SYBASE iAnywhere

SQL Remote®

バージョン 12.0.1

2012 年 1 月

バージョン 12.0.1
2012 年 1 月

目次

はじめに

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

SQL Remote システム

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ーのサポート**　SQL Remote を使用すると、不定期に接続するユーザーが、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、エンドユーザーが 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ーとデバイス上で 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ー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。

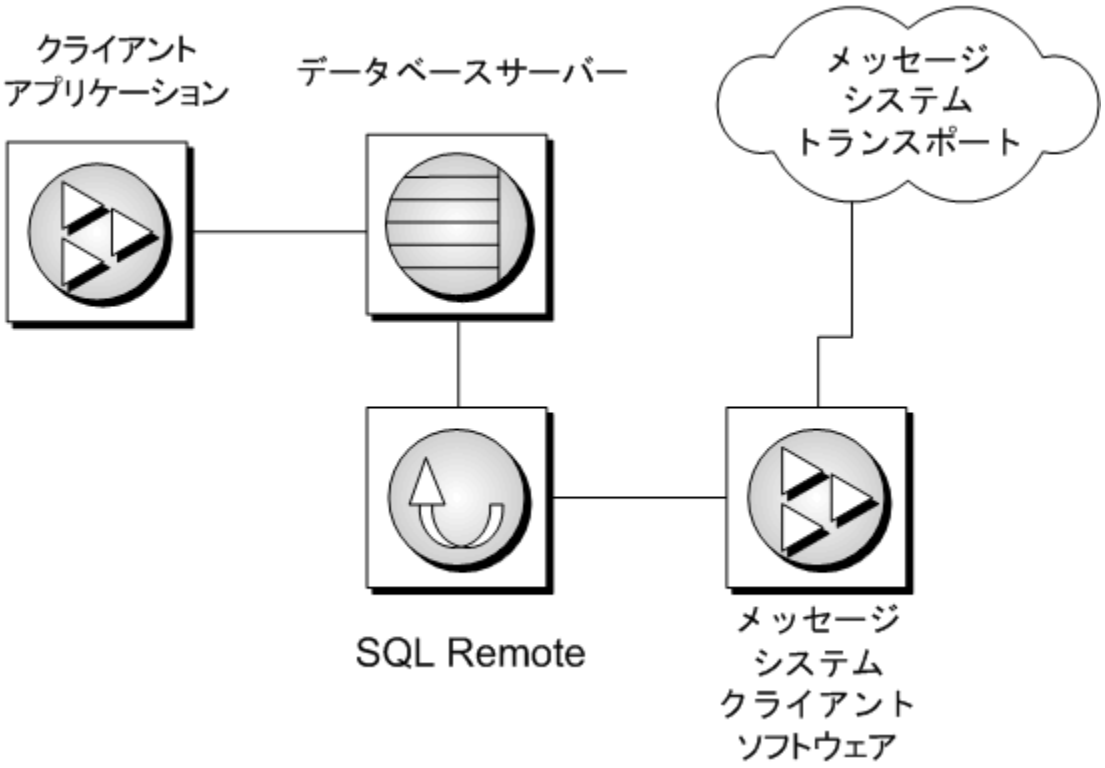
- **マルチプラットフォームサポート**　SQL Remote は、さまざまなオペレーティングシステムとメッセージリンクでサポートされています。SQL Anywhere データベースは、1 つのファイル／オペレーティングシステムから、別のファイル／オペレーティングシステムにコピーできます。

参照

- http://www.ianywhere.jp/tech/1061806-os_components.html
- 「SQL Remote システムの作成」 9 ページ
- 「SQL Remote システムの管理」 75 ページ
- 「SQL Remote のリファレンス」 183 ページ

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ー**　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／サーバー接続経由でデータベースサーバーに接続します。SQL Remote Message Agent は、データベースサーバーと同じコンピューターでも異なるコンピューターでも実行できます。

- **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ーフェイスを使用できます。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 のプログラミングインターフェイスの詳細については、次のリストを参照してください。

- 「SQL Anywhere の .NET サポート」『SQL Anywhere サーバー プログラミング』
- 「ODBC サポート」『SQL Anywhere サーバー プログラミング』
- 「OLE DB と ADO の開発」『SQL Anywhere サーバー プログラミング』
- 「Embedded SQL」『SQL Anywhere サーバー プログラミング』
- 「JDBC サポート」『SQL Anywhere サーバー プログラミング』
- 「Sybase Open Client のサポート」『SQL Anywhere サーバー プログラミング』
- 「SQL Anywhere C API のサポート」『SQL Anywhere サーバー プログラミング』
- 「Perl DBI サポート」『SQL Anywhere サーバー プログラミング』
- 「SQL Anywhere PHP 拡張」『SQL Anywhere サーバー プログラミング』
- 「Python サポート」『SQL Anywhere サーバー プログラミング』
- 「SQL Anywhere の Ruby API サポート」『SQL Anywhere サーバー プログラミング』

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。

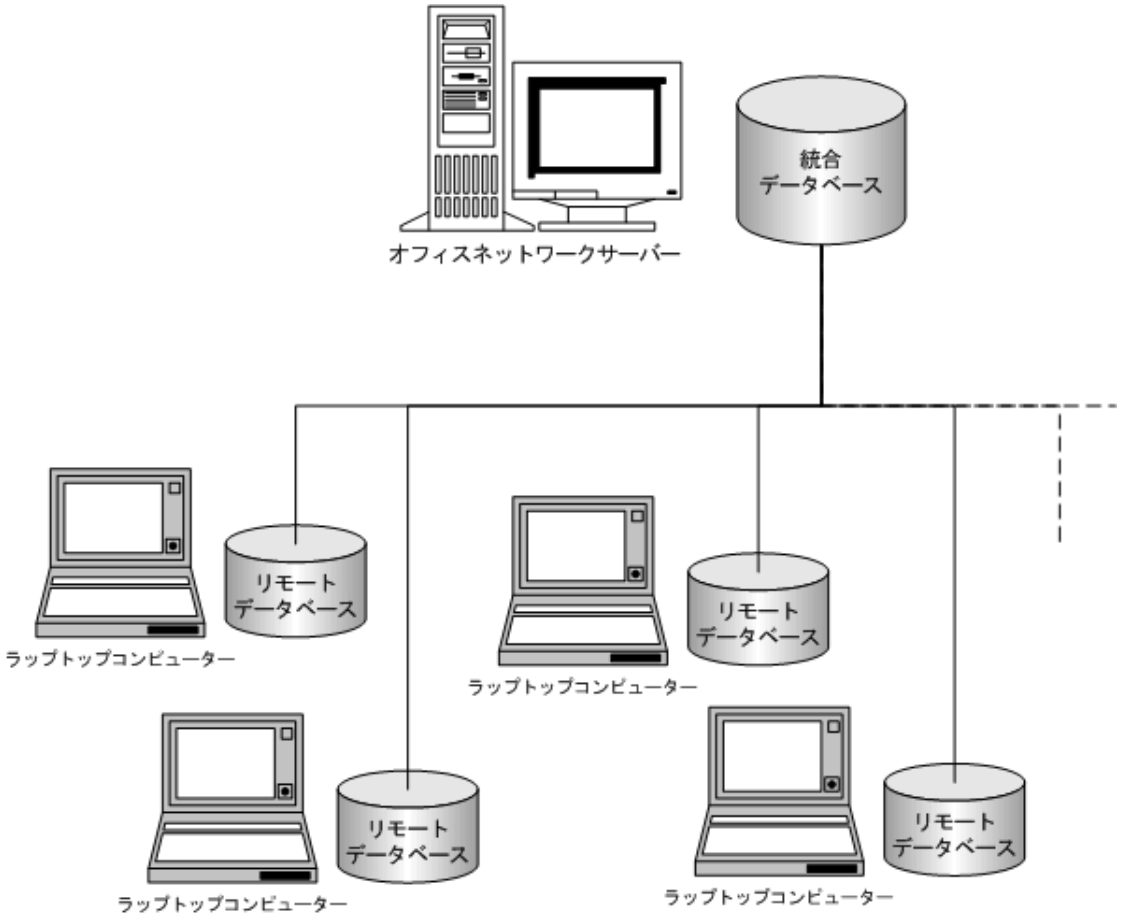
- **多数のリモートデータベース**　多数のリモートデータベースへのメッセージを同時に準備できるため、1 つのインストール環境で、数千のリモートデータベースをサポートできます。
- **随時接続**　SQL Remote は、ネットワークに時々または間接的に接続されるデータベースをサポートします。SQL Remote は、各サイトのデータを常に最新に保つような設計にはなっていません。たとえば、SMTP 電子メールシステムを使用して、レプリケーションを実行することがあります。
- **遅延時間 : 短～長**　遅延時間が長いというのは、システムにおいて、あるデータベースにデータが入力されてから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。

参照

- 「同期テクノロジの比較」『SQL Anywhere 12 紹介』
- 「同期テクノロジの注意事項」『SQL Anywhere 12 紹介』

モバイル環境でのサーバー／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ー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ーで SQL Anywhere データベースサーバーを実行します。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ーには、SQL Anywhere パーソナルサーバー、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ー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ー上の統合データベースに送信する。

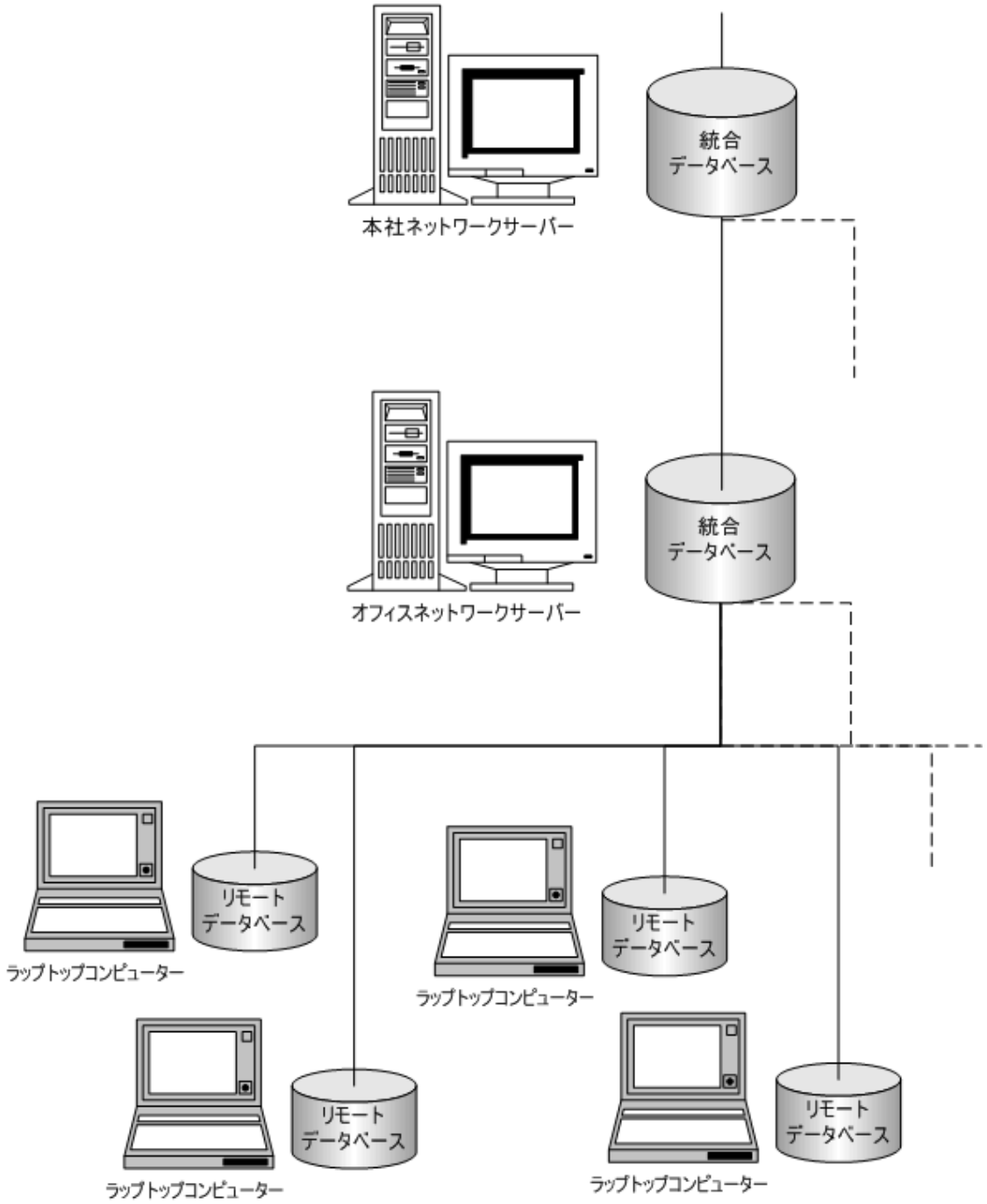
オフィスネットワークデータベースのパブリケーションの更新には、その営業担当者が取り扱っている製品の新しい特別割引や、新価格、在庫情報などがあります。これらの更新はラップトップ上の SQL Remote によって読み込まれ、自動的に営業担当者のリモートデータベースに適用されます。営業担当者による追加操作は一切不要です。

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ー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ーと本社のデータベースサーバー間で、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、サーバーの初期設定と継続したメンテナンスのみです。

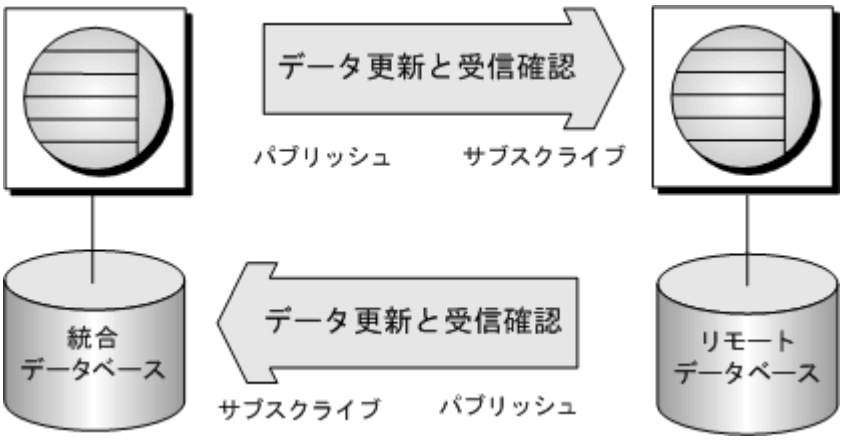
SQL Remote の階層には、レイヤーを追加できます。たとえば、各営業所のサーバーを統合データベースとして利用して、その営業所からのリモートサブスクライバーをサポートすることができます。



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。

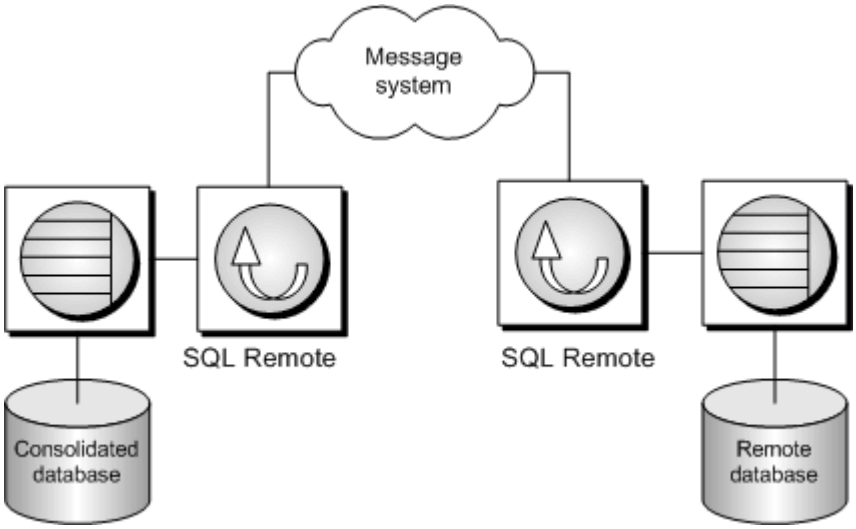
SQL Remote レプリケーションプロセス

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ー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 SQL Remote 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 SQL Remote 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ーに、関連する変更を送信します。SQL Remote 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 SQL Remote 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、SQL Remote 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. リモートユーザーは、いつでも SQL Remote 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 SQL Remote 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。

参照

- 「SQL Remote システムの作成」9 ページ
- 「SQL Remote システムの管理」75 ページ

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。以下に、SQL Remote システムを作成するために実施すべき手順の概要を示します。

♦ SQL Remote システムを作成する

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、SQL Remote がどのようにプライマリキーを使用するのかを念頭に置く必要があります (リモートデータベースが統合データベースをレプリケートする場合、プライマリキーが重複する可能性があります)。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。

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する必要があります。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ー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。

4. 統合データベースにパブリッシャーユーザーを作成します。

   パブリッシャーは PUBLISH 権限を持つユーザーです。

5. 統合データベースにリモートユーザーを作成します。

   リモートユーザーを使用して、リモートデータベースをユニークに識別します。

   リモートユーザーを作成する場合は、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ーをサブスクライブします。

7. リモートユーザーがデータをどのように使用するかを決定します。

   リモートユーザーは、自分のデータを常に読み込むことができます。また、リモートユーザーにはデータの更新、削除、挿入も許可できます。

8. 競合を解決する方法を選択します。

リモートユーザーがデータを更新、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。

9. SQL Remote システムを配備します。

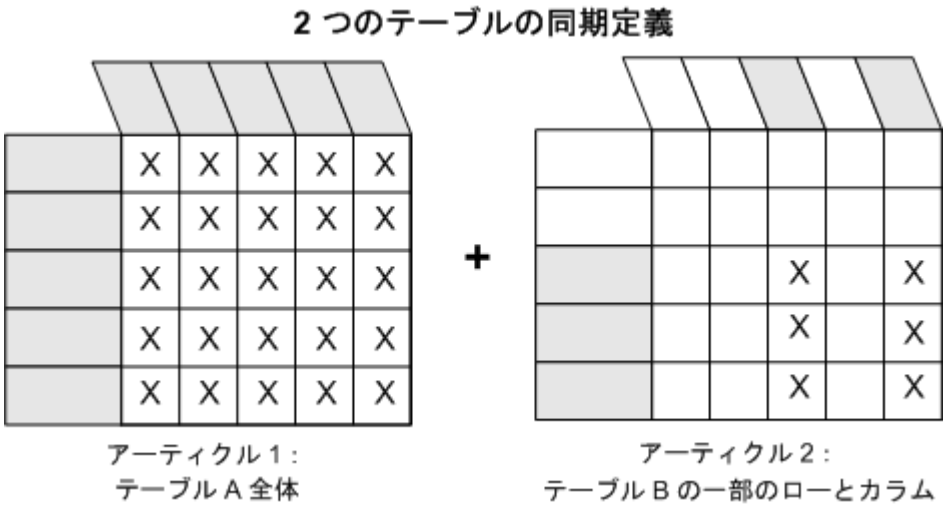
   リモートデータベースを作成して、適切なソフトウェアをインストールします。

参照

- 「重複プライマリキーエラー」53 ページ
- 「パブリケーションとアーティクル」10 ページ
- 「PUBLISH パーミッション」22 ページ
- 「REMOTE パーミッション」24 ページ
- 「メッセージタイプの作成」106 ページ
- 「サブスクリプション」32 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」34 ページ
- 「更新の競合に対するデフォルトの解決」44 ページ
- 「SQL Remote システムの管理」75 ページ

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ーを含めることができません。

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]** フォルダーに表示されます。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。

参照

- 「プロシージャーのレプリケーション」38 ページ
- 「トリガーのレプリケーション」38 ページ

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 (Sybase Central の場合)

1. **[SQL Anywhere 12]** プラグインを使用して、DBA 権限のあるユーザーとして、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]** フォルダーをクリックします。
3. **[ファイル]** » **[新規]** » **[パブリケーション]** をクリックします。
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クリックします。**[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 (SQL の場合)

1. DBA 権限のあるユーザー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。

   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
);
```

参照

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

テーブル内の一部のカラムだけ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 (Sybase Central の場合)

1. **[SQL Anywhere 12]** プラグインを使用して、DBA 権限のあるユーザーとして、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]** フォルダーを展開します。
3. **[ファイル]** » **[新規]** » **[パブリケーション]** をクリックします。
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クリックします。**[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クリックし、**[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のみをパブリッシュする (SQL の場合)

1. DBA 権限のあるユーザー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 サーバー SQL リファレンス』
- 「参照整合性エラー」51 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ーが、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。サブスクライバーが受信するローは、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。

- **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ーは、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ーは、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。

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;
```

参照

- 「SUBSCRIBE BY 句を使用して一部のローのみをパブリッシュする」 14 ページ
- 「WHERE 句を使用して一部のローだけをパブリッシュする」 16 ページ
- 「CREATE PUBLICATION 文 [Mobile Link] [SQL Remote]」『SQL Anywhere サーバー SQL リファレンス』
- 「CREATE SUBSCRIPTION 文 [SQL Remote]」『SQL Anywhere サーバー 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

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」 13 ページを参照してください。

♦ SUBSCRIBE BY 句を使用してパブリケーションを作成する (Sybase Central の場合)

1. **[SQL Anywhere 12]** プラグインを使用して、DBA 権限のあるユーザーとして、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]** フォルダーをクリックします。
3. **[ファイル]** » **[新規]** » **[パブリケーション]** をクリックします。

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. In the**[使用可能なテーブル]** リストでテーブルをクリックします。**[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クリックし、**[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。
   b. **[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クリックします。
10. **[完了]** をクリックします。

♦ SUBSCRIBE BY 句を使用してパブリケーションを作成する (SQL の場合)

1. DBA 権限のあるユーザー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

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ーを一致させます。

```
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;
```

ユーザーは、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だけをパブリッシュする」16 ページ
- 「データ分割の切断」60 ページ
- 「CREATE PUBLICATION 文 [Mobile Link] [SQL Remote]」『SQL Anywhere サーバー SQL リファレンス』
- 「CREATE SUBSCRIPTION 文 [SQL Remote]」『SQL Anywhere サーバー 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」32 ページ

WHERE 句を使用して一部のローだけをパブリッシュする

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」13 ページを参照してください。

♦ WHERE 句を使用したパブリケーションの作成 (Sybase Central の場合)

1. **[SQL Anywhere 12]** プラグインを使用して、DBA 権限のあるユーザーとして、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]** フォルダーをクリックします。
3. **[ファイル]** » **[新規]** » **[パブリケーション]** をクリックします。
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. In the**[使用可能なテーブル]** リストでテーブルをクリックします。**[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クリックし、**[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 句を使用したパブリケーションの作成 (SQL の場合)

1. DBA 権限のあるユーザーとして、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;
   ```

ユーザーは、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 句とプライマリキー」 50 ページ
- 「CREATE PUBLICATION 文 [Mobile Link] [SQL Remote]」『SQL Anywhere サーバー SQL リファレンス』
- 「CREATE SUBSCRIPTION 文 [SQL Remote]」『SQL Anywhere サーバー SQL リファレンス』

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」 135 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 (Sybase Central の場合)

1. **[SQL Anywhere 12]** プラグインを使用して、パブリケーションを所有するユーザー、または DBA 権限を持つユーザー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]** フォルダーをクリックします。
3. 変更するアーティクルを右クリックし、**[プロパティ]** をクリック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 (SQL の場合)

1. パブリケーションを所有するユーザー、または DBA 権限を持つユーザー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 サーバー 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」 135 ページを参照してください。

♦ パブリケーションの削除 (Sybase Central の場合)

1. **SQL Anywhere 12** プラグインを使用して、DBA 権限のあるユーザーとして、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]** フォルダーを展開します。
3. 目的のパブリケーションを右クリックして、**[削除]** をクリックします。

♦ パブリケーションの削除 (SQL の場合)

1. DBA 権限のあるユーザーとして、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;
   ```

参照

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

ユーザーパーミッション

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ーを管理するための、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ーは、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ーが必要です。パブリッシャーを作成するには、1 人のユーザーに PUBLISH パーミッションを付与します。パブリッシャーユーザーは、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、パブリッシャー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ーユーザー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ー名によって、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ーに 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ーに 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、リモート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ーに付与されます。

参照

- 「抽出ユーティリティ (dbxtract)」 194 ページ

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ーに 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、統合データベースのパブリッシャーに CONSOLIDATE パーミッションを付与します。

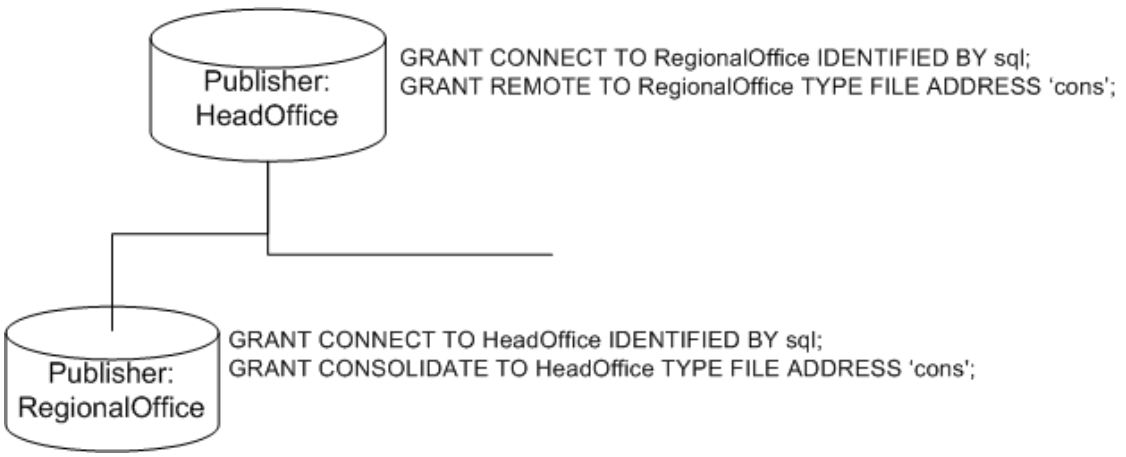
たとえば、統合データベースのパブリッシャーによって識別される統合データベース HeadOffice と、リモートデータベースのパブリッシャーによって識別されるリモートデータベース RegionalOffice があるとします。

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 のパブリッシャーと同じ名前のユーザーを作成します。
- 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。

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 のパブリッシャーと同じ名前のユーザーを作成します。
- 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。



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ーに付与されます。

参照

- 「抽出ユーティリティ (dbxtract)」194 ページ

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ー 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。

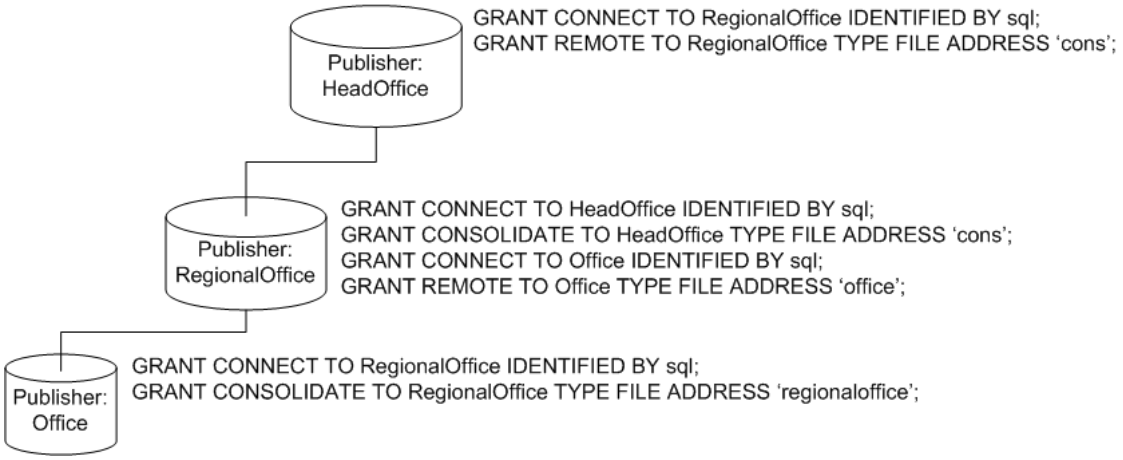
- リモートデータベース RegionalOffice のパブリッシャーと同じ名前のユーザーを作成します。
- ユーザー RegionalOffice に REMOTE パーミッションを付与します。このことによって、RegionalOffice が HeadOffice からメッセージを受信するデータベースとして識別されます。

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 のパブリッシャーと同じ名前のユーザーを作成します。
- 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ーを作成します。
- 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 のパブリッシャーと同じ名前のユーザーを作成します。
- RegionalOffice ユーザーに 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、パブリッシャーが必要です。パブリッシャーは、PUBLISH パーミッションを持つユニークなユーザーです。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、パブリッシャー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ー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ー名に 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ーに継承されません。

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

パブリッシャーの作成

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

♦ パブリッシャーを作成する (Sybase Central の場合)

1. **SQL Anywhere 12** プラグインを使用して、DBA 権限のあるユーザーとして、データベースに接続します。
2. パブリッシャーの役割を果たすユーザーを作成します (まだ存在しない場合)。
   a. 左ウィンドウ枠で、**[ユーザーとグループ]** フォルダーをクリックします。
   b. **[ファイル]** » **[新規]** » **[ユーザー]** をクリックします。
   c. **ユーザー作成ウィザード**の指示に従います。ユーザーにパスワードを割り当てます。
3. **[ユーザーとグループ]** フォルダーで、作成したユーザーを右クリックして **[パブリッシャーに変更]** をクリックします。

♦ パブリッシャーを作成する (SQL の場合)

1. DBA 権限のあるユーザーとして、データベースに接続します。
2. CREATE USER 文を実行して、パブリッシャーの役割を果たすユーザーを作成します。ユーザー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ーを作成します。

