのセクション) に対して行われたすべてのコミット済みトランザクションをメッセージにパッケージします。次に、統合データベースの Message Agent は、そのパブリケーションに対してサブスクライブされているリモート・ユーザに、関連する変更を送信します。Message Agent が変更内容を送信するときは、メッセージング・システムを使用します。SQL Remote では、SMTP 電子メール・システム、FTP、FILE をサポートしています。
3. リモート・データベースの Message Agent は、統合データベースから送信されたメッセージを受信し、メッセージの送信元である統合データベースに確認メッセージを送信します。次に、Message Agent はトランザクションをリモート・データベースに適用します。
4. リモート・ユーザは、いつでも Message Agent を実行して、リモート・データベースで行われたトランザクションをメッセージにパッケージし、統合データベースにそのメッセージを送信できます。
5. 統合サイトの Message Agent は、リモート・データベースからのメッセージを処理し、そのトランザクションを統合データベースに適用します。

SQL Remote のレプリケーション設計と設定

目次

SQL Remote システムの作成

統合データベースを使用して、すべての SQL Remote 管理タスクを実行します。

♦ **SQL Remote システムを作成するには、次の手順に従います。**

1. SQL Anywhere 統合データベースを選択するか、または新しい SQL Anywhere データベースを作成します。統合データベースからリモート・データベース (これも SQL Anywhere データベースです) が作成されます。

   新しい SQL Anywhere データベースを作成する場合は、SQL Remote でプライマリ・キーをどのように使用するかについて検討します。たとえば、プライマリ・キー・カラムのデータ型に、グローバル・オートインクリメントを使用した BIGINT を選択すると実用的です。「重複プライマリ・キー・エラー」 61 ページを参照してください。

2. レプリケートするデータを決定します。

   効率的なレプリケーション・システムを作成するには、使用するテーブル、そのテーブルのカラム、レプリケートするローのサブセットを決定します。必要な情報のみを含めるようにしてください。

3. 統合データベースにパブリケーションを作成します。

   SQL Remote では、パブリッシュ／サブスクライブ・モデルを採用しており、適切な情報が目的のユーザに必ず届けられます。統合データベースのパブリケーションにレプリケートするデータを整理してください。「パブリケーションとアーティクル」 16 ページを参照してください。

4. 統合データベースにパブリッシャ・ユーザを作成します。

   パブリッシャは、PUBLISH 権限を持つユーザで、統合データベースをユニークに識別するために使用されます。「PUBLISH パーミッション」 27 ページを参照してください。

5. リモート・ユーザを作成します。

   リモート・ユーザを使用して、リモート・データベースをユニークに識別します。「REMOTE パーミッション」 30 ページを参照してください。

   リモート・ユーザを作成する場合は、データを転送するときに使用するメッセージ・タイプを定義し、必要に応じてデータを送信する頻度を定義します。

6. サブスクリプションを作成し、パブリケーションに対してリモート・ユーザをサブスクライブします。「サブスクリプション」 38 ページを参照してください。

7. リモート・ユーザがデータをどのように使用するかを決定します。

   リモート・ユーザは、自分のデータを常に読み込むことができます。また、リモート・ユーザにはデータの更新、削除、挿入も許可できます。「トランザクション・ログ・ベースのレプリケーションの概要」 40 ページを参照してください。

8. 競合を解決する方法を選択します。

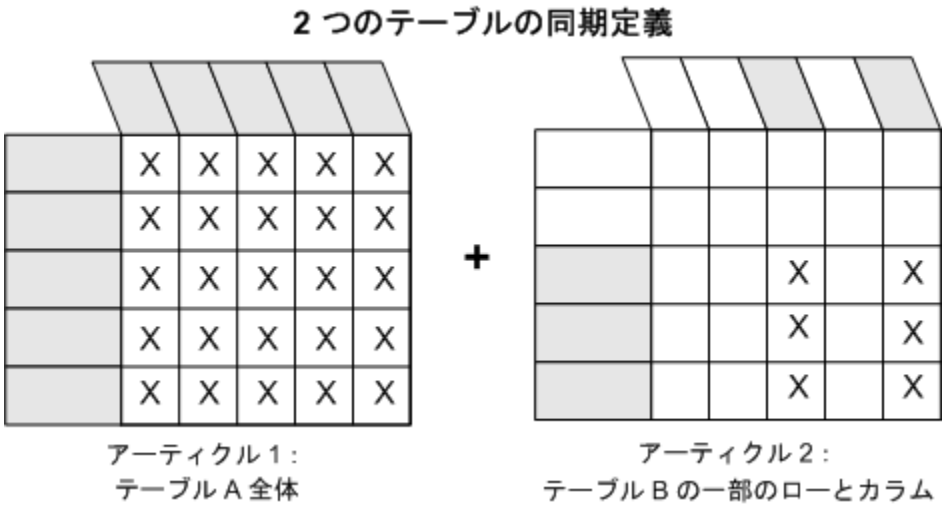
   リモート・ユーザがデータを更新、削除、挿入すると、レプリケーション時に競合が発生する可能性があります。競合を解決する方法を実装する必要があります。「更新の競合に対するデフォルトの解決」 49 ページを参照してください。

9. SQL Remote システムを配備します。

   リモート・データベースを作成して、適切なソフトウェアをインストールします。「SQL Remote の管理の概要」 88 ページを参照してください。

パブリケーションとアーティクル

「パブリケーション」では、レプリケートされるデータ・セットを定義します。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。「アーティクル」は、パブリケーションに含まれるテーブルを表します。パブリケーション内の各アーティクルは、テーブル全体または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。



制限事項

パブリケーションには、ビューまたはストアド・プロシージャを含めることができません。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ト」 43 ページと「トリガのレプリケート」 43 ページを参照してください。

パブリケーションとアーティクルの表示 (Sybase Central の場合)

Sybase Central では、パブリケーションは左ウィンドウ枠の **[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。

パブリケーションの作成

パブリケーションは、統合データベース内の既存のテーブルに基づいて作成します。

次の手順を使用して、テーブルのすべてのカラムとローで構成されるパブリケーションを作成します。

♦ **テーブル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統合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] - [新規] - [パブリケーション]** を選択します。
4. **[新しいパブリケーションの名前を指定してください。]** フィールドに、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. **[次へ]** をクリックします。
6. **[使用可能なテーブル]** リストでテーブルを選択します。**[追加]** をクリックします。
7. **[完了]** をクリックします。

♦ 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統合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。「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。

   たとえば、次の文では、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
   );
   ```

   次の文では、SalesOrders、SalesOrderItems、Products の各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 PubSales (
     TABLE SalesOrders,
     TABLE SalesOrderItems,
     TABLE Products
   );
   ```

テーブル内の一部のカラムのみをパブリッシュする

次の手順を使用して、テーブルのすべてのローと一部のカラムのみが含まれるパブリケーションを作成します。

♦ 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 (Sybase Central の場合)。

1. DBA 権限のあるユーザとして統合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]** フォルダを展開します。
3. **[ファイル] - [新規] - [パブリケーション]** を選択します。
4. **[新しいパブリケーションの名前を指定してください。]** フィールドに、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. **[次へ]** をクリックします。

6. **[使用可能なテーブル]** リストでテーブルを選択します。**[追加]** をクリックします。**[次へ]** をクリックします。
7. **[使用可能なカラム]** 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。パブリッシュする各カラムを選択し、**[追加]** をクリックします。**[次へ]** をクリックします。
8. **[完了]** 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統合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。

   たとえば、次の文では、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。このパブリケーションでは、Customers テーブルの Surname、GivenName、Street、State、Country、PostalCode、Phone カラムはパブリッシュされません。

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

参照

- 「CREATE PUBLICATION 文 [Mobile Link] [SQL Remote]」 『SQL Anywhere サーバ - SQL リファレンス』
- 「参照整合性エラー」 58 ページ

テーブル内の一部のローのみをパブリッシュする

テーブルの一部のローのみが含まれるパブリケーションを作成するには、パブリッシュするローのみに一致する検索条件を記述する必要があります。検索条件で次のいずれかの句を使用します。

- **SUBSCRIBE BY 句** SUBSCRIBE BY 句は、パブリケーションに対する複数のサブスクライバが、テーブルから異なるローを受信する場合に使用します。

  SQL Remote システムで多数のサブスクリプションが必要な場合に、SUBSCRIBE BY 句をおすすめします。SUBSCRIBE BY 句を使用すると、複数の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。サブスクライバが受信するローは、指定された式の値によって異なります。

  SUBSCRIBE BY 句を使用すると、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。

「SUBSCRIBE BY 句を使用して一部のローのみをパブリッシュする」 20 ページを参照してください。

- **WHERE 句**　WHERE 句は、アーティクルにローのサブセットを追加する場合に使用します。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。

  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。デフォルト値が設定されていない場合は、リモート・データベースが統合データベースから新しいローを挿入しようとすると、エラーが発生します。

  アーティクルでは WHERE 句を結合できます。

  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。WHERE 句を使用しても、複数の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ることはできません。ただし、SUBSCRIBE BY 句では、この関連付けができます。

  「WHERE 句を使用して一部のローのみをパブリッシュする」 21 ページを参照してください。

例

各営業担当者が次の操作を実行できるパブリケーションが必要です。

- 受注に対してサブスクライブする。
- 受注をローカルで更新する。
- 統合データベースに売り上げをレプリケートする。

WHERE 句を使用すると、営業担当者ごとに個別のパブリケーションを作成する必要があります。次のパブリケーションは、Sam Singer という名前の営業担当者用です。他のそれぞれの営業担当者にも、同様のパブリケーションが必要になります。

```
CREATE PUBLICATION PubOrdersSamSinger (
  TABLE SalesOrders
    WHERE Active = 1
);
```

次の文では、PubsOrdersSamSinger パブリケーションに対して Sam Singer をサブスクライブします。

```
CREATE SUBSCRIPTION
TO PubOrdersSamSinger
FOR Sam_Singer;
```

SUBSCRIBE BY 句を使用する場合、必要なパブリケーションは 1 つのみです。すべての営業担当者が、次のパブリケーションを使用できます。

```
CREATE PUBLICATION PubOrders (
  TABLE SalesOrders
  SUBSCRIBE BY SalesRepresentativeID
);
```

次の文では、Sam Singer の ID 8887 で、PubsOrders パブリケーションに対して Sam Singer をサブスクライブします。

```
CREATE SUBSCRIPTION
TO PubOrders ('8887')
FOR Sam_Singer;
```

SUBSCRIBE BY 句を使用して一部のローのみをパブリッシュする

次の手順を使用して、SUBSCRIBE BY 句を使用してパブリケーションを作成します。SUBSCRIBE BY 句とその代替手段である WHERE 句を使用する方法については、「テーブル内の一部のローのみをパブリッシュする」 18 ページを参照してください。

♦ SUBSRIBE BY 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統合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]** - **[新規]** - **[パブリケーション]** を選択します。
4. **[新しいパブリケーションの名前を指定してください。]** フィールドに、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. **[次へ]** をクリックします。
6. **[使用可能なテーブル]** リストでテーブルを選択します。**[追加]** をクリックします。**[次へ]** をクリックします。
7. **[使用可能なカラム]** 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。パブリッシュする各カラムを選択し、**[追加]** をクリックします。**[次へ]** をクリックします。
8. **[次へ]** をクリックします。
9. **[SUBSCRIBE BY 制限の指定]** ページで、次の手順を実行します。
   a. **[アーティクル]** リストでテーブルをクリックします。
      [カラム] をクリックし、ドロップダウン・リストからカラムを選択します。
10. **[完了]** をクリックします。

♦ SUBSRIBE BY 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. SUBSCRIBE BY 句が含まれる CREATE PUBLICATION 文を実行します。

例を示します。

次の文では、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 カラムの値を使用し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
);
```

次の文では、パブリケーションに対して 2 人の従業員をサブスクライブします。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してサブスクライブできます。また、単一のパブリケーションに対して複数のサブスクリプションを作成することもできます。

参照

- 「WHERE 句を使用して一部のローのみをパブリッシュする」 21 ページ
- 「切断データ分割の使用」 69 ページ
- 「CREATE PUBLICATION 文 [Mobile Link] [SQL Remote]」 『SQL Anywhere サーバ - SQL リファレンス』
- 「パブリケーションに対するサブスクリプション」 38 ページ

WHERE 句を使用して一部のローのみをパブリッシュする

次の手順を使用して、WHERE 句を使用するパブリケーションを作成し、テーブルのすべてのカラムと一部のローのみを追加します。WHERE 句とその代替手段である SUBSCRIBE BY 句を使用する方法については、「テーブル内の一部のローのみをパブリッシュする」 18 ページを参照してください。

♦ 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統合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]** - **[新規]** - **[パブリケーション]** を選択します。
4. **[新しいパブリケーションの名前を指定してください。]** フィールドに、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. **[次へ]** をクリックします。

6. **[使用可能なテーブル]** リストでテーブルを選択します。**[追加]** をクリックします。**[次へ]** をクリックします。
7. **[使用可能なカラム]** 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。パブリッシュする各カラムを選択し、**[追加]** をクリックします。**[次へ]** をクリックします。
8. **[WHERE 句の指定]** ページで、次の手順に従います。
   a. **[アーティクル]** リストでテーブルをクリックします。
   b. **[選択したアーティクルには次の WHERE 句があります]** フィールドに WHERE 句を入力します。
9. **[完了]** 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. WHERE 句を使用する CREATE PUBLICATION 文を実行して、パブリケーションの対象とするローを追加します。

   たとえば、次の文では、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。Status カラムはパブリッシュされません。

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

   次の文では、同じパブリケーションに対して 2 人の従業員をサブスクライブします。Ann Taylor と Sam Singer は同じデータを受信します。

   ```
   CREATE SUBSCRIPTION
   TO PubCustomers
   FOR Ann_Taylor;

   CREATE SUBSCRIPTION
   TO PubCustomers)
   FOR Sam_Singer;
   ```

ユーザは、複数のパブリケーションに対してサブスクライブできます。また、単一のパブリケーションに対して複数のサブスクリプションを作成することもできます。

「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。

参照

- 「WHERE 句とプライマリ・キー」 57 ページ

パブリケーションの変更

パブリケーションを変更するには、アーティクルを追加、修正、または削除するか、パブリケーションの名前を変更します。

警告
動作中の SQL Remote システムで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、レプリケーション・システムのデータが失われるおそれがあります。「アップグレードと再同期」 148 ページを参照してください。

♦ パブリケーションを変更するには、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. 左ウィンドウ枠の **[パブリケーション]** フォルダを選択します。
3. 変更するアーティクルを右クリックし、**[プロパティ]** を選択してパブリケーションを編集します。

♦ パブリケーションを変更するには、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。

   たとえば、次の文では、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
   );
   ```

   たとえば、次の文では、PubContacts パブリケーションの Customers アーティクルを再定義します。

   ```
   ALTER PUBLICATION PubContacts (
     ALTER ARTICLE Customers ( name, surname )
   );
   ```

参照

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

パブリケーションの削除

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。

警告
動作中の SQL Remote システムでパブリケーションを削除すると、レプリケーション・エラーが発生し、レプリケーション・システムのデータが失われるおそれがあります。「アップグレードと再同期」 148 ページを参照してください。

♦ **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. 左ウィンドウ枠で、**[パブリケーション]** フォルダを展開します。
3. 目的のパブリケーションを右クリックして、**[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。

   たとえば、次の文では、PubOrders という名前のパブリケーションを削除します。

   ```
   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。

ユーザ・パーミッション

SQL Remote では、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つユーザを管理するための、一貫性のあるシステムを採用しています。

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 1 つ以上のパーミッションで識別されます。

- **PUBLISH**　SQL Remote システム内のすべてのデータベースが情報をパブリッシュします。したがって、どのデータベースにもパブリッシャが必要です。パブリッシャを作成するには、1 人のユーザに PUBLISH パーミッションを付与します。パブリッシャ・ユーザは、SQL Remote システム全体でユニークとする必要があります。データを送信するとき、パブリッシャはそのデータベースを表しています。たとえば、データベースがメッセージを送信するとき、メッセージにはデータベースのパブリッシャ・ユーザ名が含まれています。データベースがメッセージを受信する場合は、メッセージに含まれるパブリッシャ名によって、メッセージを送信したデータベースを識別できます。
- **REMOTE**　統合データベースなど、他のデータベースにメッセージを送信するデータベースでは、メッセージの送信先となるリモート・データベースを指定する必要があります。統合データベースでこれらのリモート・データベースを指定するには、リモート・データベースのパブリッシャに REMOTE パーミッションを付与します。REMOTE パーミッションによって、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が識別されます。
- **CONSOLIDATE**　各リモート・データベースでは、メッセージを受信する統合データベースを指定する必要があります。リモート・データベースで統合データベースを指定するには、統合データベースのパブリッシャに CONSOLIDATE パーミッションを付与します。リモート・データベースは、1 つの統合データベースからのメッセージのみを受信できます。CONSOLIDATE パーミッションによって、現在のデータベースにメッセージを送信するデータベースが識別されます。

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おり、他のデータベース権限やパーミッションからは独立しています。

抽出ユーティリティ (dbxtract) によるパーミッションの自動設定

抽出ユーティリティ (dbxtract) と**データベース抽出ウィザード**では、デフォルトで、適切な PUBLISH パーミッションと CONSOLIDATE パーミッションが、リモート・データベース内のユーザに付与されます。

単層階層

単層階層では、1 つの統合データベースの下に 1 つ以上のリモート・データベースが存在します。このような階層では、統合データベースによって、リモート・データベースのパブリッシャに REMOTE 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ます。

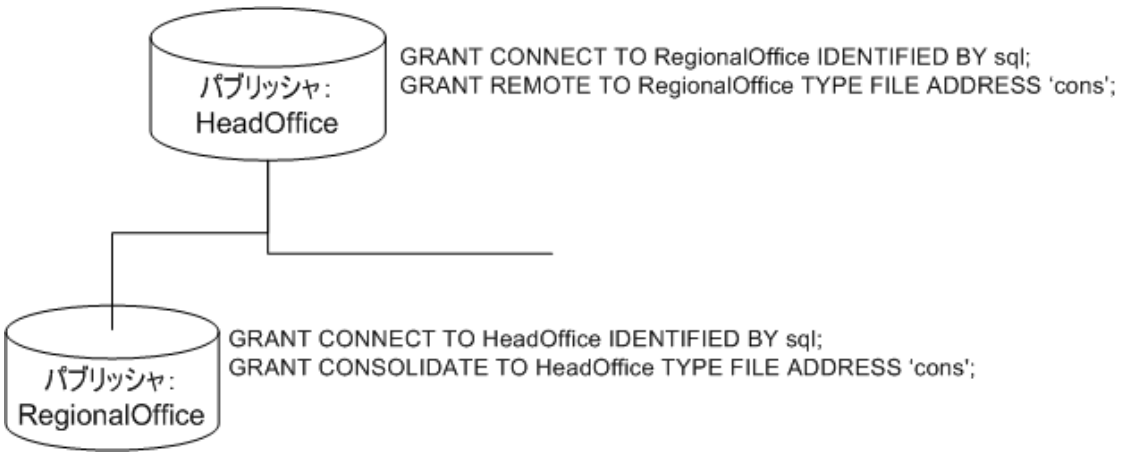
たとえば、統合データベースのパブリッシャによって識別される統合データベース HeadOffice と、リモート・データベースのパブリッシャによって識別されるリモート・データベース RegionalOffice があるとします。

統合データベース HeadOffice で、次の処理を実行します。

- リモート・データベース RegionalOffice のパブリッシャと同じ名前のユーザを作成します。
- RegionalOffice に REMOTE パーミッションを付与します。これにより、RegionalOffice が HeadOffice のリモート・データベースとして識別されます。

リモート・データベース RegionalOffice で、次の処理を実行します。

- 統合データベース HeadOffice のパブリッシャと同じ名前のユーザを作成します。
- RegionalOffice に CONSOLIDATE パーミッションを付与します。これにより、RegionalOffice が HeadOffice のリモート・データベースとして識別されます。



Dbxtract によるパーミッションの自動設定

抽出ユーティリティ (dbxtract) と**データベース抽出ウィザード**では、デフォルトで、適切な PUBLISH パーミッションと CONSOLIDATE パーミッションが、リモート・データベース内のユーザに付与されます。

多層階層

多層階層では、現在のデータベースの直下にあるすべてのリモート・データベースに REMOTE パーミッションが付与されます。階層内の現在のデータベースの直上にあるデータベースには CONSOLIDATE パーミッションが付与されます。

たとえば、統合データベースのパブリッシャ HeadOffice によって識別される統合データベースがあり、このデータベースにはリモート・データベース RegionalOffice が存在するとします。ただし、RegionalOffice データベースにもリモート・データベース Office が存在します。

統合データベース HeadOffice で、次の処理を実行します。

- リモート・データベース RegionalOffice のパブリッシャと同じ名前のユーザを作成します。

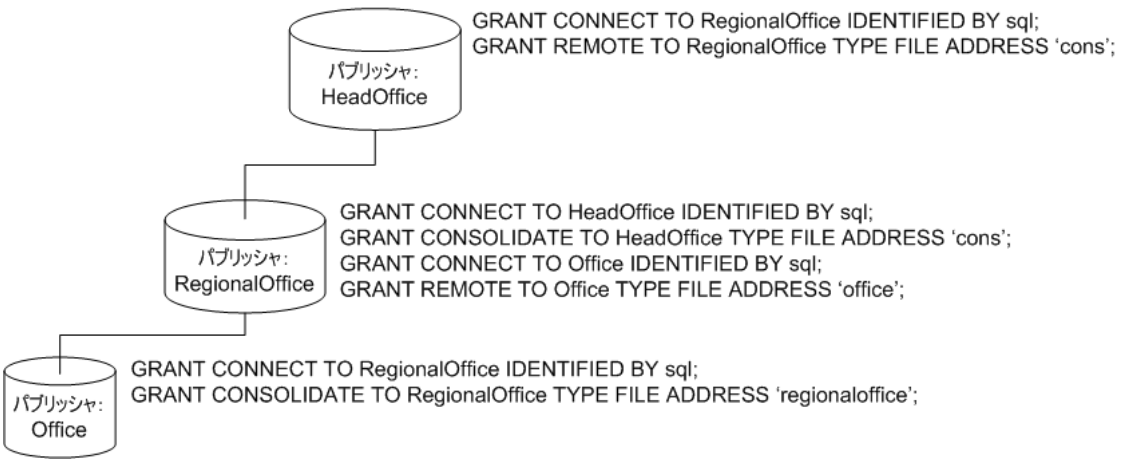
- ユーザ RegionalOffice に REMOTE パーミッションを付与します。このことによって、RegionalOffice が HeadOffice からメッセージを受信するデータベースとして識別されます。

RegionalOffice データベースで、次の処理を実行します。

- 統合データベース HeadOffice のパブリッシャと同じ名前のユーザを作成します。
- HeadOffice に CONSOLIDATE パーミッションを付与します。このことによって、HeadOffice が RegionalOffice の統合データベースとして識別されます。つまり、HeadOffice が、RegionalOffice にメッセージを送信するデータベースとなります。
- RegionalOffice の直下にあるデータベースと同じ名前のデータベース Office と同じ名前のユーザを作成します。
- Office に REMOTE パーミッションを付与します。このことによって、Office が RegionalOffice からメッセージを受信するデータベースとして識別されます。

Office データベースで、次の処理を実行します。

- 統合データベース RegionalOffice のパブリッシャと同じ名前のユーザを作成します。
- RegionalOffice ユーザに CONSOLIDATE パーミッションを付与します。このことによって、RegionalOffice が Office の統合データベースとして識別されます。つまり、RegionalOffice が Office にメッセージを送信します。



PUBLISH パーミッション

SQL Remote システムのすべてのデータベースには、パブリッシャが必要です。パブリッシャは、PUBLISH パーミッションを持つユニークなユーザです。パブリケーションの更新と受信確認を含む、SQL Remote のすべての出力メッセージは、パブリッシャによって識別されます。SQL Remote システムのすべてのデータベースは、受信確認を送信します。

PUBLISH パーミッションには、出力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。

GROUP パーミッションを持つユーザ名に 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継承されません。

パブリッシャを作成するには、ユーザに PUBLISH パーミッションを付与します。

パブリッシャの作成

次の手順を使用して、ユーザを作成して PUBLISH パーミッションを付与します。

♦ **パブリッシャを作成するには、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. パブリッシャの役割を果たすユーザを作成します (まだ存在しない場合)。
   a. 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。
   b. **[ファイル]** - **[新規]** - **[ユーザ]** を選択します。
   c. **ユーザ作成ウィザード**の指示に従います。ユーザにパスワードを割り当てます。
3. **[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 **[パブリッシャに変更]** を選択します。

♦ **パブリッシャを作成するには、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 USER 文を実行して、パブリッシャの役割を果たすユーザを作成します。ユーザにパスワードを割り当てます。

   たとえば、次の例では、パスワード sql を指定して cons という名前のユーザを作成します。

   ```
   CREATE USER cons IDENTIFIED BY SQL;
   ```
3. GRANT CONNECT 文を実行して、ユーザに CONNECT パーミッションを付与します。「GRANT 文」『SQL Anywhere サーバ - SQL リファレンス』を参照してください。

   たとえば、次の文では、ユーザ cons に CONNECT パーミッションを付与します。

   ```
   GRANT CONNECT TO cons IDENTIFIED BY SQL;
   ```
4. GRANT PUBLISH 文を実行して、ユーザに PUBLISH パーミッションを付与します。「GRANT PUBLISH 文 [SQL Remote]」『SQL Anywhere サーバ - SQL リファレンス』を参照してください。

   たとえば、次の文では、ユーザ cons に PUBLISH パーミッションを付与します。

   ```
   GRANT PUBLISH TO cons;
   ```


抽出ユーティリティ (dbxtract)

抽出ユーティリティ (dbxtract) または**データベース抽出ウィザード**でリモート・データベースを抽出すると、リモート・ユーザがリモート・データベースのパブリッシャになり、PUBLISH パーミッションが付与されます。

参照

- 「PUBLISH パーミッションの取り消し」 29 ページ
- 「パブリッシャの表示」 29 ページ

PUBLISH パーミッションの取り消し

次の手順を使用して、ユーザから PUBLISH パーミッションを取り消します。

警告
リモート・データベースまたは統合データベースでパブリッシャを変更すると、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。「実行中のシステムに行ってはならない変更」 148 ページを参照してください。

♦ **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)。**

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. 左ウィンドウ枠で、**[ユーザとグループ]** フォルダを選択します。
3. PUBLISH パーミッションを持つユーザを右クリックして、**[パブリッシャの取り消し]** を選択します。

♦ **PUBLISH パーミッションを取り消すには、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. REVOKE PUBLISH 文を実行して、現在のパブリッシャから PUBLISH パーミッションを取り消します。「REVOKE PUBLISH 文 [SQL Remote]」 『SQL Anywhere サーバ - SQL リファレンス』を参照してください。

   次に例を示します。

   ```
   REVOKE PUBLISH FROM cons;
   ```

参照

- 「PUBLISH パーミッション」 27 ページ
- 「パブリッシャの表示」 29 ページ

パブリッシャの表示

♦ **パブリッシャを確認するには、次の手順に従います (Sybase Central の場合)。**

- 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。

右ウィンドウ枠で、**[タイプ]** が **[パブリッシャ]** となっているユーザがパブリッシャを表します。

♦ **パブリッシャを確認するには、次の手順に従います (SQL の場合)。**

- CURRENT PUBLISHER 定数を使用します。次の文で、パブリッシャ・ユーザ名を取得します。

```
SELECT CURRENT PUBLISHER;
```

参照

- 「PUBLISH パーミッション」 27 ページ
- 「PUBLISH パーミッションの取り消し」 29 ページ

REMOTE パーミッション

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。SQL Remote 階層内で現在のデータベースの直下にあるデータベースのパブリッシャには、現在のデータベースによって REMOTE パーミッションが付与されます。

ユーザに REMOTE パーミッションを付与する場合は、次の設定を行う必要があります。

- **メッセージ・システム**　データベース内で最低 1 つのメッセージ・システムが定義されるまで、新規リモート・ユーザを作成することはできません。「SQL Remote メッセージ・システム」 121 ページを参照してください。
- **送信頻度**　SQL 文を使用して REMOTE パーミッションを付与する場合、送信頻度の設定はオプションです。「送信頻度の設定」 100 ページを参照してください。

ユーザに REMOTE パーミッションを付与するには、次の処理を実行します。

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

データベースのパブリッシャは、同じデータベースの REMOTE パーミッションと CONSOLIDATE パーミッションを持つことはできません。これにより、パブリッシャが、出力メッセージの送信者と受信者の両方として識別されます。

グループへの REMOTE パーミッションの付与

グループに REMOTE パーミッションを付与できますが、REMOTE パーミッションは、グループ内のすべてのユーザに自動的には適用されません。グループ内の各ユーザには REMOTE パーミッションを明示的に付与する必要があります。

REMOTE パーミッションの付与

次の手順を使用して、リモート・ユーザを追加します。

♦ **リモート・ユーザを作成するには、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. 左ウィンドウ枠で、**[SQL Remote ユーザ]** フォルダを選択します。
3. **[ファイル]** - **[新規]** - **[SQL Remote ユーザ]** を選択します。
4. **リモート・ユーザ作成ウィザード**の指示に従います。

♦ **既存のユーザをリモート・ユーザにするには、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. 左ウィンドウ枠で、**[SQL Remote ユーザ]** フォルダを選択します。
3. ユーザを右クリックして、**[リモート・ユーザに変更]** を選択します。
4. このウィンドウで、メッセージ・タイプの選択、アドレスの入力、送信頻度の選択を行い、**[OK]** をクリックします。

♦ **リモート・ユーザを作成するには、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 USER 文を実行して、ユーザを作成します。

   次に例を示します。

   ```
   CREATE USER remote1;
   ```

3. GRANT CONNECT 文を実行して、ユーザの CONNECT パーミッションを付与します。「GRANT 文」『SQL Anywhere サーバ - SQL リファレンス』を参照してください。

   次に例を示します。

   ```
   GRANT CONNECT TO remote1;
   ```

4. GRANT REMOTE 文を実行して、ユーザの REMOTE パーミッションを付与します。「GRANT REMOTE 文 [SQL Remote]」『SQL Anywhere サーバ - SQL リファレンス』を参照してください。

   次に例を示します。

   ```
   GRANT REMOTE TO remote1;
   ```

   たとえば、次の文では、ユーザ S_Beaulieu に REMOTE パーミッションを付与し、リモート・データベースで次の処理を実行することを指定します。

   - SMTP 電子メールをメッセージ・システムとして使用する。
   - 電子メール・アドレス s_beaulieu@acme.com を使用してメッセージを送信する。

- 毎日午後 10 時にメッセージを送信する。

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

次の文では、ユーザ rem1 に REMOTE パーミッションを付与し、rem1 のリモート・データベースでアドレス rem1 の FILE メッセージ・システムを使用することを指定します。

```
GRANT CONNECT TO rem1 IDENTIFIED BY SQL
GRANT REMOTE TO rem1 TYPE FILE ADDRESS 'rem1';
```

参照

- 「REMOTE パーミッションの取り消し」 32 ページ

REMOTE 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には、次の処理を実行します。

- SQL Remote システムからユーザを削除します。
- ユーザまたはグループを通常のユーザまたはグループに戻します。
- すべてのパブリケーションから、ユーザまたはグループのサブスクリプションを削除します。

♦ **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. 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. 左ウィンドウ枠で、**[ユーザとグループ]** フォルダまたは **[SQL Remote ユーザ]** フォルダのいずれかを展開します。
3. リモート・ユーザまたはグループを右クリックして、**[リモートの取り消し]** を選択します。

♦ **REMOTE パーミッションを取り消すには、次の手順に従います (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. REVOKE REMOTE 文を実行して、現在のユーザまたはグループの REMOTE パーミッションを取り消します。「REVOKE REMOTE 文 [SQL Remote]」 『SQL Anywhere サーバ - SQL リファレンス』を参照してください。

   たとえば、次の文では、ユーザ S_Beaulieu から REMOTE パーミッションを取り消します。

