SYBASE®
An SAP® Company

デザイン・ガイド

Replication Server® 15.7.1

ドキュメント ID： DC35493-01-1571-01
改訂： 2012 年 4 月

Sybase, Inc., One Sybase Drive, Dublin, CA 94568.

目次

表記の規則

ここでは、Sybase® マニュアルで使用しているスタイルおよび構文の表記規則について説明します。

表記の規則

構文要素	定義
`mono-spaced(fixed-width)`	• SQL およびプログラム・コード • 表示されたとおりに入力する必要のあるコマンド • ファイル名 • ディレクトリ名
`italic mono-spaced`	SQL またはプログラム・コードのスニペット内では、ユーザ指定の値のプレースホルダ (以下の例を参照)
italic	• ファイルおよび変数の名前 • 他のトピックまたはマニュアルとの相互参照 • 本文中では、ユーザ指定の値のプレースホルダ (以下の例を参照) • 用語解説に含まれているテキスト内の用語
bold san serif	• コマンド、関数、ストアド・プロシージャ、ユーティリティ、クラス、メソッドの名前 • 用語解説のエントリ (用語解説内) • メニュー・オプションのパス • 番号付きの作業または手順内では、クリックの対象となるボタン、チェック・ボックス、アイコンなどのユーザ・インタフェース (UI) 要素

必要に応じて、プレースホルダ (システムまたは設定固有の値) の説明が本文中に追加されます。次に例を示します。

次のコマンドを実行します。

```
installation directory¥start.bat
```

installation directory はアプリケーションがインストールされた場所です。

構文の表記規則

構文要素	定義
{ }	中カッコで囲まれたオプションの中から必ず 1 つ以上を選択する。コマンドには中カッコは入力しない。
[ ]	角カッコは、オプションを選択しても省略してもよいことを意味する。コマンドには角カッコは入力しない。
( )	このカッコはコマンドの一部として入力する。
|	縦線はオプションのうち 1 つのみを選択できることを意味する。
,	カンマは、表示されているオプションを必要な数だけ選択でき、選択したものをコマンドの一部として入力するときにカンマで区切ることを意味する。
...	省略記号 (...) は、直前の要素を必要な回数だけ繰り返し指定できることを意味する。省略記号はコマンドには入力しない。

大文字と小文字の区別

- すべてのコマンド構文およびコマンドの例は、小文字で表記しています。ただし、複写コマンド名では、大文字と小文字が区別されません。たとえば、**RA_CONFIG**、**Ra_Config**、**ra_config** は、すべて同じです。
- 設定パラメータの名前では、大文字と小文字が区別されます。たとえば、**Scan_Sleep_Max** は、**scan_sleep_max** とは異なり、パラメータ名としては無効になります。
- データベース・オブジェクト名は、複写コマンド内では、大文字と小文字が区別されません。ただし、複写コマンドで大文字と小文字が混在したオブジェクト名を使用する場合 (プライマリ・データベースの大文字と小文字が混在したオブジェクト名と一致させる場合)、引用符でオブジェクト名を区切ります。次に例を示します。**pdb_get_tables** "*TableName*"
- 識別子および文字データでは、使用しているソート順によっては大文字と小文字が区別されます。
  - “binary” などの大文字と小文字を区別するソート順を使用する場合には、識別子や文字データは、大文字と小文字を正しく入力してください。
  - “nocase” などの大文字と小文字を区別しないソート順を使用する場合には、識別子や文字データは、大文字と小文字をどのような組み合わせでも入力できます。

用語

Replication Agent™ は、Adaptive Server® Enterprise、Oracle、IBM DB2 UDB、Microsoft SQL Server 用の Replication Agent を表現するために使用される一般的な用語です。具体的な名前は、次のとおりです。

- RepAgent － Adaptive Server Enterprise 用の Replication Agent スレッド
- Replication Agent for Oracle
- Replication Agent for Microsoft SQL Server
- Replication Agent for UDB － Linux、Unix、Windows 用の IBM DB2