   ```
   CREATE USER cons IDENTIFIED BY SQL;
   ```

3. GRANT CONNECT 文を実行して、ユーザーに CONNECT パーミッションを付与します。

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

   たとえば、次の文では、ユーザー 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、リモートユーザーがリモートデータベースのパブリッシャーになり、PUBLISH パーミッションが付与されます。

参照

- 「GRANT 文」『SQL Anywhere サーバー SQL リファレンス』
- 「GRANT PUBLISH 文 [SQL Remote]」『SQL Anywhere サーバー SQL リファレンス』
- 「PUBLISH パーミッションの取り消し」23 ページ
- 「パブリッシャーの表示」24 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、ユーザーから 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ーを変更すると、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」136 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 (Sybase Central の場合)

1. **SQL Anywhere 12** プラグインを使用して、DBA 権限のあるユーザーとして、データベースに接続します。
2. 左ウィンドウ枠で、**[ユーザーとグループ]** フォルダーをクリックします。

3. PUBLISH パーミッションを持つユーザーを右クリックして、**[パブリッシャーの取り消し]** をクリックします。

♦ PUBLISH パーミッションを取り消す (SQL の場合)

1. DBA 権限のあるユーザーとして、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ーから PUBLISH パーミッションを取り消します。

   次に例を示します。

   ```
   REVOKE PUBLISH FROM cons;
   ```

参照

- 「PUBLISH パーミッション」 22 ページ
- 「REVOKE PUBLISH 文 [SQL Remote]」『SQL Anywhere サーバー SQL リファレンス』
- 「パブリッシャーの表示」 24 ページ

パブリッシャーの表示

♦ パブリッシャーを確認する (Sybase Central の場合)

- 左ウィンドウ枠で、**[ユーザーとグループ]** フォルダーをクリックします。

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

♦ パブリッシャーを確認する (SQL の場合)

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

  ```
  SELECT CURRENT PUBLISHER;
  ```

参照

- 「PUBLISH パーミッション」 22 ページ
- 「PUBLISH パーミッションの取り消し」 23 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ー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ーには、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。

ユーザーに 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ーを作成することはできません。

- **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。

ユーザーに 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。

●ユーザーをリモートユーザーとして識別します。

●このリモートユーザー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ー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ーは、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、パブリッシャーが、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ーに自動的には適用されません。グループ内の各ユーザーには REMOTE パーミッションを明示的に付与してください。

参照

- 「SQL Remote メッセージシステム」105 ページ
- 「送信頻度の設定」86 ページ
- 「GRANT REMOTE 文 [SQL Remote]」『SQL Anywhere サーバー SQL リファレンス』

REMOTE パーミッションの付与

次の手順を使用して、リモートユーザーを追加します。

♦ リモートユーザーを作成する (Sybase Central の場合)

1. **SQL Anywhere 12** プラグインを使用して、DBA 権限のあるユーザーとして、データベースに接続します。
2. 左ウィンドウ枠で、**[SQL Remote ユーザー]** フォルダーをクリックします。
3. **[ファイル]** » **[新規]** » **[SQL Remote ユーザー]** をクリックします。
4. **リモートユーザー作成ウィザード**の指示に従います。

♦ 既存のユーザーをリモートユーザーにする (Sybase Central の場合)

1. **SQL Anywhere 12** プラグインを使用して、DBA 権限のあるユーザーとして、データベースに接続します。
2. 左ウィンドウ枠で、**[SQL Remote ユーザー]** フォルダーをクリックします。

3. ユーザーを右クリックして、**[リモートユーザーに変更]** をクリックします。
4. このウィンドウで、メッセージタイプのクリック、アドレスの入力、送信頻度のクリックを行い、**[OK]** をクリックします。

♦ リモートユーザーを作成する (SQL の場合)

1. DBA 権限のあるユーザーとして、データベースに接続します。
2. CREATE USER 文を実行して、ユーザーを作成します。

   次に例を示します。

   ```
   CREATE USER remote1;
   ```

3. GRANT CONNECT 文を実行して、ユーザーの CONNECT パーミッションを付与します。

   次に例を示します。

   ```
   GRANT CONNECT TO remote1;
   ```

4. GRANT REMOTE 文を実行して、ユーザーの REMOTE パーミッションを付与します。

   次に例を示します。

   ```
   GRANT REMOTE TO userid;
   ```

   たとえば、次の文では、ユーザー 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、ユーザー 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

- 「GRANT 文」『SQL Anywhere サーバー SQL リファレンス』
- GRANT REMOTE 文 [SQL Remote]
- 「REMOTE パーミッションの取り消し」27 ページ

REMOTE パーミッションの取り消し

ユーザーの 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ーを削除します。

●ユーザーまたはグループを通常のユーザー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、ユーザー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 (Sybase Central の場合)

1. **[SQL Anywhere 12]** プラグインを使用して、DBA 権限のあるユーザーとして、統合データベースに接続します。
2. 左ウィンドウ枠で、**[ユーザーとグループ]** フォルダーまたは **[SQL Remote ユーザー]** フォルダーのいずれかを展開します。
3. リモートユーザーまたはグループを右クリックして、**[リモートの取り消し]** をクリックします。

♦ REMOTE パーミッションを取り消す (SQL の場合)

1. DBA 権限のあるユーザーとして、データベースに接続します。
2. REVOKE REMOTE 文を実行して、現在のユーザーまたはグループの REMOTE パーミッションを取り消します。

   たとえば、次の文では、ユーザー 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」25 ページ
- 「REVOKE REMOTE 文 [SQL Remote]」『SQL Anywhere サーバー SQL リファレンス』

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。

ユーザーに 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、新規統合ユーザーを作成することはできません。
- **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。

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。

●ユーザーを統合ユーザーとして識別します。

●この統合ユーザー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ー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ーは、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、パブリッシャーが、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。

参照

- 「SQL Remote メッセージシステム」105 ページ
- 「送信頻度の設定」86 ページ

CONSOLIDATE パーミッションの付与

次の手順を使用して、ユーザーに 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ーに 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 (Sybase Central の場合)

1. **SQL Anywhere 12** プラグインを使用して、DBA 権限のあるユーザーとして、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をクリックして、**[ファイル]** » **[プロパティ]** をクリックします。
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]** をクリックします。
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 (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ーに CONNECT パーミッションを付与します。

   次に例を示します。

   ```
   GRANT CONNECT TO cons;
   ```

2. GRANT CONSOLIDATE 文を実行して、統合ユーザーに CONSOLIDATE パーミッションを付与します。

   たとえば、次の文では、hq_user ユーザーに 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 ユーザーに 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

- 「GRANT 文」『SQL Anywhere サーバー SQL リファレンス』
- 「GRANT CONSOLIDATE 文 [SQL Remote]」『SQL Anywhere サーバー SQL リファレンス』
- 「CONSOLIDATE パーミッションの取り消し」29 ページ

CONSOLIDATE パーミッションの取り消し

ユーザーの 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ーを削除します。

●ユーザーまたはグループを通常のユーザー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、ユーザーまたはグループのサブスクリプションを削除します。

♦ CONSOLIDATE パーミッションを取り消す (Sybase Central の場合)

1. **SQL Anywhere 12** プラグインを使用して、DBA 権限のあるユーザーとして、データベースに接続します。
2. **[ユーザーとグループ]** フォルダーまたは **[SQL Remote ユーザー]** フォルダーを展開します。
3. 統合ユーザーまたはグループを右クリックして、**[統合の取り消し]** をクリックします。
4. **[はい]** をクリックします。

♦ CONSOLIDATE パーミッションを取り消す (SQL の場合)

1. DBA 権限のあるユーザーとして、データベースに接続します。
2. REVOKE CONSOLIDATE 文を実行して、現在のユーザーまたはグループから CONSOLIDATE パーミッションを取り消します。

   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」28 ページ
- 「REVOKE CONSOLIDATE 文 [SQL Remote]」『SQL Anywhere サーバー SQL リファレンス』

REMOTE DBA 権限

REMOTE DBA 権限を持つユーザー名は、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 権限のあるユーザーのみです。

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ーは、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 ユーザーのユーザー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ー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 権限を持つユーザーには、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ーを作成するときに、統合データベースのパブリッシャーと各リモートユーザーに 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、リモートユーザーがリモートデータベースのパブリッシャーになり、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、ユーザーの管理が簡単になります。SQL Remote から (ユーザーに完全な 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 権限ではユーザーに追加のパーミッションを付与しません) に関わらず、各リモートユーザーにはデータベースに接続するための 1 つのユーザー名のみが必要です。

参照

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

REMOTE DBA 権限の付与

♦ REMOTE DBA 権限を付与する (Sybase Central の場合)

1. **SQL Anywhere 12** プラグインを使用して、DBA 権限のあるユーザーとして、データベースに接続します。
2. 左ウィンドウ枠で、**[ユーザーとグループ]** フォルダーまたは **[SQL Remote ユーザー]** フォルダーのいずれかをクリックします。
3. ユーザーを右クリックして、**[プロパティ]** をクリックします。
4. **[権限]** タブをクリックして、**[リモート DBA]** オプションをクリックします。
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 (SQL の場合)

1. DBA 権限のあるユーザーとして、データベースに接続します。
2. GRANT REMOTE DBA 文を実行して、ユーザーに 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 サーバー データベース管理』
- 「GRANT REMOTE DBA 文 [Mobile Link] [SQL Remote]」『SQL Anywhere サーバー 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、ユーザーは 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。

次の手順を使用して、ユーザーの REMOTE DBA 権限を取り消します。

♦ REMOTE DBA 権限を取り消す (Sybase Central の場合)

1. **SQL Anywhere 12** プラグインを使用して、DBA 権限のあるユーザーとして、データベースに接続します。
2. 左ウィンドウ枠で、**[ユーザーとグループ]** フォルダーまたは **[SQL Remote ユーザー]** フォルダーのいずれかをクリックします。
3. ユーザーを右クリックして、**[プロパティ]** をクリックします。
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 (SQL の場合)

1. DBA 権限のあるユーザーとして、データベースに接続します。
2. REVOKE REMOTE DBA 文を実行して、ユーザーの 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

- 「REMOTE DBA 権限」30 ページ
- 「REVOKE REMOTE DBA 文 [SQL Remote]」『SQL Anywhere サーバー SQL リファレンス』
- 「REMOTE パーミッションの付与」25 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ー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ー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ーをサブスクライブするには、次の情報が必要です。

- **パブリケーション名**　ユーザー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 SUBSCRIBE 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ーの従業員 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。
- **Subscriber-id**　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ー。統合データベースでは、サブスクリプションをリモートユーザーに作成する場合、リモートユーザーには REMOTE パーミッションが付与されている必要があります。リモートデータベースでは、サブスクリプションを統合ユーザーに作成する場合、そのユーザーには CONSOLIDATED パーミッションが付与されている必要があります。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ーに付与されます。

♦ パブリケーションに対してユーザーをサブスクライブする (Sybase Central の場合)

1. **SQL Anywhere 12** プラグインを使用して、DBA 権限のあるユーザーとして、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]** フォルダーをクリックします。
3. パブリケーションをクリックします。
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]** をクリックします。
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ーをサブスクライブする (SQL の場合)

1. DBA 権限のあるユーザーとして、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ー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ー名 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ー名 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 サーバー SQL リファレンス』
- 「GRANT REMOTE 文 [SQL Remote]」『SQL Anywhere サーバー 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」 14 ページ

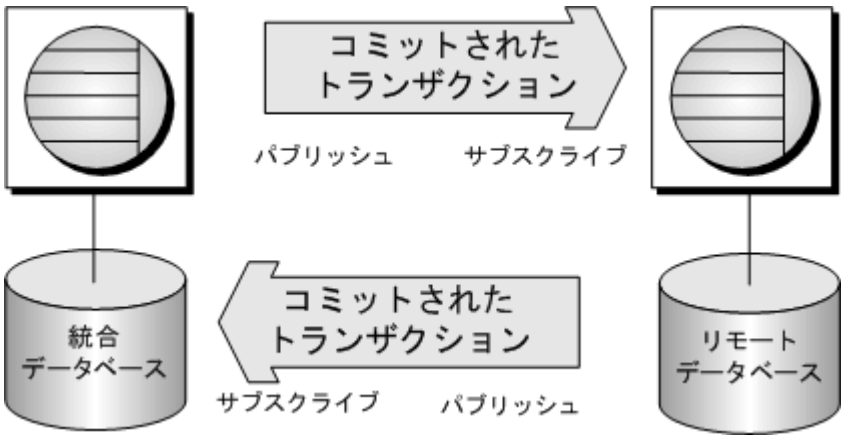
トランザクションログベースのレプリケーション

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ブされた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ー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ーは、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ー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ー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、リモートユーザーごとに適切なメッセージを生成します。

参照

- 「SQL Remote Message Agent (dbremote)」 89 ページ
- 「トランザクションログ」『SQL Anywhere サーバー データベース管理』

INSERT および DELETE 文のレプリケーション

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ー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」 53 ページ
- 「ローが見つからないエラー」 50 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ー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ーに送信されます。
- UPDATE 文は、いずれかのリモートユーザー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ーに送信されます。
- UPDATE 文は、いずれかのリモートユーザー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ーに送信されます。

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ー 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

ID	Rep	Dept
1	Ann	101
2	Marc	101
3	Marc	101
4	Rob	102

>

Ann

ID	Rep
1	Ann

Marc

ID	Rep
2	Marc
3	Marc

ManagerSteve

ID	Rep
1	Ann
2	Marc
3	Marc

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 人のユーザーをサブスクライブします。

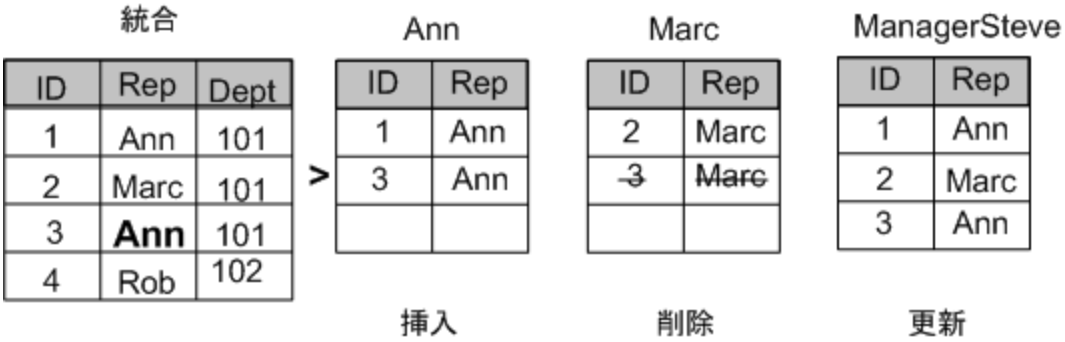
```
CREATE SUBSCRIPTION
  TO "cons"."p1"( 'Ann' )
  FOR "Ann";
CREATE SUBSCRIPTION
  TO "cons"."p1"( 'Marc' )
  FOR "Marc";
CREATE SUBSCRIPTION
  TO "cons"."p1"( 'Ann' )
  FOR "ManagerSteve";
CREATE SUBSCRIPTION
  TO "cons"."p1"( 'Marc' )
  FOR "ManagerSteve";
```

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。



サブスクライバー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」43 ページ
- 「UPDATE 文」『SQL Anywhere サーバー SQL リファレンス』

プロシージャーのレプリケーション

SQL Remote は、プロシージャーをレプリケートするときに、プロシージャーの動作をレプリケートします。プロシージャーコールはレプリケートされません。代わりに、プロシージャーの個々の動作 (INSERT 文、UPDATE 文、DELETE 文) がレプリケートされます。

トリガーのレプリケーション

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ーと同じトリガーが定義されています。

デフォルトでは、SQL Remote はトリガー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ー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、その複製トリガー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 トリガーのレプリケーション**　競合解析または RESOLVE UPDATE トリガーが実行する動作は、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。

- **BEFORE トリガーのレプリケーション**　更新中のローを変更する BEFORE トリガーの動作は、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ーカラムをロー内で増加させる BEFORE UPDATE トリガーは、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 トリガー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 トリガーは、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、サブスクライバーデータベースに BEFORE UPDATE トリガーが存在しないこと、または BEFORE UPDATE トリガー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。

トリガー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ーの動作をすべてレプリケートするには、SQL Remote Message Agent (dbremote) の -t オプションを使用します。

-t オプションを使用する場合は、トリガー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ー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ーを起動するとき) 実行されないようにします。

トリガーの動作が 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。

●IF CURRENT REMOTE USER IS NULL ...END IF 文で、トリガーの本文を囲みます。

- SQL Remote ユーザー名に対して、SQL Anywhere の fire_triggers オプションを Off に設定します。

トリガーエラーの回避

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ーは、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。このようなトリガー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ー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、条件付きのトリガーアクションにします。
- NULL 値を返さない object_id 関数を使用して、条件付きのトリガーアクションにします。object_id 関数は、テーブルやその他のオブジェクトを引数として取得し、そのオブジェクトの ID 番号か NULL 値 (オブジェクトが存在しない場合) を返します。
- ローが存在するかどうか決定する SELECT 文を使用して、条件付きのトリガー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**によってトリガー定義が抽出されます。

参照

- 「更新の競合に対するデフォルトの解決」44 ページ
- 「CURRENT REMOTE USER 特別値の使用」47 ページ
- 「SQL Remote Message Agent ユーティリティ (dbremote)」183 ページ
- 「fire_triggers オプション」『SQL Anywhere サーバー データベース管理』
- 「CURRENT PUBLISHER 特別値」『SQL Anywhere サーバー SQL リファレンス』
- 「システム関数」『SQL Anywhere サーバー SQL リファレンス』

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。

参照

- 「SQL Remote のパススルーモード」137 ページ

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 サーバー 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。

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 オプション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

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。

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。ユーザー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 サーバー SQL リファレンス』
- 「blob_threshold オプション [SQL Remote]」『SQL Anywhere サーバー データベース管理』
- 「verify_threshold オプション [SQL Remote]」『SQL Anywhere サーバー データベース管理』

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 サーバー データベース管理』
- 「sr_time_format オプション [SQL Remote]」『SQL Anywhere サーバー データベース管理』
- 「sr_timestamp_format [SQL Remote]」『SQL Anywhere サーバー データベース管理』
- 「date_order オプション」『SQL Anywhere サーバー 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、トリガーとプロシージャーを使用して競合を適切に解決できます。

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**　あるユーザーがロー (特定のプライマリキー値を持つ) を削除します。別のユーザー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、後者のユーザーが UPDATE 文または DELETE 文を送信したときに、ローが見つからないという理由でエラーが発生します。

- **参照整合性エラー**　外部キーを持つカラムがパブリケーションに含まれていて、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。

- **重複プライマリキーエラー**　2 人のユーザー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、あるユーザーがプライマリキーを更新し、別のユーザー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。

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」99 ページを参照してください。

参照

- 「更新の競合に対するデフォルトの解決」44 ページ
- 「レプリケーションエラーのレポートと処理」130 ページ
- 「更新の競合」43 ページ
- 「ローが見つからないエラー」50 ページ
- 「参照整合性エラー」51 ページ
- 「重複プライマリキーエラー」53 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 つが他のユーザー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。

  ○デフォルトによる競合解決を使用する。

  ○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。

  ○トリガーを使用する、カスタマイズされた競合解決を使用する。

  注意
  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ー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。

参照

- 「更新の競合に対するデフォルトの解決」44 ページ
- 「VERIFY 句を使用したカスタム競合解決」45 ページ
- 「トリガーを使用したカスタム競合解決」47 ページ

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。

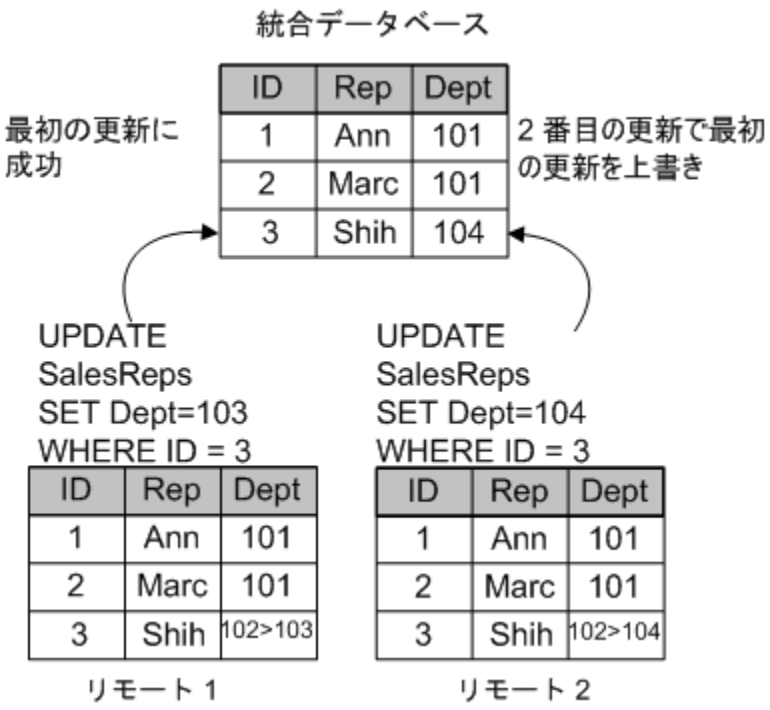
1. ユーザー 1 が、リモートサイト 1 でローを更新します。
2. ユーザー 2 が、リモートサイト 2 で同じローを更新します。
3. ユーザー 1 による更新内容が、統合データベースに送信されて適用されます。
4. ユーザー 2 による更新内容が、統合データベースに送信されます。

   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、ユーザー 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、ユーザー 1 の更新)。

   c. 競合はレポートされません。



参照

- 「ローが見つからないエラー」 50 ページ

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ー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. ユーザー 1 が、リモートサイト 1 でローを更新します。
2. ユーザー 2 が、リモートサイト 2 で同じローを更新します。
3. ユーザー 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. ユーザー 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、ユーザー 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ーを起動します。

      「競合解決トリガー」を定義して、更新の競合を処理します。競合解決トリガーは、リモートユーザー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。

   b. UPDATE 文を実行します。

   c. 競合解決トリガー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、その競合をトリガー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、トリガーアクションをレプリケートしません。これは、トリガー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。競合解決トリガーは統合データベースでのみ起動されるため、このトリガー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 トリガーは起動されません。

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。

参照

- 「トリガーを使用したカスタム競合解決」47 ページ

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* パラメーターにテーブルが 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ー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 サーバー SQL リファレンス』

verify_all_columns オプション

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 トリガーが起動します。
- 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ー用のいずれかに設定できます。

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。

参照

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

トリガー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。

●2 つ以上の更新の結果に対して計算を実行します。

●競合をテーブルにレポートします。

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

RESOLVE UPDATE 競合解決トリガーの使用

RESOLVE UPDATE トリガーの起動後に、各ローが更新されます。RESOLVE UPDATE トリガーの構文は、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。その他のカラムは、エラーを返します。

CURRENT REMOTE USER 特別値の使用

CURRENT REMOTE USER 特別値は、メッセージをデータベースに適用するときに、SQL Remote の受信フェーズによって設定されます。CURRENT REMOTE USER 特別値は、適用される操作が SQL Remote の受信フェーズによって適用されるかどうか、および、そのことが当てはまる場合に、適用される操作をどのリモートユーザーが生成したかをトリガーで判別する場合に、非常に役立ちます。

参照

- 「日付の競合解決」48 ページ
- 「在庫競合解決」48 ページ
- 「CREATE TRIGGER 文」『SQL Anywhere サーバー SQL リファレンス』
- 「UPDATE 文 [SQL Remote]」『SQL Anywhere サーバー SQL リファレンス』
- 「トリガーのレプリケーション」38 ページ
- -t オプション、SQL Remote Message Agent ユーティリティ (dbremote)190 ページ

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 トリガーは、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。

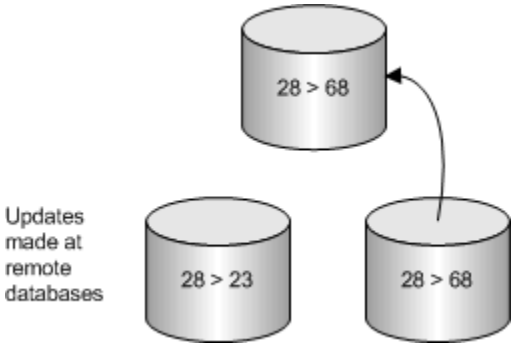
```
CREATE TRIGGER contact_date RESOLVE UPDATE
 ON Contacts
 REFERENCING OLD AS old_name
  NEW AS new_name
 FOR EACH ROW
 BEGIN
   IF new_name.contact_date <
     old_name.contact_date THEN
         SET new_name.contact_date
          = old_name.contact_date
   END IF
 END;
```

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。

在庫競合解決

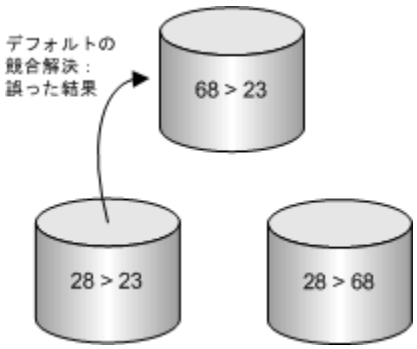
スポーツ用品メーカーのための倉庫システムがあるとします。製品情報のテーブルには、各製品の在庫数を記録する Quantity カラムがあります。このカラムへの更新は、通常は在庫数を引いていくことであり、新しく製品が搬入される場合は、在庫数を追加します。

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。

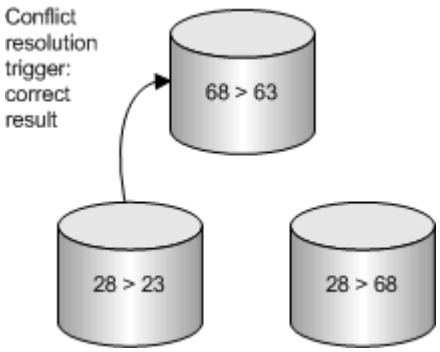


このエントリがデータベースに追加され、現在の Quantity カラムは、在庫に 68 枚の S サイズのタンクトップ T シャツがあることを示しています。ここで最初の営業担当者からの更新を受信すると、28 から 23 に変更するという内容にもかかわらず、実際のカラム値は 68 であるという競合が SQL Anywhere で検出されます。

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 トリガーでは、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、このトリガー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 トリガーによる変更内容が、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

あるユーザーがロー (特定のプライマリキー値を持つ) を削除します。別のユーザー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、後者のユーザーが 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には、宣言されたプライマリキーか、一意性制約がある必要があります。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。

参照

- 「レプリケーションエラーのレポートと処理」130 ページ
- 「更新の競合に対するデフォルトの解決」44 ページ
- 「INSERT および DELETE 文のレプリケーション」35 ページ