```
REVOKE REMOTE FROM S_Beaulieu;
```

参照

- 「REMOTE パーミッションの付与」 31 ページ

CONSOLIDATE パーミッション

SQL Remote 階層内で現在のデータベースの直上にあるデータベースには、現在のデータベースによって 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、統合データベースに CONSOLIDATE パーミッションを付与する必要があります。

CONSOLIDATE パーミッションは、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に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が送信されるためです。

ユーザに CONSOLIDATE パーミッションを付与する場合は、次の設定を行う必要があります。

- **メッセージ・システム**　データベース内で最低 1 つのメッセージ・システムが定義されるまで、新規統合ユーザを作成することはできません。「SQL Remote メッセージ・システム」 121 ページを参照してください。
- **送信頻度**　SQL 文を使用して CONSOLIDATE パーミッションを付与する場合、送信頻度の設定はオプションです。「送信頻度の設定」 100 ページを参照してください。

CONSOLIDATE パーミッションを付与するには、次の処理を実行します。

- ユーザを統合ユーザとして識別します。
- この統合ユーザでメッセージを交換するときに使用するメッセージ・タイプを指定します。
- メッセージの送信先アドレスを指定します。
- メッセージが統合ユーザに送信される頻度を指示します。

データベースのパブリッシャは、同じデータベースの REMOTE パーミッションと CONSOLIDATE パーミッションを持つことはできません。これにより、パブリッシャが、出力メッセージの送信者と受信者の両方として識別されます。

抽出ユーティリティ (dbxtract)

抽出ユーティリティ (dbxtract) または**データベース抽出ウィザード**でリモート・データベースを抽出すると、リモート・データベースで GRANT CONSOLIDATE 文が自動的に実行されます。

CONSOLIDATE パーミッションの付与

次の手順を使用して、ユーザに CONSOLIDATE パーミッションを付与します。

統合データベースのパブリッシャに CONSOLIDATE パーミッションを付与することをおすすめします。

♦ **統合データベースを指定するには、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接続します。
2. 左ウィンドウ枠でデータベースをクリックして、**[ファイル] - [プロパティ]** を選択します。
3. **[SQL Remote]** タブをクリックします。

4. **[このリモート・データベースに対応する統合データベースが存在する]** を選択します。
5. **[メッセージ・タイプ]**、**[アドレス]**、**[送信頻度]** の各設定を行います。
6. **[OK]** をクリックして **[DBA データベースのプロパティ]** ウィンドウを閉じます。

♦ 統合データベースを指定するには、次の手順に従います (SQL の場合)。

1. GRANT CONNECT 文を実行して、統合データベースのパブリッシャに CONNECT パーミッションを付与します。「GRANT 文」『SQL Anywhere サーバ - SQL リファレンス』を参照してください。

   次に例を示します。

   ```
   GRANT CONNECT TO cons;
   ```

2. GRANT CONSOLDIATE 文を実行して、統合ユーザに CONSOLIDATE パーミッションを付与します。「GRANT CONSOLIDATE 文 [SQL Remote]」『SQL Anywhere サーバ - SQL リファレンス』を参照してください。

   たとえば、次の文では、hq_user ユーザに CONSOLIDATE パーミッションを付与し、統合データベースで SMTP 電子メール・システムを使用することを指定します。

   ```
   GRANT CONSOLIDATE TO hq_user
   TYPE SMTP
   ADDRESS 'hq_address';
   ```

   この GRANT CONSOLIDATE 文には SEND 句がありません。したがって、SQL Remote は統合データベースにメッセージを送信してから停止します。

   たとえば、次の文では、cons ユーザに CONSOLIDATE パーミッションを付与し、統合データベースで FILE システムを使用することを指定します。

   ```
   GRANT CONNECT TO "cons" IDENTIFIED BY SQL;
   GRANT CONSOLIDATE TO "cons" TYPE "FILE" ADDRESS 'cons';
   GRANT CONNECT TO "rem1" IDENTIFIED BY SQL;
   GRANT PUBLISH TO "rem1";
   CREATE REMOTE MESSAGE TYPE FILE ADDRESS 'rem1';
   ```

参照

- 「CONSOLIDATE パーミッションの取り消し」 34 ページ

CONSOLIDATE パーミッションの取り消し

ユーザの CONSOLIDATE パーミッションを取り消す場合、SQL Anywhere では次の処理を実行します。

- SQL Remote システムからユーザを削除します。
- ユーザまたはグループを通常のユーザまたはグループに戻します。
- すべてのパブリケーションから、ユーザまたはグループのサブスクリプションを削除します。

♦ **CONSOLDIATE 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. 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. **[ユーザとグループ]** フォルダまたは **[SQL Remote ユーザ]** フォルダを展開します。
3. 統合ユーザまたはグループを右クリックして、**[統合の取り消し]** を選択します。
4. **[はい]** をクリックします。

♦ **CONSOLIDATE パーミッションを取り消すには、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. REVOKE CONSOLIDATE 文を実行して、現在のユーザまたはグループから CONSOLIDATE パーミッションを取り消します。「REVOKE CONSOLIDATE 文 [SQL Remote]」 『SQL Anywhere サーバ - SQL リファレンス』を参照してください。

   次に例を示します。

```
REVOKE CONSOLIDATE FROM Cons_User;
```

参照

- 「CONSOLIDATE パーミッションの付与」 33 ページ

REMOTE DBA 権限の付与

REMOTE DBA 権限を持つユーザ名は、SQL Remote から接続した場合にのみデータベースの完全な DBA 権限を付与されます。REMOTE DBA 権限がある場合、SQL Remote はデータベースに完全にアクセスでき、メッセージで指定された変更を行うことができます。SQL Remote を実行できるのは、REMOTE DBA 権限または DBA 権限のあるユーザのみです。

特別な権限として、REMOTE DBA には次のような性質があります。

- **SQL Remote 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は、SQL Remote 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 ユーザのユーザ名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名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。
- **SQL Remote からの完全な DBA パーミッション**　SQL Remote から接続している場合、REMOTE DBA 権限を持つユーザには、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 権限を付与するタイミング

統合データベースでユーザを作成するときに、統合データベースのパブリッシャと各リモート・ユーザに REMOTE DBA 権限を付与することをおすすめします。抽出ユーティリティ (dbxtract) または**データベース抽出ウィザード**でリモート・データベースを抽出すると、リモート・ユーザ

がリモート・データベースのパブリッシャになり、PUBLISH パーミッションと REMOTE DBA 権限が付与されます。

この推奨内容を実行すると、ユーザの管理が簡単になります。SQL Remote から (ユーザに完全な DBA 権限を付与します)、または他のクライアント・アプリケーションから (この場合、REMOTE DBA 権限ではユーザに追加のパーミッションを付与しません) に関わらず、各リモート・ユーザにはデータベースに接続するための 1 つのユーザ名のみが必要です。

♦ **REMOTE DBA 権限を付与するには、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. 左ウィンドウ枠で、**[ユーザとグループ]** フォルダまたは **[SQL Remote ユーザ]** フォルダのいずれかを選択します。
3. ユーザを右クリックして、**[プロパティ]** を選択します。
4. **[権限]** タブをクリックして、**[リモート DBA]** オプションを選択します。
5. **[適用]** をクリックしてから **[OK]** をクリックします。

♦ **REMOTE DBA 権限を付与するには、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. GRANT REMOTE DBA 文を実行して、ユーザに REMOTE DBA 権限を付与します。

   次に例を示します。

```
GRANT REMOTE DBA TO dbremote
IDENTIFIED BY dbremote;
```

参照

- 「REMOTE DBA 権限」 『SQL Anywhere サーバ - データベース管理』
- 「GRANT REMOTE DBA 文 [Mobile Link] [SQL Remote]」 『SQL Anywhere サーバ - SQL リファレンス』

REMOTE DBA 権限の取り消し

REMOTE DBA 権限を取り消すと、ユーザは SQL Remote から接続したときにデータベースに対する完全な DBA 権限が付与されなくなります。「REMOTE DBA 権限の付与」 35 ページを参照してください。

次の手順を使用して、ユーザの REMOTE DBA 権限を取り消します。

♦ **REMOTE DBA 権限を取り消すには、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. 左ウィンドウ枠で、**[ユーザとグループ]** フォルダまたは **[SQL Remote ユーザ]** フォルダのいずれかを選択します。

3. ユーザを右クリックして、**[プロパティ]** を選択します。
4. **[権限]** タブをクリックして、**[リモート DBA]** オプションをクリアします。
5. **[OK]** をクリックします。

♦ **REMOTE DBA 権限を取り消すには、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. REVOKE REMOTE DBA 文を実行して、ユーザの REMOTE DBA 権限を取り消します。

   次に例を示します。

   ```
   REVOKE REMOTE DBA FROM dbremote;
   ```

参照

- 「REVOKE REMOTE DBA 文 [SQL Remote]」『SQL Anywhere サーバ - SQL リファレンス』
- 「REMOTE パーミッションの付与」 31 ページ

サブスクリプション

パブリケーションに対してユーザをサブスクライブするには、「サブスクリプション」を作成します。パブリケーションに含まれる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すべて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。

パブリケーションに対するサブスクリプション

パブリケーションに対してユーザをサブスクライブするには、次の情報が必要です。

- **ユーザ名**　パブリケーションに対してサブスクライブされるユーザ。このユーザには REMOTE パーミッションが必要です。
- **パブリケーション名**　ユーザがサブスクライブされるパブリケーションの名前。
- **サブスクリプション値**　サブスクリプション値は、パブリケーションに SUBSCRIBE BY 句が含まれている場合にのみ適用されます。サブスクリプション値とは、パブリケーションの SUBSCRIBE BY 句に対してテストを行った値です。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 SUBCRIBE BY 句として保持する場合は、サブスクリプションを作成するときに、サブスクライブされるユーザの従業員 ID の値を指定する必要があります。サブスクリプション値は常に文字列です。

  この値は、パブリケーションに SUBSCRIBE BY 句が含まれている場合にのみ必要です。「SUBSCRIBE BY 句を使用して一部のローのみをパブリッシュする」 20 ページを参照してください。

♦ パブリケーションに対してユーザをサブスクライブするには、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. パブリケーションを選択します。
4. 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。
5. **[ファイル] - [新規] - [SQL Remote サブスクリプション]** を選択します。
6. **SQL Remote サブスクリプション作成ウィザード**の指示に従います。

   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。

♦ パブリケーションに対してリモート・ユーザをサブスクライブするには、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 SUBSCRIPTION 文を実行して、パブリケーションに対してユーザをサブスクライブします。

   たとえば、次の文では、CustomerPub パブリケーションに対してユーザ名 SamS のサブスクリプションを作成します。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 CustomerPub
     FOR SamS;
   ```

   たとえば、次の文では、PubOrders パブリケーションに対してユーザ名 SamS のサブスクリプションを作成します。このパブリケーションには、サブスクリプション式 SalesRepresentative が定義されていて、Sam 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;
   ```

参照

- 「CREATE SUBSCRIPTION 文 [SQL Remote]」『SQL Anywhere サーバ - SQL リファレンス』
- 「SUBSCRIBE BY 句を使用して一部のローのみをパブリッシュする」 20 ページ

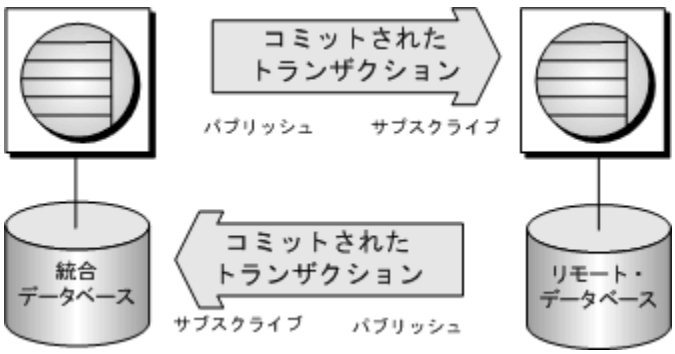
トランザクション・ログ・ベースのレプリケーションの概要

SQL Remote は、次の変更をレプリケートします。

- **コミットされた変更**　データベースに対して行われ、トランザクション・ログに記録された変更。
- **パブリケーションに含まれているデータを修正する変更**　SQL Remote は、トランザクション・ログをスキャンして、パブリケーションに含まれているローに対する変更のうち、コミットされた変更があるかどうかを調べます。さらに、SQL 文をメッセージにパッケージし、そのメッセージをサブスクライバ・データベースに送信します。

統合データベースでは、パブリケーションに含まれるトランザクション・ログ内のすべてのコミットされた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・データベースでは、パブリケーションに含まれるトランザクション・ログ内のすべてのコミットされた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。



データベース・サーバによるパブリケーションの処理

SQL Anywhere データベース・サーバは、パブリケーションを評価してトランザクション・ログに情報を書き込むコンポーネントです。パブリケーションの数が多くなると、データベース・サーバが実行する必要のある処理も多くなります。

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、式の値をトランザクション・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。

トランザクション・ログに情報を追加すると、次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**　テーブルが複数のパブリケーションに属している場合は、多数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データベース・サーバの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て、トランザクション・ログの情報が増加する場合があります。このことがパフォーマンスに影響する可能性があります。

SQL Remote で処理されるサブスクリプション

SQL Remote 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。

送信フェーズの間、SQL Remote Message Agent は現在のサブスクリプションをトランザクション・ログ内のパブリケーション情報にマップして、リモート・ユーザごとに適切なメッセージを生成します。「Message Agent (dbremote)」 104 ページを参照してください。

参照

- 「トランザクション・ログ」 『SQL Anywhere サーバ - データベース管理』

INSERT 文と DELETE 文のレプリケート

SQL Remote では、INSERT 文と DELETE 文は最も簡単にレプリケートできる文です。

- あるデータベースで INSERT 文を実行すると、この文は INSERT 文として SQL Remote システム内でサブスクライブされたデータベースに送信されます。
- あるデータベースで DELETE 文を実行すると、この文は DELETE 文として SQL Remote システム内でサブスクライブされたデータベースに送信されます。

統合データベース

SQL Remote では、統合データベースのトランザクション・ログからの INSERT 文または DELETE 文をそれぞれコピーして、挿入または削除されるローに対してサブスクライブする各リモート・データベースにこれらの文を送信します。テーブルのカラムのサブセットに対してのみサブスクライブされている場合、リモート・データベースに送信される INSERT 文にはそのカラムのみが含まれます。

リモート・データベース

SQL Remote では、リモート・データベースのトランザクション・ログからの INSERT 文または DELETE 文をそれぞれコピーして、挿入または削除されるローに対してサブスクライブする統合データベースに、この文を送信します。次に、統合データベースによって文が適用され、その結果、トランザクション・ログへの書き込みが行われます。統合データベースのトランザクション・ログが SQL Remote によって処理されると、変更内容は最終的に他のリモート・サイトに送信されます。SQL Remote は、最初に文を実行したリモート・ユーザにはその文を送信しません。

参照

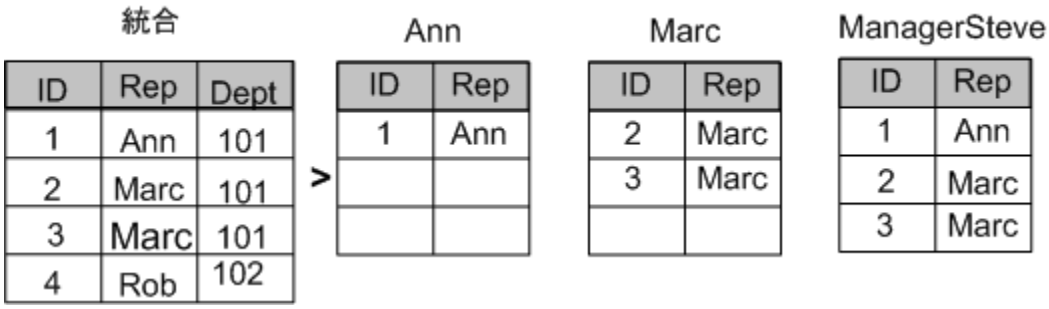
- 「重複プライマリ・キー・エラー」 61 ページ
- 「ローが見つからないエラー」 57 ページ

UPDATE 文のレプリケーション

UPDATE 文では、データベースに入力された文とまったく同じ文がレプリケートされない場合があります。次の例で、UPDATE 文がレプリケートされる仕組みについて説明します。

- 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でローを更新する場合、UPDATE 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。

UPDATE 文がレプリケートされる仕組みを説明するため、次の例では、統合データベースとユーザ Ann、Marc、ManagerSteve の 3 つのリモートデータベースを使用します。



統合データベースには、次の文を使用して作成された、cons という名前のパブリケーションが存在します。

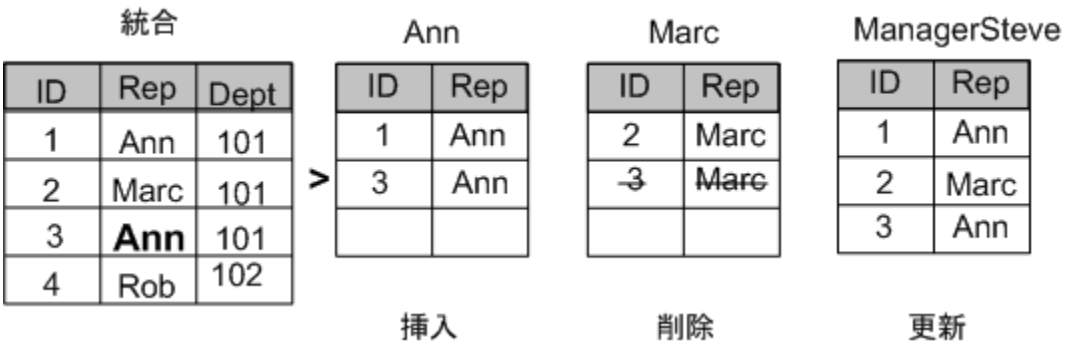
```
CREATE PUBLICATION "cons"."p1" (
 TABLE "DBA"."customers" ( "ID", "Rep") SUBSCRIBE BY repid
);
```

Ann と Marc は、cons パブリケーションに対して、それぞれの Rep カラム値でサブスクライブします。ManagerSteve は、cons パブリケーションに対して、Ann と Marc の Rep カラム値を使用してサブスクライブします。次の文は、パブリケーション cons に対して 3 人のユーザをサブスクライブします。

```
CREATE SUBSCRIPTION
  TO "cons"."p1"( 'Ann' )
  FOR "Ann";
CREATE SUBSCRIPTION
  TO "cons"."p1"( 'Marc' )
  FOR "Marc";
CREATE SUBSCRIPTION
  TO "cons"."p1"( 'Ann' )
  FOR "ManagerSteve";
CREATE SUBSCRIPTION
  TO "cons"."p1"( 'Marc' )
  FOR "ManagerSteve";
```


統合データベースでは、Marc から Ann にローの Rep 値を変更する UPDATE 文が、次のようにレプリケートされます。

- Marc に対しては DELETE 文として。
- Ann に対しては INSERT 文として。
- ManagerSteve に対しては UPDATE 文として。



サブスクライバ間のローの再割り当ては、営業担当者間に顧客を定期的に再割り当てする営業支援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。

参照

- 「更新の競合」 49 ページ

プロシージャのレプリケート

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。代わりに、プロシージャの個々の動作 (INSERT 文、UPDATE 文、DELETE 文) がレプリケートされます。

トリガのレプリケート

通常、リモート・データベースには、統合データベースに存在するトリガと同じトリガが定義されています。

デフォルトでは、SQL Remote はトリガによって実行される動作をレプリケートしません。代わりに、統合データベースでトリガを起動するアクションが、リモート・データベースにレプリケートされると、その複製トリガはリモート・データベースで自動的に起動します。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。

- **RESOLVE UPDATE トリガのレプリケーション**　競合解析または 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リモー

ト・データベースを含む、すべてのリモート・データベースにレプリケートされます。「更新の競合に対するデフォルトの解決」 49 ページを参照してください。

- **BEFORE トリガのレプリケーション**　更新中のローを変更する BEFORE トリガの動作は、UPDATE 文が動作する前にレプリケートされます。

  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 トリガは、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 文がレプリケートされると、リモート・データベースの BEFORE UPDATE トリガが起動されるためです。

  また、カラムを最終更新時間に設定する BEFORE UPDATE トリガは、UPDATE 文がレプリケートされた時間を取得します。

  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 BEFORE UPDATE トリガが存在しないこと、または BEFORE UPDATE 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。

トリガの動作をレプリケートするオプション

メッセージを送信するときにトリガの動作をすべてレプリケートするには、Message Agent (dbremote) の -t オプションを使用します。「Message Agent (dbremote)」 160 ページを参照してください。

-t 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 (レプリケートされたトリガの動作が適用されるときと、リモート・データベースでトリガを起動するとき) 実行されないようにします。

トリガの動作が 2 回実行されないようにするには、次のいずれかのオプションを使用します。

- トリガの本文を IF CURRENT REMOTE USER IS NULL ... END IF 文で囲みます。
- SQL Remote ユーザ名に対して、SQL Anywhere の fire_triggers オプションを Off に設定します。「fire_triggers オプション [互換性]」 『SQL Anywhere サーバ - データベース管理』を参照してください。

トリガ・エラーの回避

パブリケーションにデータベースのサブセットのみが含まれる場合、統合データベースのトリガは、統合データベースには存在し、リモート・データベースには存在しないテーブルやローを参照できます。このようなトリガがリモート・データベースで起動すると、エラーが発生します。こうしたエラーを回避するには、IF 文を使用してトリガの動作を条件付きとし、次の処理を実行します。

- 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。「CURRENT PUBLISHER 特別値」 『SQL Anywhere サーバ - SQL リファレンス』を参照してください。
- 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。「システム関数」 『SQL Anywhere サーバ - SQL リファレンス』を参照してください。

- 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。

抽出ユーティリティ (dbxtract)

デフォルトでは、データベース抽出ユーティリティ (dbxtract) と**データベース抽出ウィザード**によってトリガ定義が抽出されます。

参照

- 「CURRENT REMOTE USER 特殊定数の使用」 53 ページ

データ定義文

データ定義文 (CREATE、ALTER、DROP) は、パススルー・モードで実行した場合を除き、SQL Remote によってレプリケートされません。

「SQL Remote のパススルー・モード」 150 ページを参照してください。

データ型

SQL Remote は、文字セットを変換しません。

互換性のあるソート順と文字セットの使用

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。

BLOB

「BLOB」 には、LONG VARCHAR、LONG BINARY、TEXT、IMAGE の各データ型が含まれています。

SQL Remote は、INSERT 文または UPDATE 文をレプリケートすると、BLOB 値の代わりに変数を使用します。つまり、BLOB は細かく分割して各部分がレプリケートされます。分割された部分は受信データベースで SQL 変数を使用して再構成され、連結されます。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';
```

この変数によって、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。

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 値であるかのようにレプリケートされます。「blob_threshold オプション [SQL Remote]」 『SQL Anywhere サーバ - データベース管理』を参照してください。

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スレッショルドより長いと、カラムの古い値は UPDATE 文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を小さくしますが、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 を使用している場合に競合を検出するには、次の方法を使用します。

♦ verify_threshold が設定されているときに BLOB 値との競合を検出するには、次の手順に従います。

1. BLOB を更新するときに、同じテーブル内の last_modified カラムも必ず更新されるようにデータベースを設定します。
2. last_modified カラムが BLOB カラムと一緒にレプリケートされるようにパブリケーションを設定します。

   BLOB カラムと last_modified カラムがレプリケートされるときに、last_modified カラムの値を確認できます。last_modified カラムと競合している場合は、BLOB カラムとも競合しています。

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

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 SQL Remote 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。

document_in_progress テーブルを使用することをおすすめします。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。

参照

- 「SET 文」 『SQL Anywhere サーバ - SQL リファレンス』

日付と時刻

日付カラムまたは時刻カラムをレプリケートする場合、SQL Remote 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、SQL Remote に 1998 年 5 月 2 日という日付を 1998-05-02 として送信するように命令します。

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

日付と時刻をレプリケートする場合は、次の点を考慮します。

- 時刻、日付、タイムスタンプには、SQL Remote システム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用されている年、月、日の順序が、date_order データベース・オプション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。

参照

- 「sr_date_format オプション [SQL Remote]」 『SQL Anywhere サーバ - データベース管理』
- 「sr_time_format オプション [SQL Remote]」 『SQL Anywhere サーバ - データベース管理』
- 「sr_timestamp_format [SQL Remote]」 『SQL Anywhere サーバ - データベース管理』
- 「date_order オプション [データベース]」 『SQL Anywhere サーバ - データベース管理』

レプリケーションの競合とエラー

SQL Remote を使用すると、複数のデータベースでデータベースを更新できます。ただし、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設計する必要があります。

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。競合を適切に処理すると、SQL Remote では問題を起こしません。

競合は、数多くのシステムで発生します。SQL Remote では、SQL Remote システム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。「更新の競合に対するデフォルトの解決」 49 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。

- **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) を削除します。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。「ローが見つからないエラー」 57 ページを参照してください。
- **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て、関連するプライマリ・キーが含まれていない場合、その外部キーを参照する INSERT 文は失敗します。

  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。

  「参照整合性エラー」 58 ページを参照してください。
- **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。「重複プライマリ・キー・エラー」 61 ページを参照してください。

配信エラー

配信エラーとその解決方法については、「保証されたメッセージ配信システムの概要」 114 ページを参照してください。

参照

- 「レプリケーション・エラーのレポートと処理」 142 ページ

更新の競合

データが読み取り用に共有されている場合、または各ロー (プライマリ・キーによって識別される) が 1 つのデータベースのみで更新される場合、更新の競合は発生しません。複数のデータベースでデータが更新された場合にのみ、更新の競合が発生します。

UPDATE 文をレプリケートするため、SQL Remote はローごとに個別の UPDATE 文を発行します。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 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。たとえば、競合が検出されると、統合データベースでは次の処理を実行できます。

  ○ デフォルトの競合解決を使用する。「更新の競合に対するデフォルトの解決」 49 ページを参照してください。

  ○ VERIFY 句を使用する、カスタマイズされた競合解決を使用する。「VERIFY 句を使用したカスタム競合解決」 50 ページを参照してください。

  ○ トリガを使用する、カスタマイズされた競合解決を使用する。「トリガを使用したカスタム競合解決」 52 ページを参照してください。

  プライマリ・キーの更新に競合解決は適用されない
  UPDATE 文の競合は、プライマリ・キーの更新には適用されません。SQL Remote システムで、プライマリ・キーを更新しないでください。適切な設計を行って、プライマリ・キーの競合をシステムから取り除く必要があります。

- **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ます。ローが削除されたり、プライマリ・キーが別のユーザによって変更されたりしている場合は、更新すべきローが見つかりません。

  リモート・データベースでは、更新は行われません。

  統合データベースでは、更新は行われません。

- **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 つのリモート・データベースで同じローが別々に更新され、その変更が統合データベースにレプリケートされた場合は、統合データベースに到着した最初の変更が適用され、2 番目のデータベースの変更は適用されません。

  そのため、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。

更新の競合に対するデフォルトの解決

更新の競合は、複数のサイトでデータが更新された場合にのみ発生します。次に例を示します。

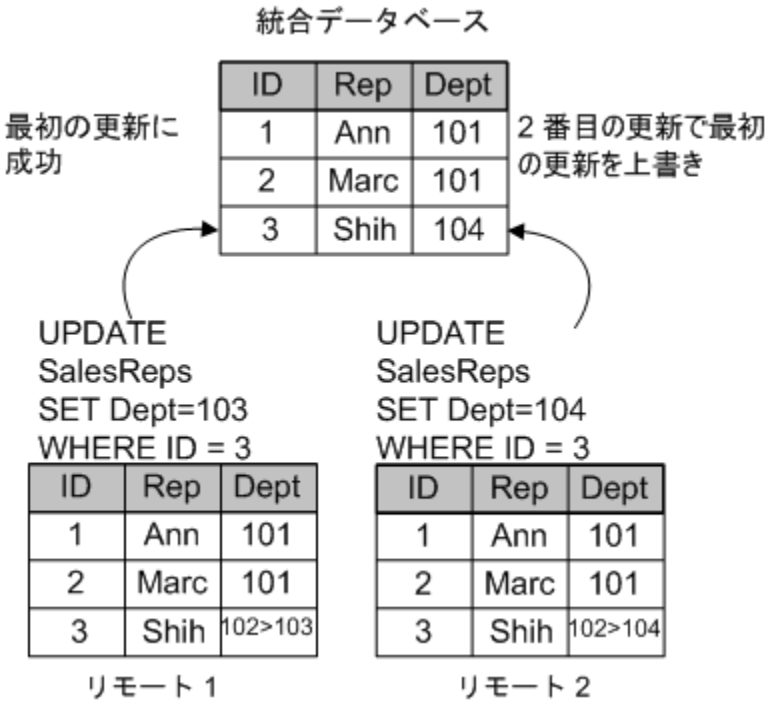
1. ユーザ 1 が、リモート・サイト 1 でローを更新します。
2. ユーザ 2 が、リモート・サイト 2 で同じローを更新します。
3. ユーザ 1 による更新内容が、統合データベースに送信されて適用されます。
4. ユーザ 2 による更新内容が、統合データベースに送信されます。

   この種の更新の競合を解決する「デフォルト」の方法は、次のとおりです。

   a. 新しい方のオペレーションが成功します (この例では、ユーザ 2 のオペレーション)。この値が統合データベースの値になります。また、この値が、ローに対してサブスクライブされるその他すべてのデータベースにレプリケートされます。

   b. その他の更新がすべて失われます (この例では、ユーザ 1 の更新)。

   c. 競合はレポートされません。



VERIFY 句を使用したカスタム競合解決

SQL Remote は、VERIFY 句を使用するメッセージの中で UPDATE 文を生成します。UPDATE 文は、既存の 1 つ以上のローの値を新しい値に変更します。VERIFY 句を含む UPDATE 文には、そのローの既存の値も含まれています。

UPDATE 文を適用すると、統合データベースは既存のローの値を、リモート・データベースで既存のローの値が変更された後に期待される値と比較します。更新の競合は、VERIFY 句の値がデータベースのローと一致しない場合に、データベース・サーバによって検出されます。

たとえば、更新の競合は、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送信されて適用されます。
4. ユーザ 2 による更新内容が、統合データベースに送信されます。

   UPDATE 文に VERIFY 句が含まれているため、統合データベースは競合を検出できます。統合データベースでは、SQL Remote がローに含まれている値を、ユーザ 2 が送信した古いロー値と比較します。これらの値は同じではないため、更新の競合が発生します。

   更新の競合が検出されると、統合データベースでは次の処理を実行します。

   a. 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を起動します。

      「競合解決トリガ」を定義して、更新の競合を処理します。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。「トリガを使用したカスタム競合解決」 52 ページを参照してください。

   b. UPDATE 文を実行します。

   c. 競合解決トリガのすべてのアクションと UPDATE 文が、すべてのリモート・データベースに送信されます。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ています。

      通常、SQL Remote は、トリガ・アクションをレプリケートしません。これは、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ためです。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。

5. リモート・データベースは、統合データベースから UPDATE 文を受信します。

   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更新の競合が含まれていると、RESOLVE UPDATE トリガは起動されません。

6. リモート・データベースで、UPDATE 文が処理されます。

   プロセスの最後では、システム全体でデータの一貫性が保たれます。

VERIFY 句を持つ UPDATE 文

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*]
[**ORDER BY** *expression* [**ASC** | **DESC**], ...]

VERIFY 句が使用できるのは、*table-list* パラメータにテーブルが 1 つしか存在しない場合のみです。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 文がデータベースで実行されても、SQL Remote 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。

参照

- 「UPDATE 文 [SQL Remote]」『SQL Anywhere サーバ - SQL リファレンス』

VERIFY_ALL_COLUMNS オプションの使用

デフォルトでは、データベース・オプション verify_all_columns は Off に設定されています。設定が Off の場合は、更新されたカラムのみが検証されます。verify_all_columns オプションが On に設定されている場合は、次のようになります。

- レプリケートされた UPDATE 文に対して、すべてのカラムが検証されます。
- いずれかのカラムが異なっているときには必ず RESOLVE UPDATE トリガが起動します。
- 各 UPDATE 文に対して送信される情報が多くなるため、メッセージのサイズが大きくなります。

verify_all_columns オプションは、PUBLIC グループ用、または SQL Remote 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。「verify_all_columns オプション [SQL Remote]」『SQL Anywhere サーバ - データベース管理』を参照してください。

抽出ユーティリティ (dbxtract)

リモート・データベースを抽出する前に統合データベースで verify_all_columns オプションが設定されている場合、抽出ユーティリティ (dbxtract) と**データベース抽出ウィザード**によってリモート・データベースの verify_all_columns オプションが設定されます。

トリガを使用したカスタム競合解決

カスタム競合解決はいくつかの手法を取ることができます。たとえば、アプリケーションによっては、解決は次のようになる可能性があります。

- 元のトランザクションの日付を比較します。「日付の競合解決」 53 ページを参照してください。
- 2 つ以上の更新の結果に対して計算を実行します。「在庫数の競合解決」 54 ページを参照してください。
- 競合をテーブルにレポートします。「CURRENT REMOTE USER 特殊定数の使用」 53 ページを参照してください。

カスタム競合解決を使用するには、RESOLVE UPDATE トリガを記述する必要があります。

RESOLVE UPDATE 競合解決トリガの使用

RESOLVE UPDATE トリガの起動後に、各ローが更新されます。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
```

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 句内のカラムのみです。その他のカラムの場合は、**カラムが見つかりません。**というエラーが返されます。

CURRENT REMOTE USER 特殊定数の使用

DBREMOTE は、マニュアルに記載されていない REMOTE USER 文を実行すると、CURRENT REMOTE USER を設定します。値は、DBREMOTE からの接続の場合を除き、NULL になります。

競合のレポートをテーブルに配置する RESOLVE UPDATE トリガに CURRENT REMOTE USER を使用して、競合を発生させたユーザを特定できます。

参照

- 「CREATE TRIGGER 文」 『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』

日付の競合解決

日付の競合を解決する方法を説明するため、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。

土曜日に更新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、月曜日に更新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更新が処理され、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。ただし、このカラムの更新の競合は、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。

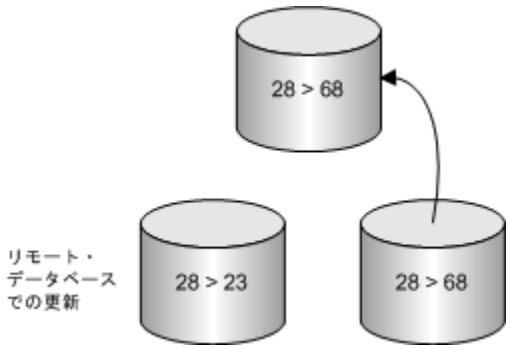
```
CREATE TRIGGER contact_date RESOLVE UPDATE
 ON Contacts
 REFERENCING OLD AS old_name
  NEW AS new_name
 FOR EACH ROW
 BEGIN
   IF new_name.contact_date <
     old_name.contact_date THEN
         SET new_name.contact_date
          = old_name.contact_date
   END IF
 END;
```

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

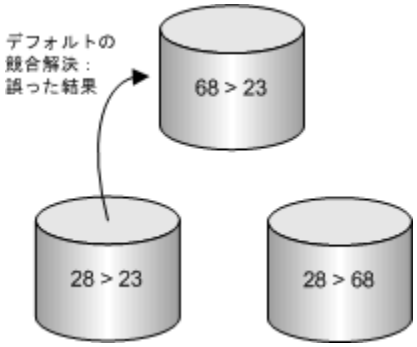
スポーツ用品メーカのための倉庫システムがあるとします。製品情報のテーブルには、各製品の在庫数を記録する Quantity カラムがあります。このカラムへの更新は、通常は在庫数を引いていくことであり、新しく製品が搬入される場合は、在庫数を追加します。

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、別の営業担当者は T シャツの返品を 40 枚受け取りました。この営業担当者は、自分のリモート・データベースに返品を入力し、倉庫での変更を統合データベースにレプリケートして、Quantity カラムの値を 40 追加して 68 とします。

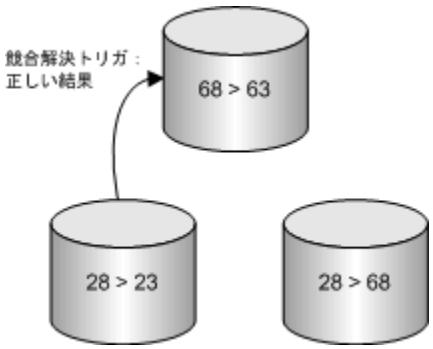


このエントリがデータベースに追加され、現在の Quantity カラムは、在庫に 68 枚の S サイズのタンクトップ T シャツがあることを示しています。ここで最初の営業担当者からの更新を受信すると、28 から 23 に変更するという内容にもかかわらず、実際のカラム値は 68 であるという競合が SQL Anywhere で検出されます。

デフォルトでは、より新しい更新が処理され、在庫レベルが間違った値 23 に設定されます。



この例の競合は、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た結果を算出し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 文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 文が実行されます。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 文によって、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。

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) を削除します。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。

UPDATE 文と DELETE 文を正しくレプリケートするには、すべてのプライマリ・キー・カラムをアーティクルに含める必要があります。

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。

WHERE 句とプライマリ・キー

プライマリ・キー・カラムは、レプリケートされた UPDATE 文と DELETE 文の WHERE 句の中で使用します。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。「INSERT 文と DELETE 文のレプリケート」 41 ページを参照してください。

参照

- 「レプリケーション・エラーのレポートと処理」 142 ページ

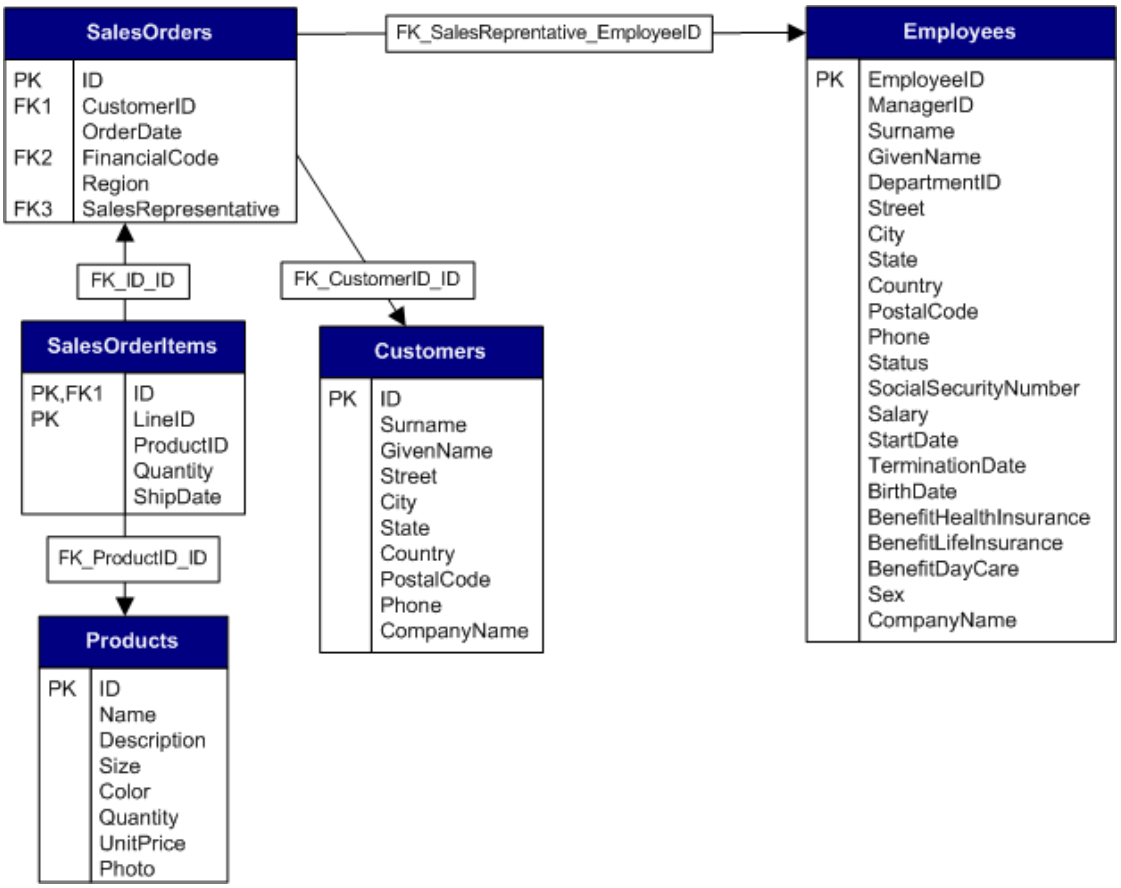
参照整合性エラー

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ることがよくあります。そのため、参照整合性制約によって、データベースの一貫性が保たれます。「エンティティ整合性と参照整合性の確保」『SQL Anywhere サーバ - SQL の使用法』を参照してください。

データベースの一部のみをレプリケートする場合は、レプリケート後のデータベースでも引き続き参照整合性が保たれていることを確認する必要があります。

参照テーブルがレプリケートされないエラーを回避する必要があります。リモート・データベースには、レプリケートされないテーブルを参照する外部キーが含まれないようにしてください。

たとえば、統合データベースの 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。SalesOrders.SalesRepresentative は、プライマリ・キー Employees.EmployeeID を参照する外部キーです。



次の例では、Employees テーブルを含まず、SalesOrder テーブル全体を含むパブリケーション PubSales を作成します。

```
CREATE PUBLICATION PubSales (
  TABLE Customers,
```



```
  TABLE SalesOrders,
  TABLE SalesOrderItems,
);
```

リモート・ユーザ Rep1 は、PubSales パブリケーションに対してサブスクライブします。次に、統合データベースから Rep1 を抽出して、Rep1 用のデータベースの作成を試みます。ただし、Rep1 には Employees テーブルがないため、データベースの作成に失敗します。この問題を回避するには、次の処理を実行します。

- **外部キー参照を削除する**　外部キー参照を除外するには、抽出ユーティリティ (dbxtract) を使用するときに -xf オプションを指定します。

  ただし、リモート・データベースから外部キー参照を削除する場合、リモート・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ことを防ぐ制約はありません。

  リモート・データベースで SalesRepresentative カラムに無効な値が挿入された場合、レプリケートされた INSERT 文は統合データベースで失敗します。

- **不足しているテーブルをパブリケーションに追加する**　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) を追加します。次に例を示します。

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

参照

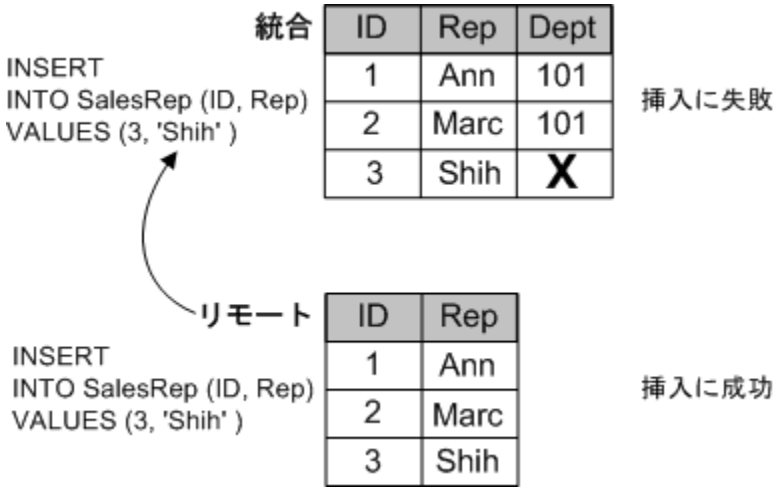
- 「レプリケーション・エラーのレポートと処理」 142 ページ

挿入エラー

リモート・データベースから統合データベースに INSERT 文をレプリケートする場合は、パブリケーションから次のカラムのみを除外できます。

- 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
この例の例外として、BEFORE トリガを使用して、INSERT 文に含まれていないカラムを管理する場合があります。

参照

- 「レプリケーションの競合とエラー」 48 ページ

重複プライマリ・キー・エラー

すべてのユーザが同じデータベースに接続する場合は、各 INSERT 文がユニークなプライマリ・キーを使用するすることが保証されるため、問題は発生しません。ユーザがプライマリ・キー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。異なるリモート・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。各リモート・データベースでは、プライマリ・キーの値がユニークであるため、各文が正常に実行されます。

ただし、この 2 人のユーザが自分のデータベースを同じ統合データベースにレプリケートすると、問題が発生します。統合データベースに対する最初のデータベースのレプリケーションは、正常に実行されます。ただし、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 番目の挿入の実行は失敗します。

必ずユニークとするプライマリ・キー値

プライマリ・キーのエラーを回避するには、データベースでローが挿入されるときに、そのプライマリ・キーがシステムのすべてのデータベース間でユニークであることが必ず保証されている必要があります。この目的を達成するには、次のようないくつかの方法があります。

1. SQL Anywhere データベース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。「グローバル・オートインクリメント・カラム」 61 ページを参照してください。
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。「プライマリ・キー・プールの使用」 62 ページを参照してください。

これらの方法のいずれか一方または両方を使用すると、プライマリ・キーの重複を回避できます。

参照

- 「レプリケーション・エラーのレポートと処理」 142 ページ

グローバル・オートインクリメント・カラム

GLOBAL AUTOINCREMENT のデフォルトを使用して、リモート・データベースごと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。

カラムに GLOBAL AUTOINCREMENT のデフォルトを指定すると、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。

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、リモート・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 ～ 11000 の範囲から選択されます。ID 番号

11 が割り当てられた別のリモート・データベースからは、11001 ～ 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。

参照

- 「GLOBAL AUTOINCREMENT デフォルト」 『SQL Anywhere サーバ - SQL の使用法』

DEFAULT GLOBAL AUTOINCREMENT の宣言

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

分割サイズ

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負でない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することをおすすめします。

例

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と顧客名を保持する文字列カラム) を持つテーブルが作成されます。ID 番号カラムである ID では GLOBAL AUTOINCREMENT のデフォルトを使用し、その分割サイズは 5000 です。

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

参照

- 「CREATE TABLE 文」 『SQL Anywhere サーバ - SQL リファレンス』
- 「ALTER TABLE 文」 『SQL Anywhere サーバ - SQL リファレンス』

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 システムで、各データベースのプライマリ・キー値のセットを格納するテーブルです。マスタ・プライマリ・キー・プール・テーブルが作成され、統合データベースに格納されます。リモート・ユーザは、統合データベースのプライマリ・キー・プール・テーブルに対してサブスクライブし、自分のプライマリ・キー値のセットを受信します。リモート・ユーザが新しいローをテーブルに挿入する場合は、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。

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。

- **プライマリ・キー・プール・テーブル**　統合データベースでは、システム内のデータベースごとに有効なプライマリ・キー値を保持するテーブルが必要です。「プライマリ・キー・プール・テーブルの作成」 63 ページを参照してください。
- **補充プロシージャ**　統合データベースでは、値を補充したキー・プール・テーブルを維持するストアド・プロシージャが必要です。「キー・プールの入力と補充」 64 ページを参照してください。
- **キー・プールの共有**　システムの各リモート・データベースは、統合データベースのキー・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ライブする必要があります。「プライマリ・キー・プールのレプリケート」 64 ページを参照してください。
- **データ入力プロシージャ**　リモート・データベースでは、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。「キー・プールのプライマリ・キーの使用」 66 ページを参照してください。

プライマリ・キー・プール・テーブルの作成

♦ **プライマリ・キー・プール・テーブルを作成するには、次の手順に従います (SQL の場合)。**

1. 統合データベースで次の文を実行して、プライマリ・キー・プール・テーブルを作成します。

```
CREATE TABLE KeyPool (
  table_name VARCHAR(128) NOT NULL,
  value INTEGER NOT NULL,
  location CHAR(12) NOT NULL,
  PRIMARY KEY (table_name, value),
);
```

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します。たとえば、統合データベースのみに営業担当者が新しく追加されると、Customers テーブルのみがプライマリ・キー・プールを必要とし、このカラムは重複します。
value	プライマリ・キー値のリストを保持します。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す。
location	受信者の識別子。システムによっては、この値が SalesReps テーブルの rep_key 値と同じになる場合があります。また、営業担当者以外のユーザが存在するシステムもあり、このようなシステムでは、この 2 つの識別子は別々である必要があります。

2. パフォーマンスを向上させるには、次の文を実行して、プライマリ・キー・テーブルにインデックスを作成します。

```
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ト

プライマリ・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できます。次の手順を使用して、プライマリ・キー・プールに個別のパブリケーションを作成し、そのパブリケーションに対してユーザをサブスクライブします。

♦ **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. 統合データベースで、プライマリ・キー・プールのデータ用の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。

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・プールの入力と補充

リモート・ユーザが新しい顧客を追加するたびに、そのリモート・ユーザが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。統合データベースのプライマリ・キー・プール・テーブルの内容は定期的に補充して、リモート・データベースに新しいプライマリ・キーをレプリケートする必要があります。

♦ **はじめにプライマリ・キー・プールを値で埋めるには、次の手順に従います (SQL の場合)。**

1. 統合データベースで、プライマリ・キー・プールを値で埋めるためのプロシージャを作成します。

トリガを使用してキー・プールを補充しない
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。

次に例を示します。

```
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;
```

2. 各ユーザのプライマリ・キー・プールにプライマリ・キーの初期値を挿入します。

   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();
   ```

   ReplenishPool 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。

3. 定期的に ReplenishPool を実行します。

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。

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、挿入を実行するストアド・プロシージャを使用します。

♦ **プライマリ・キーを使用するには、次の手順に従います (SQL の場合)。**

1. リモート・データベース上で実行するプロシージャを作成して、プライマリ・キー・プール・テーブルからプライマリ・キーを取得します。

   たとえば、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;
```

   NewKey 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。

2. リモート・データベース上で実行するプロシージャを作成して、サブスクライブされたテーブルに新しいローを挿入します。

   たとえば、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挿入します。

```
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。

リモート・データベース間でローを分割する

各リモート・データベースは、統合データベースに格納されたデータの異なるサブセットを持つことができます。パブリケーションとサブスクリプションを作成すると、リモート・データベース間でデータが分割されます。

共通部分がないように切断分割にすることも、重複を持たせて分割することもできます。たとえば、従業員ごとに独自の顧客セットを持っていて、かつ顧客を共有していない場合は、「切断」分割になります。複数のリモート・データベースに存在するように顧客を共有している場合、分割は「重複」を含みます。

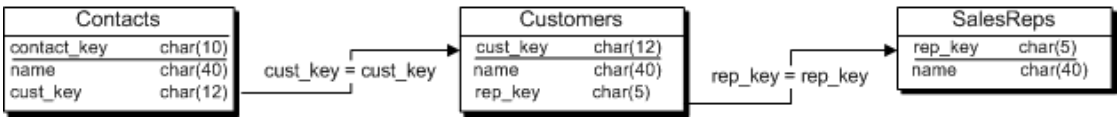
場合によっては、テーブルにサブスクリプション式がなくても、テーブルのローを分割する必要があります。「切断データ分割の使用」 69 ページを参照してください。

場合によっては、データベースに多対多の関係が存在する場合、テーブルを分割する必要があります。「重複分割の使用」 75 ページを参照してください。

切断データ分割の使用

リモート・データベースがデータを共有していない場合、データ分割は切断となります。たとえば、各営業担当者には独自の顧客セットがあり、他の営業担当者と顧客を共有していません。

次の例では、3 つのテーブル Customers、Contacts、SalesReps に、営業担当者と顧客間の対話に関する情報が格納されています。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。



Contacts、Customers、SalesReps テーブルの説明

次の表では、Customers、Contacts、SalesReps の各データベース・テーブルについて説明します。詳細については、「切断データ分割の使用」 69 ページを参照してください。

テーブル	説明	テーブル定義
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。	CREATE TABLE Contacts (contact_key CHAR(12) NOT NULL, name CHAR(40) NOT NULL, cust_key CHAR(12) NOT NULL, FOREIGN KEY REFERENCES Customers, PRIMARY KEY (contact_key));
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。	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));

テーブル	説明	テーブル定義
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。	CREATE TABLE SalesReps (rep_key CHAR(12) NOT NULL, name CHAR(40) NOT NULL, PRIMARY KEY (rep_key));

営業担当者は、次の情報を提供するパブリケーションに対してサブスクライブする必要があります。

- **社内のすべての営業担当者のリスト**　次の文では、SalesRep 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 SalesRepData (
  Table SalesReps ...)
);
```

- **営業担当者に割り当てられている顧客のリスト**　この情報は、Customers テーブルで取得できます。次の文では、Customers テーブル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には、Customers テーブルの rep_key カラムの値に一致するローが含まれています。

```
CREATE PUBLICATION SalesRepData (
  TABLE Customers SUBSCRIBE BY rep_key ...
);
```

- **割り当てられている顧客の窓口情報のリスト**　この情報は、Contacts テーブルで取得できます。Contacts テーブルは営業担当者の間で分割する必要がありますが、SalesRep テーブルの rep_key 値への参照は含みません。この問題を解決するには、Customers テーブルの rep_key カラムを参照する Contacts アーティクルで、サブクエリを使用します。

  次の文では、Contacts テーブル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には、Customers テーブルの rep_key カラムを参照するローが含まれています。

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

  Customers テーブルの 1 つのローには、Contacts テーブルの現在のローの cust_key 値が含まれています。SUBSCRIBE BY 文の中で WHERE 句を使用すると、サブクエリは必ず単一の値のみを返します。

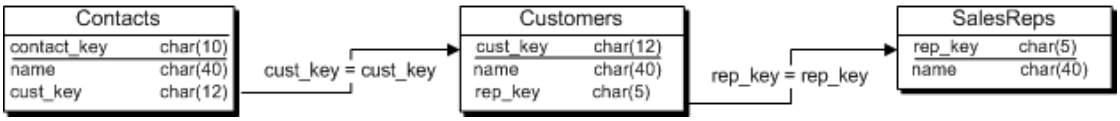
次の文では、完全なパブリケーションが作成されます。

```
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 )
);
```

BEFORE UDPATE トリガの使用

次の例では、3 つのテーブル Customers、Contacts、SalesReps に、営業担当者と顧客間の対話に関する情報が格納されています。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。



テーブルの詳細については、「Contacts、Customers、SalesReps テーブルの説明」 69 ページを参照してください。

営業担当者は、SalesRep テーブルのコピー、営業担当者に割り当てられた顧客の詳細が格納されている Customers テーブルのコピー、顧客に対応する窓口の詳細が格納された Contacts テーブルのコピーを提供するパブリケーションに対してサブスクライブします。たとえば、各営業担当者は、次のパブリケーションに対してサブスクライブします。

```
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 )
);
```

このパブリケーションの詳細については、「切断データ分割の使用」 69 ページを参照してください。

参照整合性の維持

サブスクライバ間のローの再割り当ては、営業担当者間に顧客を定期的に再割り当てする営業支援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。

統合データベースでは、新しい営業担当者に顧客が再割り当てされると、Customers テーブルの rep_key 値が更新されます。

次の文では、顧客 cust1 を別の営業担当者 rep2 に再割り当てします。

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

この更新は、次のようにレプリケートされます。

- 以前の営業担当者のリモート・データベースの Customers テーブルに対する DELETE 文として。
- 新しい営業担当者のリモート・データベースの Customers テーブルに対する INSERT 文として。

Contacts テーブルは変更されません。統合データベースのトランザクション・ログには、Contacts テーブルに関するエントリがありません。そのため、リモート・データベースの SQL Remote は、Contacts テーブルの cust_key ローを再割り当てできません。この再割り当てができないことによって、以前の営業担当者のリモート・データベースの Contacts テーブルには、すでに存在しない顧客の cust_key 値が含まれるという参照整合性の問題が発生します。

解決法としては、BEFORE UPDATE トリガを使用します。BEFORE UPDATE トリガはデータベース・テーブルをまったく変更しませんが、統合データベースのトランザクション・ログにエントリを作成します。

この BEFORE UPDATE トリガは、次のように起動する必要があります。

- UPDATE 文が実行される前。したがって、ローの BEFORE 値が評価され、トランザクション・ログに追加されます。
- 各文ではなく FOR EACH ROW。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する必要があります。

たとえば、次の文では、BEFORE UPDATE トリガを作成します。

```
CREATE TRIGGER "UpdateCustomer" BEFORE UPDATE OF "rep_key"
// only fire the trigger when we modify rep_key, not any other column
ORDER 1 ON "Cons"."Customers"
/* REFERENCING OLD AS old_name NEW AS new_name */
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
;
```

SQL Remote は、トランザクション・ログに記録された情報を使用して、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。

この文を実行した後、統合データベースのトランザクション・ログには 2 つのエントリが含まれています。

- BEFORE UPDATE トリガによって生成された、Contacts テーブルの INSERT 文と DELETE 文。

```
--BEGIN TRIGGER-1029-0000461705
--BEGIN TRANSACTION-1029-0000461708
BEGIN TRANSACTION
```



```
go
--UPDATE PUBLICATION-1029-0000461711 Cons.Contacts
--PUBLICATION-1029-0000461711-0002-NEW_SUBSCRIBE_BY-rep2
--PUBLICATION-1029-0000461711-0002-OLD_SUBSCRIBE_BY-rep1
--NEW-1029-0000461711
--INSERT INTO Cons.Contacts(contact_key,name,cust_key)
--VALUES ('5','Joe','cust1')
go
--OLD-1029-0000461711
--DELETE FROM Cons.Contacts
-- WHERE contact_key='5'
go
--END TRIGGER-1029-0000461743
```

- 実行された元の UPDATE 文と、それぞれローが挿入または削除されたユーザの INSERT 文と DELETE 文。

```
--PUBLICATION-1029-0000461746-0002-NEW_SUBSCRIBE_BY-rep2
--PUBLICATION-1029-0000461746-0002-OLD_SUBSCRIBE_BY-rep1
--NEW-1029-0000461746
--INSERT INTO Cons.Customers(cust_key,name,rep_key)
--VALUES ('cust1','company1','rep2')
go
--OLD-1029-0000461746
--DELETE FROM Cons.Customers
-- WHERE cust_key='cust1'
go
--UPDATE-1029-0000461746
UPDATE Cons.Customers
   SET rep_key='rep2'
VERIFY (rep_key)
VALUES ('1')
 WHERE cust_key='cust1'
go
--COMMIT-1029-0000461785
COMMIT WORK
```

SQL Remote は、BEFORE タグと AFTER タグのトランザクション・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 文、UPDATE 文、または DELETE 文を取得するかが決定されます。

- ユーザが BEFORE list に含まれていて AFTER list に含まれていない場合は、DELETE 文が Contacts テーブルに送信されます。
- ユーザが AFTER list に含まれていて BEFORE list に含まれていない場合は、INSERT 文が Contacts テーブルに送信されます。
- ユーザが BEFORE list と AFTER list の両方に含まれている場合は、Contacts テーブルに対して何も実行されませんが、Customers テーブルの UPDATE 文は送信されます。

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 文が送信されます。

トリガに関する注意事項

次の例では、BEFORE UPDATE トリガを使用する必要があります。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。

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

この例では、BEFORE トリガを使用します。

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

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。

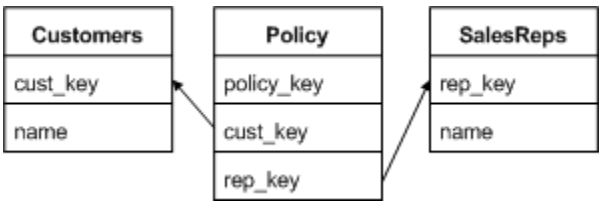
この例では、サブスクリプション式は 1 つの値を返します。ただし、複数の値を返すサブクエリも使用できます。サブスクリプション式の値は、更新を実行した後の値とする必要があります。

この例では、ローへのサブスクライバのみが新しい営業担当者になります。既存のサブスクライバと新しいサブスクライバを持つローの例については、「重複分割の使用」 75 ページを参照してください。

重複分割の使用

リモート・データベースがデータを共有している場合、データ分割は重複となります。たとえば、営業担当者は、担当者間で顧客を共有します。

3 つのテーブル Customers、Policy、SalesReps に、営業担当者と顧客間の対話に関する情報が格納されているとします。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。Policy テーブルには、Customers と SalesReps の両テーブルへの外部キーがあります。Customers と SalesReps 間には、多対多の関係があります。



Customers、Policy、SalesReps テーブルの説明

次の表では、Customers、Policy、SalesReps の各データベース・テーブルについて説明します。詳細については、「重複分割の使用」 75 ページを参照してください。

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むプライマリ・キーのカラム。
	● **name** 各顧客の名前を含むカラム。
	次の文で、このテーブルが作成されます。
	`CREATE TABLE Customers (`
	`  cust_key CHAR(12) NOT NULL,`
	`  name CHAR(40) NOT NULL,`
	`  PRIMARY KEY (cust_key)`
	`);`

テーブル	説明

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外部キーを含むカラム。
	● **rep_key**　取り引きを行う営業担当者の外部キーを含むカラム。
	次の文で、このテーブルが作成されます。
	`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 )`
	`);`

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。
	次の文で、このテーブルが作成されます。
	`CREATE TABLE SalesReps (`
	`  rep_key CHAR(12) NOT NULL,`
	`  name CHAR(40) NOT NULL,`
	`  PRIMARY KEY (rep_key)`
	`);`

データの分割

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ための新しい問題が発生します。

営業担当者は、次の情報を提供するパブリケーションに対してサブスクライブする必要があります。

- **SalesReps テーブル全体**　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,
...
```

- **データに対してサブスクライブされた営業担当者を含む、取り引きを記録した Policy テーブルのロー**　このアーティクルは、SUBSCRIBE BY 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のみを、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。

- **データに対してサブスクライブされた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**　Customers テーブルには、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への参照はありません。パブリケーションでサブクエリを使用して、この問題に対処できます。

  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、重複しているサブスクリプションがあります。

  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。

次の文では、完全なパブリケーションが作成されます。

```
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
  )
);
```

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、テーブルの分割を重複にできます。

サブスクライバ間でローを再割り当てする場合の参照整合性の維持

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。この例の Policy テーブルの変更内容は、以前の営業担当者に正しくレプリケートされます。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、以前の営業担当者にレプリケートされません。

トリガがない場合は、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る可能性があります。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

この解決法としては、Policy テーブルに変更が行われたときに起動する BEFORE トリガを記述します。この特別なトリガでは、データベース・テーブルは変更されませんが、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

たとえば、次の文では、Policy テーブルに対する挿入を追跡する BEFORE INSERT トリガを作成し、リモート・データベースに適切なデータが含まれることを保証します。

```
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

次の文では、Policy テーブルからの削除を追跡する BEFORE DELETE トリガを作成します。

```
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;
```

UPDATE PUBLICATION 文の SUBSCRIBE BY 句にはサブクエリが含まれており、このサブクエリは複数の値を返すことができます。

複数の値を返すサブクエリ

UPDATE PUBLICATION 文の SUBSCRIBE 句に含まれているサブクエリは 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。

  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ている必要があります。

- 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。

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。

  サブスクリプション・クエリの結果セットには、削除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ている必要があります。

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 テーブル上の更新については、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 INSERT と BEFORE DELETE の各トリガの機能を組み合わせる BEFORE UPDATE トリガを作成する必要があります。

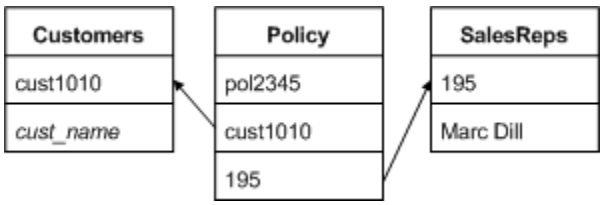
多対多の関係における subscribe_by_remote オプションの使用

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。

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で発生する問題を解決します。顧客が複数の営業担当者に所属できるため、次の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
   ),
);
```

たとえば、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 Customers テーブルに新しいローを挿入し、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。



統合データベースでは、SQL Remote によって Customers ローの挿入が実行され、この挿入の実行時に SQL Anywhere によってトランザクション・ログにサブスクリプション値が記録されます。

あとで SQL Remote がトランザクション・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 が Off に設定されていると、この新しい顧客は DELETE 文として Marc Dill に送信されます。

subscribe_by_remote が On に設定されているかぎり、SQL Remote 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 文は 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。

参照

- 「subscribe_by_remote オプション [SQL Remote]」 『SQL Anywhere サーバ - データベース管理』

各データベースへのユニークな ID 番号の割り当て

各リモート・データベースには、異なる ID 番号を割り当てる必要があります。ID 番号はさまざまな方法で作成して配布できます。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。

global_database_id オプションの使用

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、$pn + 1$ ～ $p(n + 1)$ 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 ～ 4000 になります。

global_database_id が負ではない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。この例では、別の分割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。「プライマリ・キー・プールの使用」 62 ページを参照してください。

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。

参照

- 「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』

global_database_id 値の設定

♦ **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 (SQL の場合)。**

● global_database_id オプションの値を設定します。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 = 10;
```

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 ～ 105000 の範囲から選択されます。

参照

● 「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』

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 (dbxtract) または**データベース抽出ウィザード**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、ユニークなデータベース ID 番号を設定するタスクを自動化できます。

♦ **ユニークなデータベース ID 番号の設定を自動化するには、次の手順に従います。**

1. **sp_hook_dbxtract_begin** という名前のストアド・プロシージャを作成します。

   たとえば、user_id が 101 であるリモート・ユーザ user2 のデータベースを抽出するには、次の文を実行します。

```
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';
```

   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 などとなります。

2. -v オプションを指定した抽出ユーティリティ (dbxtract) または**データベース抽出ウィザード**を実行して、リモート・データベースを抽出します。抽出ユーティリティは、次のタスクを実行します。

   a. #hook_dict という名前のテンポラリ・テーブルを作成し、次の内容を追加します。

name	value
extracted_db_global_id	抽出されるユーザ ID

sp_hook_dbxtract_begin プロシージャを作成してローの 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、DEFAULT GLOBAL 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。

- 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、global_database_id は 101 に設定されたままになります。

b. **sp_hook_dbxtract_begin** を呼び出します。

c. プロシージャ・フックのデバッグに役立つように、次の情報を出力します。

- 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

参照

- 「#Hook_dict テーブル」 182 ページ
- 「SQL Remote システム・プロシージャ」 182 ページ
- 「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』

SQL Remote の配備と管理

この項では、SQL Remote の配備と管理について説明します。

SQL Remote の管理

目次

SQL Remote の管理の概要

統合データベースから SQL Remote システムを配備して管理します。

♦ **SQL Remote システムを配備して管理するには、次の手順に従います。**

1. 統合データベースを設定します。

   「SQL Remote システムの作成」 14 ページを参照してください。

2. SQL Remote システムを確認してテストします。

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

3. リモート・データベースを作成し、設計の配備を行います。

   統合データベースの DBA として、次の手順で SQL Remote を配備します。

   a. 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作成し、そのサブスクリプションを開始します。「リモート・データベースの抽出」 90 ページを参照してください。

   b. 各リモート・ユーザのコンピュータに、SQL Anywhere データベース・サーバ、リモート・データベース、SQL Remote、クライアント・アプリケーションをインストールします。「組み込みデータベース・アプリケーションの配備」 『SQL Anywhere サーバ - プログラミング』と「SQL Remote の配備」 『SQL Anywhere サーバ - プログラミング』を参照してください。

4. Message Agent (dbremote) を実行してメッセージを交換します。

   メッセージを交換するには、次の手順を実行する必要があります。

   a. 統合データベースとリモート・データベースで Message Agent (dbremote) を継続モードとバッチ・モードのどちらで実行するかを決定します。 「Message Agent (dbremote) のモードの選択」 98 ページ。

   b. ユーザ名、Message Agent (dbremote) 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。「Message Agent (dbremote) の概要」 98 ページを参照してください。

5. メッセージを管理します。

   保証されたメッセージ配信システムを使用して、数多くのデータベース間でやりとりされるメッセージを管理します。「保証されたメッセージ配信システムの概要」 114 ページを参照してください。

6. パフォーマンスを向上させます。

   「SQL Remote のパフォーマンス向上」 104 ページを参照してください。

7. バックアップとリカバリの方式を実装します。

   統合データベースにバックアップとリカバリの方式を作成し、実装してください。「SQL Remote システムのバックアップ」 132 ページを参照してください。

8. エラーを処理します。
   「レプリケーション・エラーのレポートと処理」 142 ページを参照してください。
9. 必要に応じて、ソフトウェアとデータベース・スキーマをアップグレードします。
   「アップグレードと再同期」 148 ページを参照してください。