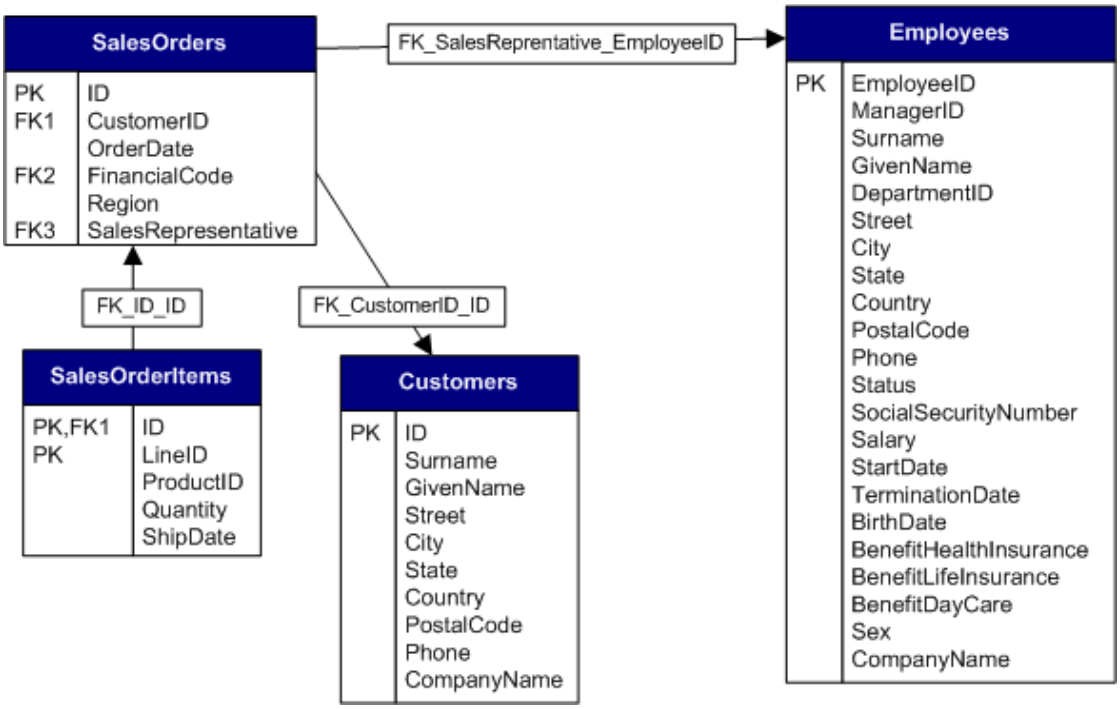
参照整合性エラー

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。

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,
);
```

リモートユーザー 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

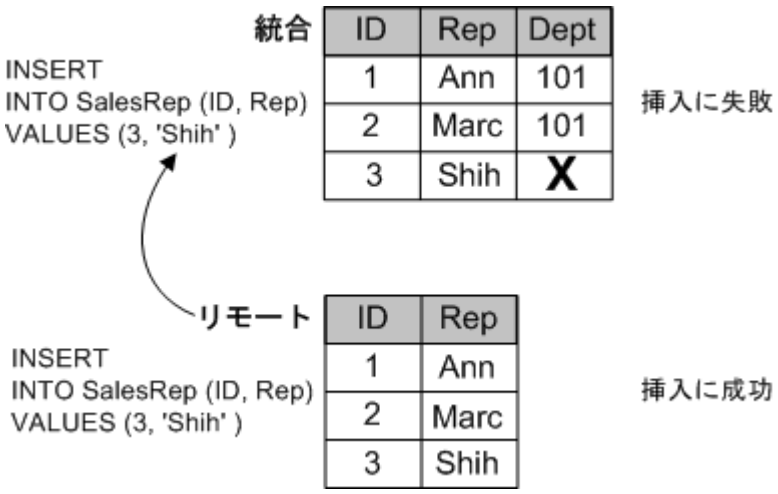
- 「レプリケーションエラーのレポートと処理」 130 ページ
- 「エンティティ整合性と参照整合性」『SQL Anywhere サーバー SQL の使用法』

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。



注意
この例の例外として、BEFORE トリガーを使用して、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」42 ページ

重複プライマリキーエラー

すべてのユーザー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。ユーザー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、ユーザー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 人のユーザーが、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 人のユーザー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。

2. プライマリキープールを使用して、各サイトで未使用のユニークなプライマリキー値のリストを管理します。

これらの方法のいずれか一方または両方を使用すると、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」130 ページ
- 「GLOBAL AUTOINCREMENT カラム」54 ページ
- 「プライマリキープール」55 ページ

GLOBAL AUTOINCREMENT 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 サーバー SQL の使用法』

デフォルト GLOBAL AUTOINCREMENT 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 サーバー SQL リファレンス』
- 「ALTER TABLE 文」『SQL Anywhere サーバー SQL リファレンス』

プライマリキープール

「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。リモートユーザーは、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。リモートユーザー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ー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ーを、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。
- **補充プロシージャー**　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ーが必要です。
- **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。
- **データ入力プロシージャー**　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ーを使用して、新しいローを入力します。

参照

- 「プライマリキープールテーブルの作成」 56 ページ
- 「プライマリキープールのレプリケート」 56 ページ
- 「キープールの入力と補充」 57 ページ
- 「キープールのプライマリキーの使用」 58 ページ

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 (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ー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ー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 (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 サーバー SQL リファレンス』
- 「CREATE SUBSCRIPTION 文 [SQL Remote]」『SQL Anywhere サーバー SQL リファレンス』

キープールの入力と補充

リモートユーザーが新しい顧客を追加するたびに、そのリモートユーザー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 (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ーを作成します。

> **注意**
> トリガー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ーを使用できません。

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. 各ユーザー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 プロシージャーには、各サブスクライバー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、各ユーザーに対して値を 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ーと単一の統合ユーザー 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 プロシージャーによって、各ユーザーのプールの値が 100 個まで増えます。指定する値は、ユーザー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。

   キープール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 プロシージャーを統合データベースで定期的に実行してください。

キープールのプライマリキーの使用

♦ プライマリキーを使用する (SQL の場合)

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ー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ーと、挿入を実行するストアドプロシージャーを使用します。

1. リモートデータベース上で実行するプロシージャーを作成して、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 プロシージャーは、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 プロシージャーは、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ーを作成して、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 プロシージャーは、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、プロシージャーを強化できます。

リモートデータベース間でのローの分割

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。

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。

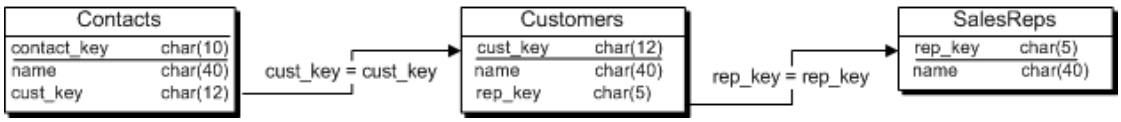
参照

- 「データ分割の切断」60 ページ
- 「重複分割」66 ページ

データ分割の切断

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。これらのテーブルの詳細については、「データ分割の切断」60 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));
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 )
);
```

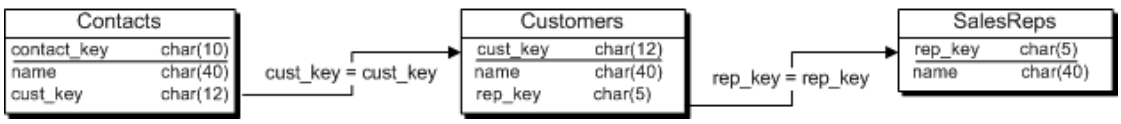
  Customers テーブルの 1 つのローには、Contacts テーブルの現在のローの cust_key 値が含まれています。SUBSCRIBE BY 文の中で WHERE 句を使用すると、サブクエリは必ず単一の値のみを返します。

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 UPDATE トリガー

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」60 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、「データ分割の切断」 60 ページを参照してください。

参照整合性の維持

サブスクライバー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 トリガーを使用します。BEFORE UPDATE トリガー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 トリガーは、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。トリガー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 トリガーを作成します。

```
CREATE TRIGGER "UpdateCustomer" BEFORE UPDATE OF "rep_key"
// only fire the trigger when rep_key is modified, 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ー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 トリガー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ーの 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ーが INSERT 文、UPDATE 文、または DELETE 文を取得するかが決定されます。

- ユーザーが 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。
- ユーザーが 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。
- ユーザーが 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、リモートユーザーがすでにローを保有しているため、UPDATE 文が送信されます。

トリガーに関する注意事項

次の例では、BEFORE UPDATE トリガーを使用する必要があります。他のコンテキストでは、BEFORE DELETE トリガーと BEFORE INSERT トリガーが必要です。

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

この例では、BEFORE トリガーを使用します。

```
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。

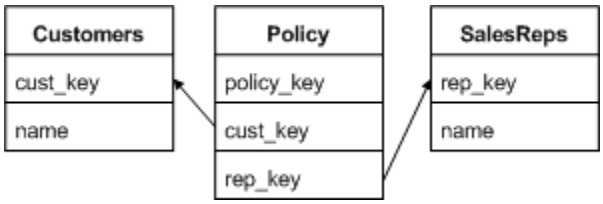
この例では、サブスクリプション式は 1 つの値を返します。ただし、複数の値を返すサブクエリも使用できます。サブスクリプション式の値は、更新を実行した後の値とする必要があります。

この例では、ローへのサブスクライバー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ーと新しいサブスクライバーを持つローの例については、「重複分割」66 ページを参照してください。

重複分割

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」 66 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ー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ー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ー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。

参照

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

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。

トリガーがない場合は、Customers テーブルに不正確なデータを持つサブスクライバー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。

トリガーを使用した問題の解決法

この問題を解決するには、Policy テーブルを変更する処理の前に起動されるトリガーを記述してください。これらの特別なトリガーによって、データベーステーブルは変更されません。代わりに、サブスクライバー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 トリガー

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 トリガーを作成し、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 トリガー

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

```
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 トリガー内のサブクエリは、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ーと関連付けられることはなく、複数のサブスクライバー間で共有されます。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。(特定のユーザーだけに 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 トリガー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 の各トリガーの機能を組み合わせる BEFORE UPDATE トリガーを作成する必要があります。

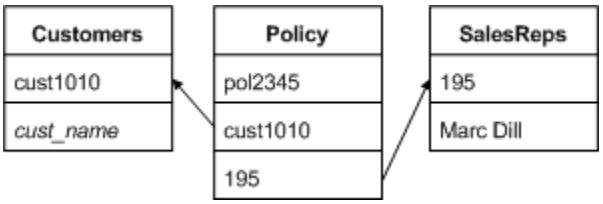
多対多の関係における subscribe_by_remote オプション

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、リモートユーザー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ー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 サーバー データベース管理』

リモートデータベースのユニークな ID 番号

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、ユーザー名など、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を挿入しようとすると、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。

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ーを設定できます。

参照

- 「global_database_id オプション」『SQL Anywhere サーバー データベース管理』
- 「プライマリキープール」 55 ページ
- 「デフォルト GLOBAL AUTOINCREMENT 宣言」 54 ページ
- 「イベントのトリガー条件」『SQL Anywhere サーバー 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 (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 = 20;
  ```

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

参照

- 「global_database_id オプション」『SQL Anywhere サーバー 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

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ーを作成して、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

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

   たとえば、user_id が 1001 であるリモートユーザー 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。

名前	値
extracted_db_global_id	抽出されるユーザー ID

   sp_hook_dbxtract_begin プロシージャーを作成してローの 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 プロシージャーを定義しない場合、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 プロシージャーを定義しても、このプロシージャーが #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. プロシージャー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ーフック

   ●プロシージャーフックが呼び出される前の #hook_dict の内容

   ●プロシージャー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」208 ページ
- 「SQL Remote システムプロシージャー」208 ページ
- 「global_database_id オプション」『SQL Anywhere サーバー データベース管理』

SQL Remote システムの管理

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

1. 統合データベースを設定します。

   「SQL Remote システムの作成」9 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. 各リモートユーザー用の 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」76 ページを参照してください。

   b. 各リモートユーザーのコンピューターに、SQL Anywhere データベースサーバー、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 サーバー プログラミング』と「SQL Remote の配備」『SQL Anywhere サーバー プログラミング』を参照してください。

4. SQL Remote 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 SQL Remote 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。「SQL Remote Message Agent (dbremote) モード」84 ページを参照してください。

   b. ユーザー名、SQL Remote 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。「SQL Remote Message Agent (dbremote)」83 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」99 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 パフォーマンス」89 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 システムバックアップ」121 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」130 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」135 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

リモートユーザー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ー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。

- **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。

  効率を高める 1 つの方法は、複数のリモートデータベースを作成することです。

参照

- 「リモートデータベースの自動抽出」76 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」78 ページ
- 「複数のリモートデータベースの作成」81 ページ

リモートデータベースの自動抽出

再ロードファイルへのリモートデータベースの抽出の詳細については、「リモートデータベースの抽出」76 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 (Sybase Central の場合)

1. **[SQL Anywhere 12]** プラグインを使用して、DBA 権限のあるユーザーとして、統合データベースに接続します。
2. **[ツール]** メニューから、**[SQL Anywhere 12]** » **[データベースの抽出]** をクリックします。
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]** をクリックします。

   プロンプトが表示されたら、**[構造とデータを抽出]** をクリックします。
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、リモートユーザー、パブリケーション、サブスクリプション、トリガー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、SQL Remote Message Agent が開始されないため、メッセージが交換されません。

♦ リモートデータベースを自動的に抽出する (SQL の場合)

1. DBA 権限のあるユーザー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。

   -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ー用であり、*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、リモートユーザー、パブリケーション、サブスクリプション、トリガー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、SQL Remote Message Agent が開始されないため、メッセージが交換されません。

参照

- 「SQL Remote Message Agent (dbremote)」 83 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」 78 ページ
- 「抽出ユーティリティ (dbxtract)」 194 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

データベースの自動抽出の詳細については、「リモートデータベースの抽出」76 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、スクリプト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 スクリプトファイル。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)」194 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)」194 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のに必要な SQL 文が記述され、次のオブジェクトを作成する文が含まれています。

- パブリッシャー、リモートユーザー、統合ユーザー
- 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
- トリガー
- プロシージャー

注意
リモートデータベースを作成する場合は、*reload.sql* を編集する必要があるかもしれません。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。

♦ *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

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ー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。

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* スクリプト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、リモートユーザー、パブリケーション、サブスクリプション、トリガー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、SQL Remote Message Agent が開始されないため、メッセージが交換されません。

参照

- 「リモートデータベースの自動抽出」 76 ページ
- 「reload.sql ファイルの編集」 79 ページ
- 「SQL Remote Message Agent (dbremote)」 83 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* スクリプト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。

プロシージャー、トリガー、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ー、トリガー、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ーには、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、プロシージャーには、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ー、トリガー、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

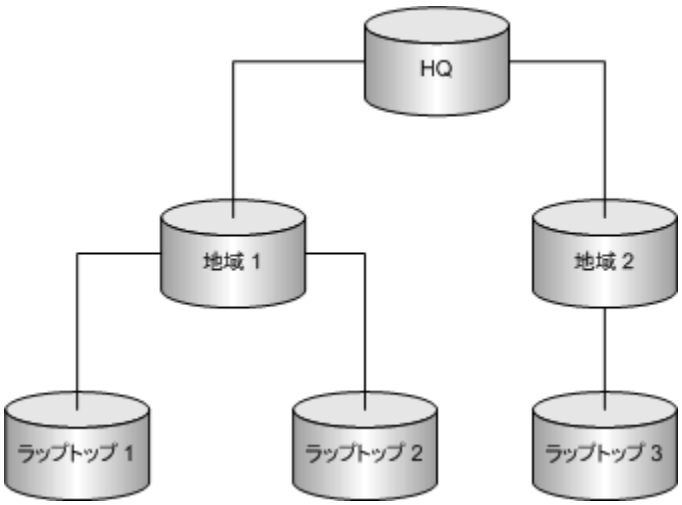
「多層階層システムのデータベースの抽出」80 ページを参照してください。

参照

- 「複数のリモートデータベースの作成」81 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」78 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

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、ユーザー 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ー (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 レベルのユーザー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) はリモートユーザーに影響しません。「抽出ユーティリティ (dbxtract)」 194 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」 81 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」 78 ページ

複数のリモートデータベースの作成

♦ 複数のリモートデータベースを作成する

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。

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ー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 SQL Remote 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、サブスクライバー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ーに送信できます。

      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。

   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ーのデータと一致しますが、他のユーザー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 SQL Remote Message Agent が実行されていた場合は、SQL Remote 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、各ユーザー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**を使用してユーザー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)」194 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)」194 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。

4. リモートデータベースごとに、各リモートユーザーに固有の SQL Remote 定義を定義します。

5. リモートユーザーごとに、統合データベースからユーザー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)」194 ページの -d オプションを参照してください。

6. 各リモートユーザー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、SQL Remote Message Agent (dbremote) を実行すると、各ユーザー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。

参照

- 「多層階層システムのデータベースの抽出」 80 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」 78 ページ
- 「reload.sql ファイルの編集」 79 ページ
- 「サブスクリプションの開始」 143 ページ
- 「SQL Remote Message Agent (dbremote)」 83 ページ
- 「START SUBSCRIPTION 文 [SQL Remote]」『SQL Anywhere サーバー SQL リファレンス』
- 「REMOTE RESET 文 [SQL Remote]」『SQL Anywhere サーバー SQL リファレンス』
- 「ユーザーパーミッション」 19 ページ

SQL Remote Message Agent (dbremote)

SQL Remote 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。SQL Remote 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。SQL Remote Message Agent (dbremote) では、メッセージの送受信処理を行います。

SQL Remote Message Agent の機能は次のとおりです。

- **メッセージ送信時の SQL Remote Message Agent (dbremote) のタスク**

  ○各パブリッシャー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ーへのメッセージに変換する。

  ○サブスクライバーにメッセージを送信する。

  ○SQL Remote 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。

- **メッセージ受信時の SQL Remote 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。

接続

SQL Remote Message Agent (dbremote) は、データベースサーバー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**　SQL Remote 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。
- **各ワーカー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。

参照

- 「メッセージを送信するタスク」95 ページ
- 「メッセージを受信するタスク」89 ページ

SQL Remote Message Agent (dbremote) モード

SQL Remote 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、SQL Remote Message Agent (dbremote) は各リモートユーザー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。

- **バッチモード**　バッチモードでは、SQL Remote 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。

SQL Remote 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 SQL Remote 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**　SQL Remote Message Agent (dbremote) を実行するユーザーには、REMOTE DBA 権限または DBA 権限が必要です。
- **システム内にある各 SQL Remote 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、SQL Remote Message Agent (dbremote) の -l オプションを使用して変更できます。

参照

- 「継続モードでの SQL Remote Message Agent (dbremote) の実行」 85 ページ
- 「バッチモードでの SQL Remote Message Agent (dbremote) の実行」 87 ページ
- 「REMOTE DBA 権限」 30 ページ
- 「SQL Remote Message Agent ユーティリティ (dbremote)」 183 ページ

継続モードでの SQL Remote 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。

♦ 継続モードでの SQL Remote Message Agent (dbremote) の実行

1. REMOTE 権限を持つすべてのユーザーが 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、SQL Remote Message Agent (dbremote) は各リモートユーザー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。

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

   Windows では、SQL Remote 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 SQL Remote Message Agent (dbremote) を起動することもできます。

   たとえば、次の文ではデータベースファイル *c:¥mydata.db* で dbremote を継続モードで実行します。接続にはユーザー名 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
   ```

   ユーザー名 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。

参照

- 「SQL Remote Message Agent (dbremote)」 83 ページ
- 「SQL Remote Message Agent ユーティリティ (dbremote)」 183 ページ
- 「SQL Remote Message Agent (dbremote) の稼働条件」 84 ページ
- 「Mac OS X での SQL Remote Message Agent (dbremote) の実行」 88 ページ
- 「UNIX での SQL Remote Message Agent (dbremote) の実行」 88 ページ
- 「送信頻度の設定」 86 ページ

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

SQL Remote Message Agent (dbremote) を継続モードで実行する場合、サーバーの稼働中は常に SQL Remote Message Agent (dbremote) を実行させておくことができます。このためには、SQL

Remote 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ー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。

参照

- 「Windows 用サービスユーティリティ (dbsvc)」『SQL Anywhere サーバー データベース管理』

送信頻度の設定

統合データベースなどで、SQL Remote Message Agent (dbremote) を継続モードで実行する場合は、すべての REMOTE ユーザー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、SQL Remote 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。

SQL Remote 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 を設定したユーザー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ー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ー全員に、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。あるユーザー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)**　ユーザーが SEND AT 句または SEND EVERY 句を指定していない場合は、SQL Remote 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。

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 (Sybase Central の場合)

1. **SQL Anywhere 12** プラグインを使用して、DBA 権限のあるユーザーとして、データベースに接続します。
2. 左ウィンドウ枠で、**[SQL Remote ユーザー]** ディレクトリをクリックします。

3. ユーザーを右クリックし、**[プロパティ]** をクリックします。
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]** のいずれかをクリックし、時間を指定します。

参照

- 「GRANT REMOTE 文 [SQL Remote]」『SQL Anywhere サーバー SQL リファレンス』
- 「バッチモードでの SQL Remote Message Agent (dbremote) の実行」 87 ページ

バッチモードでの SQL Remote Message Agent (dbremote) の実行

♦ バッチモードでの SQL Remote 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。

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ーが最低 1 つあることを確認します。

   リモートユーザーの「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 SQL Remote Message Agent (dbremote) を起動してください。

2. SQL Remote Message Agent (dbremote) を起動します。

   Windows では、SQL Remote 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 SQL Remote Message Agent (dbremote) を起動することもできます。

   たとえば、次の文ではデータベースファイル *c:¥mydata.db* で dbremote をバッチモードで実行します。接続にはユーザー名 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"
   ```

   SQL Remote 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。

   ユーザー名 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。

参照

- 「SQL Remote Message Agent ユーティリティ (dbremote)」 183 ページ
- 「SQL Remote Message Agent (dbremote) の稼働条件」 84 ページ
- 「SQL Remote Message Agent (dbremote)」 83 ページ
- 「Mac OS X での SQL Remote Message Agent (dbremote) の実行」 88 ページ
- 「UNIX での SQL Remote Message Agent (dbremote) の実行」 88 ページ

Mac OS X での SQL Remote Message Agent (dbremote) の実行

SQL Anywhere には、Mac OS X で SQL Remote 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 SQL Remote Message Agent を起動することもできます。

♦ SyncConsole を起動する

1. Finder で、*/Applications/SQLAnywhere12* 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]** をクリックします。

   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. SQL Remote 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ーでは、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 12 Demo"
   ```

   ユーザー名には、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。

参照

- 「SQL Remote Message Agent (dbremote) の稼働条件」 84 ページ
- 「SQL Remote Message Agent ユーティリティ (dbremote)」 183 ページ

UNIX での SQL Remote 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、SQL Remote Message Agent (dbremote) をデーモンとして実行します。「SQL Remote Message Agent ユーティリティ (dbremote)」 183 ページ の -ud オプションを参照してください。

ユーザー名には、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。「SQL Remote Message Agent (dbremote) の稼働条件」 84 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、「SQL Remote Message Agent ユーティリティ (dbremote)」 183 ページを参照してください。

SQL Remote パフォーマンス

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ー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ーには削除文、別のサブスクライバーには更新文、また別のサブスクライバーには挿入文が送信されます。

どのサブスクライバー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ーと SQL Remote Message Agent (dbremote) が分担します。

データベースサーバー

データベースサーバーは、パブリケーションを処理します。

SQL Remote Message Agent (dbremote)

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

SQL Remote 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ーのサブスクリプション値と照合します。SQL Remote 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ーに送信します。

サブスクライバーの数が非常に多くてもデータベースサーバーのパフォーマンスは低下しませんが、SQL Remote 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。

参照

- 「データベースサーバーによるパブリケーションの処理」35 ページ

メッセージを受信するタスク

SQL Remote 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、SQL Remote 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。

- **メッセージの読み込み**　メッセージが着信すると、SQL Remote 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。

  欠落しているメッセージがあり、SQL Remote Message Agent (dbremote) が継続モードで実行されている場合、SQL Remote 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。SQL Remote 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。

○SQL Remote 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、SQL Remote Message Agent (dbremote) の待機時間が切れた場合、SQL Remote Message Agent (dbremote) は、パブリッシャー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。

- **メッセージの適用**　SQL Remote Message Agent (dbremote) は、サブスクライバーデータベースに正しい順序でメッセージを適用します。
- **サブスクライバー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、パブリッシャーには確認メッセージが返送されます。パブリッシャーの SQL Remote 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。

参照

- 「新しいメッセージをチェックするための間隔調整のポーリング」91 ページ
- 「受信メッセージのキャッシュによるスループットの調整」92 ページ
- 「メッセージを再送するための要求調整」93 ページ
- 「ワーカースレッド数の調整」94 ページ
- 「保証されたメッセージ配信システム」99 ページ

メッセージを受信する場合のパフォーマンス

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、SQL Remote 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。

●SQL Remote Message Agent (dbremote) が受信メッセージを確認する頻度。

●SQL Remote 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。

●SQL Remote 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。

●受信したメッセージの処理に使用されるワーカースレッドの数。

参照

- 「新しいメッセージをチェックするための間隔調整のポーリング」91 ページ
- 「受信メッセージのキャッシュによるスループットの調整」92 ページ
- 「メッセージを再送するための要求調整」93 ページ
- 「ワーカースレッド数の調整」94 ページ

新しいメッセージをチェックするための間隔調整のポーリング

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、SQL Remote 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。

ポーリング間隔の延長

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 12 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、メールサーバー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、SQL Remote Message Agent (dbremote) の待機時間も調整してください。待機時間とは、SQL Remote 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。

ポーリング間隔の短縮

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 12 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

- 「メッセージを受信する場合のパフォーマンス」90 ページ
- 「受信メッセージのキャッシュによるスループットの調整」92 ページ
- 「メッセージを再送するための要求調整」93 ページ
- 「ワーカースレッド数の調整」94 ページ
- 「メッセージを送信する場合のパフォーマンス」96 ページ

受信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、SQL Remote 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ースレッド間の競合。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、SQL Remote 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

SQL Remote 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、SQL Remote 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。「SQL Remote Message Agent ユーティリティ (dbremote)」183 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 SQL Remote Message Agent (dbremote) を起動します。

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

参照

- 「メッセージを受信する場合のパフォーマンス」90 ページ
- 「新しいメッセージをチェックするための間隔調整のポーリング」91 ページ
- 「メッセージを再送するための要求調整」93 ページ
- 「ワーカースレッド数の調整」94 ページ
- 「メッセージを送信する場合のパフォーマンス」96 ページ

メッセージを再送するための要求調整

メッセージがシーケンスから欠落している場合、SQL Remote 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。SQL Remote 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、SQL Remote Message Agent (dbremote) の待機時間は 1 です。

SQL Remote 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、SQL Remote Message Agent (dbremote) は何もしません。代わりに、SQL Remote 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、SQL Remote 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 SQL Remote 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。

例

user1 と user2 という 2 人のリモートユーザーがいて、両ユーザー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 SQL Remote Message Agent (dbremote) を実行します。たとえば、これらのユーザーは、次のコマンドを使用してそれぞれの SQL Remote Message Agent (dbremote) を実行します。

```
dbremote -c "DSN=SQL Anywhere 12 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 はユーザー名であり、*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。SQL Remote Message Agent (dbremote) では、メッセージは両方のユーザーについて 1 番から開始するものと考えます。

0 秒後

1. SQL Remote Message Agent (dbremote) が user1.1 と user2.4 を読み込みます。
2. SQL Remote 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 SQL Remote Message Agent (dbremote) の待機時間は user1 が N/A、user2 が 3 です。

30 秒後

1. SQL Remote 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. SQL Remote Message Agent (dbremote) での適用：なし
3. 現時点の SQL Remote Message Agent (dbremote) の待機時間：user1：N/A、user2：2

60 秒後

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

90 秒後

1. SQL Remote Message Agent (dbremote) でのメッセージの読み込み：user1.4
2. SQL Remote Message Agent (dbremote) での適用：なし
3. SQL Remote Message Agent (dbremote) の待機時間：user1：3、user2：0
4. SQL Remote Message Agent (dbremote) が、user2 に再送を要求します。

ユーザー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 SQL Remote Message Agent (dbremote) の待機時間はリセットされます。

120 秒後

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

参照

- 「メッセージを受信する場合のパフォーマンス」90 ページ
- 「新しいメッセージをチェックするための間隔調整のポーリング」91 ページ
- 「受信メッセージのキャッシュによるスループットの調整」92 ページ
- 「ワーカースレッド数の調整」94 ページ
- 「メッセージを送信する場合のパフォーマンス」96 ページ

ワーカースレッド数の調整

次の手順では、SQL Remote 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、ヘッダー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ー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ースレッドを使用すると、リモートユーザーからの受信メッセージを並列で適用できます。「SQL Remote Message Agent ユーティリティ (dbremote)」183 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ー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ーからのメッセージが並列で適用される。
- リモートユーザー 1 人からのメッセージは直列で適用される。

  たとえば、リモートユーザー 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 つのワーカー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ー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ー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ーがあるときに、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ー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ースレッドを使用する場合、ワーカースレッドによってユーザー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ースレッド数を設定する

- 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ースレッド数を設定します。

  たとえば、次のコマンドでは、ワーカースレッド数を 5 に設定します。

  ```
  dbremote -c "DSN=SQL Anywhere 12 Demo"  -w 5
  ```

参照

- 「継続モードでの SQL Remote Message Agent (dbremote) の実行」85 ページ
- 「メッセージを受信する場合のパフォーマンス」90 ページ
- 「新しいメッセージをチェックするための間隔調整のポーリング」91 ページ
- 「受信メッセージのキャッシュによるスループットの調整」92 ページ
- 「メッセージを再送するための要求調整」93 ページ
- 「メッセージを送信する場合のパフォーマンス」96 ページ

メッセージを送信するタスク

SQL Remote 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。

- **パブリッシャーのトランザクションログのスキャン**　SQL Remote Message Agent (dbremote) は、パブリッシャー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ー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、SQL Remote 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。サブスクライバーデータベースの SQL Remote 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**　SQL Remote Message Agent (dbremote) は、各リモートユーザー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。

  SQL Remote 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。SQL Remote 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。

- **リモートデータベースからの再送要求の処理**　ユーザーがメッセージの再送要求を発行すると、パブリッシャーデータベースの SQL Remote 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。

- **パブリッシャー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、パブリッシャーには確認メッセージが返送されます。

参照

- 「送信遅延調整」97 ページ
- 「送信メッセージのキャッシュによるスループットの調整」97 ページ
- 「再送要求処理速度」98 ページ
- 「保証されたメッセージ配信システム」99 ページ

メッセージを送信する場合のパフォーマンス

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、SQL Remote 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。
- メッセージのサイズ。
- 再送要求処理の緊急度。

参照

- 「送信遅延調整」97 ページ
- 「メッセージを受信する場合のパフォーマンス」90 ページ
- 「再送要求処理速度」98 ページ

送信遅延調整

送信するメッセージを作成するため、SQL Remote 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、リモートユーザー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 12 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 12 Demo"  -sd 60
```

一般的に、送信間隔が長いと、SQL Remote Message Agent (dbremote) によって、送信前にメッセージの作成処理の大部分が実行されます。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。

参照

- 「送信メッセージのキャッシュによるスループットの調整」97 ページ
- 「再送要求処理速度」98 ページ
- 「SQL Remote Message Agent ユーティリティ (dbremote)」183 ページ

送信メッセージのキャッシュによるスループットの調整

SQL Remote 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ーのグループには、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、SQL Remote Message Agent (dbremote) のコマンドで指定します。

-m オプションは、SQL Remote 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。「SQL Remote Message Agent ユーティリティ (dbremote)」183 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 SQL Remote Message Agent (dbremote) を起動します。

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

参照

●「送信遅延調整」97 ページ
●「再送要求処理速度」98 ページ

再送要求処理速度

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、SQL Remote Message Agent (dbremote) は再送要求の処理を遅らせます。デフォルトでは、SQL Remote Message Agent (dbremote) は、再送を要求したリモートユーザー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、SQL Remote 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 SQL Remote 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。SQL Remote 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 12 Demo" -ru 1h
```

参照

- 「送信遅延調整」97 ページ
- 「送信メッセージのキャッシュによるスループットの調整」97 ページ
- 「メッセージを再送するための要求調整」93 ページ
- 「SQL Remote Message Agent ユーティリティ (dbremote)」183 ページ

保証されたメッセージ配信システム

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。
- レプリケートされたオペレーションが欠落しない。
- 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。

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ーに対応したローがあり、そこにはそのサブスクライバー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ー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、SQL Remote Message Agent (dbremote) が管理しています。

  サブスクライバーデータベースでは、SQL Remote Message Agent (dbremote) はパブリッシャー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。