リモート・データベースの抽出

リモート・ユーザ用のデータベースを作成するには、統合データベースからリモート・データベースを「抽出」します。

データベース抽出ウィザードまたは抽出ユーティリティ (dbxtract) のいずれかを使用して、特定のリモート・ユーザ用のリモート・データベースを統合データベースから抽出できます。どちらの方法を使用しても、次の 1 つ以上のタスクを実行できます。

- **自動的にスキーマとデータを抽出して新規または既存のデータベースに直接再ロードする**　これは、SQL Remote を学習する場合に適した方法です。この方法を使用した場合、データの中間コピーはディスク上に作成されません。この方法により、データのセキュリティが向上します。ただし、実装にかかる時間は長くなります。「リモート・データベースの自動抽出」 90 ページを参照してください。
- **スキーマとデータをファイルに抽出してから、新規または既存のデータベースにこれらをロードする**　SQL Remote を配備する場合は、この方法をおすすめします。スキーマ・ファイルを編集して、リモート・データベースの抽出と作成をカスタマイズできます。「再ロード・ファイルへのリモート・データベースの抽出」 92 ページを参照してください。

  複数のリモート・データベースを効率的に作成する方法については、「複数のリモート・データベースの作成」 95 ページを参照してください。

リモート・データベースの自動抽出

リモート・データベースの自動抽出と、その代替となる再ロード・ファイルへのリモート・データベースの抽出については、「リモート・データベースの抽出」 90 ページを参照してください。

次の手順を使用して、統合データベースを抽出し、スキーマとデータを新しいデータベースに再ロードします。データの中間コピーはディスク上に作成されません。

♦ リモート・データベースを自動的に抽出するには、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統合データベースに接続します。
2. **[ツール] - [SQL Anywhere 11] - [データベースの抽出]** を選択します。
3. プロンプトが表示されたら、**[新しいデータベースへの抽出と再ロード]** を選択します。

   プロンプトが表示されたら、**[構造とデータを抽出]** を選択します。
4. ウィザードの指示に従い、デフォルト値をそのまま使用します。

   適切なスキーマ、リモート・ユーザ、パブリケーション、サブスクリプション、トリガを含む新しいリモート・データベースが作成されます。デフォルトでは、統合データベースのデータがリモート・データベースに抽出され、サブスクリプションが開始されます。ただし、ウィザードでは、Message Agent が開始されないため、メッセージが交換されません。「Message Agent (dbremote) の概要」 98 ページを参照してください。

♦ **リモート・データベースを自動的に抽出するには、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統合データベースに接続します。
2. 抽出ユーティリティ (dbxtract) を実行し、-ac オプションを指定して既存のデータベースに抽出するか、または -an オプションを指定して新しいデータベースに抽出します。「抽出ユーティリティ (dbxtract)」 170 ページを参照してください。

   -an オプションを指定する場合は、空のデータベースを作成してから、抽出ユーティリティ (dbxtract) を実行してください。たとえば、次のコマンドは *mydata.db* という名前の空のデータベースを作成します。

   ```
   dbinit c:¥remote¥mydata.db
   ```

   次のコマンドを実行して、*c:¥consolidateddata.db* にある統合データベースから新しいリモート・データベースを抽出します。新しいデータベースは、field_user という名前のリモート・ユーザ用であり、*c:¥remote¥mydata.db* に作成されます。

   ```
   dbxtract -c "UID=DBA;PWD=sql;DBF=c:¥consolidateddata.db"
   -an c:¥remote¥mydata.db field_user
   ```

   適切なスキーマ、リモート・ユーザ、パブリケーション、サブスクリプション、トリガを含む新しいリモート・データベース *mydata.db* が作成されます。デフォルトでは、統合データベースのデータがリモート・データベースに抽出され、サブスクリプションが開始されます。ただし、抽出ユーティリティ (dbxtract) では、Message Agent が開始されないため、メッセージが交換されません。「Message Agent (dbremote) の概要」 98 ページを参照してください。

参照

● 「再ロード・ファイルへのリモート・データベースの抽出」 92 ページ

再ロード・ファイルへのリモート・データベースの抽出

再ロード・ファイルへのリモート・データベースの抽出と、その代替となるリモート・データベースの自動抽出については、「リモート・データベースの抽出」 90 ページを参照してください。

ほとんどの配備シナリオでは、リモート・データベースの抽出と作成をカスタマイズする必要があります。カスタム抽出を作成するには、コマンド・ファイルと一連のテキスト・ファイルへのデータベースの抽出を選択します。次に、必要に応じてこれらのファイルを編集できます。

データベースをファイルに抽出する場合は、作成するファイルを次のいずれかから選択します。

- ***reload.sql* という名前の SQL コマンド・ファイル。このファイルには、リモート・データベース・スキーマの構築に必要な文が含まれています。** 「抽出ユーティリティ (dbxtract)」 170 ページの -n オプションを参照してください。

  たとえば、次のコマンドを実行します。

  ```
  dbxtract -c "UID=DBA;PWD=sql;DBF=c:¥cons¥cons.db" -n "c:¥remote¥reload.sql" UserName
  ```

- **一連のデータ・ファイル。それぞれに、データベース・テーブルの内容が含まれています。** 一連のデータ・ファイル。それぞれに、データベース・テーブルの内容が含まれています。データ・ファイルを格納する、*extract* という名前の新しいディレクトリが作成されます。これらのファイルを使用して、既存のリモート・データベースにデータをロードできます。「抽出ユーティリティ (dbxtract)」 170 ページの -d オプションを参照してください。

  たとえば、次のコマンドを実行します。

  ```
  dbxtract -c "UID=DBA;PWD=sql;DBF=c:¥cons¥cons.db" -d "c:¥remote1¥" UserName
  ```

- ***reload.sql* ファイルとデータ・ファイルの両方。** データ・ファイルを格納する、*extract* という名前の新しいディレクトリが作成されます。*reload.sql* ファイルには、データ・ファイルをロードする命令が含まれています。たとえば、次のコマンドを実行します。

  ```
  dbxtract -c "UID=DBA;PWD=sql;DBF=c:¥cons¥cons.db" "c:¥remote1¥reload.sql" UserName
  ```

reload.sql ファイル

reload.sql ファイルには、データベース・スキーマを構築する SQL 文が記述され、次のオブジェクトを作成するコマンドが含まれています。

- パブリッシャ、リモート・ユーザ、統合ユーザ
- パブリケーションとサブスクリプション
- メッセージ型
- テーブル
- ビュー
- トリガ
- プロシージャ

reload.sql の編集が必要な場合があります。
抽出ユーティリティ (dbxtract) 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。「reload.sql ファイルの編集」 93 ページを参照してください。

次の手順を使用して、reload.sql ファイルからリモート・データベースを作成します。

♦ reload.sql ファイル (コマンド・ライン) 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. 抽出ユーティリティ (dbxtract) を使用して、データベース・スキーマとデータをファイルに抽出します。たとえば、次のコマンドを実行します。

   ```
   dbxtract -c "UID=DBA;PWD=sql;DBF=c:¥cons¥cons.db" "c:¥remote¥reload.sql" UserName
   ```

   デフォルトでは、指定されたリモート・ユーザのサブスクリプションが自動的に開始されます。
2. 必要に応じて、*reload.sql* を編集します。「reload.sql ファイルの編集」 93 ページを参照してください。
3. 空の SQL Anywhere データベースを作成します。

   たとえば、次のコマンドを実行します。

   ```
   dbinit c:¥rem1¥rem1.db
   ```
4. 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。

   たとえば、次のコマンドを実行します。

   ```
   READ remote¥reload.sql
   ```

   適切なスキーマ、リモート・ユーザ、パブリケーション、サブスクリプション、トリガを含む新しいリモート・データベース *rem1.db* が作成されます。ただし、抽出ユーティリティ (dbxtract) では、Message Agent が開始されないため、メッセージが交換されません。「Message Agent (dbremote) の概要」 98 ページを参照してください。

参照

- 「リモート・データベースの自動抽出」 90 ページ

reload.sql ファイルの編集

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。たとえば、次の場合に *reload.sql* ファイルを編集します。

リモート・データベースへのレプリケートされないテーブルの追加

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。抽出ユーティリティ (dbxtract) と**データベース抽出ウィザード**は、レプリケートされないテーブルを統合データベースから抽出できません。

データベースを抽出した後、*reload.sql* を編集してこのようなテーブルを追加してください。

プロシージャ、トリガ、ビューの抽出

デフォルトでは、抽出ユーティリティ (dbxtract) と**データベース抽出ウィザード**は、すべてのストアド・プロシージャ、トリガ、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。たとえば、プロシージャには、データベースの、リモート・サイトに含まれない部分を参照するものがあります。

データベースを抽出した後、*reload.sql* を編集して、不必要なプロシージャ、トリガ、ビューを削除してください。

多層システムでの抽出ユーティリティ (dbxtract) の使用

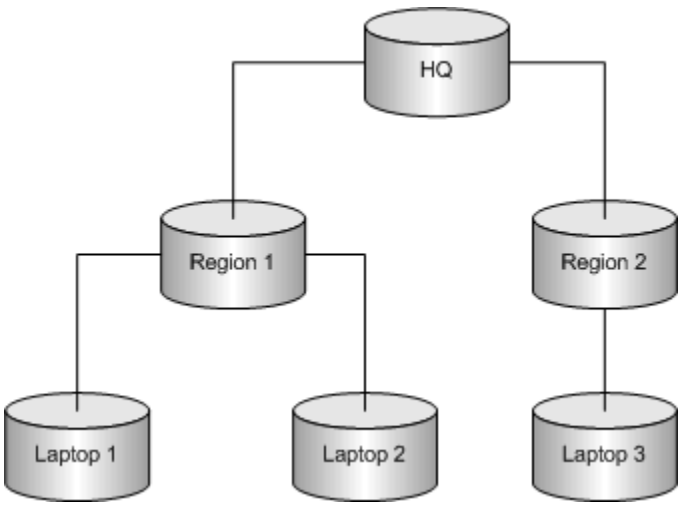
「多層階層システムのデータベースの抽出」 94 ページを参照してください。

参照

- 「複数のリモート・データベースの作成」 95 ページ
- 「再ロード・ファイルへのリモート・データベースの抽出」 92 ページ

多層階層システムのデータベースの抽出

多層配置での抽出ユーティリティ (dbxtract) と**データベース抽出ウィザード**の役割について理解するために、3 層の SQL Remote システムについて検討してみます。次の図は、このシステムを表したものです。



♦ **3 層のシステムにリモート・データベースを作成するには、次の手順に従います。**

1. 最上位レベルの統合データベース HQ で抽出ユーティリティ (dbxtract) を使用して、第 2 レベルのデータベース Region 1 と Region 2 を作成します。

2. 第 2 レベルのデータベース Region 1 と Region 2 で抽出ユーティリティ (dbxtract) を使用して、ユーザ Laptop 1、Laptop 2、Laptop 3 用の第 3 レベルのデータベースを作成します。第 2 レベルのデータベースは、第 1 レベルのデータベース HQ のリモート・データベースであり、第 3 レベルのデータベース Laptop 1、Laptop 2、Laptop 3 の統合データベースです。

多層階層システムでのデータベースの再抽出

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用の**スキーマ**を再抽出する必要がある場合、抽出ユーティリティ (dbxtract) によってリモート・ユーザ (Laptop 1、Laptop 2、Laptop 3) が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されます。このため、これらの第 3 レベルのユーザとそのサブスクリプションを手動で再作成してください。

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の**データ**のみを再抽出する必要がある場合、抽出ユーティリティ (dbxtract) はリモート・ユーザに影響しません。「抽出ユーティリティ (dbxtract)」 170 ページの -d オプションを参照してください。

完全に修飾されたパブリケーション定義

完全に修飾されたパブリケーション定義には、WHERE 句と SUBSCRIBE BY 句があります。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、リモート・データベースは、すべてのローをレプリケートし、統合データベースに戻します。

参照

- 「複数のリモート・データベースの作成」 95 ページ
- 「再ロード・ファイルへのリモート・データベースの抽出」 92 ページ

複数のリモート・データベースの作成

次の手順を使用して、複数のリモート・データベースを効率的に作成します。

♦ **複数のリモート・データベースを作成するには、次の手順に従います。**

1. 統合データベースのコピーを作成し、統合データベースからリモート・ユーザのサブスクリプションを開始します。次に例を示します。

   a. サブスクリプションを開始し、その後直ちに統合データベースと Message Agent (実行されている場合) を停止します。

      統合データベースの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。データベースの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。統合データベースでサブスクリプションを開始すると、サブスクライバのデータベースがまだ存在しない場合でも、メッセージをパッケージしてサブスクライバに送信できます。「START SUBSCRIPTION 文 [SQL Remote]」『SQL Anywhere サーバ - SQL リファレンス』と「サブスクリプションの開始」 155 ページを参照してください。

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

b. 統合データベースをコピーします。

デフォルトでは、抽出ユーティリティ (dbxtract) と**データベース抽出ウィザード**はともに、独立性レベル 3 で実行されます。この独立性レベルでは、抽出されたデータベース内のデータは、データベース・サーバのデータと一致しますが、他のユーザがデータベースを使用できなくなる場合があります。統合データベースのコピーに対してリモート・データベースを抽出することをおすすめします。

c. 統合データベースを再起動します。統合データベースで Message Agent が実行されていた場合は、Message Agent も再起動します。

2. 統合データベースのコピーからリモート・データベースのスキーマを抽出します。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。ただし、多数のリモート・データベースがある場合は、この処理に時間がかかることがあります。

リモート・データベースのスキーマを抽出する場合、次のオプションを選択します。

a. リモート・データベースのスキーマのみを抽出する。

デフォルトでは、抽出ユーティリティ (dbxtract) と**データベース抽出ウィザード**はともに、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。ただし、ほとんどの配備シナリオでは、各リモート・データベースで使用するデータは異なりますが、スキーマは同じです。抽出ユーティリティ (dbxtract) または**データベース抽出ウィザード**を使用してユーザごとにスキーマとデータの両方を抽出すると、同じスキーマが繰り返し抽出されることになります。「抽出ユーティリティ (dbxtract)」 170 ページの -n オプションを参照してください。

b. プライマリ・キーを基準にデータを順序付ける。

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。データがプライマリ・キーを基準に順序付けられると、リモート・データベースへのデータのロード処理が高速になります。「抽出ユーティリティ (dbxtract)」 170 ページの -u オプションを参照してください。

3. *reload.sql* ファイルを使用して空のリモート・データベースを作成します。このデータベース・ファイルをコピーして必要な数のリモート・データベースを作成します。「再ロード・ファイルへのリモート・データベースの抽出」 92 ページを参照してください。

4. リモート・データベースごとに、各リモート・ユーザに固有の SQL Remote 定義を定義します。「ユーザ・パーミッション」 25 ページを参照してください。

5. リモート・ユーザごとに、統合データベースからユーザに対応するデータのみを抽出します。「抽出ユーティリティ (dbxtract)」 170 ページの -d オプションを参照してください。

6. 各リモート・ユーザのデータを対応するリモート・データベースにロードします。

各リモート・データベースが作成されると、その情報はライブ統合データベースより古いものになります。

しかし、Message Agent (dbremote) を実行すると、各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。「Message Agent (dbremote) の概要」 98 ページを参照してください。

参照

- 「多層階層システムのデータベースの抽出」 94 ページ
- 「reload.sql ファイルの編集」 93 ページ

Message Agent (dbremote) の概要

Message Agent (dbremote) 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 (dbremote) をインストールし、システム内の各データベースで実行してください。Message Agent (dbremote) 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。

- **メッセージ送信時の Message Agent (dbremote) のタスク**
  - 各パブリッシャ・データベースのトランザクション・ログをスキャンして、トランザクション・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。
  - サブスクライバにメッセージを送信する。
  - Message Agent (dbremote) がメッセージの再送要求を受信した場合は、要求を作成したデータベースにメッセージを再送する。
  - システム・テーブルのメッセージ情報を保持し、保証されたメッセージ配信システムを管理する。

「メッセージ送信のタスク」 110 ページを参照してください。

- **メッセージ受信時の Message Agent (dbremote) のタスク**
  - 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。
  - 欠落しているメッセージの再送を要求する。
  - システム・テーブルのメッセージ情報を保持し、保証されたメッセージ配信システムを管理する。

「メッセージ受信のタスク」 104 ページを参照してください。

接続

Message Agent (dbremote) 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 (dbremote) の起動中は常に接続されています。
- **トランザクション・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。
- **トランザクション・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。

Message Agent (dbremote) のモードの選択

Message Agent (dbremote) は次のいずれかのモードで動作します。

- **継続モード**　継続モードでは、Message Agent (dbremote) は各リモート・ユーザの送信頻度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データベースにおいて有用です。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。

  「継続モードでの Message Agent (dbremote) の実行」 99 ページを参照してください。

- **バッチ・モード**　バッチ・モードでは、Message Agent (dbremote) は受信メッセージを受信して処理し、トランザクション・ログを 1 回スキャンし、出力メッセージを作成して送信した後、停止します。

  バッチ・モードは、たまに接続するリモート・データベースにおいて有用です。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・データベースがメイン・ネットワークにダイヤルアップするようなときです。

  「バッチ・モードでの Message Agent (dbremote) の実行」 101 ページを参照してください。

Message Agent (dbremote) の稼働条件

SQL Remote には、高い柔軟性があります。システム内では、複数のデバイスと複数のオペレーティング・システム上で両方のモードの Message Agent (dbremote) を実行できます。ただし、SQL Remote には、次の稼働条件があります。

- **REMOTE DBA 権限または DBA 権限が必要**　Message Agent (dbremote) 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 権限の付与」 35 ページを参照してください。
- **システム内にある各 Message Agent (dbremote) のメッセージの最大長は、すべて同じ値にする**　このメッセージ長は、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。デフォルト値は 50000 バイトです。このメッセージ長は、Message Agent (dbremote) の -l オプションを使用して変更できます。「Message Agent (dbremote)」 160 ページを参照してください。

継続モードでの Message Agent (dbremote) の実行

通常、統合データベースは、継続モードで実行されます。継続モードとその代替となるバッチ・モードについては、「Message Agent (dbremote) の概要」 98 ページを参照してください。

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

1. REMOTE 権限を持つすべてのユーザが SEND AT または SEND EVERY の頻度を指定していることを確認します。

   継続モードでは、Message Agent (dbremote) は各リモート・ユーザのプロパティに含まれる SEND AT または SEND EVERY の頻度で指定された時刻に、メッセージを送信します。「送信頻度の設定」 100 ページを参照してください。

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

   Windows では、Message Agent (dbremote) の名前は *dbremote.exe* です。UNIX では、この名前は dbremote です。Mac OS X では、SyncConsole を使用して Message Agent (dbremote) を起動することもできます。「Mac OS X での Message Agent (dbremote) の実行」 102 ページと「UNIX での Message Agent (dbremote) の実行」 103 ページを参照してください。

   たとえば、次の文ではデータベース・ファイル *c:¥mydata.db* で dbremote を継続モードで実行します。接続にはユーザ名 ManagerSteve とパスワード sql を使用しています。

   ```
   dbremote -c "UID=ManagerSteve;PWD=sql;DBF=c:¥mydata.db" -l 40000
   ```

   ユーザ名 ManagerSteve には、REMOTE DBA 権限または DBA 権限のいずれかが必要です。-l オプションで定義されているメッセージの最大長は、システム内のすべてのデータベースで同じである必要があります。「Message Agent (dbremote) の稼働条件」 99 ページを参照してください。

   指定できる dbremote オプションの完全なリストについては、「Message Agent (dbremote)」 160 ページを参照してください。

継続モードでのサービスとしての Message Agent (dbremote) の実行

Message Agent (dbremote) を継続モードで実行する場合、サーバの稼働中は常に Message Agent (dbremote) を実行させておくことができます。このためには、Message Agent (dbremote) 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たとき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、「Windows 用サービス・ユーティリティ (dbsvc)」 『SQL Anywhere サーバ - データベース管理』を参照してください。

送信頻度の設定

統合データベースなどで、Message Agent (dbremote) を継続モードで実行する場合は、すべての REMOTE ユーザが送信頻度を指定していることを確認してください。継続モードでは、Message Agent (dbremote) は SEND AT または SEND EVERY プロパティで指定された時刻に、メッセージを送信します。

Message Agent (dbremote) では、次の送信頻度の値をサポートしています。

- **SEND EVERY**　メッセージを送信する間隔の待機時間を指定します。

  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メッセージが同時に送信されます。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。

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

- **SEND AT**　メッセージを送信する時刻を指定します。

毎日、指定した時刻に更新内容が送信されます。送信時間をずらすことなく、この設定の時間をできるだけ変えずに使用してください。データベースがビジーでない時刻を選択してください。

- **デフォルト設定 (SEND 句なし)** ユーザが SEND AT 句または SEND EVERY 句を指定していない場合は、Message Agent (dbremote) はバッチ・モードで起動し、起動のたびにメッセージを送信して停止します。「バッチ・モードでの Message Agent (dbremote) の実行」 101 ページを参照してください。

非常に頻繁なメッセージの送信

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、送信間隔をあまり短くしないでください。

♦ **送信頻度を設定するには、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. 左ウィンドウ枠で、**[SQL Remote ユーザ]** ディレクトリを選択します。
3. ユーザを右クリックし、**[プロパティ]** を選択します。
4. **[SQL Remote]** タブをクリックします。
5. **[次の間隔で送信]** または **[毎日次の時刻に送信]** のいずれかを選択し、時間を指定します。

参照

- 「GRANT REMOTE 文 [SQL Remote]」 『SQL Anywhere サーバ - SQL リファレンス』

バッチ・モードでの Message Agent (dbremote) の実行

次の手順を使用して、SQL Remote をバッチ・モードで実行します。バッチ・モードとその代替となる継続モードについては、「Message Agent (dbremote) の概要」 98 ページを参照してください。

♦ **バッチ・モードで Message Agent (dbremote) を実行するには、次の手順に従います。**

1. 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・ユーザが最低 1 つあることを確認します。

   リモート・ユーザの「すべて」に SEND AT 句または SEND EVERY 句が定義されており、メッセージを送受信してから停止する必要がある場合は、-b オプションを使用して Message Agent (dbremote) を起動してください。

2. Message Agent (dbremote) を起動します。

   Windows では、Message Agent (dbremote) の名前は *dbremote.exe* です。UNIX では、この名前は dbremote です。Mac OS X では、**SyncConsole** を使用して Message Agent (dbremote) を起動

することもできます。「Mac OS X での Message Agent (dbremote) の実行」 102 ページと「UNIX での Message Agent (dbremote) の実行」 103 ページを参照してください。

たとえば、次の文ではデータベース・ファイル *c:¥mydata.db* で dbremote をバッチ・モードで実行します。接続にはユーザ名 ManagerSteve とパスワード sql を使用しています。

```
dbremote -c "UID=ManagerSteve;PWD=sql;DBF=c:¥mydata.db"
```

Message Agent (dbremote) は受信メッセージを受信して処理し、トランザクション・ログを 1 回スキャンし、出力メッセージを作成して送信した後、停止します。

ユーザ名 ManagerSteve には、REMOTE DBA 権限または DBA 権限のいずれかが必要です。-l オプションで定義されているメッセージの最大長は、システム内のすべてのデータベースで同じである必要があります。「Message Agent (dbremote) の稼働条件」 99 ページを参照してください。

参照

- 「Message Agent (dbremote)」 160 ページ

Mac OS X での Message Agent (dbremote) の実行

SQL Anywhere には、Mac OS X で Message Agent (dbremote) 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/SQLAnywhere11* 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・オプションのウィンドウが表示されます。
4. Message Agent (dbremote) の接続情報を指定します。

   たとえば、次の接続パラメータでは、SQL Anywhere サンプル・データベースの ODBC データ・ソースが使用されます。

   ```
   DSN="SQL Anywhere 11 Demo"
   ```

   ユーザ名には、REMOTE DBA 権限または DBA 権限のいずれかが必要です。-l オプションで定義されているメッセージ長は、システム内のすべてのデータベースで同じである必要があります。「Message Agent (dbremote) の稼働条件」 99 ページを参照してください。

   [オプション] フィールドに指定できる dbremote オプションの完全なリストについては、「Message Agent (dbremote)」 160 ページを参照してください。

UNIX での Message Agent (dbremote) の実行

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 (dbremote) をデーモンとして実行します。「Message Agent (dbremote)」 160 ページ の -ud オプションを参照してください。

ユーザ名には、REMOTE DBA 権限または DBA 権限のいずれかが必要です。-l オプションで定義されているメッセージの最大長は、システム内のすべてのデータベースで同じである必要があります。「Message Agent (dbremote) の稼働条件」 99 ページを参照してください。

指定できる dbremote オプションの完全なリストについては、「Message Agent (dbremote)」 160 ページを参照してください。

SQL Remote のパフォーマンス向上

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、ローに対してサブスクライブされたユーザにメッセージが作成されます。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。

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 (dbremote) が分担します。

データベース・サーバ

データベース・サーバは、パブリケーションを処理します。「データベース・サーバによるパブリケーションの処理」 40 ページを参照してください。

Message Agent (dbremote)

「Message Agent (dbremote)」 は、サブスクリプションを処理します。

Message Agent (dbremote) 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 (dbremote) 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データベース・サーバのパフォーマンスは低下しませんが、Message Agent (dbremote) 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。

メッセージ受信のタスク

Message Agent (dbremote) は、メッセージの受信時に次のタスクを実行します。

- **受信メッセージのポーリング**　データベースに着信した新しいメッセージをチェックするために、Message Agent (dbremote) によって新しいメッセージがポーリングされます。「新しいメッセージを確認するポーリング間隔の調整」 105 ページを参照してください。
- **メッセージの読み込み**　メッセージが着信すると、Message Agent (dbremote) によって読み込まれ、適用可能になるまでキャッシュ・メモリ内に格納されます。「メッセージのキャッシュによるスループットの調整」 106 ページを参照してください。

  欠落しているメッセージがあり、Message Agent (dbremote) が継続モードで実行されている場合、Message Agent (dbremote) は、後続のポーリングでメッセージの着信を待機します。Message Agent (dbremote) が待機するポーリング回数は、その「待機時間」と呼ばれ、-rp オプションで指定されます。

  - Message Agent (dbremote) の待機時間が切れる前に、欠落していたメッセージが着信した場合、このメッセージは正しい順序でキャッシュに追加されます。

○ 欠落しているメッセージが着信しないまま、Message Agent (dbremote) の待機時間が切れた場合、Message Agent (dbremote) は、パブリッシャ・データベースからのメッセージの再送要求を送信します。

キャッシュ・メモリ使用量を超えるまで、メッセージは継続して読み込まれ、キャッシュに追加されます。-m オプションで指定したメモリ使用量を超過すると、メッセージは削除されます。

「メッセージの再送要求の調整」 107 ページを参照してください。

- **メッセージの適用** Message Agent (dbremote) は、サブスクライバ・データベースに正しい順序でメッセージを適用します。「ワーカ・スレッド数の調整」 109 ページを参照してください。
- **サブスクライバ・データベースへのメッセージ適用の確認メッセージの待機** メッセージが受信されてサブスクライブされ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 (dbremote) は確認メッセージを受信すると、システム・テーブル内で確認メッセージを追跡します。「保証されたメッセージ配信システムの概要」 114 ページを参照してください。

メッセージ受信時のパフォーマンス向上

SQL Remote システム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・データベースからメッセージを受信して、それをデータベースに適用することです。このラグ・タイムを短縮するには、Message Agent (dbremote) を継続モードで実行している場合に、次の変数を調整します。

- Message Agent (dbremote) が受信メッセージを確認する頻度。「新しいメッセージを確認するポーリング間隔の調整」 105 ページを参照してください。
- Message Agent (dbremote) が送信するメッセージの格納に使用するメモリ量。「メッセージのキャッシュによるスループットの調整」 106 ページを参照してください。
- Message Agent (dbremote) が順序不整合のメッセージの再送を要求するまでメッセージの着信を待機する時間。「メッセージの再送要求の調整」 107 ページを参照してください。
- 受信したメッセージの処理に使用されるワーカ・スレッドの数。「ワーカ・スレッド数の調整」 109 ページを参照してください。

新しいメッセージを確認するポーリング間隔の調整

データベースに着信した新しいメッセージをチェックするために、Message Agent (dbremote) によって新しいメッセージがポーリングされます。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。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が、通常はデフォルトで十分です。

ポーリング間隔の短縮

秒単位の値を使用することによって、ポーリング頻度を上げることができます。たとえば、次のコマンドでは、30 秒ごとにポーリングが行われます。

```
dbremote -c "DSN=SQL Anywhere 11 Demo" -rd 30s
```

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、ポーリング間隔は短くしないでください。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。

- 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ることがあります。たとえば、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。
- 再送要求によってシステムに過負荷がかかる場合があります。ポーリング間隔を調整する場合は、Message Agent (dbremote) の待機時間も調整してください。待機時間とは、Message Agent (dbremote) が順序不整合のメッセージの再送を要求するまでメッセージの着信を待機するポーリング回数のことです。「メッセージの再送要求の調整」 107 ページを参照してください。

ポーリング間隔の延長

ポーリングの頻度を下げることができます。次のコマンドでは、ポーリング間隔を 5 分に設定します。

```
dbremote -c "DSN=SQL Anywhere 11 Demo" -rd 5
```

長めのポーリング間隔を設定すると、システムのメッセージ・スループット全体が向上しますが、個別のメッセージの適用にかかる時間が長くなることがあります。たとえば、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。

参照

- 「メッセージ受信時のパフォーマンス向上」 105 ページ
- 「メッセージのキャッシュによるスループットの調整」 106 ページ
- 「メッセージの再送要求の調整」 107 ページ
- 「ワーカ・スレッド数の調整」 109 ページ
- 「メッセージ送信時のパフォーマンス向上」 111 ページ

メッセージのキャッシュによるスループットの調整

メッセージが着信すると、Message Agent (dbremote) はメッセージを読み込み、メッセージが適用されるまでキャッシュ・メモリ内に格納します。このようなメッセージのキャッシュによって、次の状態を回避できます。

- 規模の大きいシステム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・システムからの順序不整合のメッセージの再読み込み。メッセージのキャッシュは、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) 上で読み込む場合に役立ちます。
- メッセージを読み込む (単一スレッド・タスク) データベース・ワーカ・スレッド間の競合。メッセージの内容がキャッシュされるためです。

メッセージをキャッシュする方法

次のいずれかの状況が生じると、Message Agent (dbremote) によって適用されるまで、メッセージはメモリ内に格納されます。

- トランザクションが非常に大きく、マルチパートのメッセージが必要とされる。
- メッセージが順序不整合の状態で着信する。

メッセージ・キャッシュ・サイズの指定

Message Agent (dbremote) の -m オプションを使用して、メッセージ・キャッシュのサイズを指定します。-m オプションは、Message Agent (dbremote) がメッセージの格納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。指定したキャッシュ・メモリ使用量を超過すると、メッセージは削除されます。

-m オプションは、単一の統合データベースと多数のリモート・データベースを使用する場合に役立ちます。「Message Agent (dbremote)」 160 ページの -m オプションを参照してください。

例

次のコマンドは、12 MB のメモリをメッセージ・キャッシュとして使用して Message Agent (dbremote) を起動します。

```
dbremote -c "DSN=SQL Anywhere 11 Demo" -m 12M
```

参照

- 「メッセージ受信時のパフォーマンス向上」 105 ページ
- 「新しいメッセージを確認するポーリング間隔の調整」 105 ページ
- 「メッセージの再送要求の調整」 107 ページ
- 「ワーカ・スレッド数の調整」 109 ページ
- 「メッセージ送信時のパフォーマンス向上」 111 ページ

メッセージの再送要求の調整

メッセージがシーケンスから欠落している場合、Message Agent (dbremote) は、指定されたポーリング回数を待機してから、欠落しているメッセージの再送を要求します。Message Agent (dbremote) が待機するポーリング回数は、その待機時間と呼ばれます。デフォルトでは、Message Agent (dbremote) の待機時間は 1 です。

Message Agent (dbremote) の待機時間が 1 であり、メッセージ 6 を受信するはずが、メッセージ 7 を受信した場合、Message Agent (dbremote) は何もしません。代わりに、Message Agent (dbremote) は、次のポーリングの結果を待機します。次のポーリングの後、メッセージ 6 が欠落したままである場合、Message Agent (dbremote) はメッセージ 6 の再送要求を発行します。

再送待機時間の延長

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とします。一般的には、順序不整合のメッセージが到着するのは 2 ～ 3 回のポーリングが完了してからになります。この例では、-rp オプションを使用して Message Agent (dbremote) の待

機時間を長くし、不要な再送要求が大量に送信されないようにすることをおすすめします。-rp 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。「新しいメッセージを確認するポーリング間隔の調整」 105 ページを参照してください。

例

user1 と user2 という 2 人のリモート・ユーザがいて、両ユーザともポーリング間隔を 30 秒、待機時間を 3 回のポーリングに設定して Message Agent (dbremote) を実行します。たとえば、これらのユーザは、次のコマンドを使用してそれぞれの Message Agent (dbremote) を実行します。

```
dbremote -c "DSN=SQL Anywhere 11 Demo"  -rd 30s -rp 3
```

次の一連の操作によってメッセージは *userX.n* とマーク付けされます。X はユーザ名であり、*n* はメッセージ番号です。たとえば、user1.5 は、user1 からの 5 番目のメッセージになります。Message Agent (dbremote) 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 (dbremote) が user1.1 と user2.4 を読み込みます。
2. Message Agent (dbremote) 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 (dbremote) の待機時間は user1 が N/A、user2 が 3 です。

30 秒後

1. Message Agent (dbremote) でのメッセージの読み込み：新着メッセージなし
2. Message Agent (dbremote) での適用：なし
3. 現時点の Message Agent (dbremote) の待機時間：user1：N/A、user2：2