- **メッセージのヘッダー内の情報**　SQL Remote Message Agent (dbremote) は、メッセージ内のヘッダー情報を読み込み、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ーには、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ー。

  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」)
  ```

参照

- 「操作の順序」101 ページ
- 「消失または壊れたメッセージ」102 ページ
- 「メッセージは 1 回だけ適用」103 ページ

操作の順序

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、パブリッシャーデータベースとサブスクライバー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。
- 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、サブスクライバー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. パブリッシャーの SQL Remote Message Agent (dbremote) はサブスクライバー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、パブリッシャーが次のメッセージを user1 に送信します。

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

   パブリッシャーの SYSREMOTEUSER システムテーブル内で、パブリッシャーは 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ーデータベースに適用されると、パブリッシャー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、サブスクライバー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. パブリッシャーの SQL Remote 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 カラムにユーザー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、パブリッシャーは、パブリッシャーの 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、パブリッシャー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. サブスクライバーデータベースの SQL Remote 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、サブスクライバー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ーデータベースの log_received カラムには、パブリッシャー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、サブスクライバーの SQL Remote Message Agent (dbremote) は、その SYSREMOTEUSER システムテーブルの confirm_received 値を設定し、パブリッシャーデータベースに確認メッセージを送信します。

参照

- 「操作の順序」 101 ページ
- 「消失または壊れたメッセージ」 102 ページ
- 「メッセージは 1 回だけ適用」 103 ページ

消失または壊れたメッセージ

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**　サブスクライバー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ー。
- **rereceive_count カラム**　SQL Remote Message Agent (dbremote) がパブリッシャーユーザー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ー。

サブスクライバー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. サブスクライバーの SQL Remote 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. サブスクライバーの SQL Remote Message Agent (dbremote) は、パブリッシャーに確認メッセージを送信します。
3. パブリッシャーが確認メッセージを受信すると、パブリッシャーの SQL Remote 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. サブスクライバーの SQL Remote 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. パブリッシャー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ーの resend_count 値を増分します。
3. パブリッシャーの 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

- 「操作の順序」 101 ページ
- 「メッセージは 1 回だけ適用」 103 ページ

メッセージは 1 回だけ適用

サブスクライバーの SQL Remote Message Agent (dbremote) は、メッセージヘッダー内の 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

- 「操作の順序」 101 ページ
- 「消失または壊れたメッセージ」 102 ページ

メッセージサイズ

この項では、SQL Remote 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。

SQL Remote 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。

- **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。

- **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。

参照

- 「SQL Remote のアップグレード」『SQL Anywhere 12 変更点とアップグレード』
- 「compression オプション [SQL Remote]」『SQL Anywhere サーバー データベース管理』
- 「エンコードでのメッセージ破壊の防止」 104 ページ
- 「SQL Remote Message Agent (dbremote) の稼働条件」 84 ページ

エンコードでのメッセージ破壊の防止

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 (*dbencod12.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。

- 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。

参照

- 「カスタムエンコードスキーム」 104 ページ
- 「SQL Remote のアップグレード」『SQL Anywhere 12 変更点とアップグレード』

カスタムエンコードスキーム

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、各ユーザー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。

ヘッダーファイル *%SQLANY12%¥SDK¥Include¥dbrmt.h* 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ーフェイ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ー encode_dll に設定します。次に例を示します。

```
SET REMOTE FTP OPTION "Public"."encode_dll" = 'c:¥¥sqlany12¥¥Bin32¥¥custom.dll';
```

注意
エンコードとデコードは互換性が必要です。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]」213 ページ

SQL Remote メッセージシステム

SQL Remote のレプリケーションでは、メッセージシステムとは、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 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。
- **FTP**　インターネットファイル転送プロトコル。
- **HTTP**　ハイパーテキスト転送プロトコル。
- **SMTP/POP**　インターネット電子メール転送プロトコル。

REMOTE または CONSOLIDATE パーミッションをユーザーに割り当てる場合は、メッセージシステムを選択します。

SQL Remote システムで使用される各メッセージシステムでは、制御パラメーター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。

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、必ずパブリッシャー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**、**HTTP**、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ー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ー 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ーアドレスを返信アドレスとして使用します。また SQL Remote 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ーアドレスを使用します。

参照

- 「FILE メッセージシステム」110 ページ
- 「FTP メッセージシステム」111 ページ
- 「SMTP メッセージシステム」118 ページ
- 「HTTP メッセージシステム」114 ページ
- 「REMOTE パーミッションの付与」25 ページ
- 「CONSOLIDATE パーミッションの付与」28 ページ
- 「サポートされるプラットフォーム」『SQL Anywhere 12 紹介』

メッセージタイプの作成

♦ メッセージタイプを追加する (Sybase Central の場合)

1. **SQL Anywhere 12** プラグインを使用して、DBA 権限のあるユーザーとして、データベースに接続します。
2. 左ウィンドウ枠で、**[SQL Remote ユーザー]** 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]** をクリックします。
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. **[パブリッシャー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ー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 (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ー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、Windows Mobile デバイス上のレジストリ値が変更され、FILE メッセージリンクのメッセージのディレクトリが Microsoft ActiveSync ディレクトリとなるよう設定されます。メッセージリンクパラメーターがリモートデータベースで定義されていない場合、SQL Remote はレジストリからのメッセージリンクパラメーターを読み込むだけだということに注意してください。Windows Mobile デバイスをデスクトップコンピューターにドッキングすると、Microsoft ActiveSync は、デスクトップコンピューターの [Microsoft 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

1. Windows Mobile デバイスをデスクトップコンピューターに接続します。
2. **[ツール]** から、**[SQL Anywhere 12]** » **[Windows Mobile メッセージタイプの編集]** をクリックします。
3. 必要な場合は、**Directory** パラメーターの値をデスクトップと同期している Windows Mobile デバイスのディレクトリとなるよう値を変更します。
4. **OK** をクリックして、Windows Mobile デバイスのレジストリに変更を保存します。

参照

- 「CREATE REMOTE MESSAGE TYPE 文 [SQL Remote]」『SQL Anywhere サーバー SQL リファレンス』
- 「メッセージタイプの変更」107 ページ
- 「メッセージタイプの削除」108 ページ

メッセージタイプの変更

パブリッシャー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 (Sybase Central の場合)

1. **SQL Anywhere 12** プラグインを使用して、DBA 権限のあるユーザーとして、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 ユーザー]** 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]** をクリックします。
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 (SQL の場合)

1. DBA 権限のあるユーザー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ー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 サーバー SQL リファレンス』
- 「メッセージタイプの作成」 106 ページ
- 「メッセージタイプの削除」 108 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、パブリッシャー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 (Sybase Central の場合)

1. **SQL Anywhere 12** プラグインを使用して、DBA 権限のあるユーザーとして、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 ユーザー]** 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]** をクリックします。
5. **[はい]** をクリックします。

♦ メッセージタイプを削除する (SQL の場合)

1. DBA 権限のあるユーザー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 サーバー SQL リファレンス』
- 「メッセージタイプの作成」 106 ページ
- 「メッセージタイプの変更」 107 ページ

リモートメッセージタイプ制御パラメーターの設定

メッセージ制御パラメーターは、データベース内に保存されます。次の手順を使用して、制御パラメーターを設定します。

♦ **メッセージ制御パラメーターを設定する (SQL の場合)**

- SET REMOTE OPTION 文を実行します。

  たとえば、次の文は、ユーザー 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';
  ```

♦ **メッセージリンクパラメーターを表示する (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;
  ```

ディスクに格納されるメッセージリンクパラメーター

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ー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ー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ーは、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ーは、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ーのうち、その代替の 1 つとして使用されるパスが格納されます。

SQL Remote 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ーの設定を使用します。設定が指定されていない場合は、そのパブリッシャー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ーのデータベースへの保存をサポートする SQL Remote 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]」213 ページ
- 「SYSREMOTEOPTION システムビュー」『SQL Anywhere サーバー 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。リモートユーザーの FILE アドレスは、すべてのメッセージが書き込まれるサブフォルダーです。メッセージを取得するには、アプリケーションがユーザー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。

SQL Remote Message Agent (dbremote) がサービスとして起動中の場合は、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、SQL Remote 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フォルダーです。このサブフォルダーは、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 ユーザー全員が使用できます。各ユーザーには、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、統合コンピューターまたはリモートコンピューター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ー

FILE メッセージシステムでは、SET REMOTE OPTION 文で設定される次の制御パラメーターを使用します。

- **directory**　メッセージが格納されるディレクトリです。このパラメーターは、SQLREMOTE 環境変数の代替となります。
- **debug**　このパラメーターの設定は、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出力ログに表示されます。
- **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ーに設定する必要があります。
- **invalid_extensions**　メッセージングシステムでのファイルの生成時に SQL Remote 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。

- **max_retries**　デフォルトでは、SQL Remote が継続モードで実行されていて、メッセージシステムにアクセスするときにエラーが発生する場合、送受信フェーズの後に停止します。このパラメーターを使用して、停止前に SQL Remote が再試行する送受信フェーズの回数を指定します。

- **pause_after_failure**　このパラメーターが使えるのは、max_retries がゼロ以外の値に指定され、SQL Remote が継続モードで実行されている場合です。メッセージシステムでエラーが発生すると、このパラメーターは、SQL Remote が送受信フェーズを再試行する間に待機する時間を秒で指定します。

- **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 Microsoft ActiveSync

FILE リンクの場合、SQL Remote 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ー上では、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ー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
```

Microsoft 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ーと 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。

参照

- 「FTP メッセージシステム」111 ページ
- 「SMTP メッセージシステム」118 ページ
- 「HTTP メッセージシステム」114 ページ
- 「CREATE REMOTE MESSAGE TYPE 文 [SQL Remote]」『SQL Anywhere サーバー SQL リファレンス』
- 「SET REMOTE OPTION 文 [SQL Remote]」213 ページ
- 「SQLREMOTE 環境変数」『SQL Anywhere サーバー データベース管理』
- 「メッセージサイズ」103 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ー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フォルダーが各ユーザーのアドレスになります。

FTP メッセージ制御パラメーター

FTP メッセージシステムでは、SET REMOTE OPTION 文で設定される次の制御パラメーターを使用します。

- **host**　FTP サーバーを実行しているコンピューターのホスト名。このパラメーターには、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ー名。
- **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**　このパラメーターは、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、デバッグ出力が出力ログに表示されます。
- **active_mode**　このパラメーターは、SQL Remote がクライアント／サーバー接続を確立する方法を制御します。このパラメーターは、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。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 サーバーがすべてのデータ接続を開始します。
- **reconnect_retries**　失敗になる前に、リンクがサーバー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。このパラメーターを設定すると、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。このパラメーターを設定すると、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**　このパラメーターは、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 サーバー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ーに設定する必要があります。
- **max_retries**　デフォルトでは、SQL Remote が継続モードで実行されていて、メッセージシステムにアクセスするときにエラーが発生する場合、送受信フェーズの後に停止します。このパラメーターを使用して、停止前に SQL Remote が再試行する送受信フェーズの回数を指定します。
- **pause_after_failure**　このパラメーターが使えるのは、max_retries がゼロ以外の値に指定され、SQL Remote が継続モードで実行されている場合です。メッセージシステムでエラーが発

生すると、このパラメーターは、SQL Remote が送受信フェーズを再試行する間に待機する時間を秒で指定します。

参照

- 「サポートされるプラットフォーム」『SQL Anywhere 12 紹介』
- 「メッセージサイズ」103 ページ
- 「FILE メッセージシステム」110 ページ
- 「SMTP メッセージシステム」118 ページ
- 「CREATE REMOTE MESSAGE TYPE 文 [SQL Remote]」『SQL Anywhere サーバー SQL リファレンス』
- 「SET REMOTE OPTION 文 [SQL Remote]」213 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ーの設定**　デバッグ出力を確認して、FTP サーバー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 サーバーの ping**　FTP リンクが FTP サーバー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 サーバーの IP アドレスと、FTP サーバーへの ping (往復) 時間が返されます。FTP サーバーの 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 サーバー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ー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 サーバーがすべてのデータ接続を開始します。FTP サーバー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 サーバー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 ユーザー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 サーバーを設定し直すか、active_mode メッセージ制御パラメーターに 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 サーバーが接続中で、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 サーバーにログインします。ディレクトリを、root_directory パラメーター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 制御パラメーター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 サーバーのパーミッションが正しく設定されていません。

HTTP メッセージシステム

HTTP を使用する SQL Remote は、ハイパーテキスト転送プロトコル (HTTP) を使用してインターネット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、HTTP 経由でターゲットデータベースに送信されます。このメッセージは、HTTP サーバーとして動作する SQL Anywhere データベースを使用して送受信されます。

HTTP 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 12 紹介』を参照してください。

HTTP メッセージサーバーの設定

SQL Remote メッセージをリモートデータベースとの間で転送する HTTP サーバーとして動作する SQL Anywhere データベースサーバーを使用します。デフォルトでは、新しく初期化された SQL Anywhere データベースには、メッセージサーバーとして動作できるように定義された Web サービスはありません。sr_add_message_server、sr_drop_message_server、sr_update_message_server という 3 つのシステムストアドプロシージャーが、新しく作成された SQL Anywhere データベースに定義されています。これにより、データベースが SQL Remote メッセージを転送するための HTTP サーバーとして動作できるよう必要なデータベースオブジェクトを定義できます。

注意
データベースは SQL Anywhere 12.0.1 で初期化され、3336 以上のビルド番号のデータベースサーバーで初期化されている必要があります。SYS.SYSHISTORY システムテーブルに問い合わせ、データベースの初期化に使用されたデータベースサーバーのバージョンとビルドを判断します。データベースが 12.0.1 と 3336 未満のビルド番号で初期化されている場合、"ALTER DATABASE UPGRADE PROCEDURE ON" を実行してデータベースを更新します。

別のデータベースサーバーを実行するか、それとも既存の統合データベースをメッセージサーバーとして使用するのかを決める必要があります。この決定を行う場合は次のことを考慮します。

1. リモートデータベースが、メッセージサーバーで認証する場合、そのリモートデータベースのパブリッシャーと認証用に提供されたパスワードが使用されます。統合データベースにユーザーが存在していても、そのユーザーは HTTP メッセージシステムの要件である定義済みパスワードを持っていないかもしれません (リモートユーザーは接続のパーミッションを持っていないかもしれません)。統合データベースのリモートユーザーに CONNECT パーミッションを付与するとセキュリティ上の懸念が生ずる場合、メッセージサーバーとして動作する別のデータベースを設定します。

2. 統合データベースの負荷が大きい場合、ここにメッセージサーバー機能を追加すると、統合データベースを実行したときにコンピュータ上のリソースに過剰な負荷がかかるかもしれません。

メッセージサーバーとして動作するデータベースに必要なデータベースオブジェクトを設定するには、データベースの SQL Remote 定義を問い合わせる sr_add_message_server ストアドプロシージャーを呼び出します。 「sr_add_message_server システムプロシージャー」206 ページを参照してください。

メッセージサーバーを別のデータベースとして作成する場合、統合データベースの定義と一致する SQL Remote 定義で 2 番目のデータベースを定義する必要があります。dbunload ユーティリティを使用して、統合データベースのコピーを作成し、-n スイッチを指定してデータではなく統合データベースのスキーマだけをアンロードします。

```
dbunload -n -an -c "ENG=cons.DBN=cons;UID=DBA;PWD=sql"
```

メッセージサーバーとして別のデータベースを使用している場合、統合データベースの SQL Anywhere 定義に変更が行われた場合には、メッセージサーバーデータベースにも対応する変更が行われる必要があります。

メッセージサーバーを設定するには、SQL Remote メッセージが格納されているディレクトリにデータベースサーバーがアクセスできる必要があります。メッセージが格納されているディレクトリを定義するには、SET REMOTE OPTION コマンドを使用して、**root_directory** HTTP メッセージパラメーターを SQL Remote メッセージが格納されているディレクトリに設定します。次に、作成される新しいオブジェクトを所有するデータベースユーザーを選択し、そのユーザーがグループであることを確認します。最後に、sr_add_message_server ストアドプロシージャーを実行して、そのオブジェクトを所有するユーザーの名前を渡します。 「SET REMOTE OPTION 文 [SQL Remote]」213 ページと「sr_add_message_server システムプロシージャー」206 ページを参照してください。

(リモートユーザーの追加や削除など) メッセージサーバーの SQL Remote 定義に変更が行われるたびに、sr_update_message_system ストアドプロシージャーを実行して、メッセージサーバーのサポートに必要なオブジェクトの定義を更新します。ストアドプロシージャーが稼働し、オブジェクトが削除されて再作成される間、短期間メッセージサーバーはレプリケーションに利用できなくなります。 「sr_update_message_server システムプロシージャー」207 ページを参照してください。

このデータベースをもうメッセージサーバーとして使用していない場合、sr_drop_message_system ストアドプロシージャーを実行して、メッセージサーバーをサポートするために作成されたオブジェクトを削除できます。 「sr_drop_message_server システムプロシージャー」207 ページを参照してください。

メッセージサーバーデータベースサーバーの起動

メッセージサーバーをサポートするのに必要なオブジェクトが作成された後、メッセージサーバーデータベースサーバーを起動したときに、-xs スイッチを使用してデータベースサーバーへの HTTP (か HTTPS またはその両方) のサポートを有効にする必要があります。 -xs の使用の詳細については、「-xs dbeng12/dbsrv12 サーバーオプション」『SQL Anywhere サーバー データベース管理』を参照してください。 メッセージサーバーに必要なオブジェクトを定義した人々に関係する HTTP サーバー側プロトコルオプションには次のものがあります。

- **ServerPort | PORT** デフォルトポートである 80 と 443 がすでにコンピュータ上で使用されている場合に、HTTP または HTTPS 要求を受信するためにデータベースサーバーが使用するポート番号を指定します。 「ServerPort (PORT) プロトコルオプション」『SQL Anywhere サーバー データベース管理』を参照してください。

- **MaxRequestSize | MAXSIZE** 1 つの HTTP 要求の最大サイズを指定します。デフォルト値は 100 KB です。SQL Remote メッセージサイズ (dbremote コマンドライン上の -l オプション) を 100 KB を超える値に定義した場合、データベースサーバーが受信できる最大 HTTP 要求のサイズも増やす必要があります。デフォルトの SQL Remote メッセージサイズは 50 KB です。 「MaxRequestSize (MAXSIZE) プロトコルオプション」『SQL Anywhere サーバー データベース管理』を参照してください。

- **Identity (HTTP のみ)** HTTPS を使用している場合、ID ファイルには、パブリック証明書とプライベートキーが含まれており、自己署名されない証明書の場合は、さらに署名を行うすべての証明書も含まれています。これには暗号化証明書も含まれます。この証明書のパスワードは、**Identity_Password** パラメーターで指定してください。 「Identity プロトコルオプション」『SQL Anywhere サーバー データベース管理』を参照してください。

- **Identity_Password (HTTP のみ)** トランスポートレイヤーセキュリティを使用している場合、このオプションにより、Identity プロトコルオプションで指定した暗号化証明書に対応するパスワードを指定します。「Identity_Password プロトコルオプション」『SQL Anywhere サーバー データベース管理』を参照してください。

HTTP アドレスとユーザー ID

SQL Remote と HTTP を使用するには、システムに参加する各データベースに HTTP アドレス、ユーザー ID、パスワードが必要です。これらは別々の識別子です。HTTP アドレスは各メッセージの送信先で、ユーザー ID とパスワードは、サーバーに認証を行うときにユーザーが入力する名前とパスワードです。

HTTP メッセージ制御パラメーター

SQL Remote 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、ユーザーは制御パラメーターのセットを自分のコンピューター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、ユーザー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として使用されます。

HTTP メッセージシステムでは、SET REMOTE OPTION 文で設定される次の制御パラメーターを使用します。

- **certificate** 安全な (HTTPS) 要求を行うには、HTTPS サーバーで使用される証明書にクライアントがアクセスできる必要があります。必要な情報は、セミコロンで区切られたキー／値

のペアの文字列で指定されます。ファイルキーを使用して証明書のファイル名を指定できます。ファイルキーと証明書キーを一緒に指定することはできません。次のキーを使用できます。

キー	省略形	説明
file		証明書のファイル名
certificate	cert	証明書自体
company	co	証明書で指定された会社
unit		証明書で指定された会社の部署
name		証明書で指定された通称

証明書は、HTTPS サーバーに対する要求、または安全でないサーバーから安全なサーバーにリダイレクトされる可能性がある要求に対してのみ必要です。PEM でフォーマットされた証明書のみがサポートされています。**certificate='***file=filename***'**

- **client_port**　SQL Remote が HTTP を使用して通信するポート番号を識別します。これは「送信」TCP/IP 接続をフィルターするファイアウォール経由で接続するためのもので、この用途にかぎり推奨されています。単一のポート番号、ポート番号の範囲、または両方の組み合わせを指定できます。指定したクライアントポートの数が少ないと、SQL Remote が前回の実行でポートを閉じた後すぐにオペレーティングシステムがポートを解放しなかった場合に、SQL Remote でメッセージの送受信ができなくなる可能性があります。
  client_port=*nnnnn[-mmmmm]***'**

- **debug**　YES に設定すると、すべての HTTP コマンドと応答が出力ログに表示されます。この情報は、HTTP 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。

- **https**　HTTPS (**https=yes**) または HTTP (**https=no**) を使用するかどうかを指定します。

- **password**　メッセージサーバーのデータベースパスワード。RFC 2617 の基本認証を使用してサードパーティの HTTP サーバーとゲートウェイに対する認証を行います。
  password='*password***'**

- **proxy_host**　プロキシサーバーの URI を指定します。SQL Remote がプロキシサーバーを介してネットワークにアクセスする場合に使用します。SQL Remote がプロキシサーバーに接続し、そのプロキシサーバーを介してメッセージサーバーに要求を送信することを示します。**proxy_host='** *http://proxy-server***[:***port-number***]'**

- **reconnect_retries**　失敗になる前に、リンクがサーバー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。このパラメーターを設定すると、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。このパラメーターを設定すると、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。
- **root_directory**　この HTTP 制御パラメーターは、クライアント側で指定された場合には無視されます。sr_add_message_server または sr_update_message_server ストアドプロシージャーを呼び出す前に、メッセージサーバーでこの制御パラメーターを定義します。メッセージサーバーがアクセスできる SQL Remote メッセージの格納されているディレクトリを指定します。HTTP メッセージシステムを使用する場合、リモートユーザーまたはパブリッシャーに指定するアドレスには、1 つのサブディレクトリのみを含めることができます。複数のサブディレクトリを含めることはできません。**root_directory='***c:¥msgs***'**
- **url**　サーバー名または IP アドレスを指定し、任意で、使用する HTTP サーバーのポート番号をセミコロンで区切って指定します。要求が Relay Server を経由する場合は、URL 拡張子を任意で追加して、要求の渡し先のサーバーファームを示すことができます。**url ='***server-name***[:***port-number***][***url-extension***]'**
- **user**　メッセージサーバーのデータベースユーザー ID。RFC 2617 の基本認証を使用してサードパーティの HTTP サーバーとゲートウェイに対する認証を行います。**user='***userid***'**

参照

- 「FILE メッセージシステム」 110 ページ
- 「SMTP メッセージシステム」 118 ページ
- 「FTP メッセージシステム」 111 ページ
- 「CREATE REMOTE MESSAGE TYPE 文 [SQL Remote]」『SQL Anywhere サーバー SQL リファレンス』
- 「チュートリアル : HTTP メッセージシステムを使用したレプリケーションシステムの設定」 155 ページ
- 「チュートリアル : 統合データベースをメッセージサーバーとして HTTP メッセージシステムを使用したレプリケーションシステムの設定」 165 ページ
- 「チュートリアル : HTTP メッセージシステムを使用し、Relay Server を経由して統合データベースをメッセージサーバーとしたレプリケーションシステムの設定」 173 ページ
- 「SET REMOTE OPTION 文 [SQL Remote]」 213 ページ

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 サーバーを使用して送信され、POP サーバーから受信されます。

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 12 紹介』を参照してください。

SMTP アドレスとユーザー 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 ユーザー 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 ユーザー ID とパスワードは、ユーザーが自分の電子メールサーバーに接続するときに入力する名前とパスワードです。

注意
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 アドレス共有」120 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ー

SQL Remote 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、ユーザーは制御パラメーターのセットを自分のコンピューター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、ユーザー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、SET REMOTE OPTION 文で設定される次の制御パラメーターを使用します。

- **local_host** ローカルコンピューターの名前。SQL Remote がローカルホスト名を決定できないコンピューターで設定すると便利です。ローカルホスト名は、任意の SMTP サーバー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 サーバーでサポートされているわけではありません。top_supported パラメーターを 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 サーバーで動作します。デフォルトは YES です。

- **smtp_authenticate** SMTP リンクがユーザー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、このパラメーターを NO に設定します。

- **smtp_userid** SMTP 認証のユーザー ID。デフォルトでは、このパラメーターは pop3_userid パラメーターと同じ値を取ります。smtp_userid は、ユーザー ID が POP サーバー上のユーザー 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、このパラメーターは pop3_password パラメーターと同じ値を取ります。smtp_password は、ユーザー ID が POP サーバー上のユーザー ID と異なる場合にのみ設定する必要があります。

- **smtp_host** SMTP サーバーが動作しているコンピューターの名前。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ーの名前。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ー ID。POP ユーザー ID は、SMTP/POP3 ログインウィンドウのユーザー 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ー 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出力ログに表示されます。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**　このパラメーターが true に設定されている場合は、メールサーバー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」103 ページを参照してください。
- **max_retries**　デフォルトでは、SQL Remote が継続モードで実行されていて、メッセージシステムにアクセスするときにエラーが発生する場合、送受信フェーズの後に停止します。このパラメーターを使用して、停止前に SQL Remote が再試行する送受信フェーズの回数を指定します。
- **pause_after_failure**　このパラメーターが使えるのは、max_retries がゼロ以外の値に指定され、SQL Remote が継続モードで実行されている場合です。メッセージシステムでエラーが発生すると、このパラメーターは、SQL Remote が送受信フェーズを再試行する間に待機する時間を秒で指定します。

参照

- 「FILE メッセージシステム」110 ページ
- 「FTP メッセージシステム」111 ページ
- 「CREATE REMOTE MESSAGE TYPE 文 [SQL Remote]」『SQL Anywhere サーバー SQL リファレンス』
- 「SET REMOTE OPTION 文 [SQL Remote]」213 ページ

SMTP/POP アドレス共有

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ー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、SQL Remote Message Agent (dbremote) が稼働している同じコンピューターから、同じアカウントとパスワードを使用して SMTP/POP3 サーバーに接続します。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 台のコンピューター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、SQL Remote 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。

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。

バックアップユーティリティ (dbbackup) と SQL Remote 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 SQL Remote 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、SQL Remote 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 SQL Remote 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 SQL Remote Message Agent (dbremote) のみを実行してください。

参照

- 「リモートデータベースのトランザクションログの管理」122 ページ
- 「リモートデータベースのバックアップ」123 ページ
- 「メディア障害からの統合データベースの保護」124 ページ

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 SQL Remote 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

1. リモートデータベースで、-x オプションを指定して SQL Remote 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、SQL Remote 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、SQL Remote 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ー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 SQL Remote Message Agent (dbremote) の接続文字列に含まれるユーザー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 SQL Remote 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

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 権限のあるユーザーとして、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:¥archive* 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* フォルダーにあります。

      ```
      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 SQL Remote 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
      ```

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 SQL Remote 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 権限のあるユーザーとして、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 SQL Remote 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 SQL Remote Message Agent (dbremote) を実行する。SQL Remote 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。

- **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。

- **統合データベースで -x オプションを指定して SQL Remote 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 SQL Remote 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。

参照

- 「SQL Remote Message Agent ユーティリティ (dbremote)」183 ページ
- 「トランザクションログミラー」『SQL Anywhere サーバー データベース管理』
- 「統合データベースのバックアップ」124 ページ

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

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 権限のあるユーザーとして、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 権限のあるユーザーとして、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
      ```

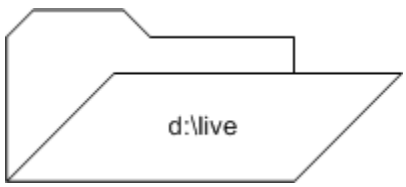
3. 現在のトランザクションログファイルにアクセスしながら SQL Remote Message Agent (dbremote) を実行します。

   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 SQL Remote 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。



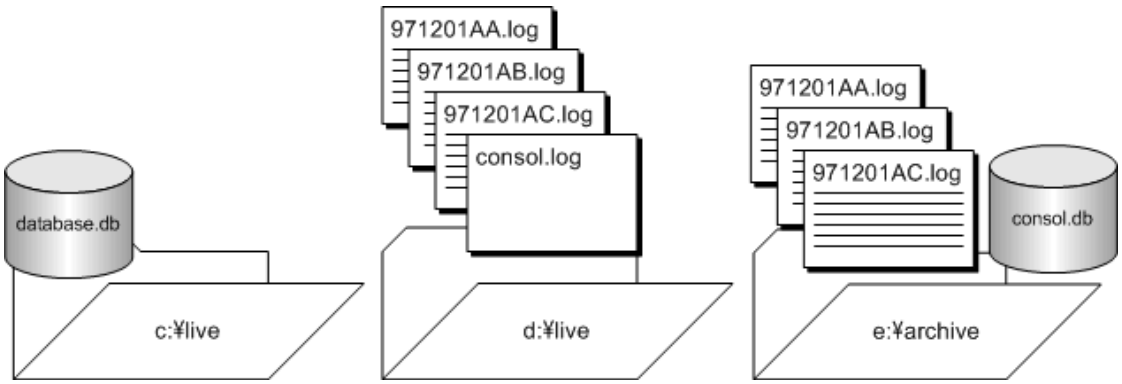
トランザクションログの名前を変更して再起動する -r オプションと -n オプションを使用して、バックアップディレクトリ *e:¥archive* にトランザクションログをバックアップすると、バックアップユーティリティ (dbbackup) が次のタスクを実行します。

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フォルダーにバックアップする。

注意
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。SQL Remote Message Agent (dbremote) は、指定されたフォルダー内で、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 フォルダーと archive フォルダー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」 124 ページ
- 「トランザクションログバックアップコピーに名前を付ける」『SQL Anywhere サーバー データベース管理』
- 「バックアップユーティリティ (dbbackup)」『SQL Anywhere サーバー 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。

♦ -a オプションを使用してデータベースをリカバリする

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 SQL Remote 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。

♦ C ドライブのメディア障害からリカバリする

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。

```
dbeng12 -a c:¥dbdir¥dateAA.log c:¥recover¥cons.db
dbeng12 -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。

   ```
   dbeng12 c:¥recover¥cons.db
   ```

7. データベースサーバーを停止します。
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

- 「統合データベースの自動リカバリ」 128 ページ
- 「-a dbeng12/dbsrv12 データベースオプション」『SQL Anywhere サーバー 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」 126 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

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ー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ーを停止します。

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 SQL Remote 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

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。

```
dbeng12 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。

   ```
   dbeng12 c:¥recover¥cons.db
   ```

9. データベースサーバーを停止します。

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 dbeng12/dbsrv12 データベースオプション」『SQL Anywhere サーバー 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」50 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」51 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」53 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、SQL Remote Message Agent (dbremote) はエラーをログ出力ウィンドウに出力します。SQL Remote 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。

SQL Remote 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 SQL Remote Message Agent (dbremote) を実行します。

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ーの実行**　デフォルトではプロシージャーを呼び出しません。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 SQL Remote Message Agent (dbremote) で呼び出すストアドプロシージャーを指定できます。

  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。

- **エラーの無視**　SQL Remote 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」135 ページを参照してください。

参照

- 「SQL Remote Message Agent ユーティリティ (dbremote)」183 ページ
- 「エラー処理プロシージャーの実行」131 ページ
- 「リモートデータベースからのエラーの収集」131 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」132 ページ

エラー処理プロシージャーの実行

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ーを呼び出します。デフォルトでは、SQL エラーの発生時にプロシージャーを呼び出しません。

呼び出すプロシージャーでは、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。プロシージャーは、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 オプションを設定する

- 次の文を実行します。*remote-user* は SQL Remote Message Agent (dbremote) コマンドのパブリッシャー名、*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ーです。

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

参照

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

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

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';
```

SQL Remote 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。

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';
```

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 SQL Remote Message Agent (dbremote) を実行します。

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

- 「SET REMOTE OPTION 文 [SQL Remote]」213 ページ
- 「SQL Remote Message Agent ユーティリティ (dbremote)」183 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」132 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 (SQL の場合)

1. **[SQL Anywhere 12]** プラグインを使用して、ユーザー 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 ユーザーに通知するストアドプロシージャーを作成します。

   たとえば、次の文を実行して sp_LogReplicationError プロシージャーを作成します。

```
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 SQL Remote 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ーを作成します。

   たとえば、次の文を実行して sp_notifiy_DBA プロシージャーを作成します。

```
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ーを呼び出す 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 プロシージャーを呼び出します。

```
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ー (パブリッシャー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ー (リモートユーザーのデータベースを識別する)。
errormsg	replication_error プロシージャーに渡されたエラーメッセージ。

6. プロシージャーをテストします。

   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 SQL Remote Message Agent (dbremote) が、その出力ログに次のメッセージを出力する。

   ```
   "cons" (0-0000000000-0) メッセージを受信しました。
   SQL 文が失敗しました : (-193) テーブル 'reptable' のプライマリキーがユニークではありません。
   INSERT INTO cons.reptable(id,text,last_contact)
   VALUES (2,'dave','1997/apr/21 16:02:38.325')
   COMMIT WORK
   ```

   ●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」131 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 (SQL の場合)**

- 既知のエラーの原因となる動作に BEFORE トリガーを作成します。このトリガーは、エラーを通知するものです。

  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 SQLE_REMOTE_STATEMENT_FAILED SQLSTATE を通知する BEFORE INSERT トリガーを作成します。INSERT 文は失敗しますが、この失敗は SQL Remote Message Agent (dbremote) の出力ログにはレポートされません。

参照

- 「リモートの文が失敗しました。」『エラーメッセージ』

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 権限を持つユーザーを使用して SQL Remote Message Agent (dbremote) に接続することをおすすめします。
- **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)」194 ページを参照してください。
- **メッセージの暗号化**　SQL Remote 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 サーバー データベース管理』を参照してください。

参照

- 「REMOTE DBA 権限」30 ページ

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 12 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。

  - **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。

参照

- 「PASSTHROUGH 文 [SQL Remote]」『SQL Anywhere サーバー SQL リファレンス』
- 「SQL Remote のパススルーモード」137 ページ
- 「サブスクリプションの再同期」140 ページ

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。

- **パブリッシャーの変更**　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ーのユーザー名を変更すると、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ーのユーザー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ー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ーのユーザー名を変更すると、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ーのユーザー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ーを再同期します。

- **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、新しいユーザーの追加、ユーザーの再同期、ユーザーの削除、およびリモートユーザーのアドレス、メッセージタイプ、送信頻度の変更があります。

- **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。
- **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。
- **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」 80 ページを参照してください。

参照

- 「ALTER PUBLICATION 文 [Mobile Link] [SQL Remote]」『SQL Anywhere サーバー SQL リファレンス』
- 「SQL Remote のパススルーモード」 137 ページ
- 「PASSTHROUGH 文 [SQL Remote]」『SQL Anywhere サーバー SQL リファレンス』
- 「パススルーモードの制限事項」 138 ページ
- 「サブスクリプションの再同期」 140 ページ
- 「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 サーバー SQL の使用法』

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。

- 新しいユーザーを追加する。
- ユーザーを再同期する。
- システムからユーザーを削除する。
- リモートユーザーのアドレス、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。
- 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、同じユーザー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。

参照

- 「実行中のシステムに行ってはならない変更」136 ページ
- 「PASSTHROUGH 文 [SQL Remote]」『SQL Anywhere サーバー SQL リファレンス』

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。

- **プロシージャーの呼び出し**　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ーを呼び出すと、次のようになります。

  ○プロシージャーが統合データベースで実行されない場合でも、プロシージャー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。

  ○プロシージャーは、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、プロシージャー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ー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。

- **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 サーバー 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

- 「PASSTHROUGH 文 [SQL Remote]」『SQL Anywhere サーバー SQL リファレンス』
- 「制御文」『SQL Anywhere サーバー SQL の使用法』
- 「サブスクリプションの再同期」140 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には PASSTHROUGH 文を使用し、停止には 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ー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ー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。

●サブスクライバー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ー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ーのパススルーセッションを停止します。

```
PASSTHROUGH STOP;
```

データ操作言語 (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 FOR 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 FOR Joe_Remote;
DROP TABLE CrucialData;
PASSTHROUGH STOP;
```

参照

- 「PASSTHROUGH 文 [SQL Remote]」『SQL Anywhere サーバー SQL リファレンス』
- 「データ修正文」『SQL Anywhere サーバー 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。

  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。

参照

- 「SQL Remote のパススルーモード」 137 ページ
- 「PASSTHROUGH 文 [SQL Remote]」『SQL Anywhere サーバー SQL リファレンス』
- 「SYNCHRONIZE SUBSCRIPTION 文 [SQL Remote]」『SQL Anywhere サーバー SQL リファレンス』
- 「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 サーバー SQL の使用法』
- 「アンロードユーティリティ (dbunload)」『SQL Anywhere サーバー データベース管理』

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、SQL Remote 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 (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 SQL Remote Message Agent を停止します。
2. **[SQL Anywhere 12]** プラグインを使用して、DBA 権限のあるユーザーとして、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クリックします。
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. **[サブスクライバー]** リストでユーザーを右クリックして、**[プロパティ]** をクリックします。
   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。

参照

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

SQL Remote 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」 141 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。

♦ 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 (Interactive SQL の場合)

1. DBA 権限のあるユーザー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。

   統合データベースの SQL Remote 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ーに送信します。SQL Remote 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。

   サブスクライバーデータベースの SQL Remote 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。

参照

- 「SYNCHRONIZE SUBSCRIPTION 文 [SQL Remote]」『SQL Anywhere サーバー SQL リファレンス』
- 「送信頻度の設定」 86 ページ

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 (Sybase Central の場合)**

1. **SQL Anywhere 12** プラグインを使用して、DBA 権限のあるユーザーとして、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クリックします。
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. **[サブスクライバー]** リストでユーザーを右クリックして、**[プロパティ]** をクリックします。
   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 (SQL の場合)**

1. DBA 権限のあるユーザーとして、データベースに接続します。
2. START SUBSCRIPTION 文を実行します。

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。

参照

- 「START SUBSCRIPTION 文 [SQL Remote]」『SQL Anywhere サーバー SQL リファレンス』
- 「REMOTE RESET 文 [SQL Remote]」『SQL Anywhere サーバー SQL リファレンス』

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 (Sybase Central の場合)**

1. **SQL Anywhere 12** プラグインを使用して、DBA 権限のあるユーザーとして、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. 目的のパブリケーションをクリックします。
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、[サブスクライバー] リスト内のユーザーをクリックし、**[プロパティ]** をクリックします。

   [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。

チュートリアル：SQL Remote システムの作成

このチュートリアルのレッスンでは、SQL Anywhere の統合データベースとリモートデータベースの両方を使用する SQL Remote レプリケーションシステムの設定について学びます。

このチュートリアルでは、次のことを学習します。

- SQL Anywhere 統合データベースと、統合データベース内のデータのサブセットを含む SQL Anywhere リモートデータベースを作成します。
- 1 つの SQL Anywhere リモートデータベースを含む、ファイル共有レプリケーションシステムを作成する。
- 統合データベースとリモートデータベース間でデータをレプリケートする。

レッスン 1：統合データベースの作成

♦ チュートリアル用の統合データベースとディレクトリを作成する

1. ディレクトリ *c:¥tutorial*、*c:¥tutorial¥hq*、および *c:¥tutorial¥field* を作成します。
2. *c:¥tutorial* ディレクトリから次のコマンドを実行して、統合データベースを作成します (hq)。

   ```
   dbinit hq.db
   ```

3. Interactive SQL から統合データベース (hq) に接続します。

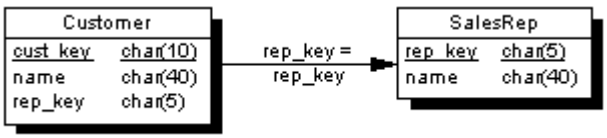
   ```
   dbisql -c "UID=DBA;PWD=sql;SERVER=server_hq;DBF=c:¥tutorial¥hq.db"
   ```

4. 次の文を実行して、統合データベース (hq) 内に 2 つのテーブルを作成します。

   ```
   CREATE TABLE SalesReps (
    rep_key CHAR(12) NOT NULL,
    name CHAR(40) NOT NULL,
    PRIMARY KEY ( rep_key )
   );

   CREATE TABLE Customers (
    cust_key CHAR(12) NOT NULL,
    name CHAR(40) NOT NULL,
    rep_key CHAR(12) NOT NULL,
    FOREIGN KEY ( rep_key )
     REFERENCES SalesReps (rep_key ),
    PRIMARY KEY (cust_key)
   );
   ```

   次の数字は、チュートリアル用の統合データベース (hq) スキーマを示します。



- 各営業担当は SalesReps テーブルの 1 つのローで表されています。
- 各顧客は Customers テーブルの 1 つのローで表されています。
- 各顧客は 1 人の営業担当に割り当てられており、この割り当ては Customers テーブルから SalesReps テーブルへの外部キーとしてデータベースに組み込まれています。Customers テーブルと SalesReps テーブルとの関係は、多対 1 です。

テーブル名	説明
SalesRep	SalesReps テーブルには、会社の営業担当それぞれに 1 つのローがあります。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。
Customers	Customers テーブルには、会社と取引がある顧客それぞれに 1 つのローがあります。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。

5. 次の文を実行して、SalesReps および Customers テーブルにサンプルデータを追加します。

```
INSERT INTO SalesReps (rep_key, name)
VALUES ('rep1', 'Field User');
INSERT INTO SalesReps (rep_key, name)
VALUES ('rep2', 'Another User');
COMMIT;

INSERT INTO Customers (cust_key, name, rep_key)
VALUES ('cust1', 'Ocean Sports', 'rep1' );
INSERT INTO Customers (cust_key, name, rep_key)
VALUES ('cust2', 'Sports Plus', 'rep2' );
COMMIT;
```

6. 次の文を実行して、テーブルが作成されたことを確認します。

```
SELECT * FROM SalesReps;
```

上記のクエリは、SalesRep テーブルから次のデータを返します。

rep_key	name
rep1	Field User
rep2	Another User

```
SELECT * FROM Customers;
```

上記のクエリは、Customers テーブルから次のデータを返します。

cust_key	name	rep_key
cust1	Ocean Sports	rep1
cust2	Sports Plus	rep2

レッスン 2：統合データベースでの PUBLISH パーミッションと REMOTE パーミッションの付与

SQL Remote システムのすべてのデータベースには、パブリッシャーが必要です。パブリッシャーは、PUBLISH パーミッションを持つユニークなユーザーです。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、パブリッシャー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。

♦ 統合データベースのパブリッシャーを作成する (Interactive SQL の場合)

1. 現在、統合データベース (hq) に接続されていない場合は、次のコマンドを実行します。

   ```
   dbisql -c "UID=DBA;PWD=sql;SERVER=server_hq;DBF=c:¥tutorial¥hq.db"
   ```

2. 次の文を実行して、CONNECT パーミッションおよび PUBLISH パーミッションを持つユーザー hq_user を作成します。

   ```
   CREATE USER hq_user IDENTIFIED BY hq_pwd;
   GRANT CONNECT TO hq_user IDENTIFIED BY hq_pwd;
   GRANT PUBLISH TO hq_user;
   ```

3. 次の文を実行して、データベースのパブリッシュユーザー ID を確認します。

   ```
   SELECT CURRENT PUBLISHER;
   ```

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ーに 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。

♦ REMOTE パーミッションを付与する

● 次の文を実行して、CONNECT パーミッションおよび PUBLISH パーミッションを持つユーザー field_user とパスワード field_pwd を作成します。

```
CREATE USER field_user IDENTIFIED BY field_pwd;
GRANT CONNECT TO field_user IDENTIFIED BY field_pwd;
GRANT REMOTE TO field_user
TYPE file
ADDRESS 'field';
```

レッスン 3：パブリケーションとサブスクリプションの作成

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説明します。このレッスンでは、SalesRepData というパブリケーションを作成します。これは、SalesReps テーブルのすべてのローと、Customers テーブルのいくつかのローをレプリケートします。パブリケーションに対してユーザー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。

♦ 統合データベースでパブリケーションを作成する (Interactive SQL の場合)

1. 現在、統合データベース (hq) に接続されていない場合は、次のコマンドを実行します。

   ```
   dbisql -c "UID=DBA;PWD=sql;SERVER=server_hq;DBF=c:¥tutorial¥hq.db"
   ```

2. 次の文を実行して、SalesRepData というパブリケーションを作成します。

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

   SalesRepData パブリケーションは以下をパブリッシュします。

   ●SalesReps テーブル全体

   ●指定された rep_key 値と一致するローを除くすべての Customers テーブルのカラム

3. 次の文を実行して、SalesRepData へのサブスクリプションを作成します。

   ```
   CREATE SUBSCRIPTION
   TO SalesRepData ('rep1')
   FOR field_user;
   ```

   値 rep1 は、SalesReps テーブルのユーザー Field User の rep_key の値です。

> **注意**
> このチュートリアルでは、プライマリキーの値が重複したエントリを防ぐようにはなっていません。 詳細については、「SQL Remote システムの作成」9 ページを参照してください。

レッスン 4：SQL Remote のメッセージタイプの作成

データやメッセージを送信するときに使用するメッセージタイプを定義します。レプリケーションの一部として送信されるすべてのメッセージは、メッセージタイプを使用します。メッセージタイプの記述は、次の 2 つの部分で構成されます。

- **SQL Remote でサポートされているメッセージシステム**　このチュートリアルでは、FILE メッセージシステムを使用します。FILE メッセージシステムは、単純なファイル共有システムです。

- **FILE アドレス**　ユーザーの FILE アドレスは、すべての着信メッセージが送信されるサブディレクトリです。アプリケーションはこのディレクトリからメッセージを検索します。このチュートリアルでは、統合データベースの FILE アドレスは hq で、*c:¥tutorial* のサブディレクトリです。

♦ メッセージタイプを作成する (Interactive SQL の場合)

1. 現在、統合データベース (hq) に接続されていない場合は、次のコマンドを実行します。

   ```
   dbisql -c "UID=DBA;PWD=sql;SERVER=server_hq;DBF=c:¥tutorial¥hq.db"
   ```

2. 次の文を実行して、FILE メッセージタイプを作成します。

   ```
   CREATE REMOTE MESSAGE
   TYPE file
   ADDRESS 'hq';
   ```

レッスン 5 ：リモートデータベースの抽出

このレッスンでは、統合データベースからリモートデータベースを抽出して、リモートユーザー用のデータベースを作成します。

メッセージの送受信を行い SQL Remote システムに組み込まれるように、リモートデータベースを設定する必要があります。統合データベース (hq) と同様に、出力メッセージのソースを識別するためにリモートデータベースには CURRENT PUBLISHER が必要です。また、サブスクライバーとして識別された統合データベース (hq) も必要です。

dbxtract ユーティリティを実行して、以下を含むリモートデータベースを作成します。

●統合データベースに対するサブスクリプション

●パブリケーション

●データの現在のコピー

♦ リモートデータベースを抽出する

1. *c:¥tutorial* ディレクトリから次のコマンドを実行して、統合データベース (hq) からユーザー field_user のリモートデータベースのスキーマを抽出します。

```
dbxtract -v -c "UID=DBA;PWD=sql;SERVER=server_hq;DBF=C:¥tutorial¥hq.db" c:¥tutorial
field_user
```

このコマンドは以下を実行します。

- 現在のディレクトリに、*reload.sql* という名前の SQL スクリプトファイルを作成します。*reload.sql* ファイルには、新しいデータベースにロードするためのスキーマと命令が含まれています。
- *c:¥tutorial* ディレクトリにデータファイルを作成します。
- リモートユーザーに対するサブスクリプションを開始します。

2. *c:¥tutorial* ディレクトリから次のコマンドを実行して、リモートデータベースを作成します (field)。

```
dbinit field.db
```

警告
運用環境では、同じディレクトリに 2 つのレプリケートデータベースを格納しないでください。

3. データベース情報をリモートデータベース (field) にロードします。

DBA 権限を持つユーザーとして、Interactive SQL からリモートデータベース (field) に接続します。

```
dbisql -c "UID=DBA;PWD=sql;SERVER=server_field;DBF=c:¥tutorial¥field.db"
```

4. 次の文を実行して、*reload.sql* ファイルを読み込みます。

```
READ C:¥tutorial¥reload.sql;
```

reload.sql スクリプトファイルは以下を実行します。

- リモートデータベースでメッセージタイプを作成します。
- リモートデータベース (field) に PUBLISH パーミッションを付与します。
- リモートデータベース (field) に SalesReps テーブルと Customers テーブルを作成します。これらのテーブルは、統合データベース (hq) 中のデータと同じデータを含みます。
- レプリケートするデータを示すパブリケーションを作成する。
- 統合データベース (hq) 用のサブスクリプションを作成し、開始する。

5. 次の文を実行して、テーブルが作成されたことを確認します。

```
SELECT * FROM SalesReps;
```

上記のクエリは、SalesRep テーブルから次のデータを返します。

rep_key	name
rep1	Field User
rep2	Another User

```
SELECT * FROM Customers;
```

上記のクエリは、Customers テーブルから次のデータを返します。

cust_key	name	rep_key
cust1	Ocean Sports	rep1

レッスン 6：統合データベースからリモートデータベースにデータを送信する

このレッスンでは、データは統合データベース (hq) からリモートデータベース (field) にレプリケートされています。

♦ 統合データベース (hq) でデータを入力する (Interactive SQL の場合)

1. 現在、統合データベース (hq) に接続されていない場合は、次のコマンドを実行します。

   ```
   dbisql -c "UID=DBA;PWD=sql;SERVER=server_hq;DBF=c:¥tutorial¥hq.db"
   ```

2. 次の文を実行して、SalesReps および Customers テーブルにサンプルデータを追加します。

   ```
   INSERT INTO SalesReps ( rep_key, name )
   VALUES ( 'rep3', 'Example User' );

   INSERT INTO Customers ( cust_key, name, rep_key )
   VALUES ( 'cust3', 'Land Sports', 'rep1' );
   INSERT INTO Customers ( cust_key, name, rep_key )
   VALUES ( 'cust4', 'Air Plus', 'rep2' );
   COMMIT;
   ```

3. 次の文を実行して、データが入力されたことを確認します。

   ```
   SELECT * FROM SalesReps;
   SELECT * FROM Customers;
   ```

リモートデータベース (field) にローを送信するには、統合データベース (hq) で Message Agent を実行してください。

♦ 統合データベース (hq) からリモートデータベース (field) へデータを送信する

1. 統合データベース (hq) で *c:¥tutorial* ディレクトリから Message Agent を実行します。

   ```
   dbremote -c "UID=DBA;PWD=sql;SERVER=server_hq;DBF=c:¥tutorial¥hq.db"
   ```

このコマンドは、統合データベース (hq) が現在デフォルトサーバーで実行されていることを前提としています。データベースが実行されていない場合は、DBN パラメーターの代わりに DBF パラメーターを、データベースファイル名とともに指定します。

2. [Message Agent] ウィンドウに **実行が完了しました。** と表示されたら、[シャットダウン] をクリックします。

3. *c:¥tutorial¥field* を参照します。

   ファイル名 *hq.0* がディレクトリにリストされています。このファイルは、統合データベース (hq) から送信された変更を含みます。

レッスン 7：リモートデータベースでのデータの受信

このレッスンでは、リモートデータベース (field) で Message Agent を実行し、統合データベース (hq) から送信されたデータを受信します。

♦ リモートデータベース (field) でデータを受信する

1. 現在、リモートデータベース (field) に接続されていない場合は、次のコマンドを実行します。

   ```
   dbisql -c "UID=DBA;PWD=sql;SERVER=server_field;DBF=c:¥tutorial¥field.db"
   ```

2. リモートデータベース (field) で *c:¥tutorial* ディレクトリから Message Agent を実行します。

   ```
   dbremote -c "UID=DBA;PWD=sql;SERVER=server_field;DBF=c:¥tutorial¥field.db;"
   ```

3. [Message Agent] ウィンドウに **実行が完了しました。** と表示されたら、[シャットダウン] をクリックします。

   c:¥tutorial¥field¥hq.0 ファイルは *c:¥tutorial¥hq¥field.0* というファイルに置き換えられました。*field.0* ファイルは受信確認を含みます。

4. 次の方法でリモートデータベース (field) がデータを含んでいることを確認してください。
   a. 次の文を実行して、SalesReps テーブルをの内容を表示します。

      ```
      SELECT * FROM SalesReps;
      ```

      統合データベース (hq) で入力したローが両方とも SalesReps テーブルにあります。これは、SalesRepsData パブリケーションに SalesReps テーブルからのすべてのデータが含まれていたからです。

rep_key	name
rep1	Field User
rep2	Another User
rep3	Example User

b. 次の文を実行して、Customers テーブルをの内容を表示します。

```
SELECT * FROM Customers;
```

Customers テーブルには現在、統合データベース (hq) で入力された Land Sports の顧客データを持つローも含まれています。

cust_key	name	rep_key
cust1	Ocean Sports	rep1
cust3	Land Sports	rep1

5. 統合データベース (hq) で *c:¥tutorial* ディレクトリから Message Agent を実行します。

```
dbremote -c "UID=DBA;PWD=sql;SERVER=server_hq;DBF=c:¥tutorial¥hq.db"
```

c:¥tutorial¥hq ディレクトリで、ファイル *field.0* が消去されます。

レッスン 8：リモートデータベースから統合データベースにデータを送信する

このレッスンでは、リモートデータベース (field) から統合データベース (hq) にデータを送信します。

♦ リモートデータベース (field) から統合データベース (hq) にデータをレプリケートする (Interactive SQL の場合)

1. 現在、リモートデータベース (field) に接続されていない場合は、次のコマンドを実行します。

```
dbisql -c "UID=DBA;PWD=sql;SERVER=server_hq;DBF=c:¥tutorial¥field.db"
```

2. 次の文を実行し、リモートデータベース (field) でローを挿入します。

```
INSERT INTO Customers ( cust_key, name, rep_key )
VALUES ( 'cust5', 'North Land Trading', 'rep1' );
COMMIT;
```

3. *c:¥tutorial* ディレクトリから、リモートデータベース (field) に対して dbremote ユーティリティを実行します。

```
dbremote -c "UID=DBA;PWD=sql;SERVER=server_field;DBF=c:¥tutorial¥field.db"
```

c:¥tutorial¥hq ディレクトリに、ファイル *field.1* が表示されます。

♦ 統合データベース (hq) でデータを受信する

1. 現在、統合データベース (hq) に接続されていない場合は、次のコマンドを実行します。

```
dbisql -c "UID=DBA;PWD=sql;SERVER=server_hq;DBF=c:¥tutorial¥hq.db
```

2. 統合データベース (hq) で *c:¥tutorial* ディレクトリから Message Agent を実行します。

```
dbremote -c "UID=DBA;PWD=sql;SERVER=server_hq;DBF=c:¥tutorial¥hq.db"
```

3. [Message Agent] ウィンドウに **実行が完了しました。** と表示されたら、[シャットダウン] をクリックします。
4. *c:¥tutorial¥field* を参照します。

   hq.1 ファイルは *hq.2* というファイルに置き換えられました。*hq.2* ファイルは受信確認を含みます。
5. 次の文を実行して、統合データベース (hq) 内の Customers テーブルのデータを表示します。

   ```
   SELECT * FROM Customers;
   ```

   このクエリは、次の結果を返します。

cust_key	name	rep_key
cust1	Ocean Sports	rep1
cust2	Sports Plus	rep2
cust3	Land Sports	rep1
cust4	Air Plus	rep2
cust5	North Landing Trading	rep1

チュートリアル：HTTP メッセージシステムを使用したレプリケーションシステムの設定

このチュートリアルのレッスンでは、SQL Anywhere の統合データベースとリモートデータベースの両方を使用する SQL Remote レプリケーションシステムの設定について学びます。統合データベースは、FILE メッセージシステムを使用して変更をレプリケートし、リモートデータベースは、HTTP メッセージシステムを使用して変更をレプリケートします。

このチュートリアルでは、次のことを学習します。

- SQL Anywhere 統合データベースと、統合データベース内のすべてのデータを含む SQL Anywhere リモートデータベースを作成する。
- SQL Remote によって生成されたメッセージを格納するディレクトリ構造を作成する。統合データベースは、FILE メッセージシステムを使用してファイルにアクセスし、リモートデータベースは HTTP メッセージシステムを使用する。
- HTTP プロトコルを使用してリモートデータベースからメッセージを受信する Web サーバーとして動作するメッセージサーバー SQL Anywhere データベースを作成する。
- 統合データベースとリモートデータベース間でデータをレプリケートする。

レッスン 1：統合データベースの作成

このレッスンでは、データベースとそのトランザクションログを格納するのに必要なディレクトリ、そしてメッセージのディレクトリ構造を作成します。また、リモートユーザーおよびデータのレプリケートに必要なパブリケーションとサブスクリプションなど、統合データベースのスキーマを定義します。SQL Remote が統合データベースに対して稼働する場合、FILE メッセージシステムを使用してメッセージを送受信しますが、リモートデータベースは HTTP メッセージシステムを使用します。

♦ チュートリアル用の統合データベースとディレクトリを作成する

1. 統合データベース、リモートデータベース、メッセージサーバーデータベースを格納するために、次のディレクトリを作成します。
   - *c:¥tutorial*
   - *c:¥tutorial¥cons*
   - *c:¥tutorial¥rem*
   - *c:¥tutorial¥msgsrv*
2. 統合データベースとリモートデータベースが生成したメッセージファイルを格納するために、次のディレクトリを作成します。
   - *c:¥tutorial¥messages*
   - *c:¥tutorial¥messages¥cons*
   - *c:¥tutorial¥messages¥rem*

3. *c:¥tutorial¥cons* ディレクトリから次のコマンドを実行して、統合データベース (cons) を作成します。

   ```
   dbinit cons.db
   ```

4. Interactive SQL を使用して、DBA 権限のあるユーザーとして統合データベース (cons) に接続し、AutoStop 接続パラメーターに対して AutoStop=NO を指定して切断を行ってもデータベースは稼働状態となるようにします。

   ```
   dbisql -c "UID=DBA;PWD=sql;SERVER=cons;DBF=c:¥tutorial¥cons¥cons.db;autostop=no"
   ```

5. 統合データベース (cons) のグローバルデータベース ID を設定するには、次の文を実行します (GLOBAL AUTOINCREMENT デフォルトを使用する場合には、すべてのデータベースに排他的なプライマリキーが選択されるようグローバルデータベース ID が必要です)。

   ```
   SET OPTION public.global_database_id=0;
   ```

6. このチュートリアルのデータベースのスキーマは、レプリケートする 1 つのテーブルから構成され、テーブルのすべてのカラムとローはすべてのリモートユーザーにレプリケートされます。統合データベース (cons) で次の文を実行し、データベースに 1 つのテーブルを作成します。

   ```
   CREATE TABLE employees (
       employee_id BIGINT NOT NULL DEFAULT GLOBAL AUTOINCREMENT(1000000) PRIMARY
   KEY,
       first_name VARCHAR(128) NOT NULL,
       last_name VARCHAR(128) NOT NULL,
       hire_date TIMESTAMP NOT NULL DEFAULT TIMESTAMP
   );
   ```