60 秒後

1. Message Agent (dbremote) でのメッセージの読み込み：user1.3
2. Message Agent (dbremote) での適用：新着メッセージなし
3. Message Agent (dbremote) の待機時間：user1：3、user2：1

90 秒後

1. Message Agent (dbremote) でのメッセージの読み込み：user1.4
2. Message Agent (dbremote) での適用：なし
3. Message Agent (dbremote) の待機時間：user1：3、user2：0
4. Message Agent (dbremote) 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 (dbremote) の待機時間はリセットされます。

120 秒後

1. Message Agent (dbremote) が user1.2 と user2.2 を読み込みます。
2. Message Agent (dbremote) が user1.2、user1.3、user1.4、user2.2 を適用します。
3. Message Agent (dbremote) の待機時間：user1：N/A、user2：N/A

参照

- 「メッセージ受信時のパフォーマンス向上」 105 ページ
- 「新しいメッセージを確認するポーリング間隔の調整」 105 ページ
- 「メッセージのキャッシュによるスループットの調整」 106 ページ
- 「ワーカ・スレッド数の調整」 109 ページ
- 「メッセージ送信時のパフォーマンス向上」 111 ページ

ワーカ・スレッド数の調整

次の手順では、Message Agent (dbremote) がどのように受信メッセージを適用するかを示します。

1. メッセージを読み込みます。メッセージが読み込まれ、ヘッダ情報が検索されます (正しい適用順を決定するため)。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。
2. メッセージを適用します。読み込まれたメッセージが、適用されるデータベース・ワーカ・スレッドに渡されます。

   通常、リモート・データベースでは、メッセージは直列で適用されます。多層システムでは、リモート・データベースが他のリモートの統合データベースになることもあります。このタイプのリモート・データベースでは、統合データベースと同様にメッセージが適用されます。

   統合データベースでは、デフォルトでメッセージが直列に適用されます。追加のデータベース・ワーカ・スレッドを使用すると、リモート・ユーザからの受信メッセージを並列で適用できます。「Message Agent (dbremote)」 160 ページの -w オプションを参照してください。

   統合データベースでデータベース・ワーカ・スレッドを使用すると、次のようになります。

   - 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る。
   - 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る。

     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。

データベース・ワーカ・スレッドを使用する場合の利点

統合データベースでデータベース・ワーカ・スレッドを使用する場合、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。分散されたドライブ・アレイがあるシステム上にデータベース・サーバがあるときに、パフォーマンス上の利点が顕著に表れます。

データベース・ワーカ・スレッドを使用する場合の欠点

統合データベースでデータベース・ワーカ・スレッドを使用する場合、ワーカ・スレッドによってユーザ間に多数のロックが生じると、スループットが低下することがあります。

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。

♦ **データベース・ワーカ・スレッド数を設定するには、次の手順に従います。**

- 統合データベースで、-w オプションを使用してデータベース・ワーカ・スレッド数を設定します。

  たとえば、次のコマンドでは、ワーカ・スレッド数を 5 に設定します。

  ```
  dbremote -c "DSN=SQL Anywhere 11 Demo"  -w 5
  ```

参照

- 「継続モードでの Message Agent (dbremote) の実行」 99 ページ
- 「メッセージ受信時のパフォーマンス向上」 105 ページ
- 「新しいメッセージを確認するポーリング間隔の調整」 105 ページ
- 「メッセージのキャッシュによるスループットの調整」 106 ページ
- 「メッセージの再送要求の調整」 107 ページ
- 「メッセージ送信時のパフォーマンス向上」 111 ページ

メッセージ送信のタスク

Message Agent (dbremote) は、次のタスクを実行してメッセージを送信します。

- **パブリッシャのトランザクション・ログのスキャン**　Message Agent (dbremote) は、パブリッシャ・データベースのトランザクション・ログをスキャンして、トランザクション・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します。-l オプションで定義されているメッセージの最大長は、システム内のすべてのデータベースで同じである必要があります。

  大きなトランザクションの場合、Message Agent (dbremote) はマルチパートのメッセージを作成します。これらのメッセージには、トランザクション内でのそれぞれの位置を追跡するシーケンス番号が個別に付けられています。サブスクライバ・データベースの Message Agent (dbremote) は、シーケンス番号を使用して、メッセージが正しい順序で適用され、失われることがないようにします。

- **リモート・データベースへのメッセージの送信**　Message Agent (dbremote) は、各リモート・ユーザの送信頻度プロパティで指定された時刻にメッセージを送信します。「送信遅延の調整」 111 ページを参照してください。

  Message Agent (dbremote) は、キャッシュ・メモリが設定値を超えた場合、指定時刻より前にメッセージを送信します。Message Agent (dbremote) は、メッセージをキャッシュ・メモリに格納します。使用中のキャッシュ・メモリが指定された値を超えると、メッセージが送信されます。「メッセージのキャッシュによるスループットの調整」 112 ページを参照してください。

- **リモート・データベースからの再送要求の処理**　ユーザがメッセージの再送要求を発行すると、パブリッシャ・データベースの Message Agent (dbremote) は通常のメッセージ送信処理を中断し、再送要求を処理します。

  このような再送要求の緊急度は、-ru オプションを使用して制御します。「再送要求処理の緊急度の調整」 112 ページを参照してください。

- **パブリッシャ・データベースへの確認メッセージの送信**　メッセージが受信されてサブスクライブされ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。「保証されたメッセージ配信システムの概要」 114 ページを参照してください。

メッセージ送信時のパフォーマンス向上

メッセージの送信におけるパフォーマンス上の主な問題は、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ーンアラウンド・タイムです。このラグ・タイムを短縮するには、Message Agent (dbremote) でのメッセージの送信時に、次の変数を調整できます。

- リモート・データベースにメッセージを送信する頻度。「送信遅延の調整」 111 ページを参照してください。
- メッセージのサイズ。「メッセージ受信時のパフォーマンス向上」 105 ページを参照してください。
- 再送要求処理の緊急度。「再送要求処理の緊急度の調整」 112 ページを参照してください。

送信遅延の調整

送信するメッセージを作成するため、Message Agent (dbremote) はトランザクション・ログから新しいデータをポーリングします。送信遅延は、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。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。送信遅延は、-sd オプションを使用して設定できますが、通常はデフォルトで十分です。送信遅延は、リモート・ユーザの送信頻度より短いかまたは同じ時間にしてください。

送信遅延の短縮

秒単位の値を使用することによって、ポーリング頻度を上げることができます。たとえば、次のコマンドでは、30 秒ごとにポーリングが行われます。

```
dbremote -c "DSN=SQL Anywhere 11 Demo"  -sd 30s ...
```

送信遅延の延長

ポーリングの頻度を下げることができます。次のコマンドでは、ポーリング間隔を 60 分に設定します。

```
dbremote -c "DSN=SQL Anywhere 11 Demo"  -sd 60
```

長めの間隔を設定すると、メッセージを送信する前に、Message Agent (dbremote) がメッセージ作成処理の大部分を実行する必要があります。メッセージ作成処理を分散するために、通常は短めの間隔をおすすめします。

「Message Agent (dbremote)」 160 ページを参照してください。

参照

- 「メッセージのキャッシュによるスループットの調整」 112 ページ
- 「再送要求処理の緊急度の調整」 112 ページ

メッセージのキャッシュによるスループットの調整

Message Agent (dbremote) は、送信するメッセージをメモリの設定可能エリアにキャッシュします。

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。メッセージは次の場合に送信されます。

- 送信頻度に達したとき
- 使用中のキャッシュ・メモリが -m の値を超えたとき
- メッセージのサイズがその最大サイズ (-l オプションで指定される) に達したとき

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 (dbremote) のコマンドで指定します。

-m オプションは、Message Agent (dbremote) 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。

-m オプションは、単一の統合データベースと多数のリモート・データベースを使用する場合に役立ちます。「Message Agent (dbremote)」 160 ページの -m オプションを参照してください。

例

次のコマンドは、12 MB のメモリをメッセージ・キャッシュとして使用して Message Agent (dbremote) を起動します。

```
dbremote -c "DSN=SQL Anywhere 11 Demo" -m 12M
```

参照

- 「送信遅延の調整」 111 ページ
- 「再送要求処理の緊急度の調整」 112 ページ

再送要求処理の緊急度の調整

メッセージを再送すると、通常のメッセージ送信処理が中断されるため、Message Agent (dbremote) は再送要求の処理を遅らせます。デフォルトでは、Message Agent (dbremote) は、再送を要求したリモート・ユーザの送信頻度の半分の時間を待機します。

メッセージを再送する場合、Message Agent (dbremote) は次のタスクを実行します。

- トランザクション・ログのスキャンを停止し、新しいメッセージの作成を停止する。
- キャッシュに格納された送信待機中の現在のメッセージを削除する。トランザクション・ログの読み込み時とそれらのメッセージの作成時に Message Agent (dbremote) が実行したすべての作業が失われます。

- 再送要求で要求されたオフセットからトランザクション・ログを再読み込みする。Message Agent (dbremote) はメッセージを作成し、そのキャッシュに格納します。
- 次の送信頻度の時刻まで待機した後、メッセージを送信する。

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。他のメッセージが到着するまで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。たとえば、次のコマンドでは、再送要求を処理する前に 1 時間待機します。

```
dbremote -c "DSN=SQL Anywhere 11 Demo" -ru 1h
```

「Message Agent (dbremote)」 160 ページを参照してください。

参照

- 「送信遅延の調整」 111 ページ
- 「メッセージのキャッシュによるスループットの調整」 112 ページ
- 「メッセージの再送要求の調整」 107 ページ

保証されたメッセージ配信システムの概要

保証されたメッセージ配信システムでは、次のことが保証されます。

- 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る。「正しい順序でのオペレーション適用の確保」 115 ページを参照してください。
- レプリケートされたオペレーションが欠落しない。「消失または壊れたメッセージの再送」 117 ページを参照してください。
- レプリケートされたオペレーションが二重に適用されない。「一度かぎりのメッセージ適用の確保」 118 ページを参照してください。

保証されたメッセージ配信システムでは、次の情報が使用されます。

- **SYSREMOTEUSER システム・テーブルで管理されているステータス情報**　この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。次に例を示します。
  - ○ 統合データベースでは、SYSREMOTEUSER システム・テーブルに各リモート・ユーザに対応したローがある。
  - ○ 各リモート・データベースでは、SYSREMOTEUSER システム・テーブルに統合データベースの情報を含むローが 1 つある。

  SYSREMOTEUSER システム・テーブルは、Message Agent (dbremote) が管理しています。

  サブスクライバ・データベースでは、Message Agent (dbremote) 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 SYSREMOTEUSER システム・テーブルが正しく管理されていることを確認します。

  「ISYSREMOTEUSER システム・テーブル」『SQL Anywhere サーバ - SQL リファレンス』を参照してください。
- **メッセージのヘッダ内の情報**　Message Agent (dbremote) は、メッセージ内のヘッダ情報を読み込み、この情報を使用して SYSREMOTEUSER システム・テーブルを更新します。各メッセージのヘッダには、次の情報が含まれています。
  - ○ **メッセージの resend_count**　データベースがメッセージを失った受信の回数を追跡するカウンタ。

    次の例では、resend_count は 1 です。

    ```
    Current message's header: (「1」-0000942712-0001119170-0)
    ```
  - ○ **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**　次の例では、前のメッセージの最後のコミットのトランザクション・ログ・オフセットは、0000942712 です。

    ```
    Previous message's header:(0-0000923357-「0000942712」-0)
    Current message's header: (0-「0000942712」-0001119170-0)
    ```
  - ○ **現在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**　次の例では、現在のメッセージの最後のコミットは、0001119170 です。


```
Current message's header: (0-0000942712-「0001119170」-0)
```

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、両方のトランザクション・ログ・オフセットは、最後のメッセージに COMMIT が含まれるまで同じになることがあります。

次の例では、4 番目のメッセージまで COMMIT は発生していません。

```
(0-「0000942712」-「0000942712」-0)
(0-「0000942712」-「0000942712」-1)
(0-「0000942712」-「0000942712」-2)
(0-0000942712-0001119170-3)
```

○ **シーケンス番号**　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に、このシーケンス番号が使用されます。

シーケンス番号 0 は、次のことを示す場合があります。

- トランザクション・ログ・オフセットが異なる場合、メッセージはマルチパートのメッセージの一部ではない。

  次の例では、メッセージはマルチパートのメッセージの一部ではありません。

  ```
  「
  (0-0000923200-0000923357-「0」)
  (0-0000923357-0000942712-「0」)」
  ```

- トランザクション・ログ・オフセットが同一の場合、メッセージはマルチパートのメッセージの最初のメッセージである。

  次の例では、最初のメッセージはマルチパートのメッセージの一部です。

  ```
  「(0-0000942712-0000942712-「0」)」
  (0-0000942712-「0000942712」-「1」)
  (0-0000942712-0000942712-「2」)
  (0-0000942712-0001119170-「3」)
  ```

正しい順序でのオペレーション適用の確保

レプリケートされた文が正しい順序で適用されるようにするため、保証されたメッセージ配信システムは、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・オフセットを使用します。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、トランザクション・ログ・オフセット値を比較して決定されます。

各メッセージには、次のトランザクション・ログ・オフセットが含まれています。

- 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シーケンス番号がトランザクションにあります。「正しい順序でのオペレーション適用の確保」 115 ページを参照してください。
- 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。

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に、トランザクション内のシーケンス番号が使用されます。

メッセージの送信

SYSREMOTEUSER システム・テーブルの 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。

次の手順では、メッセージが送信されたときに SYSREMOTEUSER システム・テーブルがどのように更新されるかを示します。

1. パブリッシャの Message Agent (dbremote) はサブスクライバにメッセージを送信すると、送信したメッセージの最後の COMMIT のトランザクション・ログ・オフセット値を log_sent 値に設定します。

   たとえば、パブリッシャが次のメッセージを user1 に送信します。

   ```
   「
   (0-0000923200-0000923357-「0」)」
   ```

   パブリッシャの SYSREMOTEUSER システム・テーブル内で、パブリッシャは log_sent 値に user1 の 0000923357 を設定します。

2. メッセージが受信されてサブスクライバ・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送信されます。確認メッセージには、サブスクライバ・データベースによって適用された最新のトランザクション・ログ・オフセットが含まれています。

   たとえば、確認メッセージでは、user1 がトランザクション・ログ・オフセット 0000923357 以前のすべてのトランザクションを適用したことが確認されます。

3. パブリッシャの Message Agent (dbremote) は確認メッセージを受信すると、SYSREMOTEUSER システム・テーブルの confirm_sent カラムにユーザの確認メッセージのオフセット値を設定します。

   たとえば、パブリッシャは、パブリッシャの SYSREMOTEUSER システム・テーブルの confirm_sent カラムに user1 の 0000923357 を設定します。

log_sent と confirm_sent の両方の値には、パブリッシャのトランザクション・ログのトランザクション・ログ・オフセットが含まれています。confirm_sent 値は、log_sent 値より後のオフセットにはなりません。

メッセージの受信

次の手順では、メッセージが受信されたときに SYSREMOTEUSER システム・テーブルがどのように更新されるかを示します。

1. サブスクライバ・データベースの Message Agent (dbremote) 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 SYSREMOTEUSER システム・テーブルの log_received カラムが更新されます。

たとえば、サブスクライバが次のメッセージを受信して適用すると、SYSREMOTEUSER システム・テーブルの log_received 値に 0000923357 が設定されます。

```
「
(0-0000923200-0000923357-「0」)」
```

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・ログでの、トランザクション・ログ・オフセットが入ります。

2. オペレーションが受信され適用されると、サブスクライバの Message Agent (dbremote) は、その SYSREMOTEUSER システム・テーブルの confirm_received 値を設定し、パブリッシャ・データベースに確認メッセージを送信します。

参照

- 「消失または壊れたメッセージの再送」 117 ページ
- 「一度かぎりのメッセージ適用の確保」 118 ページ

消失または壊れたメッセージの再送

SYSREMOTEUSER システム・テーブルには、メッセージの再送を管理する 2 つのカラムが含まれています。

- **resend_count カラム**　サブスクライバ・データベースがメッセージを失った回数を追跡するカウンタ。
- **rereceive_count カラム**　Message Agent (dbremote) がパブリッシャ・ユーザからのメッセージが失われたと判断した回数を追跡するカウンタ。

サブスクライバ・データベースでメッセージが正しい順序で受信されると、次の処理が実行されます。

1. サブスクライバの Message Agent (dbremote) は、メッセージを正しい順序で適用し、その SYSREMOTEUSER システム・テーブルを更新します。
2. サブスクライバの Message Agent (dbremote) は、パブリッシャに確認メッセージを送信します。
3. パブリッシャが確認メッセージを受信すると、パブリッシャの Message Agent (dbremote) はその SYSREMOTEUSER システム・テーブルを更新します。

メッセージが正しい順序で受信されなかった場合は、次の処理が実行されます。

1. サブスクライバの Message Agent (dbremote) は、再送要求を送信し、その SYSREMOTEUSER システム・テーブルの rereceive_count 値を増分します。
2. パブリッシャは再送要求を受信すると、その SYSREMOTEUSER システム・テーブルでサブスクライバの resend_count 値を増分します。
3. パブリッシャの SYSREMOTEUSER システム・テーブル内で、log_sent 値に confirm_sent カラムの値が設定されます。log_sent 値を設定し直すと、オペレーションが再送されます。

参照

- 「正しい順序でのオペレーション適用の確保」 115 ページ
- 「一度かぎりのメッセージ適用の確保」 118 ページ

一度かぎりのメッセージ適用の確保

サブスクライバの Message Agent (dbremote) は、メッセージ・ヘッダ内の resend_count 値と、その SYSREMOTEUSER システム・テーブル内の rereceive_count を比較します。resend_count 値が rereceive_count よりも小さい場合、メッセージは適用されることなく削除されます。この動作により、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。

参照

- 「正しい順序でのオペレーション適用の確保」 115 ページ
- 「消失または壊れたメッセージの再送」 117 ページ

メッセージ・サイズの制御

この項では、Message Agent (dbremote) のメッセージのエンコードと圧縮スキームについて説明します。

Message Agent には、次のエンコードと圧縮の機能があります。

- **互換性**　システムは、SQL Anywhere 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。「SQL Remote のアップグレード」『SQL Anywhere 11 - 変更点とアップグレード』を参照してください。
- **圧縮**　メッセージの圧縮レベルを選択できます。

  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。ただし、圧縮の実行にかなりの時間がかかります。「compression オプション [SQL Remote]」『SQL Anywhere サーバ - データベース管理』を参照してください。
- **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。「エンコード：メッセージ破壊の回避」 119 ページを参照してください。

メッセージの最大長に関する SQL Remote の稼働条件については、「Message Agent (dbremote) の稼働条件」 99 ページを参照してください。

エンコード：メッセージ破壊の回避

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。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。

- メッセージ・システムでバイナリ形式のメッセージが使用できる場合、メッセージはエンコードされません。
- SMTP のように、メッセージ・システムでテキストベースのメッセージ形式が必要とされる場合は、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。

  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。「カスタム・エンコード・スキームの作成」 120 ページを参照してください。
- 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使用されます。「SQL Remote のアップグレード」『SQL Anywhere 11 - 変更点とアップグレード』を参照してください。

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・スキームを実装するには、カスタム・エンコード DLL を構築します。このカスタム 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などの統計を取ったりすることができます。

ヘッダ・ファイル *dbrmt.h* は、*install-dir* ディレクトリにインストールされています。このファイルには、カスタム・エンコード・スキームの構築に使用できるアプリケーション・プログラミング・インタフェースが含まれています。

自分のカスタム DLL を使用するには、そのカスタム 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:¥¥sany11¥¥Bin32¥¥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。

参照

- 「SET REMOTE OPTION 文 [SQL Remote]」 『SQL Anywhere サーバ - SQL リファレンス』

SQL Remote メッセージ・システム

SQL Remote は、基本となるメッセージ・システムを 1 つ以上使用して、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。「FILE メッセージ・システム」 125 ページを参照してください。
- **FTP**　インターネット・ファイル転送プロトコル。「FTP メッセージ・システム」 127 ページを参照してください。
- **SMTP/POP**　インターネット電子メール転送プロトコル。「SMTP メッセージ・システム」 129 ページを参照してください。

REMOTE または CONSOLIDATE パーミッションをユーザに割り当てる場合は、メッセージ・システムを選択します。「REMOTE パーミッションの付与」 31 ページと「CONSOLIDATE パーミッションの付与」 33 ページを参照してください。

SQL Remote システム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。

すべてのメッセージ・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オペレーティング・システムの完全なリストについては、「サポートされるプラットフォーム」 『SQL Anywhere 11 - 紹介』を参照してください。

メッセージ・システムの設定

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。

各メッセージ・タイプの定義には、メッセージ・システムのタイプ名 (**FILE**、**FTP**、または **SMTP**) 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が含まれます。

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。

抽出ユーティリティ (dbxtract)

リモート・データベースの作成時に抽出ユーティリティ (dbxtract) と**データベース抽出ウィザード**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 (dbremote) でも、FILE システムの受信メッセージの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。

参照

- 「FILE メッセージ・システム」 125 ページ
- 「FTP メッセージ・システム」 127 ページ
- 「SMTP メッセージ・システム」 129 ページ

メッセージ・タイプの作成

♦ **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. 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. 左ウィンドウ枠で、**[SQL Remote ユーザ]** ディレクトリを展開します。
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]** - **[新規]** - **[メッセージ・タイプ]** を選択します。
5. **[新しいメッセージ・タイプの名前を指定してください。]** フィールドに、メッセージ・タイプの名前を入力します。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・アドレスを指定してください。]** フィールドにパブリッシャ・アドレスを入力します。**[完了]** をクリックします。

♦ **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作成されていることを確認します。
2. CREATE REMOTE MESSAGE TYPE 文を実行します。

```
CREATE REMOTE MESSAGE TYPE message-type ADDRESS publisher-address
```

次に例を示します。

```
CREATE REMOTE MESSAGE TYPE FILE
ADDRESS 'company';
```

Windows Mobile でのリモート・メッセージ・タイプの作成

Windows Mobile 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 同期用に設定できます。このオプションにより、FILE メッセージ・リンクのメッセージのディレクトリが、ActiveSync ディレクトリとして設定されます。Windows Mobile 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] ディレクトリにあるファイルと、Windows Mobile の [ActiveSync] ディレクトリにあるファイルの同期をとります。

♦ **SQL Remote の ActiveSync 同期を設定するには、次の手順に従います。**

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. **[ツール]** - **[SQL Anywhere 11]** - **[Windows Mobile メッセージ・タイプの編集]** を選択します。

参照

- 「CREATE REMOTE MESSAGE TYPE 文 [SQL Remote]」 『SQL Anywhere サーバ - SQL リファレンス』
- 「メッセージ・タイプの変更」 123 ページ
- 「メッセージ・タイプの削除」 123 ページ

メッセージ・タイプの変更

パブリッシャのアドレスを変更するには、そのメッセージ・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。

♦ **リモート・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. 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. 左ウィンドウ枠で、データベースの **[SQL Remote ユーザ]** ディレクトリを展開します。
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. 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、**[プロパティ]** を選択します。
5. メッセージ・タイプのプロパティを更新し、**[OK]** をクリックします。

♦ **リモート・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. 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. 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。
3. ALTER REMOTE MESSAGE TYPE 文を実行します。

参照

- 「ALTER REMOTE MESSAGE TYPE 文 [SQL Remote]」 『SQL Anywhere サーバ - SQL リファレンス』
- 「メッセージ・タイプの作成」 122 ページ
- 「メッセージ・タイプの削除」 123 ページ

メッセージ・タイプの削除

メッセージ・タイプを削除すると、パブリッシャ・アドレスが定義から削除されます。

♦ **メッセージ・タイプを削除するには、次の手順に従います (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. 左ウィンドウ枠で、データベースの **[SQL Remote ユーザ]** ディレクトリを展開します。
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. 右ウィンドウ枠で、削除するメッセージ・タイプを右クリックし、**[削除]** を選択します。
5. **[はい]** をクリックします。

♦ **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)。**

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 REMOTE MESSAGE TYPE 文を実行します。

参照

- 「DROP REMOTE MESSAGE TYPE 文 [SQL Remote]」 『SQL Anywhere サーバ - SQL リファレンス』
- 「メッセージ・タイプの作成」 122 ページ
- 「メッセージ・タイプの変更」 123 ページ

リモート・メッセージ・タイプ制御パラメータの設定

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。次の手順を使用して、制御パラメータを設定します。

♦ **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 (SQL の場合)。**

- SET REMOTE OPTION 文を実行します。

  たとえば、次の文は、ユーザ myuser 用の FTP リンクに対し、FTP ホストを *ftp.mycompany.com* に設定します。

  ```
  SET REMOTE FTP OPTION myuser.host = 'ftp.mycompany.com';
  ```

  「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。

♦ **メッセージ・リンク・パラメータを表示するには、次の手順に従います (SQL の場合)。**

- SYSREMOTEOPTION システム・ビューを問い合わせます。

  次に例を示します。

  ```
  SELECT * from SYSREMOTEOPTION;
  ```

  「SYSREMOTEOPTION システム・ビュー」 『SQL Anywhere サーバ - SQL リファレンス』を参照してください。

ディスクに格納されるメッセージ・リンク・パラメータ

SQL Remote 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。メッセージ・リンク・パラメータの外部保存はおすすめしません。

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 キーの下のキーに格納されます。

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

SQLREMOTE 環境変数には、FILE メッセージ・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格納されます。

Message Agent (dbremote) 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 Message Agent (dbremote) 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。

参照

- 「SET REMOTE OPTION 文 [SQL Remote]」 『SQL Anywhere サーバ - SQL リファレンス』

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、電子メール・システムまたは FTP 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 FILE 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。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。

Windows サービスとして起動中の場合は、Message Agent (dbremote) 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必要です。正しいパーミッションが割り当てられていない場合、Message Agent はネットワーク・ドライブにアクセスできません。

アドレスのルート・ディレクトリ

一般的に、FILE メッセージ・システムのアドレスは、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、統合コンピュータまたはリモート・コンピュータのディレクトリにメッセージを入れることもできます。簡易ファイル転送メカニズムを使用してファイルを交換し、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**　メッセージが格納されるディレクトリです。この設定は、SQLREMOTE 環境変数の代替となります。
- **Debug**　このパラメータの設定は、YES か NO です。デフォルトは NO です。設定が YES の場合、FILE リンクによるすべての FILE 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・スキームを使用している場合は、作成したカスタム・エンコード DLL のフル・パスをこのパラメータに設定する必要があります。「メッセージ・サイズの制御」 119 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 Message Agent (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**　ファイル削除に失敗した後、次にファイルの削除を試みるまでの秒数です。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 秒間一時停止するように設定されています。

Windows Mobile と ActiveSync

FILE リンクの場合、Message Agent (dbremote) は *C:¥My Documents¥Synchronized Files* を調べます。統合データベース・コンピュータ上では、FILE リンク用の SQLREMOTE 環境変数または directory メッセージ・リンク・パラメータを次のディレクトリに設定してください。ここで、*userid* と *Windows-mobile-device-name* に適切な値を設定します。

```
%SystemRoot%¥Profiles¥userid¥Personal¥Windows-mobile-device-name¥Synchronized Files
```

ActiveSync はこのシステムを使用して、統合データベース・コンピュータと Windows Mobile デバイス間でメッセージ・ファイルを自動的に同期します。

FILE の同期がアクティブ化されていることを確認するには、**[モバイル・デバイス]** - **[ツール]** - **[ActiveSync]** オプションをチェックします。

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」 125 ページを参照してください。

参照

- 「FTP メッセージ・システム」 127 ページ

FTP メッセージ・システム

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・システムのリストについては、「サポートされるプラットフォーム」『SQL Anywhere 11 - 紹介』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。

- **host** FTP サーバを実行しているコンピュータのホスト名。このパラメータには、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** FTP の接続に使用される IP ポート番号。このパラーメータは通常は不要です。
- **debug** このパラメータは、YES または NO に設定されます。デフォルトは 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** このパラメータは、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。このパラメータを設定すると、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。このパラメータを設定すると、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** このパラメータは、TRUE または FALSE に設定されます。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]** ウィンドウは表示されません。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。
- **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・パスをこのパラメータに設定する必要があります。

参照

- 「メッセージ・サイズの制御」 119 ページ

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・システムの問題によって発生します。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** デバッグ出力を確認して、FTP サーバに接続しているかどうかを判断します。接続している場合は、どの FTP コマンドに問題があるかがデバッグ出力に示されます。
- **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ます。たとえば、次のコマンドを実行します。

  ```
  ping FTP-server-name
  ```

  FTP サーバの IP アドレスと、FTP サーバへの ping (往復) 時間が返されます。FTP 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、FTP 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、ファイアウォールが 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するため、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。

  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ある場合、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設定されているか、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。エラーが返される場合は、FTP サーバの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

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・フォーマットにエンコードされ、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 サーバを使用して送信され、POP サーバから受信されます。

SMTP をサポートしているオペレーティング・システムのリストについては、「サポートされるプラットフォーム」 『SQL Anywhere 11 - 紹介』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、システムに参加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、ユーザが自分の電子メール・サーバ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 メッセージを送受信するには、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。「SMTP/POP アドレスの共有」 130 ページを参照してください。

SMTP メッセージ制御パラメータ

Message Agent (dbremote) 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必要があります。設定していない場合は、ユーザに必要な情報の指定を要求するプロンプトが表示されます。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。

- **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。TOP_supported パラメータを NO に設定すると、SQL Remote は 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 認証を無効にする場合は、このパラメータを 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 サーバが動作しているコンピュータの名前。SMTP/POP3 ログイン・ウィンドウの SMTP ホスト・フィールドに対応しています。
- **pop3_host**　POP ホストを実行しているコンピュータの名前。一般的には、SMTP ホストと同じ名前です。SMTP/POP3 ログイン・ウィンドウの 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するユーザ ID。POP ユーザ ID は、SMTP/POP3 ログイン・ウィンドウのユーザ 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するパスワード。SMTP/POP3 ログイン・ウィンドウのパスワード・フィールドに対応しています。
- **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。この情報は、SMTP/POP のサポート問題のトラブルシューティングに使用できます。デフォルトは NO です。
- **Suppress_dialogs**　このパラメータが 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]** ウィンドウは表示されません。代わりに、エラーが発生します。
- **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・サイズの制御」 119 ページを参照してください。

SMTP/POP アドレスの共有

SQL Remote メッセージには専用の電子メール・アカウントを使用してください。個人的な電子メール・メッセージまたは業務上の電子メール・メッセージに使用する同じ電子メール・アカウントで SQL Remote メッセージを送受信することはおすすめしません。

SQL Remote メッセージと通常の電子メール・メッセージで同じ電子メール・アカウントを共有する必要がある場合は、電子メール・プログラムがメール・サーバからのすべてのメッセージ (SQL Remote の電子メール・メッセージと個人的なメッセージを含む) をダウンロードして削除しないようにします。通常の電子メール・メッセージのダウンロード時に、SQL Remote メッセージを変更したり、削除したりしないように電子メール・プログラムを設定してください。SQL Remote メッセージの件名には、---SQL Remote-- という文字が含まれています。

SMTP リンクの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 (dbremote) が稼働している同じコンピュータから、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ます。SMTP メッセージ・リンクが動作することがわかったら、このプログラムを無効にします。

電子メールが正しく動作していることの確認

SQL Remote メッセージが適切に送受信されない場合、電子メール・メッセージ・システムを使用しているときは、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 システムのバックアップ

SQL Remote 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と、古いトランザクション・ログへのアクセスに依存して行われます。実装するバックアップ方式には、SQL Remote のトランザクション・ログの管理を組み込んでください。

現在のトランザクション・ログと古いトランザクション・ログが不要になるまでは、Message Agent (dbremote) がこれらのログにアクセスできるようにしてください。

統合データベースがそのトランザクション・ログを必要としなくなるのは、すべてのリモート・データベースが受信を完了し、トランザクション・ログに含まれるメッセージが正常に適用されたことを確認したときです。

リモート・データベースがそのトランザクション・ログを必要としなくなるのは、統合データベースが受信を完了し、トランザクション・ログに含まれるメッセージを正常に適用したことを確認したときです。

リモート・データベースのバックアップ

リモート・データベースの場合、次のいずれかを選択する必要があります。

- **バックアップ方法として統合データベースへのレプリケーションに頼る**　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。

  この方法を選択する場合は、リモート・データベースのトランザクション・ログを管理する方式を作成してください。「リモート・データベースのトランザクション・ログの管理」 133 ページを参照してください。

- **リモート・データベースのバックアップ方式を作成する**　リモート・データベースに行われた変更が重要な場合は、トランザクション・ログの管理を含むリモート・データベースのバックアップ方式を作成する必要があります。「リモート・データベースのバックアップ」 134 ページを参照してください。

統合データベースのバックアップ

トランザクション・ログの管理を含む統合データベースのバックアップ方式が必要です。「メディア障害からの統合データベースの保護」 135 ページを参照してください。

バックアップ・ユーティリティ (dbbackup) と Message Agent (dbremote) の -x オプション

データベースでは、-x オプションを指定した Message Agent (dbremote) とバックアップ・ユーティリティ (dbbackup) の両方を実行しないでください。

-x オプションは、レプリケーションのためのトランザクション・ログの管理に使用されます。-x オプションを使用すると、Message Agent は古いトランザクション・ログにアクセスし、これらのトランザクション・ログが不要になると削除します。-x オプションでは、トランザクション・ログはバックアップされません。

バックアップ・ユーティリティ (dbbackup) は、現在のトランザクション・ログのバックアップに使用されます。バックアップ・ユーティリティ (dbbackup) が -r オプションと -n オプションを使用して実行される場合、バックアップ・ユーティリティは、現在のトランザクション・ログをバックアップ・ディレクトリにバックアップし、現在のトランザクション・ログの名前を変更し