7. 統合データベース (cons) で次の文を実行し、サンプルデータを従業員テーブルに追加します。

   ```
   INSERT INTO employees (first_name, last_name) VALUES ('Kelly', 'Meloy');
   INSERT INTO employees (first_name, last_name) VALUES ('Melisa', 'Boysen');
   COMMIT;
   ```

8. 統合データベース (cons) で次の文を実行し、そのテーブルが作成され、データが挿入されていることを確認します。

   ```
   SELECT * FROM employees;
   ```

   クエリは、従業員テーブルから次のデータを返します。ただし、hire_date カラムには、次のテーブルにある値ではなく、そのローを挿入した時間が記載されます。

employee_id	first_name	last_name	hire_date
1	Kelly	Meloy	2011-03-25 08:27:56.310
2	Melisa	Boysen	2011-03-25 08:27:56.310

9. このチュートリアルでは、パブリッシャーとリモートユーザーにはパスワードは割り当てられません。したがって、ユーザーはデータベースに存在しますが、それらのユーザーではデー

タベースに接続できません。次の文を実行して、CONNECT パーミッションおよび PUBLISH パーミッションを持つユーザー cons を作成します。

```
GRANT CONNECT TO cons;
GRANT PUBLISH TO cons;
```

10. パフォーマンス上の理由から、HTTP メッセージシステムはリモートデータベースのみで使用することができ、統合データベースでは使用できません。次の文は、統合データベースで FILE ベースのメッセージシステムを使用する設定を行うものです。

```
CREATE REMOTE MESSAGE TYPE FILE ADDRESS 'cons';
SET REMOTE FILE OPTION public.directory='c:¥¥tutorial¥¥messages';
SET REMOTE FILE OPTION public.debug='yes';
```

11. 次の文を実行して、リモートユーザー rem をパスワードなしで作成します。次に、FILE メッセージシステムにユーザーのアドレスを定義する間に、REMOTE パーミッションを付与します。

```
GRANT CONNECT TO rem;
GRANT REMOTE TO rem TYPE FILE ADDRESS 'rem';
```

12. 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説明します。従業員テーブルのすべてのローをレプリケートする pub_employees という名前のパブリケーションを作成します。パブリケーションに対してユーザー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。

```
CREATE PUBLICATION pub_employees ( TABLE employees );
CREATE SUBSCRIPTION TO pub_employees FOR rem;
```

13. Interactive SQL との接続を切断します。

14. 「レッスン 2：メッセージサーバーの作成」 157 ページに進みます。

レッスン 2：メッセージサーバーの作成

統合データベースをメッセージサーバーとして使用することは可能ですが、このチュートリアルでは、別のデータベースサーバーを使用してメッセージサーバーを実行します。こうすることにより、この 2 つのデータベースサーバー間でメッセージを処理するために実行される作業の量が分散されます。また、統合データベースへの HTTP アクセスを開放していないため、セキュリティのレベルが上がります。

♦ メッセージサーバーを作成する

1. *c:¥tutorial¥msgsrv* ディレクトリから次のコマンドを実行して、メッセージサーバーデータベース (msgsrv) を作成します。

```
dbinit msgsrv.db
```

2. メッセージサーバーを起動します。

```
dbeng12 -n msgsrv c:¥tutorial¥msgsrv¥msgsrv.db -xs http(port=8033)
```

これは、リモートデータベースからの HTTP 要求を受信し、*c:¥tutorial¥messages* ディレクトリに存在するメッセージファイルにアクセスするデータベースサーバーであるため、コマン

ドラインには **-xs http(8033)** が必要です。データベースサーバーが起動した時点では Web サービスは定義されていませんが、このレッスンで作成されます。その上、パーソナルデータベースサーバーだけが起動しているため、このコンピューター上の SQL Remote プロセスだけが HTTP を使用してメッセージサーバーと通信できます。生産環境では通常、他のコンピューター上の SQL Remote プロセスも Web サービスにアクセスできるよう、ネットワークサーバーを使用するはずです。 -xs の使用の詳細については、「-xs dbeng12/dbsrv12 サーバーオプション」『SQL Anywhere サーバー データベース管理』を参照してください。

3. 別のメッセージサーバーを作成する場合、統合データベースのスキーマの多くをそのメッセージサーバーにコピーする必要があります。とりわけ定義されたリモートユーザーとアドレスについての情報などのコピーが必要です。これは手動で行うことができますが、このタスクを行うもっとも簡単な方法は、dbunload ユーティリティを使用して、統合データベースと同じスキーマで新しいデータベースを作成することです。

   ```
   dbunload -n -xx -ac "SERVER=msgsrv;DBN=msgsrv;UID=DBA;PWD=sql" -c
   "SERVER=cons;DBN=cons;UID=DBA;PWD=sql"
   ```

   dbunload コマンドで使用されるオプションは、次のことを行います。

   - **-n**　スキーマだけがアンロードされ、統合データベースのデータはまったくメッセージサーバーに追加されないことを示します。
   - **-xx**　外部アンロードと再ロードを実行します。これは、両方の関連するデータベースがすでに稼働している場合に必要です。
   - **-ac "SERVER=msgsrv;DBN=msgsrv;UID=DBA;PWD=sql"**　アンロードの送信先接続を定義します。このレッスンでは、メッセージサーバーです。
   - **-c "SERVER=cons;DBN=cons;UID=DBA;PWD=sql"**　アンロードの送信元接続を定義します。このレッスンでは、統合データベースです。

4. Interactive SQL を使用して、DBA 権限を持つユーザーとしてメッセージサーバーデータベース (msgsrv) に接続します。

   ```
   dbisql -c "SERVER=msgsrv;DBN=msgsrv;UID=DBA;PWD=sql"
   ```

   レッスン 1 では、パブリッシャー (cons) とリモートユーザー (rem) のパスワードを作成しませんでした。そのため、これらのユーザーはいずれも統合データベースにはアクセスできません。メッセージサーバーではこれらのユーザーにパスワードが必要です。なぜなら、リモートユーザーからの HTTP 要求では、リモートデータベースのパブリッシャーおよびメッセージサーバーでの認証用に提供されたパスワードが使用されるためです。メッセージサーバーデータベース (msgsrv) で次の文を実行して、パブリッシャーとリモートユーザーのパスワードを定義します。

   ```
   GRANT CONNECT TO cons IDENTIFIED BY cons;
   GRANT CONNECT TO rem IDENTIFIED BY rem;
   ```

5. データベースが最初に初期化されたとき、リモートユーザーからの HTTP 要求を受け付けるのに必要な Web サービスはどれも定義されておらず、メッセージファイルが格納されているディレクトリにデータベースサーバーがアクセスするのを許可する定義はありません。これらのオブジェクトの作成は、誰がすべてのオブジェクトを所有するのかを指定するオプションのパラメーターを持つ sr_add_message_server ストアドプロシージャーの使用により自動化されています。メッセージサーバーデータベース (msgsrv) に次の文を実行して、メッセージ

サーバーに必要なすべてのオブジェクトを定義し、すべてのオブジェクトが cons ユーザーによって所有されていることを指定します。

```
GRANT GROUP TO cons;
SET REMOTE http OPTION cons.root_directory='c:¥¥tutorial¥¥messages';
CALL sr_add_message_server( 'cons' );
COMMIT;
```

詳細については、「sr_add_message_server システムプロシージャー」206 ページを参照してください。

6. Interactive SQL との接続を切断します。
7. 「レッスン 3：リモートデータベースの作成」159 ページに進みます。

レッスン 3：リモートデータベースの作成

このレッスンでは、リモートデータベースを抽出してから、リモートデータベースの FILE メッセージシステムを HTTP メッセージシステムと置き換えます。

♦ リモートデータベースの作成

1. *c:¥tutorial¥rem* ディレクトリから次のコマンドを実行して、リモートデータベースを作成します (rem)。

   ```
   dbinit rem.db
   ```

2. このレッスンでは、dbxtract を使用してリモートデータベースを作成します。次のコマンドを実行して統合データベースから rem ユーザーのデータベースを抽出し、抽出後もそのリモートデータベースのデータベースサーバーを稼働状態にしておきます。

   ```
   dbxtract -xx -ac "SERVER=rem;DBN=rem;dbf=c:¥tutorial¥rem
   ¥rem.db;UID=DBA;PWD=sql;autostop=no" -c "SERVER=cons;DBN=cons;UID=DBA;PWD=sql" rem
   ```

3. 現在、リモートデータベース (rem) に接続されていない場合は、次のコマンドを実行します。

   ```
   dbisql -c "SERVER=rem;DBN=rem;UID=DBA;PWD=sql"
   ```

4. 統合データベースは、FILE メッセージシステムを使用しているため、dbxtract が稼働すると、rem リモートデータベースも FILE メッセージシステムを使用していると仮定して SQL Remote 定義を作成します。リモートデータベースが HTTP メッセージシステムを使用するよう設定するには、リモートデータベース (rem) で次の文を実行して、このリモートデータベースの FILE メッセージシステムを削除します。

   ```
   CREATE REMOTE TYPE "FILE" ADDRESS '';
   SET REMOTE FILE OPTION public.directory='';
   SET REMOTE FILE OPTION public.debug='';
   ```

5. リモートデータベース (rem) で次の文を実行して、このリモートデータベースの HTTP メッセージシステムを設定します。

   ```
   CREATE REMOTE TYPE "HTTP" ADDRESS 'rem';
   GRANT CONSOLIDATE TO "cons" TYPE "HTTP" ADDRESS 'cons';
   ```

```
SET REMOTE HTTP OPTION public.user_name='rem';
SET REMOTE HTTP OPTION public.password='rem';
SET REMOTE HTTP OPTION public.debug='yes';
SET REMOTE HTTP OPTION public.https='no';
SET REMOTE HTTP OPTION public.url='localhost:8033';
COMMIT;
```

6. リモートデータベース (rem) に、抽出後の統合データベースに存在したこの 2 ローのデータが含まれることを確認します。次の文を実行して、従業員テーブルの内容を表示します。

   ```
   SELECT * FROM employees;
   ```

   クエリは、従業員テーブルから次のデータを返します。ただし、hire_date カラムには、次のテーブルにある値ではなく、そのローを挿入した時間が記載されます。

employee_id	first_name	last_name	hire_date
1	Kelly	Meloy	2011-03-25 08:27:56.310
2	Melisa	Boysen	2011-03-25 08:27:56.310

7. Interactive SQL との接続を切断します。

8. 「レッスン 4：統合データベースとリモートデータベースにデータを追加しレプリケートする」 160 ページに進みます。

レッスン 4：統合データベースとリモートデータベースにデータを追加しレプリケートする

このレッスンでは、統合データベースとリモートデータベースにデータを追加し、SQL Remote を実行して変更をレプリケートし、両方のデータベースでデータが一致していることを確認します。

♦ 統合データベースにデータを追加する

1. 現在、統合データベース (cons) に接続されていない場合は、次のコマンドを実行します。

   ```
   dbisql -c "SERVER=cons;DBN=cons;UID=DBA;PWD=sql"
   ```

2. 統合データベース (cons) で次の文を実行し、追加サンプルデータを従業員テーブルに追加します。

   ```
   INSERT INTO employees (first_name, last_name) VALUES ('Javier', 'Spoor');
   COMMIT;
   ```

3. Interactive SQL との接続を切断します。

♦ データをリモートデータベースに追加する

1. 現在、リモートデータベース (rem) に接続されていない場合は、次のコマンドを実行します。

   ```
   dbisql -c "SERVER=rem;DBN=rem;UID=DBA;PWD=sql"
   ```

2. リモートデータベース (rem) で次の文を実行し、追加サンプルデータを従業員テーブルに追加します。

   ```
   INSERT INTO employees (first_name, last_name) VALUES ('Nelson', 'Kreitzer');
   COMMIT;
   ```

3. Interactive SQL との接続を切断します。

♦ 統合データベースとリモートデータベース間で変更をレプリケートする

1. 統合データベース (cons) で、Message Agent を実行します。

   ```
   dbremote -c "SERVER=cons;DBN=cons;UID=DBA;PWD=sql" -qc -v -o c:¥tutorial¥cons1.txt
   ```

   これにより、統合データベース (cons) のトランザクションログがスキャンされ、FILE メッセージシステムを使用してリモートデータベース (rem) のメッセージが生成されます。デバッグメッセージシステムパラメーターは、統合データベースの FILE メッセージシステム用に設定されているため、*c:¥tutorial¥cons1.txt* ファイルを見て、メッセージが *c:¥tutorial¥messages¥rem* ディレクトリに書き込まれていることを示すデバッグメッセージがあることを確認できます。次に例を示します。

   ```
   I. 2011-03-25 11:03:31. Processing transactions from active transaction log
   I. 2011-03-25 11:03:31. Sending message to "rem" (0-0000000000-0000550994-0)
   I. 2011-03-25 11:03:31. sopen "c:¥tutorial¥messages¥rem¥cons.0"
   I. 2011-03-25 11:03:31. write " c:¥tutorial¥messages¥rem¥cons.0"
   I. 2011-03-25 11:03:31. close " c:¥tutorial¥messages¥rem¥cons.0"
   ```

2. リモートデータベース (rem) で、Message Agent を実行します。

   ```
   dbremote -c "SERVER=rem;DBN=rem;UID=DBA;PWD=sql" -qc -v -o c:¥tutorial¥rem.txt
   ```

   このコマンドは、HTTP メッセージシステムを使用して、統合データベースで生成されたばかりのメッセージを受信して適用します。次に、トランザクションログをスキャンし、メッセージをリモートデータベースに追加された新しいローとともに統合データベースに送り返します。デバッグメッセージシステムパラメーターは、リモートデータベースの HTTP メッセージシステム用に設定されているため、*c:¥tutorial¥rem.txt* ファイルを見て、HTTP メッセージシステムが使用されていることを示すデバッグメッセージがあることを確認できます。次に例を示します。

   ```
   I. 2011-03-25 11:10:02. Sending message to "cons" (0-0000000000-0000557411-0)
   I. 2011-03-25 11:10:02. HTTPWriteMessage "rem.0"
   I. 2011-03-25 11:10:02. HTTPWriteMessage: success -- filename  "rem.0"
   I. 2011-03-25 11:10:02. HTTPDisconnect
   ```

3. 統合データベース (cons) で、Message Agent を実行します。

   ```
   dbremote -c "SERVER=cons;DBN=cons;UID=DBA;PWD=sql" -qc -v -o c:¥tutorial¥cons2.txt
   ```

   このコマンドは、FILE ベースのメッセージシステムを使用してリモートデータベースによって生成されたばかりのメッセージを受信して適用します。

♦ 統合データベースとリモートデータベースでデータを確認する

1. 現在、統合データベース (cons) に接続されていない場合は、次のコマンドを実行します。

   ```
   dbisql -c "SERVER=cons;DBN=cons;UID=DBA;PWD=sql"
   ```

2. 統合データベースに 4 ローのデータがすべて含まれていることを確認するには、次の文を実行して従業員テーブルの内容を表示します。

   ```
   SELECT * FROM employees;
   ```

   クエリは、従業員テーブルから次のデータを返します。ただし、hire_date カラムには、次のテーブルにある値ではなく、そのローを挿入した時間が記載されます。

employee_id	first_name	last_name	hire_date
1	Kelly	Meloy	2011-03-25 08:27:56.310
2	Melisa	Boysen	2011-03-25 08:27:56.310
3	Javier	Spoor	2011-03-25 08:30:26.110
102000001	Nelson	Kreitzer	2011-03-25 08:31:51.970

3. Interactive SQL との接続を切断します。

4. 現在、リモートデータベース (rem) に接続されていない場合は、次のコマンドを実行します。

   ```
   dbisql -c "SERVER=rem;DBN=rem;UID=DBA;PWD=sql"
   ```

   次の文を実行して従業員テーブルの内容を表示することにより、リモートデータベース (rem) に 4 ローのデータがすべて含まれていることを確認します。

   ```
   SELECT * FROM employees;
   ```

   クエリは、従業員テーブルから次のデータを返します。ただし、hire_date カラムには、次のテーブルにある値ではなく、そのローを挿入した時間が記載されます。

employee_id	first_name	last_name	hire_date
1	Kelly	Meloy	2011-03-25 08:27:56.310
2	Melisa	Boysen	2011-03-25 08:27:56.310

employee_id	first_name	last_name	hire_date
3	Javier	Spoor	2011-03-25 08:30:26.110
102000001	Nelson	Kreitzer	2011-03-25 08:31:51.970

5. Interactive SQL との接続を切断します。
6. 「レッスン 5：クリーンアップ」163 ページに進みます。

レッスン 5：クリーンアップ

最後のレッスンでは、このチュートリアルで起動した 3 つのデータベースサーバーを停止します。

♦ データベースサーバーを停止する

1. 次のコマンドを実行して、リモートデータベースを停止します。

   ```
   dbstop -y -c "SERVER=rem;DBN=rem;UID=DBA;PWD=sql"
   ```

2. 次のコマンドを実行して、メッセージサーバーデータベースを停止します。

   ```
   dbstop -y -c "SERVER=msgsrv;DBN=msgsrv;UID=DBA;PWD=sql"
   ```

3. 次のコマンドを実行して、統合データベースを停止します。

   ```
   dbstop -y -c "SERVER=cons;DBN=cons;UID=DBA;PWD=sql"
   ```


チュートリアル：統合データベースをメッセージサーバーとして HTTP メッセージシステムを使用したレプリケーションシステムの設定

このチュートリアルのレッスンでは、SQL Anywhere の統合データベースとリモートデータベースの両方を使用する SQL Remote レプリケーションシステムの設定について学びます。統合データベースは、FILE メッセージシステムを使用して変更をレプリケートし、リモートデータベースは、HTTP メッセージシステムを使用して変更をレプリケートします。

このチュートリアルでは、次のことを学習します。

- SQL Anywhere 統合データベースと、統合データベース内のすべてのデータを含む SQL Anywhere リモートデータベースを作成する。
- SQL Remote によって生成されたメッセージを格納するディレクトリ構造を作成する。統合データベースは、FILE メッセージシステムを使用してファイルにアクセスし、リモートデータベースは HTTP メッセージシステムを使用する。
- HTTP メッセージシステム用メッセージサーバーとして動作する統合データベースを設定する。
- HTTP メッセージシステムを使用してメッセージを送信するリモートデータベースを作成する。
- 統合データベースとリモートデータベース間でデータをレプリケートする。

レッスン 1：統合データベースの作成

このレッスンでは、データベースとそのトランザクションログを格納するのに必要なディレクトリ、そしてメッセージのディレクトリ構造を作成します。また、リモートユーザーおよびデータのレプリケートに必要なパブリケーションとサブスクリプションなど、統合データベースのスキーマを定義します。SQL Remote が統合データベースに対して稼働する場合、FILE メッセージシステムを使用してメッセージを送受信しますが、リモートデータベースは HTTP メッセージシステムを使用します。

♦ チュートリアル用の統合データベースとディレクトリを作成する

1. 統合データベースとリモートデータベースを保持するため次のディレクトリを作成します。
   - *c:¥tutorial*
   - *c:¥tutorial¥cons*
   - *c:¥tutorial¥rem*

2. 統合データベース、リモートデータベース、メッセージサーバーデータベースが生成したメッセージファイルを格納するために、次のディレクトリを作成します。

   - *c:¥tutorial¥messages*
   - *c:¥tutorial¥messages¥cons*
   - *c:¥tutorial¥messages¥rem*

3. *c:¥tutorial¥cons* ディレクトリから次のコマンドを実行して、統合データベース (cons) を作成します。

   ```
   dbinit cons.db
   ```

4. 統合データベースを起動します。

   ```
   dbeng12 -n cons c:¥tutorial¥cons¥cons.db -xs http(port=8033)
   ```

   これは、リモートデータベースからの HTTP 要求を受信し、*c:¥tutorial¥messages* ディレクトリに存在するメッセージファイルにアクセスするデータベースサーバーであるため、コマンドラインには **-xs http(8033)** が必要です。データベースサーバーが起動した時点では Web サービスは定義されていませんが、次のレッスンで作成されます。このレッスンでは、パーソナルデータベースサーバーのみを起動します。したがって、このコンピューターの SQL Remote プロセスだけが、HTTP を使用してメッセージサーバーと通信できます。生産環境では通常、他のコンピューター上の SQL Remote プロセスも Web サービスにアクセスできるよう、ネットワークサーバーを使用するはずです。 -xs の使用の詳細については、「-xs dbeng12/dbsrv12 サーバーオプション」『SQL Anywhere サーバー データベース管理』を参照してください。

5. Interactive SQL を使用して、DBA 権限を持つユーザーとして統合データベース (cons) に接続します。

   ```
   dbisql -c "UID=DBA;PWD=sql;SERVER=cons;DBN"
   ```

6. 統合データベース (cons) のグローバルデータベース ID を設定するには、次の文を実行します (GLOBAL AUTOINCREMENT デフォルトを使用する場合には、すべてのデータベースに排他的なプライマリキーが選択されるようグローバルデータベース ID が必要です)。

   ```
   SET OPTION public.global_database_id=0;
   ```

7. このチュートリアルのデータベースのスキーマは、1 つのテーブルから構成され、テーブルのすべてのカラムとローはすべてのリモートユーザーにレプリケートされます。統合データベース (cons) 用に次の文を実行し、データベースに 1 つのテーブルを作成します。

   ```
   CREATE TABLE employees (
       employee_id BIGINT NOT NULL DEFAULT GLOBAL AUTOINCREMENT(1000000) PRIMARY
   KEY,
       first_name VARCHAR(128) NOT NULL,
       last_name VARCHAR(128) NOT NULL,
       hire_date TIMESTAMP NOT NULL DEFAULT TIMESTAMP
   );
   ```

8. 統合データベース (cons) で次の文を実行し、サンプルデータを従業員テーブルに追加します。

   ```
   INSERT INTO employees (first_name, last_name) VALUES ('Kelly', 'Meloy');
   INSERT INTO employees (first_name, last_name) VALUES ('Melisa', 'Boysen');
   COMMIT;
   ```


9. 統合データベース (cons) で次の文を実行し、そのテーブルが作成され、データが挿入されていることを確認します。

```
SELECT * FROM employees;
```

10. クエリは、従業員テーブルから次のデータを返します。ただし、hire_date カラムには、次のテーブルにある値ではなく、そのローを挿入した時間が記載されます。

employee_id	first_name	last_name	hire_date
1	Kelly	Meloy	2011-03-25 08:27:56.310
2	Melisa	Boysen	2011-03-25 08:27:56.310

11. このチュートリアルでは、パブリッシャーとリモートユーザーにはパスワードが割り当てられます。これは、統合データベースが HTTP メッセージシステム用のメッセージサーバーとして動作するためです。次の文を実行して、CONNECT パーミッションおよび PUBLISH パーミッションを持つユーザー cons を作成します。

```
GRANT CONNECT TO cons;
GRANT PUBLISH TO cons;
```

12. パフォーマンス上の理由から、HTTP メッセージシステムはリモートデータベースのみで使用することができ、統合データベースでは使用できません。次の文は、統合データベースで FILE ベースのメッセージシステムを使用する設定を行うものです。

```
CREATE REMOTE MESSAGE TYPE FILE ADDRESS 'cons';
SET REMOTE FILE OPTION public.directory='c:¥¥tutorial¥¥messages';
SET REMOTE FILE OPTION public.debug='yes';
```

13. 次の文を実行して、リモートユーザー rem をパスワードなしで作成します。次に、REMOTE パーミッションを付与し、FILE メッセージシステムにユーザーのアドレスを定義します。

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

14. 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説明します。従業員テーブルのすべてのローをレプリケートする pub_employees という名前のパブリケーションを作成します。パブリケーションに対してユーザー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。

```
CREATE PUBLICATION pub_employees ( TABLE employees );
CREATE SUBSCRIPTION TO pub_employees FOR rem;
```

15. Interactive SQL との接続を切断します。

16. 「レッスン 2：メッセージサーバーとして動作する統合データベースの設定」168 ページに進みます。

レッスン 2 : メッセージサーバーとして動作する統合データベースの設定

このレッスンでは、HTTP メッセージシステム用メッセージサーバーとして動作する統合データベースを設定します。メッセージサーバーとして動作する別のデータベースとデータベースサーバーを設定することも可能です。

♦ メッセージサーバーとして動作する統合データベースを設定する

1. Interactive SQL を使用して、DBA 権限を持つユーザーとして統合データベースに接続します。

   ```
   dbisql -c "SERVER=cons;DBN=cons;UID=DBA;PWD=sql"
   ```

2. データベースが最初に初期化されたとき、リモートユーザーからの HTTP 要求を受け付けるのに必要な Web サービスはどれも定義されておらず、メッセージファイルが格納されているディレクトリにデータベースサーバーがアクセスするのを許可する定義はありません。これらのオブジェクトの作成は、誰がすべてのオブジェクトを所有するのかを指定するオプションのパラメーターを持つ sr_add_message_server ストアドプロシージャーの使用により自動化されています。統合データベース (cons) に次の文を実行して、メッセージサーバーに必要なすべてのオブジェクトを定義して、すべてのオブジェクトが cons ユーザーによって所有されていることを指定します。

   ```
   GRANT GROUP TO cons;
   SET REMOTE http OPTION cons.root_directory='c:¥¥tutorial¥¥messages';
   CALL sr_add_message_server( 'cons' );
   COMMIT;
   ```

3. Interactive SQL との接続を切断します。

4. 「レッスン 3 : リモートデータベースの作成」 168 ページに進みます。

レッスン 3 : リモートデータベースの作成

このレッスンでは、リモートデータベースを抽出してから、リモートデータベースの FILE メッセージシステムを HTTP メッセージシステムと置き換えます。

♦ リモートデータベースの作成

1. *c:¥tutorial¥rem* ディレクトリから次のコマンドを実行して、リモートデータベースを作成します (rem)。

   ```
   dbinit rem.db
   ```

2. このレッスンでは、dbxtract を使用してリモートデータベースを作成します。次のコマンドを実行して統合データベースから rem ユーザーのデータベースを抽出し、抽出後もそのリモートデータベースのデータベースサーバーを稼働状態にしておきます。

   ```
   dbxtract -xx -ac "SERVER=rem;DBN=rem;dbf=c:¥tutorial¥rem
   ¥rem.db;UID=DBA;PWD=sql;autostop=no" -c "SERVER=cons;DBN=cons;UID=DBA;PWD=sql" rem
   ```

   現在、リモートデータベース (rem) に接続されていない場合は、次のコマンドを実行します。


```
dbisql -c "SERVER=rem;DBN=rem;UID=DBA;PWD=sql"
```

3. 統合データベースは、FILE メッセージシステムを使用しているため、dbxtract が稼働すると、統合データベースは、rem リモートデータベースも FILE メッセージシステムを使用していると仮定して SQL Remote 定義を作成します。リモートデータベースが HTTP メッセージシステムを使用するよう設定するには、リモートデータベース (rem) で次の文を実行して、このリモートデータベースの FILE メッセージシステムを削除します。

   ```
   CREATE REMOTE TYPE "FILE" ADDRESS '';
   SET REMOTE FILE OPTION public.directory='';
   SET REMOTE FILE OPTION public.debug='';
   ```

4. リモートデータベース (rem) で次の文を実行して、このリモートデータベースの HTTP メッセージシステムを設定します。

   ```
   CREATE REMOTE TYPE "HTTP" ADDRESS 'rem';
   GRANT CONSOLIDATE TO "cons" TYPE "HTTP" ADDRESS 'cons';
   SET REMOTE HTTP OPTION public.user_name='rem';
   SET REMOTE HTTP OPTION public.password='rem';
   SET REMOTE HTTP OPTION public.debug='yes';
   SET REMOTE HTTP OPTION public.https='no';
   SET REMOTE HTTP OPTION public.url='localhost:8033';
   COMMIT;
   ```

5. リモートデータベース (rem) の従業員テーブルに、抽出後の統合データベースに存在したこの 2 ローのデータが含まれることを確認します。次の文を実行して、従業員テーブルの内容を表示します。

   ```
   SELECT * FROM employees
   ```

   クエリは、従業員テーブルから次のデータを返します。ただし、hire_date カラムには、次のテーブルにあるデータではなく、そのローを挿入した時間が記載されます。

employee_id	first_name	last_name	hire_date
1	Kelly	Meloy	2011-03-25 08:27:56.310
2	Melisa	Boysen	2011-03-25 08:27:56.310

6. Interactive SQL との接続を切断します。

7. 「レッスン 4：統合データベースとリモートデータベースにデータを追加しレプリケートする」 170 ページに進みます。

レッスン 4 : 統合データベースとリモートデータベースにデータを追加しレプリケートする

このレッスンでは、統合データベースとリモートデータベースにデータを追加し、SQL Remote を実行して変更をレプリケートし、両方のデータベースでデータが一致していることを確認します。

♦ 統合データベースにデータを追加する

1. 現在、統合データベース (cons) に接続されていない場合は、次のコマンドを実行します。

   ```
   dbisql -c "SERVER=cons;DBN=cons;UID=DBA;PWD=sql"
   ```

2. 統合データベース (cons) で次の文を実行し、追加サンプルデータを従業員テーブルに追加します。

   ```
   INSERT INTO employees (first_name, last_name) VALUES ('Javier', 'Spoor');
   COMMIT;
   ```

3. Interactive SQL との接続を切断します。

♦ データをリモートデータベースに追加する

1. 現在、リモートデータベース (rem) に接続されていない場合は、次のコマンドを実行します。

   ```
   dbisql -c "SERVER=rem;DBN=rem;UID=DBA;PWD=sql"
   ```

2. リモートデータベース (rem) で次の文を実行し、追加サンプルデータを従業員テーブルに追加します。

   ```
   INSERT INTO employees (first_name, last_name) VALUES ('Nelson', 'Kreitzer');
   COMMIT;
   ```