て再起動します。バックアップ・ユーティリティ (dbbackup) は、前回のバックアップ後に名前を変更して再起動したトランザクション・ログと現在のトランザクション・ログが同じであると想定します。

-x オプションを指定した Message Agent とバックアップ・ユーティリティ (dbbackup) を同じデータベースで実行しようとすると、相互に干渉します。両方が実行されている場合、トランザクション・ログが失われる可能性があります。

バックアップされていないリモート・データベースでは、-x オプションを指定した Message Agent (dbremote) のみを実行してください。

リモート・データベースのトランザクション・ログの管理

統合データベースへのレプリケーションに頼ってリモート・データベースをバックアップする場合は、次の手順を使用してリモート・データベースのトランザクション・ログを管理します。つまり、リモート・データベースとトランザクション・ログではバックアップ・ユーティリティ (dbbackup) を実行しません。

警告
バックアップ中のデータベースに対しては、-x オプションを指定して Message Agent (dbremote) を実行しないでください。

♦ **リモート・データベースのトランザクション・ログを管理するには、次の手順に従います。**

1. リモート・データベースで、-x オプションを指定して Message Agent (dbremote) を実行し、トランザクション・ログのサイズを指定します。このオプションにより、トランザクション・ログが指定したサイズを超えると、Message Agent (dbremote) はトランザクション・ログの名前を変更して再起動します。

   次の文では、トランザクション・ログが 1 MB より大きくなると、削除されます。

   ```
   dbremote -x 1M  -c "UID=ManagerSteve;PWD=sql;DBF=c:¥mydata.db"
   ```

2. リモート・データベースで、delete_old_logs オプションを On に設定します。delete_old_logs オプションの設定により、古いトランザクション・ログ・ファイルは、レプリケーションで不要になると、Message Agent (dbremote) によって自動的に削除されます。

   トランザクション・ログが不要になるのは、そのトランザクション・ログ・ファイルに記録されている変更をすべて受信して正常に適用したという確認を、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 (dbremote) 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。

   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';
   ```

リモート・データベースのバックアップ

次の手順を使用して、リモート・データベースをバックアップします。この手順には、SQL Remote がトランザクション・ログを使用する場合の管理方式が含まれています。この手順を使用して、-x オプションを指定した Message Agent (dbremote) を実行しないでください。

♦ バックアップ・ユーティリティ (dbbackup) を使用してリモート・データベースをバックアップするには、次の手順に従います。

1. リモート・データベースのフル・バックアップを作成します。
   a. DBA 権限のあるユーザとしてデータベースに接続します。
   b. -r オプションと -n オプションを指定して dbbackup を実行します。

      たとえば、バックアップ・ディレクトリを *e:¥live* と想定すると、データベース・ファイルは *c:¥live* ディレクトリにあり、これに対応するトランザクション・ログ・ファイルは *d:¥live* ディレクトリにあります。

      ```
      dbbackup -r -n -c "UID=DBA;PWD=sql;DBF=c:¥live¥remotedatabase.db" e:¥archive
      ```

      d:¥live ディレクトリ内のトランザクション・ログは、フル・バックアップによって変更されません。
   c. *e:¥archive* ディレクトリにあるバックアップ・ファイルを、外部のドライブまたは DVD にコピーします。
   d. 現在のトランザクション・ログ・ファイルにアクセスしながら Message Agent (dbremote) を実行するには、次のコマンドを使用します。

      ```
      dbremote -c "UID=DBA;PWD=sql;DBF=c:¥live¥database.db" d:¥live
      ```

> **警告**
> バックアップ中のデータベースに対しては、-x オプションを指定して Message Agent (dbremote) を実行しないでください。

2. バックアップ・ユーティリティ (dbbackup) を設定し、リモート・データベースのトランザクション・ログのインクリメンタル・バックアップを作成します。
   a. DBA 権限のあるユーザとしてデータベースに接続します。
   b. -r オプション、-n オプション、-t オプションを指定して dbbackup を実行します。

      次に例を示します。

      ```
      dbbackup -r -n -t -c "UID=DBA;PWD=sql;DBF=c:¥live¥remotedatabase.db" e:¥archive
      ```
   c. 現在のトランザクション・ログ・ファイルにアクセスしながら Message Agent (dbremote) を実行するには、次のコマンドを使用します。

      ```
      dbremote -c "UID=DBA;PWD=sql;DBF=c:¥live¥remotedatabase.db" d:¥live
      ```


メディア障害からの統合データベースの保護

メディア障害から SQL Remote レプリケーション・システムを保護するには、次の手順に従います。

- **バックアップしたトランザクションのみをレプリケートする**　バックアップしたトランザクションのみを含むメッセージを送信します。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・システムを保護できます。これは、次の方法で実現できます。
  - -u オプションを指定して Message Agent (dbremote) を実行する。Message Agent (dbremote) が -u オプションを使用して稼働しているときは、バックアップ済みのコミットされたトランザクションのみがメッセージにパッケージされて送信されます。

    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。「Message Agent (dbremote)」 160 ページを参照してください。
- **トランザクション・ログ・ミラーを使用する**　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・ミラー」『SQL Anywhere サーバ - データベース管理』を参照してください。
- **統合データベースで -x オプションを指定して Message Agent (dbremote) を実行しない**　バックアップ中のデータベースでは、-x オプションを指定して Message Agent (dbremote) を実行しないでください。-x オプションでは、バックアップやリカバリのためでなく、レプリケーションのためにトランザクション・ログが管理されます。統合データベースをバックアップする場合は、「統合データベースのバックアップ」 135 ページを参照してください。

統合データベースのバックアップ

統合データベースとトランザクション・ログのフル・バックアップを作成して統合データベースをバックアップした後、トランザクション・ログのインクリメンタル・バックアップを作成します。

♦ **SQL Remote 統合データベースをバックアップするには、次の手順に従います。**

1. 統合データベースとそのトランザクション・ログのフル・バックアップを作成します。
   a. DBA 権限のあるユーザとしてデータベースに接続します。
   b. -r オプションと -n オプションを指定して dbbackup を実行します。

      次に例を示します。

      ```
      dbbackup -r -n -c "UID=DBA;PWD=sql;DBF=c:¥live¥database.db" e:¥archive
      ```
2. 統合データベースのトランザクション・ログのインクリメンタル・バックアップを作成します。トランザクション・ログのバックアップ時に、トランザクション・ログの名前変更と再起動を選択します。
   a. DBA 権限のあるユーザとしてデータベースに接続します。
   b. -r オプション、-n オプション、-t オプションを指定して dbbackup を実行します。

次に例を示します。

```
dbbackup -r -n -t -c "UID=DBA;PWD=sql;DBF=c:¥live¥database.db" e:¥archive
```

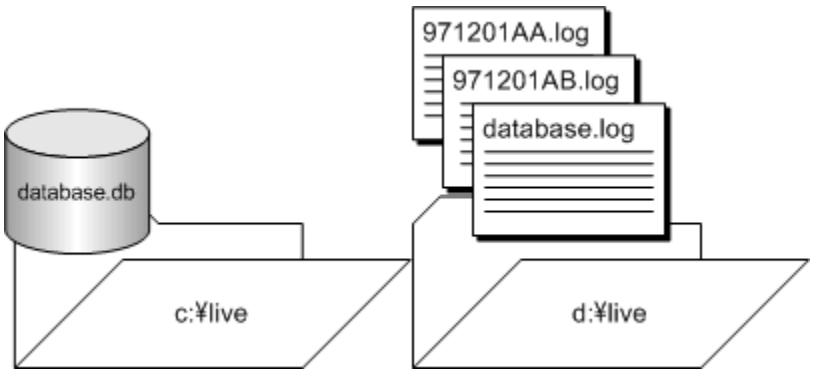
3. 現在のトランザクション・ログ・ファイルにアクセスしながら Message Agent (dbremote) を実行します。

   次に例を示します。

```
dbremote -c "UID=DBA;PWD=sql;DBF=c:¥live¥database.db" d:¥live
```

警告
バックアップ中のデータベースに対しては、-x オプションを指定して Message Agent (dbremote) を実行しないでください。

次の図は、*c:¥live* ディレクトリ内のデータベース *database.db* と *d:¥live* ディレクトリ内のトランザクション・ログ *database.log* を示します。



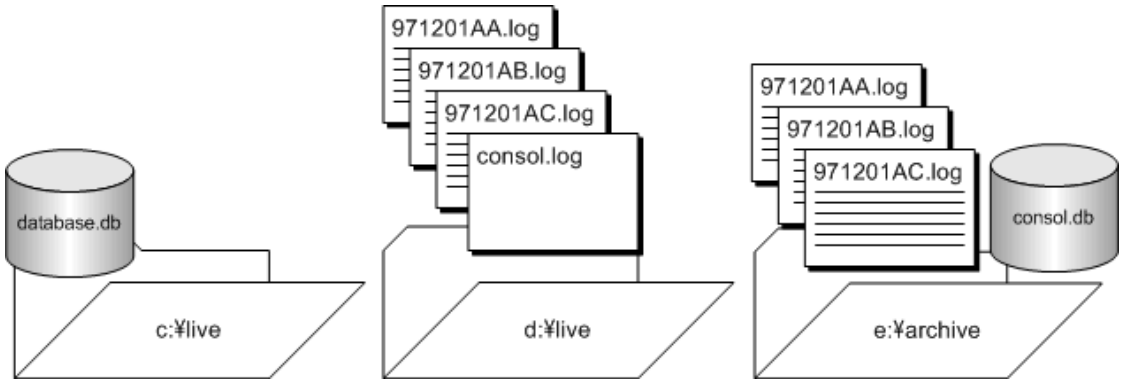
トランザクション・ログの名前を変更して再起動する -r オプションと -n オプションを使用して、バックアップ・ディレクトリ *e:¥archive* にトランザクション・ログをバックアップすると、バックアップ・ユーティリティ (dbbackup) が次のタスクを実行します。

1. 現在のトランザクション・ログ・ファイルの名前を *971201xx.log* (*xx* は *AA* から *ZZ* までの連続した英字) に変更する。
2. バックアップ・ファイル *971201xx.log* を作成して、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する。

8.0.1 より前の古いトランザクション・ログ名
リリース 8.0.1 より前の SQL Anywhere では、古いトランザクション・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、古いトランザクション・ログをより多く保存できるようにするためです。Message Agent (dbremote) 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。

3. *database.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。



参照

- 「メディア障害からの統合データベースの保護」 135 ページ
- 「バックアップ中にトランザクション・ログのバックアップ・コピーの名前を変更する」『SQL Anywhere サーバ - データベース管理』
- 「バックアップ・ユーティリティ (dbbackup)」 『SQL Anywhere サーバ - データベース管理』

統合データベースの手動リカバリ

次の手順では、各トランザクション・ログをデータベースに適用して統合データベースをリカバリする方法について説明します。SQL Anywhere データベース・サーバによって統合データベースを自動的にリカバリする場合は、「統合データベースの自動リカバリ」 140 ページを参照してください。

♦ -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. データベースとトランザクション・ログ・ファイルのコピーを作成します。この手順では、データベース・ファイルは事前にバックアップされており、テープなどに保存されていることを想定しています。
2. テンポラリ・ディレクトリを作成します。
3. データベース (*.db*) ファイルの最新のバックアップ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。トランザクション・ログ・ファイルはリストア**しません**。

   テンポラリ・ディレクトリで、次の処理を実行します。

   a. データベースのバックアップ・コピーを開始します。

   b. -a オプションを使用して、古いトランザクション・ログを適用します。

   c. データベースを停止します。

   d. 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。

   e. データベースを停止します。

   f. データベースをバックアップします。
4. データベースを運用ディレクトリにコピーします。
5. データベースを起動します。

   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：個別のトランザクション・ログの適用

c:¥dbdir¥cons.db という名前の統合データベース・ファイル、トランザクション・ログ・ファイル *c:¥dbdir¥cons.log*、トランザクション・ログ・ミラー・ファイル *d:¥mirdir¥cons.mlg* があると想定します。

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 -n -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 (dbremote) を実行します。次に例を示します。


```
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想定します。

次の手順を使用して、メディア障害からリカバリします。

♦ 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. トランザクション・ログ・ミラー・ファイル *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。

   ```
   dbeng11 -a c:¥dbdir¥dateAA.log c:¥recover¥cons.db
   dbeng11 -a c:¥dbdir¥dateAB.log c:¥recover¥cons.db
   ```

5. 現在のトランザクション・ログ *d:¥mirdir¥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。

   ```
   dbeng11 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。

参照

- 「-a データベース・オプション」 『SQL Anywhere サーバ - データベース管理』

統合データベースの自動リカバリ

次の手順では、統合データベースを自動的にリカバリする方法について説明します。トランザクション・ログを手動で適用する場合は、「統合データベースの手動リカバリ」 138 ページを参照してください。

♦ **-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. データベースとトランザクション・ログ・ファイルのコピーを作成します。この手順では、データベース・ファイルは事前にバックアップされており、テープなどに保存されていることを想定しています。
2. データベース (*.db*) ファイルの最新のバックアップ・コピー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。トランザクション・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、次の処理を実行します。
   a. 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。

      -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、トランザクション・ログ・オフセットに基づいてログを適用します。
   b. 現在のトランザクション・ログをテンポラリ・ディレクトリにコピーします。
   c. データベースを起動し、現在のトランザクション・ログを適用します。
   d. データベース・サーバを停止します。
   e. データベースとトランザクション・ログをバックアップします。
4. データベースとトランザクション・ログ・ファイルを適切な運用ディレクトリにコピーします。
5. システムを通常どおり再起動します。

   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 という名前の統合データベース・ファイル、トランザクション・ログ・ファイル *c:¥dbdir¥cons.log*、トランザクション・ログ・ミラー・ファイル *d:¥mirdir¥cons.mlg* があると想定します。

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 -n 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 -n -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 (dbremote) を実行するものとします。次に例を示します。

```
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想定します。

♦ 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. トランザクション・ログ・ミラー・ファイル *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。

   ```
   dbeng11 c:¥recover¥cons.db -ad c:¥dbdir
   ```
7. 現在のトランザクション・ログ *d:¥mirdir¥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。

   ```
   dbeng11 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。

参照

- 「-ad データベース・オプション」 『SQL Anywhere サーバ - データベース管理』

レプリケーション・エラーのレポートと処理

SQL Remote システムで次のエラーが発生する可能性があります。

- **ローが見つからないエラー**　「ローが見つからないエラー」 57 ページを参照してください。
- **参照整合性エラー**　「参照整合性エラー」 58 ページを参照してください。
- **重複プライマリ・キー・エラー**　「重複プライマリ・キー・エラー」 61 ページを参照してください。

デフォルトでは、エラーが発生すると、Message Agent (dbremote) はエラーをログ出力ウィンドウに出力します。Message Agent (dbremote) は、メッセージ・ウィンドウよりも出力メッセージ・ファイルに多くの情報を出力できます。

Message Agent (dbremote) のメッセージ・ログ・ファイルには、次の情報が含まれます。

- 適用されたメッセージ
- 失敗した SQL 文
- その他のエラー

出力ログ・ファイルにエラーを出力するには、-o オプションを指定して Message Agent (dbremote) を実行します。「Message Agent (dbremote)」 160 ページを参照してください。

エラーが発生したときに、次の処理を実行するように SQL Remote を設定できます。

- **エラー処理プロシージャの実行**　デフォルトではプロシージャを呼び出しません。ただし、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 (dbremote) で呼び出すストアド・プロシージャを指定できます。「エラー処理プロシージャの実行」 142 ページを参照してください。

  たとえば、次の処理を実行するように SQL Remote を設定できます。

  - リモート・データベースの出力ログの一部を統合データベースに送信し、ファイルに書き込みます。「リモート・データベースからのエラーの収集」 143 ページを参照してください。
  - リモート・データベースでエラーが発生したときに電子メールによる通知を送信します。「電子メールによるリモート・データベースのエラーに関する通知の受信」 144 ページを参照してください。
- **エラーの無視**　Message Agent (dbremote) がエラーをレポートしないようにする場合があります。たとえば、エラーが発生する状況を理解しており、エラーによってデータの不一致が発生しないことが確実である場合は、エラーの無視を選択できます。「レプリケーション・エラーの無視」 146 ページを参照してください。

エラー処理プロシージャの実行

replication_error オプションを設定して、SQL エラーが発生したときにプロシージャを呼び出します。デフォルトでは、SQL エラーの発生時にプロシージャを呼び出しません。

呼び出すプロシージャでは、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。「replication_error オプション [SQL Remote]」『SQL Anywhere サーバ - データベース管理』を参照してください。

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

● 次の文を実行します。*remote-user* は Message Agent (dbremote) コマンドのパブリッシャ名、*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。

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

リモート・データベースからのエラーの収集

次の手順を使用して、リモート・データベースの出力ログの一部を統合データベースに送信します。情報はファイルに書き込まれます。このファイルには、システム内の一部またはすべてのリモート・データベースからの出力ロギング情報を格納できます。

♦ リモート・データベースから出力ログ情報を収集するように SQL Remote を設定するには、次の手順に従います。

1. 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します。
   a. SET REMOTE 文と output_log_send_on_error オプションを使用して、エラーの発生時にログ情報を送信します。

      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';
      ```

      Message Agent (dbremote) は、エラーを示す **E** で始まるメッセージを読み込むと、統合データベースに出力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。

   b. この手順はオプションです。統合データベースに送信される情報の量を制限する場合は、SET REMOTE 文に output_log_send_limit オプションを設定します。output_log_send_limit オプションには、統合データベースに送信されるバイト数を指定します。これは、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)。デフォルトは 5 K です。

      output_log_send_limit に最大メッセージ・サイズを超える値を指定すると、SQL Remote は output_log_send_limit の値を上書きし、最大メッセージ・サイズ以下のメッセージだけを送信します。

      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_limit = '7K';
      ```

      「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. ログ情報を受信するように統合データベースを設定します。

   統合データベースで、-ro オプションまたは -rt オプションを指定して Message Agent (dbremote) を実行します。

   「Message Agent (dbremote)」 160 ページを参照してください。

3. この手順はオプションです。設定をテストする場合は、出力ログ情報を統合データベースに送信する output_log_send_now オプションを設定します。

   リモート・データベースで、output_log_send_now オプションを YES に設定します。

   次のポーリング時にリモート・データベースは出力ログ情報を送信し、その後、output_log_send_now オプションが NO にリセットされます。

参照

- 「電子メールによるリモート・データベースのエラーに関する通知の受信」 144 ページ

電子メールによるリモート・データベースのエラーに関する通知の受信

次の手順を使用して、リモート・データベースでエラーが発生したときに電子メールによる通知を送信します。電子メールまたはページング・システムを使用して、通知を受信できます。

♦ 電子メールによるエラーの通知を送信するように SQL Remote を設定するには、次の手順に従います (SQL の場合)。

1. ユーザ Cons として統合データベースに接続します。
2. エラーが発生したことを電子メールで DBA ユーザに通知するストアド・プロシージャを作成します。

   たとえば、次の文を実行して sp_LogReplicationError プロシージャを作成します。

```
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;
```

3. 電子メールの送信を管理するストアド・プロシージャを作成します。

   たとえば、次の文を実行して sp_notifiy_DBA プロシージャを作成します。


```
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; Smith, Elton',
   subject='SQL Remote Error',
   "message"=msg);
 //If mail sent successfully, stop
  IF rc=0 THEN
   call xp_stopmail()
  END IF
 END IF
END;
```

4. エラーの発生を電子メールで DBA に通知するプロシージャを呼び出す replication_error データベース・オプションを設定します。

   たとえば、次の文を実行して、エラーの発生時に sp_LogReplicationError プロシージャを呼び出します。

   ```
   SET OPTION PUBLIC.replication_error =
    'cons.sp_LogReplicationError';
   ```

5. 監査テーブルを作成します。

   たとえば、次の文を実行して replication_audit テーブルを作成します。

   ```
   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)
   );
   ```

   次の表は、replication_audit テーブルのカラムを示します。

カラム	説明
pub	データベースの現在のパブリッシャ (パブリッシャが挿入されたデータベースを識別する)。
remoteuser	メッセージを適用しているリモート・ユーザ (リモート・ユーザのデータベースを識別する)。
errormsg	replication_error プロシージャに渡されたエラー・メッセージ。

6. プロシージャをテストします。

   たとえば、リモート・データベースのローと同じプライマリ・キーを使用するローを統合データベースに挿入します。このローが統合データベースからリモート・データベースにレプリケートされると、プライマリ・キーの競合エラーが発生し、次の処理が実行されます。

- リモート・データベースの Message Agent (dbremote) が、その出力ログに次のメッセージを出力する。

```
Received message from "cons" (0-0000000000-0)
SQL statement failed: (-193) primary key for table 'reptable' is not unique
INSERT INTO cons.reptable( id,text,last_contact )
VALUES (2,'dave','1997/apr/21 16:02:38.325')
COMMIT WORK
```

- 統合データベースに次の INSERT 文が送信される。

```
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;
```

- 次のメッセージを含む電子メールが John Doe と Elton Smith に送信される。

```
primary key for table 'reptable' is not unique (-193)
INSERT INTO cons.reptable( id,text,last_contact ) VALUES (2,'dave','1997/apr/21 16:02:52.605')
```

参照

- 「リモート・データベースからのエラーの収集」 143 ページ

レプリケーション・エラーの無視

♦ **レプリケーション・エラーを無視するには、次の手順に従います (SQL の場合)。**

- 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、テーブルで参照先カラムが見つからないときに発生する INSERT 文のエラーを無視する場合は、参照先カラムが存在しないときに 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 (dbremote) の出力ログにはレポートされません。「リモートの文が失敗しました。」『エラー・メッセージ』を参照してください。

セキュリティ

次の機能を使用して、データを保護できます。

- **REMOTE DBA 権限**　REMOTE DBA 権限を持つユーザを使用して Message Agent (dbremote) に接続することをおすすめします。「REMOTE DBA 権限の付与」 35 ページを参照してください。
- **データベースの暗号化**　-ek オプションを使用してデータベースを暗号化できます。「抽出ユーティリティ (dbxtract)」 170 ページを参照してください。
- **メッセージの暗号化**　Message Agent (dbremote) は、単純暗号化アルゴリズムを使用して、不意なスヌーピングからメッセージを保護します。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。データベース暗号化の詳細については、「データベースの暗号化と復号化」『SQL Anywhere サーバ - データベース管理』を参照してください。

アップグレードと再同期

SQL Remote システムをアップグレードするときは、次のことに注意してください。SQL Remote システムは、次のいずれかの方法でアップグレードできます。

- **ソフトウェアのアップグレード**　SQL Remote のアップグレードについては、「SQL Remote のアップグレード」『SQL Anywhere 11 - 変更点とアップグレード』を参照してください。
- **データベース・スキーマの変更**　データベース・スキーマを変更する場合は、次のことを実行できます。
  - **パススルー・モードの使用**　パススルー・モードによって、スキーマの変更を SQL Remote システム内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・モードの使用には慎重な計画と実行が必要です。「SQL Remote のパススルー・モード」 150 ページを参照してください。
  - **サブスクリプションの再同期**　再同期させるには、データの新しいコピーをリモート・データベースにコピーする必要があります。リモート・データベースが多数ある場合は、再同期に時間がかかり、作業の中断やデータの消失が発生する恐れがあります。「サブスクリプションの再同期」 153 ページを参照してください。

実行中のシステムに行ってはならない変更

次の変更は、配備済みで実行中の SQL Remote システムには行わないでください。ただし、条件が記述されている場合は除きます。

- **パブリッシャの変更**　配備済みで実行中の SQL Remote システムの統合データベースでパブリッシャのユーザ名を変更すると、問題が発生する可能性があります。統合データベースのパブリッシャのユーザ名を変更する必要がある場合は、SQL Remote システムを停止し、すべてのリモート・ユーザを再同期してください。

  リモート・データベースでパブリッシャのユーザ名を変更すると、情報が失われるなど、そのリモート・データベースが関連するサブスクリプションに問題が発生します。リモート・データベースのパブリッシャのユーザ名を変更する必要がある場合は、リモート・データベースを停止し、そのリモート・ユーザを再同期します。「サブスクリプションの再同期」 153 ページを参照してください。
- **テーブルに対して行う制限のある変更**　テーブルに対して制限のある変更を行うことはできません。たとえば、カラムを削除したり、カラムを変更して NULL 値を使用不可にしたりしないでください。これらのカラムを参照するメッセージがシステム内に存在する可能性があるためです。
- **テーブルに対して行う許容できる変更**　パススルー・モードを使用して許容できる変更を行うことは可能です。パススルー・モードを使用してリモート・データベースのスキーマとパブリケーションに変更を行います。許容できる変更には、新しいテーブルやカラムの追加、新しいユーザの追加、ユーザの再同期、ユーザの削除、およびリモート・ユーザのアドレス、メッセージ・タイプ、送信頻度の変更があります。「SQL Remote のパススルー・モード」 150 ページを参照してください。

- **パブリケーションの変更**　パブリケーションの定義は、統合データベースとリモート・データベースの両方で管理されます。SQL Remote システム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、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ります。
- **サブスクリプションの削除**　サブスクリプションは削除できますが、パススルー・モードを使用してリモート・データベースのデータを削除する必要があります。「SQL Remote のパススルー・モード」 150 ページを参照してください。
- **データベースのアンロードと再ロード**　トランザクション・ログが正しく管理されていることを確認してください。「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。
- **多層階層で行う変更**　多層階層でのデータベース・スキーマの再抽出については、「多層階層システムのデータベースの抽出」 94 ページを参照してください。

  パススルー・モードを使用してスキーマ変更を行う方法については、「パススルー・モードの制限事項」 150 ページを参照してください。

SQL Remote のパススルー・モード

パススルー・モードを使用して、標準 SQL 文をそれらの実行が可能なリモート・データベースに渡します。

パススルー・モードを使用すると、SQL Remote システムの実行時に、次のタスクを完了できます。

- 新しいユーザを追加する。
- ユーザを再同期する。
- システム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。

警告

- SQL Remote は、システム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たときに、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。SQL Remote システムの実行時にスキーマの追加変更を実行すると、問題が発生する場合があります。「実行中のシステムに行ってはならない変更」 148 ページを参照してください。
- パススルー・オペレーションは常に、サブスクライブされたリモート・データベースのコピーがある統合データベースのコピーに対して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、運用データベースで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、同じユーザ名のリモート・データベースでは実行されません。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・モードの制限事項

- **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 システムでは、現在のレベルのすぐ下でパススルー文が動作することが重要になります。多層システムでは、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。
- **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、次のようになります。
  - プロシージャが統合データベースで実行されない場合でも、プロシージャはパススルー・コマンドを呼び出す統合データベースに存在します。「PASSTHROUGH 文 [SQL Remote]」『SQL Anywhere サーバ - SQL リファレンス』を参照してください。
  - プロシージャは、リモート・データベースにも存在します。CALL 文または EXEC 文はレプリケートされますが、プロシージャ内の文はレプリケートされません。レプリケートされるデータベース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。

- **制御文**　IF や LOOP などの制御文と、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。「制御文」『SQL Anywhere サーバ - SQL の使用法』を参照してください。
- **カーソル処理**　カーソルの処理はレプリケートされません。
- **SQL SET OPTION 文**　静的な Embedded SQL の SET OPTION 文はレプリケートされません。しかし、動的 SQL 文はレプリケートされます。「静的 SQL と動的 SQL」『SQL Anywhere サーバ - プログラミング』を参照してください。

  たとえば、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 ... "
  ```

- **バッチ文**　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、エラーが発生します。

参照

- 「サブスクリプションの再同期」 153 ページ

パススルー・モードの開始と停止

パススルー・モードの開始と停止には、PASSTHROUGH 文と PASSTHROUGH STOP 文を使用します。パススルー・セッションは、これらの PASSTHROUGH 文の間に入力された文のことを指します。パススルー・セッションに入力された文は、次のように処理されます。

- 構文エラーがチェックされます。
- ONLY キーワードを指定しない場合は、統合データベースで実行されます。ONLY が指定されている場合は、文は統合データベースで実行されずに、リモート・データベースに送信されます。

  次の文を使用すると、現在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;
  ```

- 識別されたサブスクライバのデータベースに渡されます。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、文がトランザクション・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。
- サブスクライバのデータベースで実行されます。

パススルー文の指示

次の文は、pubname パブリケーションに対してサブスクライブされたすべてのユーザ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 statement1;
```

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、すべてのリモート・ユーザのパススルー・セッションを停止します。

```
PASSTHROUGH STOP;
```

データ修正言語 (DML)

パススルー・モードは通常、データ修正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。

次の例は、リモート・データベースおよび統合データベースでテーブルを削除します。

```
-- Drop a table on the remote database
-- and at the consolidated database
PASSTHROUGH TO Joe_Remote;
DROP TABLE CrucialData;
PASSTHROUGH STOP;
```

次の例は、リモート・データベースでのみテーブルを削除します。

```
-- Drop a table on the remote database only
PASSTHROUGH ONLY TO Joe_Remote;
DROP TABLE CrucialData;
PASSTHROUGH STOP;
```

参照

- 「PASSTHROUGH 文 [SQL Remote]」『SQL Anywhere サーバ - SQL リファレンス』
- 「データ修正文」『SQL Anywhere サーバ - SQL の使用法』

サブスクリプションの再同期

リモート・データベースを作成する場合は、統合データベースからスキーマとデータの両方を抽出し、リモート・データベースの構築に使用します。この処理により、各データベースにデータの初期コピーが確実に格納されます。配備の後、次の状況ではサブスクリプションの再同期を検討することがあります。

- **統合データベースに重要な管理作業を実行した後**　たとえば、統合データベースに変更を行い、これによりデータベース内のすべてのローが更新されます。デフォルトでは、SQL Remote は更新メッセージを作成し、サブスクライブされている各リモートに送信します。これらの更新メッセージには、ローごとに UPDATE 文、DELETE 文、INSERT 文が含まれる場合があります。

  SYNCHRONIZE SUBSCRIPTION 文を使用してサブスクリプションを同期させるよう選択した場合は、サブスクライブされたテーブル内のすべてのローを削除するために必要な文と、すべての新しいローを挿入する INSERT 文の送信のみを行います。

- **リモート・データベースが統合データベースと同調していない場合**　リモート・データベースが統合データベースと同調していない場合は、パススルー・モードの使用を試みることができます。「SQL Remote のパススルー・モード」 150 ページを参照してください。

  パススルー・モードが機能しない場合は、サブスクリプションを同期させることができます。サブスクリプションを同期させる場合は、リモート・データベースを統合データベースと強制的に同調させます。SYNCHRONIZE SUBSCRIPTION 文には、リモート・データベース内のサブスクライブされたテーブルの内容を削除する文と、サブスクリプションのローを統合データベースからリモート・データベースに挿入する文が含まれています。

制限事項

- **同期はサブスクリプション全体に適用される**　1 つのテーブルを同期させることはできません。
- **同期でのデータの消失**　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リモート・データベースのデータは、失われます。

  Sybase Central の**データベース・アンロード・ウィザード**またはアンロード・ユーティリティ (dbunload) 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してから、データベースを同期させます。「データベース・アンロード・ウィザードを使用したデータのエクスポート」 『SQL Anywhere サーバ - SQL の使用法』と「アンロード・ユーティリティ (dbunload)」 『SQL Anywhere サーバ - データベース管理』を参照してください。

同期

抽出ユーティリティ (dbxtract) または**データベース抽出ウィザード**のいずれかを使用して、指定したリモート・データベース用のデータを抽出してから、そのデータをリモート・データベースに手動でロードすることをおすすめします。

> **警告**
> 抽出ユーティリティ (dbxtract) または**データベース抽出ウィザード**の実行中は、Message Agent (dbremote) を実行しないでください。

♦ **サブスクリプションを同期させるには、次の手順に従います (Sybase Central の場合)。**

1. リモート・データベースと統合データベースで Message Agent を停止します。
2. DBA 権限のあるユーザとして統合データベースに接続します。
3. 左ウィンドウ枠で、**[パブリケーション]** ディレクトリを展開します。
4. パブリケーションを選択します。
5. 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。
6. サブスクリプションを手動で同期します。
   a. **[サブスクライバ]** リストでユーザを右クリックして、**[プロパティ]** を選択します。
   b. **[詳細]** タブをクリックします。
   c. **[すぐに同期化]** をクリックします。

      [すぐに同期化] ボタンをクリックすると、サブスクリプションが処理されます。プロパティ・ウィンドウで **[キャンセル]** を続けてクリックしても、同期アクションはキャンセルされません。
7. **[OK]** をクリックします。

Message Agent (dbremote) を使用した同期

抽出ユーティリティ (dbxtract) または**データベース抽出ウィザード**を使用してサブスクリプションを同期させることをおすすめします。「同期」 153 ページを参照してください。

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、データベースにアクセスするときの処理速度が低下します。SEND AT 句を使用すると、時間を指定して統合データベースへのアクセスが少ないときに同期させることができます。「送信頻度の設定」 100 ページを参照してください。

♦ **サブスクリプションをメッセージ・システムと同期させるには、次の手順に従います (Interactive SQL の場合)。**

1. DBA 権限のあるユーザとして統合データベースに接続します。
2. SYNCHRONIZE SUBSCRIPTION 文を実行します。「SYNCHRONIZE 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。

   統合データベースの Message Agent (dbremote) 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。Message Agent (dbremote) は、適切なデータベース・スキーマがリモート・データベースに設定されていると想定します。

   サブスクライバ・データベースの Message Agent (dbremote) は、同期メッセージを受信し、サブスクライブされているテーブルの現在の内容を新しいコピーに置き換えます。

警告

- **SYNCHRONIZE SUBSCRIPTION 文をリモート・データベースで実行しない**　SYNCHRONIZE SUBSCRIPTION 文は、統合データベースで実行します。
- **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必要となる可能性があります。メッセージのサイズがリモート・データベースのサイズを超える可能性もあります。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。

  多くの場合、リモート・データベースを抽出してから、データを手動でロードすることをおすすめします。

サブスクリプションの開始

♦ **サブスクリプションを開始するには、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. 左ウィンドウ枠で、**[パブリケーション]** ディレクトリを展開します。
3. パブリケーションを選択します。
4. 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。
5. サブスクリプションを手動で同期します。
   a. **[サブスクライバ]** リストでユーザを右クリックして、**[プロパティ]** を選択します。
   b. **[詳細]** タブをクリックします。
   c. **[すぐに同期化]** をクリックします。

      [すぐに同期化] ボタンをクリックすると、サブスクリプションが処理されます。プロパティ・ウィンドウで **[キャンセル]** を続けてクリックしても、同期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を開始するには、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. START SUBSCRIPTION 文を実行します。「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。

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

サブスクリプションの停止

♦ **サブスクリプションを停止するには、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. 左ウィンドウ枠で、**[パブリケーション]** ディレクトリを展開します。
3. 必要なパブリケーションを選択します。
4. 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。
5. サブスクリプションを手動で停止するには、[サブスクライバ] リスト内のユーザを選択し、**[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、**[すぐに停止]** をクリックしてサブスクリプションを停止します。

   [すぐに停止] ボタンをクリックすると、サブスクリプションが処理されます。プロパティ・ウィンドウで **[キャンセル]** を続けてクリックしても、停止アクションはキャンセルされません。

SQL Remote のリファレンス

この項では、SQL Remote のリファレンス情報を紹介します。

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 (dbremote)

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	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、ファイル難読化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。「ファイル難読化ユーティリティ (dbfhide)」『SQL Anywhere サーバ - データベース管理』を参照してください。 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** というデータベース・サーバの **field** というデータベースに接続する SQL Remote プロセス用に、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、改行を含めたり、あらゆるオプションの設定を格納したりできます。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** というデータベース・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。

オプション	説明
-b	バッチ・モードで実行します。このモードでは、Message Agent は受信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。
-c *"keyword=value; ..."*	接続パラメータを指定します。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 *c:¥mydata.db* にあるデータベース・ファイルで dbremote を実行します。接続にはユーザ ID **DBA** とパスワード **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 権限の付与」 35 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。「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。
-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・ファイルに出力します。
-ek *key*	コマンド・プロンプトで、強力に暗号化されたデータベースの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します。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	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します。このオプションを指定すると、暗号化キーを入力するためのウィンドウが表示されます。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。

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、受信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。
-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・トランザクション・ログ・ミラー・ファイルのロケーションを指定します。このオプションを指定すると、次のどちらかの状況になった場合に、dbremote は古いトランザクション・ログ・ミラー・ファイルを削除できます。 ● オフライン・トランザクション・ログ・ミラーが、トランザクション・ログ・ミラーとは異なるディレクトリに置かれる ● dbremote がリモート・データベース・サーバとは異なるコンピュータで実行されている 通常の設定では、アクティブなトランザクション・ログ・ミラーと名前が変更されたトランザクション・ログ・ミラー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コンピュータ上で実行されるため、このオプションを指定しなくても、古いトランザクション・ログ・ミラー・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 Off 以外の値に設定されている場合だけです。
-o *file*	出力ログ・ファイルにメッセージを出力します。デフォルトでは、画面に出力を表示します。
-os *size*	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、出力メッセージを出力ログ・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 から ZZ までの連続した英字で、*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、手動で古い出力ログ・ファイルを削除し、ディスク領域を解放できます。
-ot *file*	出力ログ・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、画面に出力を送信します。
-p	メッセージをパージしません。

オプション	説明
-q	Message Agent を最小化ウィンドウで起動します。このオプションは、Windows オペレーティング・システムの場合にのみ適用されます。
-qc	完了後、SQL Remote のウィンドウを閉じます。
-r	メッセージを受信します。-r と -s のいずれも指定しない場合、Message Agent は両方のフェーズを実行します。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 オプションで実行された場合、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*	受信メッセージのポーリング頻度を指定します。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。たとえば、次のコマンドでは、30 秒ごとにポーリングが行われます。 dbremote -rd 30s -rd オプションは、欠落しているメッセージの再送を要求するまでに Message Agent が待機するポーリングの回数を設定する -rp オプションとともに使用されることが多くあります。 「メッセージ受信時のパフォーマンス向上」 105 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します。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・データベースからのエラーの収集」 143 ページを参照してください。

オプション	説明
-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を指定します。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欠落していると、Message Agent はそのメッセージが失われたと判断して、メッセージの再送を要求します。このため、メッセージ・システムの処理速度が遅い場合、この動作によって必要のない再送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、再送要求の数を最小限に抑えることができます。 このオプションの設定方法については、「メッセージ受信時のパフォーマンス向上」 105 ページを参照してください。 -rp オプションは、受信メッセージのポーリング頻度を設定する -rd オプションとともに使用されることが多くあります。
-rt *filename*	起動時に出力ログ・ファイルをトランケートし、このファイルにリモート・データベースのログ出力を追加します。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。
-ru *time*	再送の宛先のログを再スキャンする待ち時間を指定します。 「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)、または d (日数) です。
-s	メッセージを送信します。-r と -s のいずれも指定しない場合、Message Agent は両方のフェーズを実行します。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*	データベース・トランザクション・ログのポーリングとポーリングの間の遅延を制御します。-sd オプションは、継続モードで実行する場合にのみ使用されます。 「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。

オプション	説明
-t	すべてのトリガをレプリケートします。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 文で囲みます。 「CURRENT REMOTE USER 特殊定数の使用」 53 ページを参照してください。
-u	オフライン・トランザクション・ログにあるトランザクションのみを処理します。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。このオプションを使用すると、出力トランザクションと入力トランザクションの確認が、オフライン・トランザクション・ログから削除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 プラットフォームで、Message Agent をデーモンとして実行します。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、「リモート・メッセージ・タイプ制御パラメータの設定」 124 ページを参照してください。
-ux	Solaris と Linux で SQL Remote のメッセージ・ウィンドウを開きます。 -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 Server 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、SQL Remote のメッセージ・ウィンドウが自動的に表示されます。

オプション	説明
-v	冗長出力を表示します。このオプションによって、メッセージに含まれる SQL 文がメッセージ・ウィンドウに表示されます。-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*	受信メッセージを適用するデータベース・ワーカ・スレッド数を指定します。このオプションは、Windows Mobile ではサポートされていません。 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。データベース・ワーカ・スレッドの最大数は 50 です。 -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、受信メッセージのスループットが向上します。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受信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た後、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。 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。

オプション	説明
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を指定します。 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。