3. Interactive SQL との接続を切断します。

♦ 統合データベースとリモートデータベース間で変更をレプリケートする

1. 統合データベース (cons) で、Message Agent を実行します。

   ```
   dbremote -c "SERVER=cons;DBN=cons;UID=DBA;PWD=sql" -qc -v -o c:¥tutorial¥cons1.txt
   ```

   これにより、統合データベース (cons) のトランザクションログがスキャンされ、FILE メッセージシステムを使用してリモートデータベース (rem) のメッセージが生成されます。デバッグメッセージシステムパラメーターは、統合データベースの FILE メッセージシステム用に設定されているため、*c:¥tutorial¥cons1.txt* ファイルを見て、メッセージが *c:¥tutorial¥messages¥rem* ディレクトリに書き込まれていることを示すデバッグメッセージがあることを確認できます。次に例を示します。

   ```
   I. 2011-03-25 11:03:31. Processing transactions from active transaction log
   I. 2011-03-25 11:03:31. Sending message to "rem" (0-0000000000-0000550994-0)
   I. 2011-03-25 11:03:31. sopen "c:¥tutorial¥messages¥rem¥cons.0"
   I. 2011-03-25 11:03:31. write " c:¥tutorial¥messages¥rem¥cons.0"
   I. 2011-03-25 11:03:31. close " c:¥tutorial¥messages¥rem¥cons.0"
   ```


2. リモートデータベース (rem) で、Message Agent を実行します。

   ```
   dbremote -c "SERVER=rem;DBN=rem;UID=DBA;PWD=sql" -qc -v -o c:¥tutorial¥rem.txt
   ```

   このコマンドは、HTTP メッセージシステムを使用して、統合データベースで生成されたばかりのメッセージを受信して適用します。次に、トランザクションログをスキャンし、メッセージをリモートデータベースに追加された新しいローとともに統合データベースに送り返します。デバッグメッセージシステムパラメーターは、リモートデータベースの HTTP メッセージシステム用に設定されているため、*c:¥tutorial¥rem.txt* ファイルを見て、HTTP メッセージシステムが使用されていることを示すデバッグメッセージがあることを確認できます。次に例を示します。

   ```
   I. 2011-03-25 11:10:02. Sending message to "cons" (0-0000000000-0000557411-0)
   I. 2011-03-25 11:10:02. HTTPWriteMessage "rem.0"
   I. 2011-03-25 11:10:02. HTTPWriteMessage: success -- filename  "rem.0"
   I. 2011-03-25 11:10:02. HTTPDisconnect
   ```

3. 統合データベース (cons) で、Message Agent を実行します。

   ```
   dbremote -c "SERVER=cons;DBN=cons;UID=DBA;PWD=sql" -qc -v -o c:¥tutorial¥cons2.txt
   ```

   このコマンドは、FILE ベースのメッセージシステムを使用してリモートデータベースによって生成されたばかりのメッセージを受信して適用します。

♦ 統合データベースとリモートデータベースでデータを確認する

1. 現在、統合データベース (cons) に接続されていない場合は、次のコマンドを実行します。

   ```
   dbisql -c "SERVER=cons;DBN=cons;UID=DBA;PWD=sql"
   ```

2. 統合データベースに 4 ローのデータがすべて含まれていることを確認するには、次の文を実行して従業員テーブルの内容を表示します。

   ```
   SELECT * FROM employees
   ```

   クエリは、従業員テーブルから次のデータを返します。ただし、hire_date カラムには、次のテーブルにある値ではなく、そのローを挿入した時間が記載されます。

employee_id	first_name	last_name	hire_date
1	Kelly	Meloy	2011-03-25 08:27:56.310
2	Melisa	Boysen	2011-03-25 08:27:56.310
3	Javier	Spoor	2011-03-25 08:30:26.110
102000001	Nelson	Kreitzer	2011-03-25 08:31:51.970

3. Interactive SQL との接続を切断します。

4. 現在、リモートデータベース (rem) に接続されていない場合は、次のコマンドを実行します。

   ```
   dbisql -c "SERVER=rem;DBN=rem;UID=DBA;PWD=sql"
   ```

   次の文を実行して従業員テーブルの内容を表示することにより、リモートデータベース (rem) に 4 ローのデータがすべて含まれていることを確認します。

   ```
   SELECT * FROM employees
   ```

   クエリは、従業員テーブルから次のデータを返します。ただし、hire_date カラムには、次のテーブルにあるデータではなく、そのローを挿入した時間が記載されます。

employee_id	first_name	last_name	hire_date
1	Kelly	Meloy	2011-03-25 08:27:56.310
2	Melisa	Boysen	2011-03-25 08:27:56.310
3	Javier	Spoor	2011-03-25 08:30:26.110
102000001	Nelson	Kreitzer	2011-03-25 08:31:51.970

5. Interactive SQL との接続を切断します。
6. 「レッスン 5 : クリーンアップ」 172 ページに進みます。

レッスン 5 : クリーンアップ

最後のレッスンでは、このチュートリアルで起動した 2 つのデータベースサーバーを停止します。

♦ データベースサーバーを停止する

1. 次のコマンドを実行して、リモートデータベースを停止します。

   ```
   dbstop -y -c "SERVER=rem;DBN=rem;UID=DBA;PWD=sql"
   ```

2. 次のコマンドを実行して、統合データベースを停止します。

   ```
   dbstop -y -c "SERVER=cons;DBN=cons;UID=DBA;PWD=sql"
   ```

3. レッスン 1 で作成したディレクトリを削除します。

チュートリアル：HTTP メッセージシステムを使用し、Relay Server を経由して統合データベースをメッセージサーバーとしたレプリケーションシステムの設定

このチュートリアルのレッスンでは、SQL Anywhere の統合データベース、HTTP トラフィックを統合データベースに転送する Relay Server、そしてリモートデータベースを使用して SQL Remote レプリケーションシステムを設定する方法について学びます。統合データベースは、FILE メッセージシステムを使用して変更をレプリケートし、リモートデータベースは、HTTP メッセージシステムを使用して変更をレプリケートします。

このチュートリアルでは、次のことを学習します。

- SQL Anywhere 統合データベースと、統合データベース内のすべてのデータを含む SQL Anywhere リモートデータベースを作成する。
- SQL Remote によって生成されたメッセージを格納するディレクトリ構造を作成する。統合データベースは、FILE メッセージシステムを使用してファイルにアクセスし、リモートデータベースは HTTP メッセージシステムを使用する。
- HTTP トラフィックを統合データベースに転送するために既存の Relay Server を設定する。
- HTTP メッセージシステムのメッセージサーバーとして動作し、転送された HTTP トラフィックを Relay Server から受信する統合データベースを設定する。
- HTTP メッセージシステムを使用してメッセージを送信するリモートデータベースを作成する。
- 統合データベースとリモートデータベース間でデータをレプリケートする。

レッスン 1：統合データベースの作成

このレッスンでは、データベースとそのトランザクションログを格納するのに必要なディレクトリ、そしてメッセージのディレクトリ構造を作成します。また、リモートユーザーおよびデータのレプリケートに必要なパブリケーションとサブスクリプションなど、統合データベースのスキーマを定義します。SQL Remote が統合データベースに対して稼働する場合、FILE メッセージシステムを使用してメッセージを送受信しますが、リモートデータベースは HTTP メッセージシステムを使用します。このチュートリアルのため、統合データベース (メッセージサーバー) が稼働しているコンピューターの名前を machine_cons とします。

♦ チュートリアル用の統合データベースとディレクトリを作成する

1. 統合データベースとリモートデータベースを保持するため次のディレクトリを作成します。
   - *c:¥tutorial*
   - *c:¥tutorial¥cons*
   - *c:¥tutorial¥rem*
2. 統合データベースとリモートデータベースが生成したメッセージファイルを格納するために、次のディレクトリを作成します。
   - *c:¥tutorial¥messages*
   - *c:¥tutorial¥messages¥cons*
   - *c:¥tutorial¥messages¥rem*
3. *c:¥tutorial¥cons* ディレクトリから次のコマンドを実行して、統合データベース (cons) を作成します。

   ```
   dbinit cons.db
   ```
4. 統合データベースを起動します。

   ```
   dbsrv12 -n cons c:¥tutorial¥cons¥cons.db -xs http(port=8033)
   ```

   このメッセージサーバーは、リモートデータベースからの HTTP 要求を受信し、*c:¥tutorial¥messages* ディレクトリに存在するメッセージファイルにアクセスするデータベースサーバーであるため、コマンドラインには **-xs http(8033)** が必要です。データベースサーバーが起動した時点では Web サービスは定義されていませんが、次のレッスンで作成されます。このレッスンでは、パーソナルデータベースサーバーのみを起動します。したがって、このコンピューターの SQL Remote プロセスだけが、HTTP を使用してメッセージサーバーと通信できます。生産環境では通常、他のコンピューター上の SQL Remote プロセスも Web サービスにアクセスできるよう、ネットワークサーバーを使用するはずです。このレッスンではネットワークサーバーを起動して、これを cons と命名しました。ネットワークにこの名前ですでに稼働している別のデータベースサーバーがある場合は、このネットワークサーバーに別の名前を選択して、この代替名を使用するために、このチュートリアルの残りの部分で接続ストリングを変更する必要があります。 -xs の使用の詳細については、「-xs dbeng12/dbsrv12 サーバーオプション」『SQL Anywhere サーバー データベース管理』を参照してください。
5. Interactive SQL を使用して、DBA 権限を持つユーザーとして統合データベース (cons) に接続します。

   ```
   dbisql -c "UID=DBA;PWD=sql;SERVER=cons;DBN"
   ```
6. 統合データベース (cons) のグローバルデータベース ID を設定するには、次の文を実行します (GLOBAL AUTOINCREMENT デフォルトを使用する場合には、すべてのデータベースに排他的なプライマリキーが選択されるようグローバルデータベース ID が必要です)。

   ```
   SET OPTION public.global_database_id=0;
   ```
7. このチュートリアルのデータベースのスキーマは、1 つのテーブルから構成され、テーブルのすべてのカラムとローはすべてのリモートユーザーにレプリケートされます。統合データベース (cons) で次の文を実行し、データベースに 1 つのテーブルを作成します。


```
CREATE TABLE employees (
    employee_id BIGINT NOT NULL DEFAULT GLOBAL AUTOINCREMENT(1000000) PRIMARY
KEY,
    first_name VARCHAR(128) NOT NULL,
    last_name VARCHAR(128) NOT NULL,
    hire_date TIMESTAMP NOT NULL DEFAULT TIMESTAMP
);
```

8. 統合データベース (cons) で次の文を実行し、サンプルデータを従業員テーブルに追加します。

   ```
   INSERT INTO employees (first_name, last_name) VALUES ('Kelly', 'Meloy');
   INSERT INTO employees (first_name, last_name) VALUES ('Melisa', 'Boysen');
   COMMIT;
   ```

9. 統合データベース (cons) で次の文を実行し、そのテーブルが作成され、データが挿入されていることを確認します。

   ```
   SELECT * FROM employees;
   ```

   クエリは、従業員テーブルから次のデータを返します。ただし、hire_date カラムには、次のテーブルにある値ではなく、そのローを挿入した時間が記載されます。

employee_id	first_name	last_name	hire_date
1	Kelly	Meloy	2011-03-25 08:27:56.310
2	Melisa	Boysen	2011-03-25 08:27:56.310

10. このチュートリアルでは、パブリッシャーとリモートユーザーにはパスワードが割り当てられます。これは、統合データベースが HTTP メッセージシステム用のメッセージサーバーとして動作するためです。次の文を実行して、CONNECT パーミッションおよび PUBLISH パーミッションを持つユーザー cons を作成します。

    ```
    GRANT CONNECT TO cons;
    GRANT PUBLISH TO cons;
    ```

11. パフォーマンス上の理由から、HTTP メッセージシステムはリモートデータベースのみで使用することができ、統合データベースでは使用できません。次の文は、統合データベースで FILE ベースのメッセージシステムを使用する設定を行うものです。

    ```
    CREATE REMOTE MESSAGE TYPE FILE ADDRESS 'cons';
    SET REMOTE FILE OPTION public.directory='c:¥¥tutorial¥¥messages';
    SET REMOTE FILE OPTION public.debug='yes';
    ```

12. 次の文を実行して、リモートユーザー rem をパスワードなしで作成します。次に、REMOTE パーミッションを付与し、FILE メッセージシステムにユーザーのアドレスを定義します。

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

13. 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説明します。従業員テーブルのすべてのローをレプリケートする pub_employees という名前のパブリケーションを作成しま

す。パブリケーションに対してユーザー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。

```
CREATE PUBLICATION pub_employees ( TABLE employees );
CREATE SUBSCRIPTION TO pub_employees FOR rem;
```

14. Interactive SQL との接続を切断します。

15. 「レッスン 2：Relay Server の設定」 176 ページに進みます。

レッスン 2：Relay Server の設定

このレッスンでは、Relay Server の設定を変更して、HTTP 要求をバックエンド SQL Anywhere データベースに転送します。Relay Server の設定は、Relay Server が使用する *rs.config* ファイルの変更と rshost プロセスの再表示と再起動のみに依存します。このチュートリアルのため、次の仮定を行います。

1. Relay Server が稼働するコンピューターの名前は machine_iis であり、IIS 7.5 を実行する Windows 2008 Server R2 です。

2. Relay Server の設定指示は、マニュアルに記載されたとおり正確に従っています。

   - Relay Server
   - 「Relay Server の配備」『Relay Server』

3. Relay Server コンポーネントは、Windows Server 2008／Windows Server 2008 R2 上の Microsoft IIS 7.0 または 7.5 に配備されています。 「Windows Server 2008／Windows Server 2008 R2 上の Microsoft IIS 7.0 または 7.5 への Relay Server コンポーネントの配備」『Relay Server』を参照してください。

♦ メッセージサーバーとして動作する統合データベースを設定する

1. machine_iis の rshost プロセスが使用する *rs.config* ファイルを変更して、メッセージサーバーとして動作するバックエンド SQL Anywhere データベースサーバーのエントリを追加します。

```
[backend_farm]
id=srhttp_tutorial_farm
description=SQL Anywhere Web Services farm for tutorial
active_cookie=yes
active_header=no
enable=yes
verbosity=5

[backend_server]
id= srhttp_tutorial_server
description=SQL Anywhere Web Services server for tutorial
farm= srhttp_tutorial_farm
enable=yes
verbosity=5
```

2. *%SQLANY12%¥RelayServer¥IIS¥Bin64¥Server* ディレクトリから、次のコマンドラインを実行して設定の更新を適用します。


```
rshost -u -f rs.config
```

3. 「レッスン 3：メッセージサーバーとして動作する統合データベースの設定」177 ページに進みます。

レッスン 3：メッセージサーバーとして動作する統合データベースの設定

このレッスンでは、HTTP メッセージシステム用メッセージサーバーとして動作する統合データベースを設定します。メッセージサーバーとして動作する別のデータベースとデータベースサーバーを設定することも可能です。

♦ メッセージサーバーとして動作する統合データベースを設定する

1. Interactive SQL を使用して、DBA 権限のあるユーザーとして接続します。

```
dbisql -c "SERVER=cons;DBN=cons;UID=DBA;PWD=sql"
```

2. データベースが最初に初期化されたとき、リモートユーザーからの HTTP 要求を受け付けるのに必要な Web サービスはどれも定義されておらず、メッセージファイルが格納されているディレクトリにデータベースサーバーがアクセスするのを許可する定義はありません。これらのオブジェクトの作成は、誰がすべてのオブジェクトを所有するのかを指定するオプションのパラメーターを持つ sr_add_message_server ストアドプロシージャーの使用により自動化されています。統合データベース (cons) に次の文を実行して、メッセージサーバーに必要なすべてのオブジェクトを定義して、すべてのオブジェクトが cons ユーザーによって所有されていることを指定します。

```
GRANT GROUP TO cons;
SET REMOTE http OPTION cons.root_directory='c:¥¥tutorial¥¥messages';
CALL sr_add_message_server( 'cons' );
COMMIT;
```

3. Relay Server が HTTP 要求をバックエンド SQL Anywhere サーバーに転送する場合には追加設定が必要となります。いくつかのノードが読み込み専用、いくつかのノードが読み込み／書き込みノードとして定義されているバックエンド SQL Anywhere サーバーのために、高可用性環境を設定することが可能です。このチュートリアルでは、システムに 1 つのデータベースサーバーしかないため、このデータベースは読み込み／書き込みノードとして定義する必要があります。統合データベース (cons) で次の文を実行し、Relay Server がこのデータベースサーバーを読み込み／書き込みノードとして認識するよう必要なオブジェクトをすべて定義します。

```
CREATE PROCEDURE sp_oe_read_status()
RESULT (doc LONG VARCHAR)
BEGIN
DECLARE res LONG VARCHAR;
SET res='AVAILABLE=TRUE';
CALL sa_set_http_header('Content-Length', LENGTH(res) );
SELECT res;
END;
GO

CREATE SERVICE oe_read_status
```

```
TYPE 'raw'
AUTHORIZATION OFF
SECURE OFF
USER DBA
AS CALL sp_oe_read_status();
GO
```

4. Interactive SQL との接続を切断します。

5. Outbound Enabler は、Relay Server とバックエンド SQL Anywhere データベースの間のチャネルとして動作します。machine_cons コンピュータ上で、次のコマンドラインにより Relay Server Outbound Enabler (RSOE) を起動します。

```
rsoe -cr "host=machine_iis;port=80;url_suffix=/rs/server/rs_server.dll"
-cs "host=machine_cons;port=8033;status_url=/oe_read_status"
-f srhttp_tutorial_farm -id srhttp_tutorial_server -v 5 -o rsoe.log
```

6. 「レッスン 4：リモートデータベースの作成」178 ページに進みます。

レッスン 4：リモートデータベースの作成

このレッスンでは、リモートデータベースを抽出してから、リモートデータベースの FILE メッセージシステムを HTTP メッセージシステムと置き換えます。

♦ リモートデータベースの作成

1. *c:¥tutorial¥rem* ディレクトリから次のコマンドを実行して、リモートデータベースを作成します (rem)。

```
dbinit rem.db
```

2. このレッスンでは、dbxtract を使用してリモートデータベースを作成します。次のコマンドを実行して統合データベースから rem ユーザーのデータベースを抽出し、抽出後もそのリモートデータベースのデータベースサーバーを稼働状態にしておきます。

```
dbxtract -xx -ac "SERVER=rem;DBN=rem;DBF=c:¥tutorial¥rem
¥rem.db;UID=DBA;PWD=sql;autostop=no" -c "SERVER=cons;DBN=cons;UID=DBA;PWD=sql" rem
```

3. 現在、Interactive SQL からリモートデータベース (rem) に接続されていない場合は、次のコマンドを実行します。

```
dbisql -c "SERVER=rem;DBN=rem;UID=DBA;PWD=sql"
```

4. 統合データベースは、FILE メッセージシステムを使用しているため、dbxtract が稼働すると、rem リモートデータベースも FILE メッセージシステムを使用していると仮定して SQL Remote 定義を作成しますリモートデータベースが HTTP メッセージシステムを使用するよう設定するには、リモートデータベース (rem) で次の文を実行して、このリモートデータベースの FILE メッセージシステムを削除します。

```
CREATE REMOTE TYPE "FILE" ADDRESS '';
SET REMOTE FILE OPTION public.directory='';
SET REMOTE FILE OPTION public.debug='';
```


5. リモートデータベース (rem) で次の文を実行して、このリモートデータベースの HTTP メッセージシステムを設定します。

```
CREATE REMOTE TYPE "HTTP" ADDRESS 'rem';
GRANT CONSOLIDATE TO "cons" TYPE "HTTP" ADDRESS 'cons';
SET REMOTE HTTP OPTION public.user_name='rem';
SET REMOTE HTTP OPTION public.password='rem';
SET REMOTE HTTP OPTION public.debug='yes';
SET REMOTE HTTP OPTION public.https='no';
SET REMOTE HTTP OPTION public.url='machine_iis:80/rs/client/rs_client.dll/srhttp_tutorial_farm';
COMMIT;
```

6. リモートデータベース (rem) に、抽出後の統合データベースに存在したこの 2 ローのデータが含まれることを確認します。次の文を実行して、従業員テーブルの内容を表示します。

```
SELECT * FROM employees;
```

クエリは、従業員テーブルから次のデータを返します。ただし、hire_date カラムには、次のテーブルにあるデータではなく、そのローを挿入した時間が記載されます。

employee_id	first_name	last_name	hire_date
1	Kelly	Meloy	2011-03-25 08:27:56.310
2	Melisa	Boysen	2011-03-25 08:27:56.310

7. Interactive SQL との接続を切断します。

8. 「レッスン 5：統合データベースとリモートデータベースにデータを追加しレプリケートする」 179 ページに進みます。

レッスン 5：統合データベースとリモートデータベースにデータを追加しレプリケートする

このレッスンでは、統合データベースとリモートデータベースにデータを追加し、SQL Remote を実行して変更をレプリケートし、両方のデータベースでデータが一致していることを確認します。

♦ 統合データベースにデータを追加する

1. 現在、統合データベース (cons) に接続されていない場合は、次のコマンドを実行します。

```
dbisql -c "SERVER=cons;DBN=cons;UID=DBA;PWD=sql"
```

2. 統合データベース (cons) で次の文を実行し、追加サンプルデータを従業員テーブルに追加します。

```
INSERT INTO employees (first_name, last_name) VALUES ('Javier', 'Spoor');
COMMIT;
```

3. Interactive SQL との接続を切断します。

♦ データをリモートデータベースに追加する

1. 現在、リモートデータベース (rem) に接続されていない場合は、次のコマンドを実行します。

```
dbisql -c "SERVER=rem;DBN=rem;UID=DBA;PWD=sql"
```

2. リモートデータベース (rem) で次の文を実行し、追加サンプルデータを従業員テーブルに追加します。

```
INSERT INTO employees (first_name, last_name) VALUES ('Nelson', 'Kreitzer');
COMMIT;
```

3. Interactive SQL との接続を切断します。

♦ 統合データベースとリモートデータベース間で変更をレプリケートする

1. 統合データベース (cons) で、Message Agent を実行します。

```
dbremote -c "SERVER=cons;DBN=cons;UID=DBA;PWD=sql" -qc -v -o c:¥tutorial¥cons1.txt
```

これにより、統合データベース (cons) のトランザクションログがスキャンされ、FILE メッセージシステムを使用してリモートデータベース (rem) のメッセージが生成されます。デバッグメッセージシステムパラメーターは、統合データベースの FILE メッセージシステム用に設定されているため、*c:¥tutorial¥cons1.txt* ファイルを見て、メッセージが *c:¥tutorial¥messages¥rem* ディレクトリに書き込まれていることを示すデバッグメッセージがあることを確認できます。次に例を示します。

```
I. 2011-04-12 09:33:03. Processing transactions from active transaction log
I. 2011-04-12 09:33:03. Sending message to "rem" (0-0000000000-0000550994-0)
I. 2011-04-12 09:33:03. sopen "c:¥tutorial¥messages¥rem¥cons.0"
I. 2011-04-12 09:33:03. write " c:¥tutorial¥messages¥rem¥cons.0"
I. 2011-04-12 09:33:03. close " c:¥tutorial¥messages¥rem¥cons.0"
```

2. リモートデータベース (rem) で、Message Agent を実行します。

```
dbremote -c "SERVER=rem;DBN=rem;UID=DBA;PWD=sql" -qc -v -o c:¥tutorial¥rem.txt
```

このコマンドは、HTTP メッセージシステムを使用して、統合データベースで生成されたばかりのメッセージを受信して適用します。次に、トランザクションログをスキャンし、メッセージをリモートデータベースに追加された新しいローとともに統合データベースに送り返します。デバッグメッセージシステムパラメーターは、リモートデータベースの HTTP メッセージシステム用に設定されているため、*c:¥tutorial¥rem.txt* ファイルを見て、HTTP メッセージシステムが使用されていることを示すデバッグメッセージがあることを確認できます。次に例を示します。

```
I. 2011-04-12 09:34:03. Sending message to "cons" (0-0000000000-0000576448-0)
I. 2011-04-12 09:34:03. HTTPWriteMessage "rem.0"
I. 2011-04-12 09:34:03. HTTPWriteMessage: success -- filename  "rem.0"
I. 2011-04-12 09:34:03. HTTPDisconnect
```

3. また、RSOE が生成した出力ファイルを見て、その情報がログに印刷されていることを確認することにより、要求が Relay Server を通過したことを確認できます。


```
I. 2011-04-12 09:34:03. <UpChannel-0000> PacketRead packet-len:257
I. 2011-04-12 09:34:03. <UpChannel-0000> PacketRead packet-opcode:0xf004
I. 2011-04-12 09:34:03. <UpChannel-0000> packet read..
I. 2011-04-12 09:34:03. <UpChannel-0000> successful packet read.. processing it..
I. 2011-04-12 09:34:03. <UpChannel-0000> 259 RS_CLI_SESSION_BEGIN(snum=0006
sfp=4e0e5291 ridx=0)
I. 2011-04-12 09:34:03. <UpChannel-0000> Notifying worker thread
```

4. 統合データベース (cons) で、Message Agent を実行します。

   ```
   dbremote -c "SERVER=cons;DBN=cons;UID=DBA;PWD=sql" -qc -v -o c:¥tutorial¥cons2.txt
   ```

   このコマンドは、FILE ベースのメッセージシステムを使用してリモートデータベースによって生成されたばかりのメッセージを受信して適用します。

♦ 統合データベースとリモートデータベースでデータを確認する

1. 現在、統合データベース (cons) に接続されていない場合は、次のコマンドを実行します。

   ```
   dbisql -c "SERVER=cons;DBN=cons;UID=DBA;PWD=sql"
   ```

2. 統合データベースに 4 ローのデータがすべて含まれていることを確認するには、次の文を実行して従業員テーブルの内容を表示します。

   ```
   SELECT * FROM employees;
   ```

   クエリは、従業員テーブルから次のデータを返します。ただし、hire_date カラムには、次のテーブルにある値ではなく、そのローを挿入した時間が記載されます。

employee_id	first_name	last_name	hire_date
1	Kelly	Meloy	2011-03-25 08:27:56.310
2	Melisa	Boysen	2011-03-25 08:27:56.310
3	Javier	Spoor	2011-03-25 08:30:26.110
102000001	Nelson	Kreitzer	2011-03-25 08:31:51.970

3. Interactive SQL との接続を切断します。

4. 現在、リモートデータベース (rem) に接続されていない場合は、次のコマンドを実行します。

   ```
   dbisql -c "SERVER=rem;DBN=rem;UID=DBA;PWD=sql"
   ```

   次の文を実行して従業員テーブルの内容を表示することにより、リモートデータベース (rem) に 4 ローのデータがすべて含まれていることを確認します。

   ```
   SELECT * FROM employees;
   ```

クエリは、従業員テーブルから次のデータを返します。ただし、hire_date カラムには、次のテーブルにあるデータではなく、そのローを挿入した時間が記載されます。

employee_id	first_name	last_name	hire_date
1	Kelly	Meloy	2011-03-25 08:27:56.310
2	Melisa	Boysen	2011-03-25 08:27:56.310
3	Javier	Spoor	2011-03-25 08:30:26.110
102000001	Nelson	Kreitzer	2011-03-25 08:31:51.970

5. Interactive SQL との接続を切断します。
6. 「レッスン 6：クリーンアップ」182 ページに進みます。

レッスン 6：クリーンアップ

最後のレッスンでは、RSOE と統合データベースとリモートデータベースを停止します。

♦ データベースサーバーを停止する

1. 次のコマンドを実行して、RSOE を停止します。

```
rsoe -s -cr "host=machine_iis;port=80;url_suffix=/rs/server/rs_server.dll"
-cs "host=machine_cons-t3500;port=8033;status_url=/oe_read_status"
-f srhttp_tutorial_farm -id srhttp_tutorial_server
```

2. 次のコマンドを実行して、統合データベースを停止します。

```
dbstop -y -c "SERVER=cons;DBN=cons;UID=DBA;PWD=sql"
```

次のコマンドを実行して、リモートデータベースを停止します。

```
dbstop -y -c "SERVER=rem;UID=DBA;PWD=sql"
```


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

SQL Remote Message Agent ユーティリティ (dbremote)

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	説明
@data	指定された環境変数または設定ファイルからオプションを読み込みます。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」『SQL Anywhere サーバー データベース管理』を参照してください。 設定ファイル内のパスワードなどの情報を保護する場合は、ファイル非表示ユーティリティを使用して、設定ファイルの内容を読みにくくすることができます。「ファイル非表示ユーティリティ (dbfhide)」『SQL Anywhere サーバー 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ーの **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 "Server=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ーの **field** というデータベースに接続する SQL Remote 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 "Server=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、SQL Remote 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; ..."*	接続パラメーターを指定します。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。接続にはユーザー 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" SQL Remote Message Agent を実行するユーザーには、REMOTE DBA 権限または DBA 権限が必要です。「REMOTE DBA 権限」 30 ページを参照してください。 SQL Remote Message Agent は、SQL Anywhere 接続パラメーターの全種類をサポートします。「接続パラメーター」『SQL Anywhere サーバー データベース管理』を参照してください。
-dl	[SQL Remote 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 SQL Remote Message Agent に指示します。デフォルトのオペレーション数は 20 です。*n* の値を大きくするとコミットが少なくなるため、入力メッセージの処理速度が向上します。ただし、トランザクションのサイズが大きくなると、デッドロックやブロックの原因になることもあります。
-l *length*	送信する各メッセージの最大長を指定します。**size** には、メモリ容量をバイト単位で指定します。単位をキロバイト、メガバイト、またはギガバイトで指定するには、それぞれ **k**、**m**、または **g** を使用してください。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。