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。

Message Agent はいくつかのデータベース接続を使用します。「Message Agent (dbremote)」 160 ページを参照してください。

REMOTE DBA 権限の詳細については、「REMOTE DBA 権限の付与」 35 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
  ```

レジストリ設定のリストについては、「SQL Remote メッセージ・システム」 121 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。

抽出ユーティリティ (dbxtract)

SQL Anywhere の統合データベースからリモート・データベースを抽出します。

構文

dbxtract [*options*] [*directory*] *subscriber*

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します。「@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、ファイル難読化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。「ファイル難読化ユーティリティ (dbfhide)」『SQL Anywhere サーバ - データベース管理』を参照してください。
-ac *"keyword=value; ..."*	接続文字列で指定したデータベースに接続して、再ロードします。 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:¥field.db* にロードします。 `dbxtract -c "UID=DBA;PWD=sql;DBF=c:¥cons.db" -ac "UID=DBA;PWD=sql;DBF=c:¥field.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。
-al *filename*	-an オプションを使用している場合は、新しいデータベースのトランザクション・ログ・ファイル名を指定します。

オプション	説明
-an *database*	抽出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します。 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:¥field.db* を作成し、そこに *c:¥cons.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:¥cons.db" -an c:¥field.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。
-ap *size*	新しいデータベースのページ・サイズを設定します。-an が使用されていない場合、このオプションは無視されます。データベースのページ・サイズには、2048、4096、8192、16384、32768 バイトのいずれかを指定できます。デフォルトは元のデータベースのページ・サイズです。データベース・サーバですでにデータベースが実行中の場合、サーバのページ・サイズ (-gp オプションで設定) は新規ページ・サイズを処理するのに十分な容量でなければなりません。「-gp 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。
-b	サブスクリプションを開始しません。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します。 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 データベース・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	データのみを抽出します。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。

オプション	説明
-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します。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 認定のアルゴリズム) 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。 セキュリティを強化するには、128 ビットの場合は AES、256 ビットの場合は AES256 の強力な暗号化を指定します。FIPS 認定の暗号化を使用するには、128 ビットの場合は AES_FIPS、256 ビットの場合は AES256_FIPS をそれぞれ指定してください。強力な暗号化を使用するためには、-ek または -ep オプションも指定する必要があります。強力な暗号化の詳細については、「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。 暗号化されていないデータベースを作成するには、-ea none を指定するか、-ea オプションを指定しません (-e、-et、-ep、-ek オプションも指定しません)。 -ea オプションを指定しない場合、デフォルトの動作は次のようになります。 ● -ea none (-ek、-ep、-et が指定されない場合) ● -ea AES (-ek または -ep が指定される場合) (-et の指定とは無関係) ● -ea simple (-ek または -ep を指定せずに -et を指定する場合) アルゴリズム名の大文字と小文字は区別されません。 **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 認定の暗号化には、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 11 - 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します。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** 強力な暗号化が適用されたデータベースの場合、キーのコピーは必ず安全な場所に保管してください。暗号化キーがわからなくなった場合は、保守契約を結んでいるサポート・センタに依頼してもデータにはアクセスできません。アクセスできなくなったデータベースは、廃棄して、新しくデータベースを作成する必要があります。
-ep	新しいデータベースで使用する暗号化キーの入力を要求します。このオプションを使用すると、ウィンドウ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。
-er	アンロード中に暗号化されたテーブルの暗号化を解除します。 テーブル暗号化が有効であるデータベースから抽出するとき、-er または -et を指定して、新しいデータベースのテーブルで暗号化を有効にするかどうかを指定する必要があります。このオプションを指定しないと、データを新しいデータベースにロードしようとしたときにエラーが発生します。 次のコマンドは、テーブルの暗号化を解除してデータベース (*cons.db*) を抽出し、テーブル暗号化が有効になっていない新しいデータベース (*field.db*) に再ロードします。 dbxtract -an c:¥field.db -er -c "UID=DBA;PWD=sql;DBF=c:¥cons.db;DBKEY=29bN8cj1z field_user"

オプション	説明
-et	新しいデータベースのテーブル暗号化を有効にします (-an または -ar も一緒に指定する必要があります)。-ea オプションを指定せずに -et オプションを指定すると、AES アルゴリズムが使用されます。-et オプションを指定する場合、-ep または -ek も指定しないと操作は失敗します。新しいデータベースのテーブル暗号化設定を変更して、アンロードしているデータベースのテーブル暗号化設定とは異なるものに設定できます。 テーブル暗号化が有効であるデータベースを再構築するとき、-er または -et を指定して、新しいデータベースのテーブルで暗号化を有効にするかどうかを指定する必要があります。このオプションを指定しないと、データを新しいデータベースにロードしようとしたときにエラーが発生します。 次の例は、テーブルが単純暗号化アルゴリズムによって暗号化されたデータベース (*cons.db*) を、テーブル暗号化が有効になっている新しいデータベース (*field.db*) にアンロードし、キーを 34jh として AES_FIPS アルゴリズムを使用します。 dbxtract -an c:¥field.db -et -ea AES_FIPS -ek 34jh -c "UID=DBA;PWD=sql;DBF=c:¥cons.db field_user"
-f	完全に修飾されたパブリケーションを抽出します。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します。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、内部の文ではデータベース・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します。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、内部の文ではデータベース・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します。デフォルト設定では、独立性レベルは 0 です。アクティブなデータベース・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データベース・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 (dbxtract) 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 (dbxtract)」 170 ページを参照してください。
-n	スキーマ定義のみ抽出します。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・スキーマだけを構築する 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。 dbxtract -c "UID=DBA;PWD=sql;DBF=c:¥remote¥cons¥cons.db" -n "c:¥remote¥reload.sql" UserName
-nl	構造体を抽出します (-n オプションと同じ動作)。ただし、生成される *reload.sql* ファイルには、各テーブルに対する LOAD TABLE 文または INPUT 文も含まれます。このオプションを使用した場合、ユーザ・データは抽出されません。-nl を指定するときには、LOAD/INPUT 文を生成できるようにデータ・ディレクトリも指定する必要があります。ただし、このディレクトリにファイルは書き込まれません。このオプションを指定すると、データをアンロードしない再ロード・スクリプトを生成できます。データを抽出するには、-d を指定します。データベースにデータのアンロードが不要なテーブルがある場合は、dbxtract -d -e table-name と指定することによって、データのアンロードを回避できます。
-o *file*	出力ログ・ファイルにメッセージを出力します。

オプション	説明
-p *character*	エスケープ文字を指定します。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いません。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、コマンド・ライン・ユーティリティに対してのみ使用できます。
-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します。 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ません。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。
-v	冗長メッセージを表示します。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	外部キーを除外します。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・フックを除外します。
-xi	外部アンロードと内部再ロードを実行します。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、内部の文ではデータベース・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。

オプション	説明
-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します。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、内部の文ではデータベース・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・ファイルを上書きします。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るディレクトリを指定します。-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します。

備考

デフォルトでは、抽出ユーティリティ (dbxtract) は独立性レベル 0 で実行されます。アクティブなデータベース・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データベース・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、データベース・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。データベース・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 (dbxtract) 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 (dbxtract) を実行することをおすすめします。

抽出ユーティリティ (dbxtract) 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。

バージョン 10.0.0 以降のデータベースの抽出ユーティリティ (dbxtract) または**データベース抽出ウィザード**を使用する場合は、使用する dbxtract のバージョンが、データベースへのアクセスに使用するデータベース・サーバのバージョンと一致する必要があります。dbxtract のバージョン

がデータベース・サーバのバージョンより古いまたは新しい場合は、エラーがレポートされます。

抽出ユーティリティ (dbxtract) と**データベース抽出ウィザード**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 (dbxtract) 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。

参照

- 「リモート・データベースの抽出」 90 ページ
- 「データベースの抽出」 『SQL Anywhere サーバ - SQL の使用法』

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 ～ 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 システム・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 データベースでレプリケーション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 システム・プロシージャ

このシステム・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します。
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します。

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 が起動するときに呼び出されます。

sp_hook_dbremote_end システム・プロシージャ

このシステム・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します。
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します。
exit code	**integer**	0 以外の終了コードはエラーを示します。

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 システム・プロシージャ

このシステム・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します。
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します。
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 システム・プロシージャ

このシステム・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 システム・プロシージャ

このシステム・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

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

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

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

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

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

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

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』

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 PUBLICATION 文 [Mobile Link] [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』
- 「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』

用語解説

用語解説

Adaptive Server Anywhere (ASA)

SQL Anywhere Studio のリレーショナル・データベース・サーバ・コンポーネントであり、主に、モバイル環境と埋め込み環境、または小規模および中規模のビジネス用のサーバとして使用されます。バージョン 10.0.0 で、Adaptive Server Anywhere は SQL Anywhere サーバに、SQL Anywhere Studio は SQL Anywhere にそれぞれ名前が変更されました。

参照：「SQL Anywhere」 198 ページ。

Carrier

Mobile Link システム・テーブルまたは Notifier プロパティ・ファイルに保存される Mobile Link オブジェクトで、システム起動同期で使用される通信業者に関する情報が含まれます。

参照：「サーバ起動同期」 203 ページ。

DB 領域

データ用の領域をさらに作成する追加のデータベース・ファイルです。1 つのデータベースは 13 個までのファイルに保管されます (初期ファイル 1 つと 12 の DB 領域)。各テーブルは、そのインデックスとともに、単一のデータベース・ファイルに含まれている必要があります。CREATE DBSPACE という SQL コマンドで、新しいファイルをデータベースに追加できます。

参照：「データベース・ファイル」 207 ページ。

DBA 権限

ユーザに、データベース内の管理作業を許可するレベルのパーミッションです。DBA ユーザにはデフォルトで DBA 権限が与えられています。

参照：「データベース管理者 (DBA)」 207 ページ。

EBF

Express Bug Fix の略です。Express Bug Fix は、1 つ以上のバグ・フィックスが含まれる、ソフトウェアのサブセットです。これらのバグ・フィックスは、更新のリリース・ノートにリストされます。バグ・フィックス更新を適用できるのは、同じバージョン番号を持つインストール済みのソフトウェアに対してだけです。このソフトウェアについては、ある程度のテストが行われているとはいえ、完全なテストが行われたわけではありません。自分自身でソフトウェアの妥当性を確かめるまでは、アプリケーションとともにこれらのファイルを配布しないでください。

Embedded SQL

C プログラム用のプログラミング・インタフェースです。SQL Anywhere の Embedded SQL は ANSI と IBM 規格に準拠して実装されています。

FILE

SQL Remote のレプリケーションでは、レプリケーション・メッセージのやりとりのために共有ファイルを使うメッセージ・システムのことです。これは特定のメッセージ送信システムに頼らずにテストやインストールを行うのに便利です。

参照：「レプリケーション」 215 ページ。

grant オプション

他のユーザにパーミッションを許可できるレベルのパーミッションです。

iAnywhere JDBC ドライバ

iAnywhere JDBC ドライバでは、pure Java である jConnect JDBC ドライバに比べて何らかの有利なパフォーマンスや機能を備えた JDBC ドライバが提供されます。ただし、このドライバは pure Java ソリューションではありません。iAnywhere JDBC ドライバは一般に推奨されるドライバです。

参照：

- 「JDBC」 195 ページ
- 「jConnect」 195 ページ

InfoMaker

レポート作成とデータ管理用のツールです。洗練されたフォーム、レポート、グラフ、クロスタブ、テーブルを作成できます。また、これらを基本的な構成要素とするアプリケーションも作成できます。

Interactive SQL

データベース内のデータの変更や問い合わせ、データベース構造の修正ができる、SQL Anywhere のアプリケーションです。Interactive SQL では、SQL 文を入力するためのウィンドウ枠が表示されます。また、クエリの進捗情報や結果セットを返すウィンドウ枠も表示されます。

JAR ファイル

Java アーカイブ・ファイルです。Java のアプリケーションで使用される 1 つ以上のパッケージの集合からなる圧縮ファイルのフォーマットです。Java プログラムをインストールしたり実行したりするのに必要なリソースが 1 つの圧縮ファイルにすべて収められています。

Java クラス

Java のコードの主要な構造単位です。これはプロシージャや変数の集まりで、すべてがある一定のカテゴリに関連しているためグループ化されたものです。

jConnect

JavaSoft JDBC 標準を Java で実装したものです。これにより、Java 開発者は多層／異機種環境でもネイティブなデータベース・アクセスができます。iAnywhere JDBC ドライバは一般に推奨されるドライバです。

参照：

- 「JDBC」 195 ページ
- 「iAnywhere JDBC ドライバ」 194 ページ

JDBC

Java Database Connectivity の略です。Java アプリケーションからリレーショナル・データにアクセスすることを可能にする SQL 言語プログラミング・インタフェースです。推奨 JDBC ドライバは、iAnywhere JDBC ドライバです。

参照：

- 「jConnect」 195 ページ
- 「iAnywhere JDBC ドライバ」 194 ページ

Listener

Mobile Link サーバ起動同期に使用される、dblsn という名前のプログラムです。Listener はリモート・デバイスにインストールされ、Push 通知を受け取ったときにデバイス上でアクションが開始されるように設定されます。

参照：「サーバ起動同期」 203 ページ。

LTM

LTM (Log Transfer Manager) は、Replication Agent とも呼ばれます。Replication Server と併用することで、LTM はデータベース・トランザクション・ログを読み込み、コミットされた変更を Sybase Replication Server に送信します。

参照：「Replication Server」 198 ページ。

Mobile Link

Ultra Light と SQL Anywhere のリモート・データベースを統合データベースと同期させるために設計された、セッションベース同期テクノロジです。

参照：

- 「統合データベース」 222 ページ
- 「同期」 222 ページ
- 「Ultra Light」 199 ページ

Mobile Link クライアント

2 種類の Mobile Link クライアントがあります。SQL Anywhere リモート・データベース用の Mobile Link クライアントは、dbmlsync コマンド・ライン・ユーティリティです。Ultra Light リモート・データベース用の Mobile Link クライアントは、Ultra Light ランタイム・ライブラリに組み込まれています。

Mobile Link サーバ

Mobile Link 同期を実行する、mlsrv11 という名前のコンピュータ・プログラムです。

Mobile Link システム・テーブル

Mobile Link の同期に必要なシステム・テーブルです。Mobile Link 設定スクリプトによって、Mobile Link 統合データベースにインストールされます。

Mobile Link モニタ

Mobile Link の同期をモニタするためのグラフィカル・ツールです。

Mobile Link ユーザ

Mobile Link ユーザは、Mobile Link サーバに接続するのに使用されます。Mobile Link ユーザをリモート・データベースに作成し、統合データベースに登録します。Mobile Link ユーザ名はデータベース・ユーザ名から完全に独立しています。

Notifier

Mobile Link サーバ起動同期に使用されるプログラムです。Notifier は Mobile Link サーバに統合されており、統合データベースに Push 要求がないか確認し、Push 通知を送信します。

参照：

- 「サーバ起動同期」 203 ページ
- 「Listener」 195 ページ

ODBC

Open Database Connectivity の略です。データベース管理システムに対する Windows の標準的なインタフェースです。ODBC は、SQL Anywhere がサポートするインタフェースの 1 つです。

ODBC アドミニストレータ

Windows オペレーティング・システムに付属している Microsoft のプログラムです。ODBC データ・ソースの設定に使用します。

ODBC データ・ソース

ユーザが ODBC からアクセスするデータと、そのデータにアクセスするために必要な情報の仕様です。

PDB

Palm のデータベース・ファイルです。

PowerDesigner

データベース・モデリング・アプリケーションです。これを使用すると、データベースやデータ・ウェアハウスの設計に対する構造的なアプローチが可能となります。SQL Anywhere には、PowerDesigner の Physical Data Model コンポーネントが付属します。

PowerJ

Java アプリケーション開発に使用する Sybase 製品です。

Push 通知

QAnywhere では、メッセージ転送を開始するよう QAnywhere クライアントに対して指示するために、サーバから QAnywhere クライアントに配信される特殊なメッセージです。Mobile Link サーバ起動同期では、Push 要求データや内部情報を含むデバイスに Notifer から配信される特殊なメッセージです。

参照：

- 「QAnywhere」 197 ページ
- 「サーバ起動同期」 203 ページ

Push 要求

Mobile Link サーバ起動同期において、Push 通知をデバイスに送信する必要があるかどうかを判断するために Notifier が確認する、結果セット内の値のローです。

参照：「サーバ起動同期」 203 ページ。

QAnywhere

アプリケーション間メッセージング (モバイル・デバイス間メッセージングやモバイル・デバイスとエンタープライズの間のメッセージングなど) を使用すると、モバイル・デバイスや無線デバイスで動作しているカスタム・プログラムと、集中管理されているサーバ・アプリケーションとの間で通信できます。

QAnywhere Agent

QAnywhere では、クライアント・デバイス上で動作する独立のプロセスのことです。クライアント・メッセージ・ストアをモニタリングし、メッセージを転送するタイミングを決定します。

REMOTE DBA 権限

SQL Remote では、Message Agent (dbremote) で必要なパーミッションのレベルを指します。Mobile Link では、SQL Anywhere 同期クライアント (dbmlsync) で必要なパーミッションのレベルを指します。Message Agent (dbremote) または同期クライアントがこの権限のあるユーザとして接続した場合、DBA のフル・アクセス権が与えられます。Message Agent (dbremote) または同期クライアント (dbmlsync) から接続しない場合、このユーザ ID にはパーミッションは追加されません。

参照：「DBA 権限」 193 ページ。

Replication Agent

参照：「LTM」 195 ページ。

Replication Server

SQL Anywhere と Adaptive Server Enterprise で動作する、Sybase による接続ベースのレプリケーション・テクノロジです。Replication Server は、少数のデータベース間でほぼリアルタイムのレプリケーションを行うことを目的に設計されています。

参照：「LTM」 195 ページ。

SQL

リレーショナル・データベースとの通信に使用される言語です。SQL は ANSI により標準が定義されており、その最新版は SQL-2003 です。SQL は、公認されてはいませんが、Structured Query Language の略です。

SQL Anywhere

SQLAnywhere のリレーショナル・データベース・サーバ・コンポーネントであり、主に、モバイル環境と埋め込み環境、または小規模および中規模のビジネス用のサーバとして使用されます。SQL Anywhere は、SQL Anywhere RDBMS、Ultra Light RDBMS、Mobile Link 同期ソフトウェア、その他のコンポーネントを含むパッケージの名前でもあります。

SQL Remote

統合データベースとリモート・データベース間で双方向レプリケーションを行うための、メッセージベースのデータ・レプリケーション・テクノロジです。統合データベースとリモート・データベースは、SQL Anywhere である必要があります。

SQL ベースの同期

Mobile Link では、Mobile Link イベントを使用して、テーブル・データを Mobile Link でサポートされている統合データベースに同期する方法のことです。SQL ベースの同期では、SQL を直接使用したり、Java と .NET 用の Mobile Link サーバ API を使用して SQL を返すことができます。

SQL 文

DBMS に命令を渡すために設計された、SQL キーワードを含む文字列です。

参照：

- 「スキーマ」 205 ページ
- 「SQL」 198 ページ
- 「データベース管理システム (DBMS)」 207 ページ

Sybase Central

SQL Anywhere データベースのさまざまな設定、プロパティ、ユーティリティを使用できる、グラフィカル・ユーザ・インタフェースを持つデータベース管理ツールです。Mobile Link などの他の iAnywhere 製品を管理する場合にも使用できます。

SYS

システム・オブジェクトの大半を所有する特別なユーザです。一般のユーザは SYS でログインできません。

Ultra Light

小型デバイス、モバイル・デバイス、埋め込みデバイス用に最適化されたデータベースです。対象となるプラットフォームとして、携帯電話、ポケットベル、パーソナル・オーガナイザなどが挙げられます。

Ultra Light ランタイム

組み込みの Mobile Link 同期クライアントを含む、インプロセス・リレーショナル・データベース管理システムです。Ultra Light ランタイムは、Ultra Light の各プログラミング・インタフェースで使用されるライブラリと、Ultra Light エンジンの両方に含まれます。

Windows

Windows Vista、Windows XP、Windows 200x などの、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 CE

「Windows Mobile」 199 ページを参照してください。

Windows Mobile

Microsoft がモバイル・デバイス用に開発したオペレーティング・システムのファミリです。

アーティクル

Mobile Link または SQL Remote では、テーブル全体もしくはテーブル内のカラムとローのサブセットを表すデータベース・オブジェクトを指します。アーティクルの集合がパブリケーションです。

参照：

- 「レプリケーション」 215 ページ
- 「パブリケーション」 210 ページ

アップロード

同期中に、リモート・データベースから統合データベースにデータが転送される段階です。

アトミックなトランザクション

完全に処理されるかまったく処理されないことが保証される 1 つのトランザクションです。エラーによってアトミックなトランザクションの一部が処理されなかった場合は、データベースが一貫性のない状態になるのを防ぐために、トランザクションがロールバックされます。

アンロード

データベースをアンロードすると、データベースの構造かデータ、またはその両方がテキスト・ファイルにエクスポートされます (構造は SQL コマンド・ファイルに、データはカンマ区切りの ASCII ファイルにエクスポートされます)。データベースのアンロードには、アンロード・ユーティリティを使用します。

また、UNLOAD 文を使って、データから抜粋した部分だけをアンロードできます。

イベント・モデル

Mobile Link では、同期を構成する、begin_synchronization や download_cursor などの一連のイベントのことです。イベントは、スクリプトがイベント用に作成されると呼び出されます。

インクリメンタル・バックアップ

トランザクション・ログ専用のバックアップです。通常、フル・バックアップとフル・バックアップの間に使用します。

参照：「トランザクション・ログ」 209 ページ。

インデックス

ベース・テーブルにある 1 つ以上のカラムに関連付けられた、キーとポインタのソートされたセットです。テーブルの 1 つ以上のカラムにインデックスが設定されていると、パフォーマンスが向上します。

ウィンドウ

分析関数の実行対象となるローのグループです。ウィンドウには、ウィンドウ定義内のグループ化指定に従って分割されたデータの、1 つ、複数、またはすべてのローが含まれます。ウィンドウは、入力の現在のローについて計算を実行する必要があるローの数や範囲を含むように移動します。ウィンドウ構成の主な利点は、追加のクエリを実行しなくても、結果をグループ化して分析する機会が増えることです。

エージェント ID

参照：「クライアント・メッセージ・ストア ID」 202 ページ。

エンコード

文字コードとも呼ばれます。エンコードは、文字セットの各文字が情報の 1 つまたは複数のバイトにマッピングされる方法のことで、一般的に 16 進数で表現されます。UTF-8 はエンコードの例です。

参照：

- 「文字セット」 223 ページ
- 「コード・ページ」 203 ページ
- 「照合」 220 ページ

オブジェクト・ツリー

Sybase Central では、データベース・オブジェクトの階層を指します。オブジェクト・ツリーの最上位には、現在使用しているバージョンの Sybase Central がサポートするすべての製品が表示されます。それぞれの製品を拡張表示すると、オブジェクトの下位ツリーが表示されます。

参照：「Sybase Central」 199 ページ。

カーソル

結果セットへの関連付けに名前を付けたもので、プログラミング・インタフェースからローにアクセスしたり更新したりするときに使用します。SQL Anywhere では、カーソルはクエリ結果内で前方や後方への移動をサポートします。カーソルは、カーソル結果セット (通常 SELECT 文で定義される) とカーソル位置の 2 つの部分から構成されます。

参照：

- 「カーソル結果セット」 202 ページ
- 「カーソル位置」 201 ページ

カーソル位置

カーソル結果セット内の 1 つのローを指すポインタ。

参照：

- 「カーソル」 201 ページ
- 「カーソル結果セット」 202 ページ

カーソル結果セット

カーソルに関連付けられたクエリから生成されるローのセットです。

参照：

- 「カーソル」 201 ページ
- 「カーソル位置」 201 ページ

クエリ

データベースのデータにアクセスしたり、そのデータを操作したりする SQL 文や SQL 文のグループです。

参照：「SQL」 198 ページ。

クライアント／サーバ

あるアプリケーション (クライアント) が別のアプリケーション (サーバ) に対して情報を送受信するソフトウェア・アーキテクチャのことです。通常この 2 種類のアプリケーションは、ネットワークに接続された異なるコンピュータ上で実行されます。

クライアント・メッセージ・ストア

QAnywhere では、メッセージを保管するリモート・デバイスにある SQL Anywhere データベースのことです。

クライアント・メッセージ・ストア ID

QAnywhere では、Mobile Link リモート ID のことです。これによって、クライアント・メッセージ・ストアがユニークに識別されます。

グローバル・テンポラリ・テーブル

明示的に削除されるまでデータ定義がすべてのユーザに表示されるテンポラリ・テーブルです。グローバル・テンポラリ・テーブルを使用すると、各ユーザが、1 つのテーブルのまったく同じインスタンスを開くことができます。デフォルトでは、コミット時にローが削除され、接続終了時にもローが削除されます。

参照：

- 「テンポラリ・テーブル」 208 ページ
- 「ローカル・テンポラリ・テーブル」 216 ページ

ゲートウェイ

Mobile Link システム・テーブルまたは Notifier プロパティ・ファイルに保存される Mobile Link オブジェクトで、システム起動同期用のメッセージの送信方法に関する情報が含まれます。

参照：「サーバ起動同期」 203 ページ。

コード・ページ

コード・ページは、文字セットの文字を数値表示 (通常 0 ～ 255 の整数) にマッピングするエンコードです。Windows Code Page 1252 などのコード・ページがあります。このマニュアルの目的上、コード・ページとエンコードは同じ意味で使用されます。

参照：

- 「文字セット」 223 ページ
- 「エンコード」 201 ページ
- 「照合」 220 ページ

コマンド・ファイル

SQL 文で構成されたテキスト・ファイルです。コマンド・ファイルは手動で作成できますが、データベース・ユーティリティによって自動的に作成することもできます。たとえば、dbunload ユーティリティを使うと、指定されたデータベースの再構築に必要な SQL 文で構成されたコマンド・ファイルを作成できます。

サーバ・メッセージ・ストア

QAnywhere では、サーバ上のリレーショナル・データベースです。このデータベースは、メッセージを、クライアント・メッセージ・ストアまたは JMS システムに転送されるまで一時的に格納します。メッセージは、サーバ・メッセージ・ストアを介して、クライアント間で交換されます。

サーバ管理要求

XML 形式の QAnywhere メッセージです。サーバ・メッセージ・ストアを管理したり、QAnywhere アプリケーションをモニタリングするために QAnywhere システム・キューに送信されます。

サーバ起動同期

Mobile Link サーバから Mobile Link 同期を開始する方法です。

サービス

Windows オペレーティング・システムで、アプリケーションを実行するユーザ ID がログオンしていないときにアプリケーションを実行する方法です。

サブクエリ

別の SELECT 文、INSERT 文、UPDATE 文、DELETE 文、または別のサブクエリの中にネストされた SELECT 文です。

相関とネストの 2 種類のサブクエリがあります。

サブスクリプション

Mobile Link 同期では、パブリケーションと Mobile Link ユーザ間のクライアント・データベース内のリンクであり、そのパブリケーションが記述したデータの同期を可能にします。

SQL Remote レプリケーションでは、パブリケーションとリモート・ユーザ間のリンクのことで、これによりリモート・ユーザはそのパブリケーションの更新内容を統合データベースとの間で交換できます。

参照：

- 「パブリケーション」 210 ページ
- 「Mobile Link ユーザ」 196 ページ

システム・オブジェクト

SYS または dbo が所有するデータベース・オブジェクトです。

システム・テーブル

SYS または dbo が所有するテーブルです。メタデータが格納されています。システム・テーブル (データ辞書テーブルとしても知られています) はデータベース・サーバが作成し管理します。

システム・ビュー

すべてのデータベースに含まれているビューです。システム・テーブル内に格納されている情報をわかりやすいフォーマットで示します。

ジョイン

指定されたカラムの値を比較することによって 2 つ以上のテーブルにあるローをリンクする、リレーショナル・システムでの基本的な操作です。

ジョイン・タイプ

SQL Anywhere では、クロス・ジョイン、キー・ジョイン、ナチュラル・ジョイン、ON 句を使ったジョインの 4 種類のジョインが使用されます。

参照：「ジョイン」 204 ページ。

ジョイン条件

ジョインの結果に影響を及ぼす制限です。ジョイン条件は、JOIN の直後に ON 句か WHERE 句を挿入して指定します。ナチュラル・ジョインとキー・ジョインについては、SQL Anywhere がジョイン条件を生成します。

参照：

- 「ジョイン」 204 ページ
- 「生成されたジョイン条件」 221 ページ

スキーマ

テーブル、カラム、インデックス、それらの関係などを含んだデータベース構造です。

スクリプト

Mobile Link では、Mobile Link のイベントを処理するために記述されたコードです。スクリプトは、業務上の要求に適合するように、データ交換をプログラム的に制御します。

参照：「イベント・モデル」 200 ページ。

スクリプト・バージョン

Mobile Link では、同期を作成するために同時に適用される、一連の同期スクリプトです。

スクリプトベースのアップロード

Mobile Link では、ログ・ファイルを使用した方法の代わりとなる、アップロード処理のカスタマイズ方法です。

ストアド・プロシージャ

ストアド・プロシージャは、データベースに保存され、データベース・サーバに対する一連の操作やクエリを実行するために使用される SQL 命令のグループです。

スナップショット・アイソレーション

読み込み要求を発行するトランザクション用のデータのコミットされたバージョンを返す、独立性レベルの種類です。SQL Anywhere では、スナップショット、文のスナップショット、読み込み専用文のスナップショットの 3 つのスナップショットの独立性レベルがあります。スナップショット・アイソレーションが使用されている場合、読み込み処理は書き込み処理をブロックしません。

参照：「独立性レベル」 223 ページ。

セキュア機能

データベース・サーバが起動されたときに、そのデータベース・サーバで実行されているデータベースでは使用できないように -sf オプションによって指定される機能です。

セッション・ベースの同期

統合データベースとリモート・データベースの両方でデータ表現の一貫性が保たれる同期です。Mobile Link はセッション・ベースです。

ダイレクト・ロー・ハンドリング

Mobile Link では、テーブル・データを Mobile Link でサポートされている統合データベース以外のソースに同期する方法のことです。アップロードとダウンロードの両方をダイレクト・ロー・ハンドリングで実装できます。

参照：

- 「統合データベース」 222 ページ
- 「SQL ベースの同期」 199 ページ

ダウンロード

同期中に、統合データベースからリモート・データベースにデータが転送される段階です。

チェックサム

データベース・ページを使用して記録されたデータベース・ページのビット数の合計です。チェックサムを使用すると、データベース管理システムは、ページがディスクに書き込まれるときに数が一致しているかを確認することで、ページの整合性を検証できます。数が一致した場合は、ページが正常に書き込まれたとみなされます。

チェックポイント

データベースに加えたすべての変更内容がデータベース・ファイルに保存されるポイントです。通常、コミットされた変更内容はトランザクション・ログだけに保存されます。

データ・キューブ

同じ結果を違う方法でグループ化およびソートされた内容を各次元に反映した、多次元の結果セットです。データ・キューブは、セルフジョイン・クエリと相関サブクエリを必要とするデータの複雑な情報を提供します。データ・キューブは OLAP 機能の一部です。

データベース

プライマリ・キーと外部キーによって関連付けられているテーブルの集合です。これらのテーブルでデータベース内の情報が保管されます。また、テーブルとキーによってデータベースの構造が定義されます。データベース管理システムでこの情報にアクセスします。

参照：

- 「外部キー」 217 ページ
- 「プライマリ・キー」 212 ページ
- 「データベース管理システム (DBMS)」 207 ページ
- 「リレーショナル・データベース管理システム (RDBMS)」 215 ページ

データベース・オブジェクト

情報を保管したり受け取ったりするデータベース・コンポーネントです。テーブル、インデックス、ビュー、プロシージャ、トリガはデータベース・オブジェクトです。

データベース・サーバ

データベース内にある情報へのすべてのアクセスを規制するコンピュータ・プログラムです。SQL Anywhere には、ネットワーク・サーバとパーソナル・サーバの 2 種類のサーバがあります。

データベース・ファイル

データベースは 1 つまたは複数のデータベース・ファイルに保持されます。まず、初期ファイルがあり、それに続くファイルは DB 領域と呼ばれます。各テーブルは、それに関連付けられているインデックスとともに、単一のデータベース・ファイルに含まれている必要があります。

参照：「DB 領域」 193 ページ。

データベース管理システム (DBMS)

データベースを作成したり使用したりするためのプログラムの集合です。

参照：「リレーショナル・データベース管理システム (RDBMS)」 215 ページ。

データベース管理者 (DBA)

データベースの管理に必要なパーミッションを持つユーザです。DBA は、データベース・スキーマのあらゆる変更や、ユーザやグループの管理に対して、全般的な責任を負います。データベース管理者のロールはデータベース内に自動的に作成されます。その場合、ユーザ ID は DBA であり、パスワードは sql です。

データベース所有者 (dbo)

SYS が所有しないシステム・オブジェクトを所有する特別なユーザです。

参照：

- 「データベース管理者 (DBA)」 207 ページ
- 「SYS」 199 ページ

データベース接続

クライアント・アプリケーションとデータベース間の通信チャネルです。接続を確立するためには有効なユーザ ID とパスワードが必要です。接続中に実行できるアクションは、そのユーザ ID に付与された権限によって決まります。

データベース名

サーバがデータベースをロードするとき、そのデータベースに指定する名前です。デフォルトのデータベース名は、初期データベース・ファイルのルート名です。

参照：「データベース・ファイル」 207 ページ。

データ型

CHAR や NUMERIC などのデータのフォーマットです。ANSI SQL 規格では、サイズ、文字セット、照合に関する制限もデータ型に組み込めます。

参照：「ドメイン」 208 ページ。

データ操作言語 (DML)

データベース内のデータの操作に使う SQL 文のサブセットです。DML 文は、データベース内のデータを検索、挿入、更新、削除します。

データ定義言語 (DDL)

データベース内のデータの構造を定義するときに使う SQL 文のサブセットです。DDL 文は、テーブルやユーザなどのデータベース・オブジェクトを作成、変更、削除できます。

デッドロック

先へ進めない場所に一連のトランザクションが到達する状態です。

デバイス・トラッキング

Mobile Link サーバ起動同期において、デバイスを特定する Mobile Link のユーザ名を使用して、メッセージのアドレスを指定できる機能です。

参照：「サーバ起動同期」 203 ページ。

テンポラリ・テーブル

データを一時的に保管するために作成されるテーブルです。グローバルとローカルの 2 種類があります。

参照：

- 「ローカル・テンポラリ・テーブル」 216 ページ
- 「グローバル・テンポラリ・テーブル」 202 ページ

ドメイン

適切な位置に精度や小数点以下の桁数を含み、さらにオプションとしてデフォルト値や CHECK 条件などを含んでいる、組み込みデータ型のエイリアスです。ドメインには、通貨データ型のように SQL Anywhere が事前に定義したものもあります。ユーザ定義データ型とも呼ばれます。

参照：「データ型」 208 ページ。

トランザクション

作業の論理単位を構成する一連の SQL 文です。1 つのトランザクションは完全に処理されるかまったく処理されないかのどちらかです。SQL Anywhere は、ロック機能のあるトランザクション処理をサポートしているので、複数のトランザクションが同時にデータベースにアクセスしてもデータを壊すことがありません。トランザクションは、データに加えた変更を永久なものにする COMMIT 文か、トランザクション中に加えられたすべての変更を元に戻す ROLLBACK 文のいずれかで終了します。

トランザクション・ログ

データベースに対するすべての変更内容が、変更された順に格納されるファイルです。パフォーマンスを向上させ、データベース・ファイルが破損した場合でもデータをリカバリできます。

トランザクション・ログ・ミラー

オプションで設定できる、トランザクション・ログ・ファイルの完全なコピーのことで、トランザクション・ログと同時に管理されます。データベースの変更がトランザクション・ログへ書き込まれると、トランザクション・ログ・ミラーにも同じ内容が書き込まれます。

ミラー・ファイルは、トランザクション・ログとは別のデバイスに置いてください。一方のデバイスに障害が発生しても、もう一方のログにリカバリのためのデータが確保されます。

参照：「トランザクション・ログ」 209 ページ。

トランザクション単位の整合性

Mobile Link で、同期システム全体でのトランザクションの管理を保証します。トランザクション全体が同期されるか、トランザクション全体がまったく同期されないかのどちらかになります。

トリガ

データを修正するクエリをユーザが実行すると、自動的に実行されるストアド・プロシージャの特別な形式です。

参照：

- 「ロー・レベルのトリガ」 216 ページ
- 「文レベルのトリガ」 223 ページ
- 「整合性」 220 ページ

ネットワーク・サーバ

共通ネットワークを共有するコンピュータからの接続を受け入れるデータベース・サーバです。

参照：「パーソナル・サーバ」 210 ページ。

ネットワーク・プロトコル

TCP/IP や HTTP などの通信の種類です。

パーソナル・サーバ

クライアント・アプリケーションが実行されているコンピュータと同じマシンで実行されているデータベース・サーバです。パーソナル・データベース・サーバは、単一のコンピュータ上で単一のユーザが使用しますが、そのユーザからの複数の同時接続をサポートできます。

パッケージ

Java では、それぞれが互いに関連のあるクラスの集合を指します。

ハッシュ

ハッシュは、インデックスのエントリをキーに変換する、インデックスの最適化のことです。インデックスのハッシュの目的は、必要なだけの実際のロー・データをロー ID に含めることで、インデックスされた値を特定するためのローの検索、ロード、アンパックという負荷の高い処理を避けることです。

パフォーマンス統計値

データベース・システムのパフォーマンスを反映する値です。たとえば、CURRREAD 統計値は、データベース・サーバが要求したファイル読み込みのうち、現在まだ完了していないものの数を表します。

パブリケーション

Mobile Link または SQL Remote では、同期されるデータを識別するデータベース・オブジェクトのことです。Mobile Link では、クライアント上にのみ存在します。1 つのパブリケーションは複数のアーティクルから構成されています。SQL Remote ユーザは、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ことによって、パブリケーションを受信できます。Mobile Link ユーザは、パブリケーションに対して同期サブスクリプションを作成することによって、パブリケーションを同期できます。

参照：

- 「レプリケーション」 215 ページ
- 「アーティクル」 200 ページ
- 「パブリケーションの更新」 210 ページ

パブリケーションの更新

SQL Remote レプリケーションでは、単一のデータベース内の 1 つまたは複数のパブリケーションに対して加えられた変更のリストを指します。パブリケーションの更新は、レプリケーション・メッセージの一部として定期的にリモート・データベースへ送られます。

参照：

- 「レプリケーション」 215 ページ
- 「パブリケーション」 210 ページ

パブリッシャ

SQL Remote レプリケーションでは、レプリケートできる他のデータベースとレプリケーション・メッセージを交換できるデータベースの単一ユーザを指します。

参照：「レプリケーション」 215 ページ。

ビジネス・ルール

実世界の要求に基づくガイドラインです。通常ビジネス・ルールは、検査制約、ユーザ定義データ型、適切なトランザクションの使用により実装されます。

参照：

- 「制約」 220 ページ
- 「ユーザ定義データ型」 214 ページ

ヒストグラム

ヒストグラムは、カラム統計のもっとも重要なコンポーネントであり、データ分散を表します。SQL Anywhere は、ヒストグラムを維持して、カラムの値の分散に関する統計情報をオプティマイザに提供します。

ビット配列

ビット配列は、一連のビットを効率的に保管するのに使用される配列データ構造の種類です。ビット配列は文字列に似てますが、使用される要素は文字ではなく 0 (ゼロ) と 1 になります。ビット配列は、一般的にブール値の文字列を保持するのに使用されます。

ビュー

データベースにオブジェクトとして格納される SELECT 文です。ビューを使用すると、ユーザは 1 つまたは複数のテーブルのローやカラムのサブセットを参照できます。ユーザが特定のテーブルやテーブルの組み合わせのビューを使うたびに、テーブルに保持されているデータから再計算されます。ビューは、セキュリティの目的に有用です。またデータベース情報の表示を調整して、データへのアクセスが簡単になるようにする場合も役立ちます。

ファイルベースのダウンロード

Mobile Link では、ダウンロードがファイルとして配布されるデータの同期方法であり、同期変更のオフライン配布を可能にします。

ファイル定義データベース

Mobile Link では、ダウンロード・ファイルの作成に使用される SQL Anywhere データベースのことです。

参照：「ファイルベースのダウンロード」 211 ページ。

フェールオーバ

アクティブなサーバ、システム、またはネットワークで障害や予定外の停止が発生したときに、冗長な (スタンバイ) サーバ、システム、またはネットワークに切り替えることです。フェールオーバは自動的に発生します。

プライマリ・キー

テーブル内のすべてのローをユニークに識別する値を持つカラムまたはカラムのリストです。

参照：「外部キー」 217 ページ。

プライマリ・キー制約

プライマリ・キーのカラムに対する一意性制約です。テーブルにはプライマリ・キー制約を 1 つしか設定できません。

参照：

- 「制約」 220 ページ
- 「検査制約」 219 ページ
- 「外部キー制約」 218 ページ
- 「一意性制約」 217 ページ
- 「整合性」 220 ページ

プライマリ・テーブル

外部キー関係でプライマリ・キーを含むテーブルです。

プラグイン・モジュール

Sybase Central で、製品にアクセスしたり管理したりする方法です。プラグインは、通常、インストールすると Sybase Central にもインストールされ、自動的に登録されます。プラグインは、多くの場合、Sybase Central のメイン・ウィンドウに最上位のコンテナとして、その製品名 (たとえば SQL Anywhere) で表示されます。

参照：「Sybase Central」 199 ページ。

フル・バックアップ

データベース全体をバックアップすることです。オプションでトランザクション・ログのバックアップも可能です。フル・バックアップには、データベース内のすべての情報が含まれており、システム障害やメディア障害が発生した場合の保護として機能します。

参照：「インクリメンタル・バックアップ」 200 ページ。

プロキシ・テーブル

メタデータを含むローカル・テーブルです。リモート・データベース・サーバのテーブルに、ローカル・テーブルであるかのようにアクセスするときに使用します。

参照：「メタデータ」 213 ページ。

ベース・テーブル

データを格納する永久テーブルです。テーブルは、テンポラリ・テーブルやビューと区別するために、「ベース・テーブル」と呼ばれることがあります。

参照：

- 「テンポラリ・テーブル」 208 ページ
- 「ビュー」 211 ページ

ポーリング

Mobile Link サーバ起動同期において、Mobile Link Listerner などのライト・ウェイト・ポーラが Notifier から Push 通知を要求する方法です。

参照：「サーバ起動同期」 203 ページ。

ポリシー

QAnywhere では、メッセージ転送の発生時期を指定する方法のことです。

マテリアライズド・ビュー

計算され、ディスクに保存されたビューのことです。マテリアライズド・ビューは、ビュー (クエリ指定を使用して定義される) とテーブル (ほとんどのテーブルの操作をそのテーブル上で実行できる) の両方の特性を持ちます。

参照：

- 「ベース・テーブル」 213 ページ
- 「ビュー」 211 ページ

ミラー・ログ

参照：「トランザクション・ログ・ミラー」 209 ページ。

メタデータ

データについて説明したデータです。メタデータは、他のデータの特質と内容について記述しています。

参照：「スキーマ」 205 ページ。

メッセージ・システム

SQL Remote のレプリケーションでは、統合データベースとリモート・データベースの間でのメッセージのやりとりに使用するプロトコルのことです。SQL Anywhere では、FILE、FTP、SMTP のメッセージ・システムがサポートされています。

参照：

- 「レプリケーション」 215 ページ
- 「FILE」 194 ページ

メッセージ・ストア

QAnywhere では、メッセージを格納するクライアントおよびサーバ・デバイスのデータベースのことです。

参照：

- 「クライアント・メッセージ・ストア」 202 ページ
- 「サーバ・メッセージ・ストア」 203 ページ

メッセージ・タイプ

SQL Remote のレプリケーションでは、リモート・ユーザと統合データベースのパブリッシャとの通信方法を指定するデータベース・オブジェクトのことを指します。統合データベースには、複数のメッセージ・タイプが定義されていることがあります。これによって、リモート・ユーザはさまざまなメッセージ・システムを使って統合データベースと通信できることになります。

参照：

- 「レプリケーション」 215 ページ
- 「統合データベース」 222 ページ

メッセージ・ログ

データベース・サーバや Mobile Link サーバなどのアプリケーションからのメッセージを格納できるログです。この情報は、メッセージ・ウィンドウに表示されたり、ファイルに記録されたりすることもあります。メッセージ・ログには、情報メッセージ、エラー、警告、MESSAGE 文からのメッセージが含まれます。

メンテナンス・リリース

メンテナンス・リリースは、同じメジャー・バージョン番号を持つ旧バージョンのインストール済みソフトウェアをアップグレードするための完全なソフトウェア・セットです (バージョン番号のフォーマットは、*メジャー.マイナー.パッチ.ビルド*です)。バグ・フィックスとその他の変更については、アップグレードのリリース・ノートにリストされます。

ユーザ定義データ型

参照：「ドメイン」 208 ページ。

ライト・ウェイト・ポーラ

Mobile Link サーバ起動同期において、Mobile Link サーバからの Push 通知をポーリングするデバイス・アプリケーションです。

参照：「サーバ起動同期」 203 ページ。

リダイレクタ

クライアントと Mobile Link サーバ間で要求と応答をルート指定する Web サーバ・プラグインです。このプラグインによって、負荷分散メカニズムとフェールオーバ・メカニズムも実装されます。

リファレンス・データベース

Mobile Link では、Ultra Light クライアントの開発に使用される SQL Anywhere データベースです。開発中は、1 つの SQL Anywhere データベースをリファレンス・データベースとしても統合データベースとしても使用できます。他の製品によって作成されたデータベースは、リファレンス・データベースとして使用できません。

リモート ID

SQL Anywhere と Ultra Light データベース内のユニークな識別子で、Mobile Link によって使用されます。リモート ID は NULL に初期設定されていますが、データベースの最初の同期時に GUID に設定されます。

リモート・データベース

Mobile Link または SQL Remote では、統合データベースとデータを交換するデータベースを指します。リモート・データベースは、統合データベース内のすべてまたは一部のデータを共有できます。

参照：

- 「同期」 222 ページ
- 「統合データベース」 222 ページ

リレーショナル・データベース管理システム (RDBMS)

関連するテーブルの形式でデータを格納するデータベース管理システムです。

参照：「データベース管理システム (DBMS)」 207 ページ。

レプリケーション

物理的に異なるデータベース間でデータを共有することです。Sybase では、Mobile Link、SQL Remote、Replication Server の 3 種類のレプリケーション・テクノロジを提供しています。

レプリケーション・メッセージ

SQL Remote または Replication Server では、パブリッシュするデータベースとサブスクリプションを作成するデータベース間で送信される通信内容を指します。メッセージにはデータを含み、レプリケーション・システムで必要なパススルー文、情報があります。

参照：

- 「レプリケーション」 215 ページ
- 「パブリケーションの更新」 210 ページ

レプリケーションの頻度

SQL Remote レプリケーションでは、リモート・ユーザに対する設定の 1 つで、パブリッシャの Message Agent がレプリケーション・メッセージを他のリモート・ユーザに送信する頻度を定義します。

参照：「レプリケーション」 215 ページ。

ロー・レベルのトリガ

変更されているローごとに一回実行するトリガです。

参照：

- 「トリガ」 209 ページ
- 「文レベルのトリガ」 223 ページ

ローカル・テンポラリ・テーブル

複合文を実行する間だけ存在したり、接続が終了するまで存在したりするテンポラリ・テーブルです。データのセットを 1 回だけロードする必要がある場合にローカル・テンポラリ・テーブルが便利です。デフォルトでは、COMMIT を実行するとローが削除されます。

参照：

- 「テンポラリ・テーブル」 208 ページ
- 「グローバル・テンポラリ・テーブル」 202 ページ

ロール

概念データベース・モデルで、ある視点からの関係を説明する動詞またはフレーズを指します。各関係は 2 つのロールを使用して表すことができます。"contains (A は B を含む)" や "is a member of (B は A のメンバ)" などのロールがあります。

ロールバック・ログ

コミットされていない各トランザクションの最中に行われた変更のレコードです。ROLLBACK 要求やシステム障害が発生した場合、コミットされていないトランザクションはデータベースから破棄され、データベースは前の状態に戻ります。各トランザクションにはそれぞれロールバック・ログが作成されます。このログは、トランザクションが完了すると削除されます。

参照：「トランザクション」 209 ページ。

ロール名

外部キーの名前です。この外部キーがロール名と呼ばれるのは、外部テーブルとプライマリ・テーブル間の関係に名前を指定するためです。デフォルトでは、テーブル名がロール名になります。ただし、別の外部キーがそのテーブル名を使用している場合、デフォルトのロール名はテーブル名に 3 桁のユニークな数字を付けたものになります。ロール名は独自に作成することもできます。

参照：「外部キー」 217 ページ。

ログ・ファイル

SQL Anywhere によって管理されているトランザクションのログです。ログ・ファイルを使用すると、システム障害やメディア障害が発生してもデータベースを回復させることができます。また、データベースのパフォーマンスを向上させたり、SQL Remote を使用してデータをレプリケートしたりする場合にも使用できます。

参照：

- 「トランザクション・ログ」 209 ページ
- 「トランザクション・ログ・ミラー」 209 ページ
- 「フル・バックアップ」 212 ページ

ロック

複数のトランザクションを同時に実行しているときにデータの整合性を保護する同時制御メカニズムです。SQL Anywhere では、2 つの接続によって同じデータが同時に変更されないようにするために、また変更処理の最中に他の接続によってデータが読み込まれないようにするために、自動的にロックが適用されます。

ロックの制御は、独立性レベルを設定して行います。

参照：

- 「独立性レベル」 223 ページ
- 「同時性 (同時実行性)」 223 ページ
- 「整合性」 220 ページ

ワーク・テーブル

クエリの最適化の最中に中間結果を保管する内部保管領域です。

一意性制約

NULL 以外のすべての値が重複しないことを要求するカラムまたはカラムのセットに対する制限です。テーブルには複数の一意性制約を指定できます。

参照：

- 「外部キー制約」 218 ページ
- 「プライマリ・キー制約」 212 ページ
- 「制約」 220 ページ

解析ツリー

クエリを代数で表現したものです。

外部キー

別のテーブルにあるプライマリ・キーの値を複製する、テーブルの 1 つ以上のカラムです。テーブル間の関係は、外部キーによって確立されます。

参照：

- 「プライマリ・キー」 212 ページ
- 「外部テーブル」 218 ページ

外部キー制約

カラムまたはカラムのセットに対する制約で、テーブルのデータが別のテーブルのデータとどのように関係しているかを指定するものです。カラムのセットに外部キー制約を加えると、それらのカラムが外部キーになります。

参照：

- 「制約」 220 ページ
- 「検査制約」 219 ページ
- 「プライマリ・キー制約」 212 ページ
- 「一意性制約」 217 ページ

外部ジョイン

テーブル内のすべてのローを保護するジョインです。SQL Anywhere では、左外部ジョイン、右外部ジョイン、全外部ジョインがサポートされています。左外部ジョインは JOIN 演算子の左側にあるテーブルのローを保護し、右側にあるテーブルのローがジョイン条件を満たさない場合には NULL を返します。全外部ジョインは両方のテーブルに含まれるすべてのローを保護します。

参照：

- 「ジョイン」 204 ページ
- 「内部ジョイン」 223 ページ

外部テーブル

外部キーを持つテーブルです。

参照：「外部キー」 217 ページ。

外部ログイン

リモート・サーバとの通信に使用される代替のログイン名とパスワードです。デフォルトでは、SQL Anywhere は、クライアントに代わってリモート・サーバに接続するときは、常にそのクライアントの名前とパスワードを使用します。外部ログインを作成することによって、このデフォルトを上書きできます。外部ログインは、リモート・サーバと通信するときに使用する代替のログイン名とパスワードです。

競合

リソースについて対立する動作のことです。たとえば、データベース用語では、複数のユーザがデータベースの同じローを編集しようとした場合、そのローの編集権についての競合が発生します。

競合解決

Mobile Link では、競合解決は 2 人のユーザが別々のリモート・データベースの同じローを変更した場合にどう処理するかを指定するロジックのことです。

検査制約

指定された条件をカラムやカラムのセットに課す制約です。

参照：

- 「制約」 220 ページ
- 「外部キー制約」 218 ページ
- 「プライマリ・キー制約」 212 ページ
- 「一意性制約」 217 ページ

検証

データベース、テーブル、またはインデックスについて、特定のタイプのファイル破損をテストすることです。

作成者 ID

Ultra Light の Palm OS アプリケーションでは、アプリケーションが作成されたときに割り当てられる ID のことです。

参照元オブジェクト

テーブルなどのデータベースの別のオブジェクトをオブジェクト定義が直接参照する、ビューなどのオブジェクトです。

参照：「外部キー」 217 ページ。

参照整合性

データの整合性、特に異なるテーブルのプライマリ・キー値と外部キー値との関係を管理する規則を厳守することです。参照整合性を備えるには、それぞれの外部キーの値が、参照テーブルにあるローのプライマリ・キー値に対応するようにします。

参照：

- 「プライマリ・キー」 212 ページ
- 「外部キー」 217 ページ

参照先オブジェクト

ビューなどの別のオブジェクトの定義で直接参照される、テーブルなどのオブジェクトです。

参照：「プライマリ・キー」 212 ページ。

識別子

テーブルやカラムなどのデータベース・オブジェクトを参照するときに使う文字列です。A ～ Z、a ～ z、0 ～ 9、アンダースコア (_)、アットマーク (@)、シャープ記号 (#)、ドル記号 ($) のうち、任意の文字を識別子として使用できます。

述部

条件式です。オプションで論理演算子 AND や OR と組み合わせて、WHERE 句または HAVING 句に条件のセットを作成します。SQL では、unknown と評価される述部が false と解釈されます。

照合

データベース内のテキストのプロパティを定義する文字セットとソート順の組み合わせのことです。SQL Anywhere データベースでは、サーバを実行しているオペレーティング・システムと言語によって、デフォルトの照合が決まります。たとえば、英語版 Windows システムのデフォルトの照合は 1252LATIN1 です。照合は、照合順とも呼ばれ、文字列の比較とソートに使用します。

参照：

- 「文字セット」 223 ページ
- 「コード・ページ」 203 ページ
- 「エンコード」 201 ページ

世代番号

Mobile Link では、リモート・データベースがデータをアップロードしてからダウンロード・ファイルを適用するようにするためのメカニズムのことです。

参照：「ファイルベースのダウンロード」 211 ページ。

制約

テーブルやカラムなど、特定のデータベース・オブジェクトに含まれた値に関する制約です。たとえば、一意性制約があるカラム内の値は、すべて異なっている必要があります。テーブルに、そのテーブルの情報と他のテーブルのデータがどのように関係しているのかを指定する外部キー制約が設定されていることもあります。

参照：

- 「検査制約」 219 ページ
- 「外部キー制約」 218 ページ
- 「プライマリ・キー制約」 212 ページ
- 「一意性制約」 217 ページ

整合性

データが適切かつ正確であり、データベースの関係構造が保たれていることを保証する規則を厳守することです。

参照：「参照整合性」 219 ページ。

正規化

データベース・スキーマを改善することです。リレーショナル・データベース理論に基づく規則に従って、冗長性を排除したり、編成を改良します。

正規表現

正規表現は、文字列内で検索するパターンを定義する、一連の文字、ワイルドカード、演算子です。

生成されたジョイン条件

自動的に生成される、ジョインの結果に対する制限です。キーとナチュラルの 2 種類があります。キー・ジョインは、KEY JOIN を指定したとき、またはキーワード JOIN を指定したが、CROSS、NATURAL、または ON を使用しなかった場合に生成されます。キー・ジョインの場合、生成されたジョイン条件はテーブル間の外部キー関係に基づいています。ナチュラル・ジョインは NATURAL JOIN を指定したときに生成され、生成されたジョイン条件は、2 つのテーブルの共通のカラム名に基づきます。

参照：

- 「ジョイン」 204 ページ
- 「ジョイン条件」 205 ページ

接続 ID

クライアント・アプリケーションとデータベース間の特定の接続に付けられるユニークな識別番号です。現在の接続 ID を確認するには、次の SQL 文を使用します。

```
SELECT CONNECTION_PROPERTY( 'Number' );
```

接続プロファイル

ユーザ名、パスワード、サーバ名などの、データベースに接続するために必要なパラメータのセットです。便宜的に保管され使用されます。

接続起動同期

Mobile Link のサーバ起動同期の 1 つの形式で、接続が変更されたときに同期が開始されます。

参照：「サーバ起動同期」 203 ページ。

相関名

クエリの FROM 句内で使用されるテーブルやビューの名前です。テーブルやビューの元の名前か、FROM 句で定義した代替名のいずれかになります。

抽出

SQL Remote レプリケーションでは、統合データベースから適切な構造とデータをアンロードする動作を指します。この情報は、リモート・データベースを初期化するときに使用されます。

参照：「レプリケーション」 215 ページ。

通信ストリーム

Mobile Link では、Mobile Link クライアントと Mobile Link サーバ間での通信にネットワーク・プロトコルが使用されます。

転送ルール

QAnywhere では、メッセージの転送を発生させる時期、転送するメッセージ、メッセージを削除する時期を決定する論理のことです。

統合データベース

分散データベース環境で、データのマスタ・コピーを格納するデータベースです。競合や不一致が発生した場合、データのプライマリ・コピーは統合データベースにあるとみなされます。

参照：

- 「同期」 222 ページ
- 「レプリケーション」 215 ページ

統合化ログイン

オペレーティング・システムへのログイン、ネットワークへのログイン、データベースへの接続に、同一のユーザ ID とパスワードを使用するログイン機能の 1 つです。

動的 SQL

実行される前に作成したプログラムによって生成される SQL です。Ultra Light の動的 SQL は、占有容量の小さいデバイス用に設計された変形型です。

同期

Mobile Link テクノロジを使用してデータベース間でデータをレプリケートする処理です。

SQL Remote では、同期はデータの初期セットを使ってリモート・データベースを初期化する処理を表すために特に使用されます。

参照：

- 「Mobile Link」 195 ページ
- 「SQL Remote」 198 ページ

同時性 (同時実行性)

互いに独立し、場合によっては競合する可能性のある 2 つ以上の処理を同時に実行することです。SQL Anywhere では、自動的にロックを使用して各トランザクションを独立させ、同時に稼働するそれぞれのアプリケーションが一貫したデータのセットを参照できるようにします。

参照：

- 「トランザクション」 209 ページ
- 「独立性レベル」 223 ページ

独立性レベル

あるトランザクションの操作が、同時に処理されている別のトランザクションの操作からどの程度参照できるかを示します。独立性レベルには 0 から 3 までの 4 つのレベルがあります。最も高い独立性レベルには 3 が設定されます。デフォルトでは、レベルは 0 に設定されています。SQL Anywhere では、スナップショット、文のスナップショット、読み込み専用文のスナップショットの 3 つのスナップショットの独立性レベルがあります。

参照：「スナップショット・アイソレーション」 205 ページ。

内部ジョイン

2 つのテーブルがジョイン条件を満たす場合だけ、結果セットにローが表示されるジョインです。内部ジョインがデフォルトです。

参照：

- 「ジョイン」 204 ページ
- 「外部ジョイン」 218 ページ

物理インデックス

インデックスがディスクに保存されるときの実際のインデックス構造です。

文レベルのトリガ

トリガ付きの文の処理が完了した後に実行されるトリガです。

参照：

- 「トリガ」 209 ページ
- 「ロー・レベルのトリガ」 216 ページ

文字セット

文字セットは記号、文字、数字、スペースなどから成ります。"ISO-8859-1" は文字セットの例です。Latin1 とも呼ばれます。

参照：

- 「コード・ページ」 203 ページ
- 「エンコード」 201 ページ
- 「照合」 220 ページ

文字列リテラル

文字列リテラルとは、一重引用符 (') で囲まれ、シーケンスで並べられた文字のことです。

論理インデックス

物理インデックスへの参照 (ポインタ) です。ディスクに保存される論理インデックス用のインデックス構造はありません。

索引

記号

A

B

C

J

L

M

N

O

P

Q

R

S

U

V

W

あ

い

う

え

お

か

き

す

せ

そ

た

ち

つ

て

と

な

に

ね

は

ひ

ふ

へ

ほ

ま

み

め

も

ゆ

よ

り

れ

ろ

わ