オプション	説明
-m *size*	メッセージ作成と入力メッセージのキャッシュに、SQL Remote Message Agent が使用する最大メモリサイズを指定します。**size** には、メモリ容量をバイト単位で指定します。単位をキロバイト、メガバイト、またはギガバイトで指定するには、それぞれ **k**、**m**、または **g** を使用してください。デフォルトは 2048 キロバイトです (2 MB)。 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ーのグループには、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 SQL Remote 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削除されます。
-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ーとは異なるコンピューターで実行されている 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ー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。「delete_old_logs オプション [SQL Remote]」『SQL Anywhere サーバー データベース管理』を参照してください。

オプション	説明
-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。**size** には、メモリ容量をバイト単位で指定します。単位をキロバイト、メガバイト、またはギガバイトで指定するには、それぞれ **k**、**m**、または **g** を使用してください。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.dbs* に変更します。*xx* 00 で始まる番号で、1 ずつ増えていきます (xx は 2 桁以上の場合があります)。また、*yymmdd* は現在の年月日を表します。 SQL Remote 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。
-q	SQL Remote 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、SQL Remote 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 オプションで起動された場合、SQL Remote Message Agent は継続モードで動作します。メッセージ受信後に SQL Remote 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。

オプション	説明
-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、SQL Remote 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 SQL Remote 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。 「メッセージを受信する場合のパフォーマンス」90 ページを参照してください。
-rho *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」131 ページを参照してください。
-rp *number*	メッセージを消失したと判断するまでのポーリング受信回数を指定します。SQL Remote 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、SQL Remote 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、「メッセージを受信する場合のパフォーマンス」90 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。

オプション	説明
-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、-rho 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。 このオプションは **resend urgency** を制御します。再送信要求の検出から、SQL Remote 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、SQL Remote Message Agent が複数のユーザー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、SQL Remote 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。
-t	すべてのトリガー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、トリガー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ーを起動するとき、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。 トリガーの動作が 2 回実行されないようにするには、IF CURRENT REMOTE USER IS NULL ...END IF 文で、トリガーの本文を囲みます。「CURRENT REMOTE USER 特別値の使用」 47 ページを参照してください。

オプション	説明
-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、SQL Remote Message Agent をデーモンとして実行します。SQL Remote 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。 SQL Remote 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ーを格納してください。これは、SQL Remote 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ーに要求しないためです。 メッセージリンクパラメーターの詳細については、「リモートメッセージタイプ制御パラメーターの設定」 109 ページを参照してください。 SQL Remote Message Agent をデーモンとして起動すると、現在のユーザーの umask 設定によってパーミッションが制御されます。SQL Remote Message Agent に適切なパーミッションを付与するため、SQL Remote Message Agent を起動する前に umask 値を設定することをおすすめします。
-ui	X Window サーバーがサポートされている Linux で、使用可能な表示がない場合にシェルモードで SQL Remote Message Agent を起動します。
-ux	Solaris と Linux で SQL Remote Message Agent の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ー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ー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、SQL Remote Message Agent を実行するコンピューターにマルチプロセッサーを搭載しても、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。

オプション	説明
-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ー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。**size** には、メモリ容量をバイト単位で指定します。単位をキロバイト、メガバイト、またはギガバイトで指定するには、それぞれ **k**、**m**、または **g** を使用してください。デフォルトは 0 です。
-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	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* パラメーターは、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、SQL Remote 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。

備考

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 SQL Remote Message Agent を実行することができます。詳細については、*%SQLANY12%¥SDK¥Include¥*ディレクトリの *dbrmt.h* ファイルを参照してください。

SQL Remote Message Agent コマンドのユーザー ID には REMOTE DBA か DBA 権限が必要です。

SQL Remote Message Agent はいくつかのデータベース接続を使用します。

- **メッセージシステム制御パラメーター**　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。

  Windows では、メッセージリンク制御パラメーターは、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
  ```

参照

- 「SQL Remote Message Agent (dbremote)」 83 ページ
- 「SQL Remote メッセージシステム」 105 ページ
- 「REMOTE DBA 権限」 30 ページ

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込みます。「@data dbeng12/dbsrv12 サーバーオプション」『SQL Anywhere サーバー 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」『SQL Anywhere サーバー データベース管理』を参照してください。 設定ファイル内のパスワードなどの情報を保護する場合は、ファイル非表示ユーティリティを使用して、設定ファイルの内容を読みにくくすることができます。「ファイル非表示ユーティリティ (dbfhide)」『SQL Anywhere サーバー データベース管理』を参照してください。

オプション	説明
-ac *"keyword=value; ..."*	接続文字列で指定したデータベースに接続して、再ロードします。 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 サブスクライバー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。
-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 サブスクライバー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。

オプション	説明
-ap *size* [**k**]	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。**k** を使用して、キロバイトの単位を指定します (-ap 4k など)。データベースサーバーですでにデータベースが実行中の場合、サーバーのページサイズ (-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 dbeng12/dbsrv12 サーバーオプション」『SQL Anywhere サーバー 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 サーバー SQL リファレンス』を参照してください。
-c *"keyword=value; ..."*	データベース接続パラメーターを文字列として指定します。 ユーザー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 権限を持つユーザー ID を指定してください。 たとえば、次の文 (1 行で入力) は、ユーザー 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ー上で実行しているサンプルデータベースから、リモートユーザー 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 "Server=sample_server;DBN=demo; UID=DBA;PWD=sql" c:¥extract joe_remote` 接続パラメーターを指定しない場合、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ーを使用します。
-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 サーバー 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。 **注意** 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 12 紹介』を参照してください。

オプション	説明
-ek *key*	新しいデータベースで使用する暗号化キーを指定する。このオプションを使用すると、コマンドに暗号化キーを直接指定することで、強力に暗号化されたデータベースを作成できます。データベースの暗号化に使用されるアルゴリズムは、-ea オプションで指定した AES または AES_FIPS です。 -ek オプションを指定して、-ea オプションを指定しないと、AES アルゴリズムが使用されます。 **警告** 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、Sybase 製品の保守契約を結んでいるサポートセンター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 サーバー 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 (すべて 1 行で入力) 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 (すべて 1 行で入力) 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。

オプション	説明
-g	● **マテリアライズドビュー**　MANUAL REFRESH と定義されているマテリアライズドビューは、デフォルトでは再ロード後に初期化されません。このようなマテリアライズドビューを再ロードプロセス中に初期化するには、-g オプションを指定してください。-g を指定すると、データベースサーバーによって sa_refresh_materialized_views システムプロシージャーが実行されます。「sa_refresh_materialized_views システムプロシージャー」『SQL Anywhere サーバー SQL リファレンス』を参照してください。 -g オプションを指定するかどうかを決定するときには、すべてのマテリアライズドビューを初期化すると再ロードプロセスの実行時間が大幅に長くなる可能性があることを考慮に入れてください。しかし、-g オプションを指定しない場合、初期化されていないマテリアライズドビューを最初に使用しようとしたクエリはビューの初期化が完了するまで待つことになり、予想外の遅延が生じることがあります。-g オプションを指定しない場合は、再ロードの完了後に、マテリアライズドビューを手動で初期化することもできます。「マテリアライズドビューの初期化」『SQL Anywhere サーバー SQL の使用法』を参照してください。 ● **テキストインデックス**　MANUAL REFRESH と定義されているテキストインデックスは、デフォルトでは再ロード後に初期化されません。このようなテキストインデックスを再ロードプロセス中に初期化するには、-g オプションを指定してください。-g を指定すると、データベースサーバーによって sa_refresh_text_indexes システムプロシージャーが実行されます。「sa_refresh_text_indexes システムプロシージャー」『SQL Anywhere サーバー SQL リファレンス』を参照してください。
-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ー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ー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ー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ー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、他のユーザー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)」 194 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。「SYNCHRONIZE SUBSCRIPTION 文 [SQL Remote]」『SQL Anywhere サーバー SQL リファレンス』を参照してください。 パブリケーション、サブスクリプション、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、ユーザー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。

オプション	説明
-o *file*	出力ログファイルにメッセージを出力します。
-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 スクリプトファイルの名前を指定します。 再ロードスクリプト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、アンロードされた行数、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	プロシージャー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ーが実行します。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ーからの相対パスを使用します。

オプション	説明
-xp	データベースからストアドプロシージャーを抽出しません。
-xt	データベースからトリガーを抽出しません。
-xv	データベースからビューを抽出しません。
-xx	外部アンロードと外部ロードを実行する。データをアンロードする場合には、OUTPUT 文が使用されます。また、新規データベースにデータをロードする場合には、INPUT 文が使用されます。 アンロードのデフォルト動作では UNLOAD 文が使用され、ロードのデフォルト動作では LOAD TABLE 文が使用されます。内部の UNLOAD 文と LOAD TABLE 文を使用すると、OUTPUT 文と INPUT 文よりも高速で処理します。 データファイルのパスには、外部オペレーションでは dbxtract の現在の作業ディレクトリからの相対パスを使用し、内部の文ではデータベースサーバーからの相対パスを使用します。
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ーを指定します。

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ー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ー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ー上の他のユーザー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ー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、スクリプトファイルと、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スクリプトファイルを実行し、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 SQL スクリプトファイル名は *reload.sql* です。

リモートユーザーがグループの場合、グループのメンバーのユーザー 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ーに対して異なるユーザー 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ー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ー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** ユーザー 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ーの再定義など、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** ユーザー 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」 76 ページ
- 「データベース抽出」『SQL Anywhere サーバー SQL の使用法』

SQL Remote オプション

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*]

パラメーター	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。

備考

これらのオプションは SQL Remote Message Agent が使用します。SQL Remote Message Agent のコマンドで指定されたユーザー 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。

オプション	値	デフォルト
「blob_threshold オプション [SQL Remote]」『SQL Anywhere サーバー データベース管理』	整数 (バイト)	256
「compression オプション [SQL Remote]」『SQL Anywhere サーバー データベース管理』	-1 ～ 9 の整数	6

オプション	値	デフォルト
「delete_old_logs オプション [SQL Remote]」『SQL Anywhere サーバー データベース管理』	On、Off、Delay、*n* 日 (日数)	Off
「external_remote_options [SQL Remote]」『SQL Anywhere サーバー データベース管理』	On、Off	Off
「qualify_owners オプション [SQL Remote]」『SQL Anywhere サーバー データベース管理』	On、Off	On
「quote_all_identifiers オプション [SQL Remote]」『SQL Anywhere サーバー データベース管理』	On、Off	Off
「replication_error オプション [SQL Remote]」『SQL Anywhere サーバー データベース管理』	ストアドプロシージャー名	(プロシージャーなし)
「replication_error_piece オプション [SQL Remote]」『SQL Anywhere サーバー データベース管理』	ストアドプロシージャー名	(プロシージャーなし)
「save_remote_passwords オプション [SQL Remote]」『SQL Anywhere サーバー データベース管理』	On、Off	On
「sr_date_format オプション [SQL Remote]」『SQL Anywhere サーバー 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 サーバー データベース管理』	*時刻文字列*	*HH:NN:SS.SSSSSS*
「sr_timestamp_format [SQL Remote]」『SQL Anywhere サーバー 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 サーバー データベース管理』	On、Off	On

オプション	値	デフォルト
「verify_all_columns オプション [SQL Remote]」『SQL Anywhere サーバー データベース管理』	On、Off	Off
「verify_threshold オプション [SQL Remote]」『SQL Anywhere サーバー データベース管理』	整数 (バイト)	1000

SQL Remote ストアドプロシージャー

HTTP メッセージングシステムを管理するには、次のストアドプロシージャーを使用します。

sr_add_message_server システムプロシージャー

このプロシージャーは、リモートユーザーからの HTTP 要求を受信するのに必要な Web サービスを定義します。また、メッセージファイルが格納されているディレクトリにデータベースサーバーがアクセスするのを許可するための定義を行います。

構文

CALL sr_add_message_server('*owner*');

戻り値

なし。メッセージサーバーを定義するのに必要なオブジェクトの作成に問題がある場合は、エラーが戻されます。

備考

データベースが最初に初期化されたとき、リモートユーザーからの HTTP 要求を受け付けるのに必要な Web サービスはどれも定義されておらず、メッセージファイルが格納されているディレクトリにデータベースサーバーがアクセスするのを許可する定義はありません。これらのオブジェクトの作成は、誰がすべてのオブジェクトを所有するのかを指定するオプションのパラメーターを持つ sr_add_message_server ストアドプロシージャーの使用により自動化されています。オブジェクト名は重複することができません。

参照

- 「sr_drop_message_server システムプロシージャー」 207 ページ
- 「sr_update_message_server システムプロシージャー」 207 ページ

例

次の文により、メッセージサーバーデータベース (msgsrv) はメッセージサーバーに必要なすべてのオブジェクトを定義して、すべてのオブジェクトが cons ユーザー (この例では統合データベース) によって所有されることを指定します。


```
GRANT GROUP TO cons;
SET REMOTE http OPTION cons.root_directory='c:¥¥tutorial¥¥messages';
CALL sr_add_message_server( 'cons' );
COMMIT;
```

sr_drop_message_server システムプロシージャー

このプロシージャーは、sr_add_message_server によって作成されたすべてのオブジェクトを削除します。

構文

CALL sr_drop_message_server;

戻り値

なし。メッセージサーバーを定義するのに必要なオブジェクトの作成に問題がある場合は、エラーが戻されます。

備考

ストアドプロシージャーは、sr_add_message_server によって作成されたすべてのオブジェクトを削除するのに使用されます。

参照

- 「sr_add_message_server システムプロシージャー」206 ページ
- 「sr_update_message_server システムプロシージャー」207 ページ

sr_update_message_server システムプロシージャー

このプロシージャーは、メッセージサーバーの SQL Remote 定義が変更されるたびに呼び出す必要があります。

構文

CALL sr_update_message_server('*owner*');

戻り値

なし。メッセージサーバーを定義するのに必要なオブジェクトの作成に問題がある場合は、エラーが戻されます。

備考

このプロシージャーは、ストアドプロシージャーで作成されたオブジェクトを所有するユーザーというオプションのパラメーターを持ちます。

参照

- 「sr_add_message_server システムプロシージャー」206 ページ
- 「sr_drop_message_server システムプロシージャー」207 ページ

SQL Remote システムプロシージャー

以下のストアドプロシージャー名と引数は、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ーフェイスを提供します。

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ーには次の条件が適用されます。

- ストアドプロシージャーには DBA 権限が必要です。
- プロシージャーでは、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。プロシージャーのアクションは、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。
- SQL Remote 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 );
```

SQL Remote 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 SQL Remote Message Agent に返します。

sp_hook_dbremote_begin システムプロシージャー

このシステムプロシージャーを使用すると、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。

備考

この名前のプロシージャーが存在する場合、プロシージャーは、SQL Remote 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ー

このシステムプロシージャーを使用すると、SQL Remote 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。

備考

この名前のプロシージャーが存在する場合、プロシージャーは、SQL Remote 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ー

このシステムプロシージャーを使用すると、SQL Remote 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**	プロシージャーが呼び出されるときにはこのローは **false** です。プロシージャーがこのローを **true** に更新すると、SQL Remote Message Agent がシャットダウンされます。

備考

この名前のプロシージャーが存在する場合、プロシージャーは、SQL Remote 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、SQL Remote 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ー

このシステムプロシージャーを使用すると、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ー

このシステムプロシージャーを使用すると、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ーを使用すると、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ーを使用すると、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ーを使用すると、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ーを使用すると、SQL Remote Message Agent がリモートユーザーからの 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ーの名前

sp_hook_dbremote_message_apply_begin

このストアドプロシージャーを使用すると、SQL Remote Message Agent がユーザー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ーの名前

sp_hook_dbremote_message_apply_end

このストアドプロシージャーを使用すると、SQL Remote Message Agent がユーザー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ーの名前

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。次のビューを使用して SQL Remote データにアクセスできます。

- 「SYSARTICLE システムビュー」『SQL Anywhere サーバー SQL リファレンス』
- 「SYSARTICLECOL システムビュー」『SQL Anywhere サーバー SQL リファレンス』
- 「SYSPUBLICATION システムビュー」『SQL Anywhere サーバー SQL リファレンス』
- 「SYSREMOTEOPTION システムビュー」『SQL Anywhere サーバー SQL リファレンス』
- 「SYSREMOTEOPTIONTYPE システムビュー」『SQL Anywhere サーバー SQL リファレンス』
- 「SYSREMOTETYPE システムビュー」『SQL Anywhere サーバー SQL リファレンス』
- 「SYSREMOTEUSER システムビュー」『SQL Anywhere サーバー SQL リファレンス』
- 「SYSSUBSCRIPTION システムビュー」『SQL Anywhere サーバー SQL リファレンス』

SQL Remote の SQL 文

次の SQL 文を使用して、SQL Remote コマンドを実行します。

- 「ALTER PUBLICATION 文 [Mobile Link] [SQL Remote]」『SQL Anywhere サーバー SQL リファレンス』
- 「ALTER REMOTE MESSAGE TYPE 文 [SQL Remote]」『SQL Anywhere サーバー SQL リファレンス』
- 「CREATE PUBLICATION 文 [Mobile Link] [SQL Remote]」『SQL Anywhere サーバー SQL リファレンス』
- 「CREATE REMOTE MESSAGE TYPE 文 [SQL Remote]」『SQL Anywhere サーバー SQL リファレンス』
- 「CREATE SUBSCRIPTION 文 [SQL Remote]」『SQL Anywhere サーバー SQL リファレンス』
- 「CREATE TRIGGER 文」『SQL Anywhere サーバー SQL リファレンス』
- 「DROP PUBLICATION 文 [Mobile Link] [SQL Remote]」『SQL Anywhere サーバー SQL リファレンス』
- 「DROP REMOTE MESSAGE TYPE 文 [SQL Remote]」『SQL Anywhere サーバー SQL リファレンス』
- 「DROP SUBSCRIPTION 文 [SQL Remote]」『SQL Anywhere サーバー SQL リファレンス』
- 「GRANT CONSOLIDATE 文 [SQL Remote]」『SQL Anywhere サーバー SQL リファレンス』
- 「GRANT PUBLISH 文 [SQL Remote]」『SQL Anywhere サーバー SQL リファレンス』
- 「GRANT REMOTE 文 [SQL Remote]」『SQL Anywhere サーバー SQL リファレンス』
- 「GRANT REMOTE DBA 文 [Mobile Link] [SQL Remote]」『SQL Anywhere サーバー SQL リファレンス』
- 「PASSTHROUGH 文 [SQL Remote]」『SQL Anywhere サーバー SQL リファレンス』
- 「REMOTE RESET 文 [SQL Remote]」『SQL Anywhere サーバー SQL リファレンス』
- 「REVOKE CONSOLIDATE 文 [SQL Remote]」『SQL Anywhere サーバー SQL リファレンス』
- 「REVOKE PUBLISH 文 [SQL Remote]」『SQL Anywhere サーバー SQL リファレンス』
- 「REVOKE REMOTE 文 [SQL Remote]」『SQL Anywhere サーバー SQL リファレンス』
- 「REVOKE REMOTE DBA 文 [SQL Remote]」『SQL Anywhere サーバー SQL リファレンス』
- 「SET REMOTE OPTION 文 [SQL Remote]」
- 「START SUBSCRIPTION 文 [SQL Remote]」『SQL Anywhere サーバー SQL リファレンス』
- 「STOP SUBSCRIPTION 文 [SQL Remote]」『SQL Anywhere サーバー SQL リファレンス』
- 「SYNCHRONIZE SUBSCRIPTION 文 [SQL Remote]」『SQL Anywhere サーバー SQL リファレンス』
- 「UPDATE 文 [SQL Remote]」『SQL Anywhere サーバー SQL リファレンス』

SET REMOTE OPTION 文 [SQL Remote]

SQL Remote メッセージリンクのメッセージ制御パラメーターを設定します。

構文

SET REMOTE *link-name* **OPTION**
[*userid*.| **PUBLIC**.]*link-option-name* = *link-option-value*

link-name :
file
| **ftp**
| **http**
| **smtp**

link-option-name :
common-options
| *file-options*
| *ftp-options*
| *smtp-options*

common-options :
debug
| **encode_dll**
| **max_retries**
| **output_log_send_on_error**
| **output_log_send_limit**
| **output_log_send_now**
| **pause_after_failure**

file-options :
directory
| **invalid_extensions**
| **unlink_delay**

ftp-options :
active_mode
| **host**
| **invalid_extensions**
| **password**
| **port**
| **root_directory**
| **reconnect_retries**
| **reconnect_pause**
| **suppress_dialogs**
| **user**

http-options :
| **certificate**
| **client_port**
| **https**
| **password**
| **proxy**
| **reconnect_retries**
| **reconnect_pause**
| **root_directory**
| **url**
| **user**

smtp-options :
local_host
| **pop3_host**
| **pop3_password**
| **pop3_userid**
| **smtp_authenticate**
| **smtp_option**

| smtp_password
| smtp_userid
| suppress_dialogs
| top_supported

link-option-value : *string*

パラメーター

userid　*userid* を指定しない場合、現在のパブリッシャーが使用されます。

common-options　FILE、FTP、HTTP、SMTP メッセージシステムに共通のオプションは、次のとおりです。

○ **debug**　このパラメーターは、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、メッセージシステムに固有のデバッグ出力が表示されます。この情報は、メッセージシステムのトラブルシューティングに使用できます。

○ **max_retries**　デフォルトでは、SQL Remote が継続モードで実行されていて、メッセージシステムにアクセスするときにエラーが発生する場合、送受信フェーズの後に停止します。このパラメーターを使用して、停止前に SQL Remote が再試行する送受信フェーズの回数を指定します。

○ **output_log_send_on_error**　エラーが発生すると、情報を送信します。

○ **output_log_send_limit**　統合データベースに送信される情報の量を制限します。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_now**　YES に設定されると、統合データベースに出力ログ情報を送信します。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。

○ **pause_after_failure**　このパラメーターが使えるのは、max_retries がゼロ以外の値に指定され、SQL Remote が継続モードで実行されている場合です。メッセージシステムでエラーが発生すると、このパラメーターは、SQL Remote が送受信フェーズを再試行する間に待機する時間を秒で指定します。

○ **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。

file-options　これらのオプションは、ファイルメッセージシステムにのみ適用されます。

○ **directory**　メッセージが格納されるディレクトリです。このパラメーターは、SQLREMOTE 環境変数の代替となります。

○ **invalid_extensions**　メッセージングシステムでのファイルの生成時に SQL Remote 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。

○ **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。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。

ftp-options　これらのオプションは、FTP メッセージシステムにのみ適用されます。

○ **active_mode**　このパラメーターは、SQL Remote がクライアント／サーバー接続を確立する方法を制御します。このパラメーターは、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。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 サーバーがすべてのデータ接続を開始します。

○ **host**　FTP サーバーを実行しているコンピューターのホスト名。このパラメーターには、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。

○ **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。

○ **password**　FTP ホストにアクセスするためのパスワード。

○ **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。

○ **reconnect_retries**　失敗になる前に、リンクがサーバー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。このパラメーターを設定すると、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。このパラメーターを設定すると、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。

○ **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。

○ **suppress_dialogs**　このパラメーターは、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 サーバー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。

○ **user**　FTP ホストにアクセスするためのユーザー名。

http-options　これらのオプションは、HTTP メッセージシステムにのみ適用されます。

○ **certificate**　安全な (HTTPS) 要求を行うには、HTTPS サーバーで使用される証明書にクライアントがアクセスできる必要があります。必要な情報は、セミコロンで区切られたキー／値のペアの文字列で指定されます。ファイルキーを使用して証明書のファイル名を指定できます。ファイルキーと証明書キーを一緒に指定することはできません。次のキーを使用できます。

Key	Abbreviation	Description
file		証明書のファイル名

Key	Abbreviation	Description
certificate	cert	証明書自体
company	co	証明書で指定された会社
unit		証明書で指定された会社の部署
name		証明書で指定された通称

証明書は、HTTPS サーバーに対する要求、または安全でないサーバーから安全なサーバーにリダイレクトされる可能性がある要求に対してのみ必要です。PEM でフォーマットされた証明書のみがサポートされています。**certificate='***file=filename***'**

○ **client_port**　SQL Remote が HTTP を使用して通信するポート番号を識別します。これは「送信」TCP/IP 接続をフィルターするファイアウォール経由で接続するためのもので、この用途にかぎり推奨されています。単一のポート番号、ポート番号の範囲、または両方の組み合わせを指定できます。指定したクライアントポートの数が少ないと、SQL Remote が前回の実行でポートを閉じた後すぐにオペレーティングシステムがポートを解放しなかった場合に、SQL Remote でメッセージの送受信ができなくなる可能性があります。

○ **debug**　YES に設定すると、すべての HTTP コマンドと応答が出力ログに表示されます。この情報は、HTTP 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。

○ **https**　HTTPS (**https=yes**) または HTTP (**https=no**) を使用するかどうかを指定します。

○ **password**　メッセージサーバーのデータベースパスワード。パスワードは、RFC 2617 の基本認証を使用してサードパーティの HTTP サーバーとゲートウェイに対する認証を行います。

○ **proxy_host**　プロキシサーバーの URI を指定します。SQL Remote がプロキシサーバーを介してネットワークにアクセスする場合に使用します。SQL Remote がプロキシサーバーに接続し、そのプロキシサーバーを介してメッセージサーバーに要求を送信することを示します。

'

○ **reconnect_retries**　失敗になる前に、リンクがサーバー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。このパラメーターを設定すると、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。このパラメーターを設定すると、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。

○ **root_directory**　この HTTP 制御パラメーターは、クライアント側で指定された場合には無視されます。sr_add_message_server または sr_update_message_server ストアドプロシージャー

を呼び出す前に、メッセージサーバーでこの制御パラメーターを定義します。HTTP メッセージシステムを使用する場合、リモートユーザーまたはパブリッシャーに指定するアドレスには、1 つのサブディレクトリのみを含めることができます。複数のサブディレクトリを含めることはできません。

○ **url**　サーバー名または IP アドレスを指定し、任意で、使用する HTTP サーバーのポート番号をセミコロンで区切って指定します。要求が Relay Server を経由する場合は、URL 拡張子を任意で追加して、要求の渡し先のサーバーファームを示すことができます。

○ **user**　メッセージサーバーのデータベースユーザー ID。RFC 2617 の基本認証を使用してサードパーティの HTTP サーバーとゲートウェイに対する認証を行います。

smtp-options　これらのオプションは、SMTP メッセージシステムにのみ適用されます。

○ **local_host**　ローカルコンピューターの名前。SQL Remote がローカルホスト名を決定できないコンピューターで設定すると便利です。ローカルホスト名は、任意の SMTP サーバー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。

○ **pop3_host**　POP ホストを実行しているコンピューターの名前。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_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。

○ **pop3_userid**　メールの受信に使用するユーザー ID。POP ユーザー ID は、SMTP/POP3 ログインウィンドウのユーザー 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ー ID を取得してください。

○ **smtp_host**　SMTP サーバーが動作しているコンピューターの名前。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。

○ **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 サーバーでサポートされているわけではありません。top_supported パラメーターを 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 サーバーで動作します。デフォルトは YES です。

○ **smtp_authenticate**　SMTP リンクがユーザー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、このパラメーターを NO に設定します。

○ **smtp_userid**　SMTP 認証のユーザー ID。デフォルトでは、このパラメーターは pop3_userid パラメーターと同じ値を取ります。smtp_userid は、ユーザー ID が POP サーバー上のユーザー 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、このパラメーターは pop3_password パラメーターと同じ値を取ります。smtp_password は、ユーザー ID が POP サーバー上のユーザー ID と異なる場合にのみ設定する必要があります。

○ **suppress_dialogs**　このパラメーターが true に設定されている場合は、メールサーバー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。

備考

メッセージリンクの初回使用時に、ユーザーがメッセージリンクパラメーターを [メッセージリンク] ウィンドウに入力すると、SQL Remote (dbremote) Message Agent はそのパラメーターを保存します。その場合はこの文を明示的に使用する必要はありません。この文は、たくさんのデータベースから抽出を行うための統合データベースを作成する場合に非常に効果的です。

オプション名では、大文字と小文字が区別されます。オプションの値で大文字と小文字が区別されるかどうかは、オプションによって異なります。ブール値の場合、大文字と小文字は区別されません。パスワード、ディレクトリ名、その他の文字列では、ファイルシステム (ディレクトリ名の場合) またはデータベース (ユーザー ID とパスワードの場合) で大文字と小文字を区別するかどうかに従います。

パーミッション

DBA 権限。パブリッシャーは自分自身のオプションを設定できます。

関連する動作

オートコミット。

参照

- 「リモートデータベースからのエラーの収集」 131 ページ
- 「リモートメッセージタイプ制御パラメーターの設定」 109 ページ
- 「カスタムエンコードスキーム」 104 ページ
- 「SET OPTION 文」『SQL Anywhere サーバー SQL リファレンス』
- 「FTP メッセージシステム」 111 ページ
- 「FILE メッセージシステム」 110 ページ
- 「HTTP メッセージシステム」 114 ページ
- 「チュートリアル : HTTP メッセージシステムを使用したレプリケーションシステムの設定」 155 ページ
- 「SMTP メッセージシステム」 118 ページ

標準と互換性

- **SQL/2008**　ベンダー拡張。

例

次の文は、ユーザー 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';
```

次の文は、生成されるメッセージの特定ファイル拡張子を SQL Remote が使用しないようにします。

```
SET REMOTE FTP OPTION "Public"."invalid_extensions"='exe,pif,dll,bat,cmd,vbs';
```

次の文は、ユーザー myuser 用の HTTP リンクのローカルホストを指すように URL を設定します。

```
SET REMOTE HTTP OPTION myuser.url='localhost:8033';
```

次の文は、要求を srhttp ファームに転送する Relay Server を指すように HTTP URL を設定します。

```
SET REMOTE HTTP OPTION "public"."url"='iis7.company.com:80/rs/client/rs_client.dll/srhttp';
```


索引

記号

A

B

C

D

E

F

G

H

I

L

M

N

O

P

R

S

U

V

W

あ

い

え

お

か

き

く

け

こ

さ

し

す

せ

そ

た

ち

て

と

は

ひ

ふ

ほ

め

ゆ

り

れ