概要

Replication Server® を使用して、分散データ・アプリケーションを作成し、維持します。

集中データベース・システムと分散データベース・システム

企業環境における非集中化オペレーションへの動向が、分散データベース・システムに移行する方向へ組織を加速させています。

従来の企業内コンピューティング・モデルでは、情報システム部門が集中化された企業データベース・システムを管理していました。通常は、企業の本社に置かれたメインフレーム・コンピュータが、必要とされるパフォーマンス・レベルを提供していました。リモートのサイトは、情報システム部門によって提供されるアプリケーションを使用して WAN (広域ネットワーク) 経由で企業データベースにアクセスしていました。

今日の世界的な企業は、WAN と結ばれた多くの LAN (ローカル・エリア・ネットワーク) を持つと同時に、LAN 上にさらにデータ・サーバやアプリケーションを擁しています。各サイトのクライアント・アプリケーションは、LAN 経由でローカルに、または WAN 経由でリモートからデータにアクセスする必要があります。たとえば、東京のクライアントは、東京のデータ・サーバに格納されているテーブルにローカルにアクセスしたり、ニューヨークのデータ・サーバに格納されているテーブルにリモートからアクセスしたりします。

分散データベース環境では、重要な企業データを管理するために企業の本社または各地域の本社にメインフレーム・コンピュータが必要とされるのに対して、リモート・サイトのクライアントはローカル処理用にミニコンピュータやサーバ・クラスのワークステーションを使用します。

集中データベース・システムと分散データベース・システムのどちらも、リモート・アクセスに関する次のような問題に対処する必要があります。

- WAN のトラフィックが過多になるとネットワークの応答が遅くなる。たとえば、意思決定支援アプリケーションが大量のローを要求する場合、ミッション・クリティカルなトランザクション処理アプリケーションが悪影響を受けることがあります。
- 集中化されたデータ・サーバは、多数のユーザが一度にアクセスした場合のボトルネックになることがある。
- ネットワークに障害が発生するとデータを利用できなくなる。

データを複写することの利点

リモート・データベースへのアクセスに関するパフォーマンスの問題と可用性の問題は、データを送信元データベースからローカル・データベースに複写することによって解決できます。Replication Server は、データを複写するための、費用効果が高く障害に強い (フォールト・トレラントな) システムを提供します。

Replication Server は、複数のデータベースにあるデータを最新の状態に保つので、クライアントはリモートの集中型データベースの代わりに、ローカル・データにアクセスすることができます。集中型データ・システムと比べると、複写システムではシステムのパフォーマンスとデータ可用性が向上し、通信のオーバヘッドを低減します。

複写システムは、ローではなくトランザクションを転送するので、Replication Server はデータの可用性を増大させるとともに、システム中の複写データの整合性を維持します。また、ストアド・プロシージャを複写することもできるので、パフォーマンスがさらに向上します。

パフォーマンスの向上
分散複写システムでは、クライアントは WAN にアクセスしなくてもデータ要求をローカル・データ・サーバで完了できます。ローカル・クライアントのパフォーマンスは次のような理由で改善されます。

- LAN のデータ転送速度は、WAN のデータ転送速度よりも高速である。
- ローカル・クライアントは中央データ・サーバのリソースを競合しないでローカル・データ・サーバのリソースを共有する。
- ローカルの意思決定支援アプリケーションが集中化された OLTP アプリケーションから分離されるので、ロックのトラフィックと競合が大幅に減少する。

データの可用性の増大
分散複写システムでは、データはローカル・サイトとリモート・サイトで複写されるので、プライマリ・データ・ソースまたは WAN で発生することに関係なく、クライアントは動作を継続できます。

- リモート・サイトで障害が発生した場合でも、クライアントは複写データのローカル・コピーを使用し続けることができます。
- WAN に障害が発生した場合は、クライアントはローカルの複写データを使用し続けることができます。
- ローカル・データ・サーバに障害が発生した場合は、クライアントは別のサイトの複写データに切り替えることができます。

WAN 経由の通信が失敗すると、他のサイトの Replication Server がトランザクションをステーブル・キュー (ディスク記憶領域) に格納して、利用できなくなったサ

イトの複写テーブルを、通信がレジュームされたときに最新の状態にすることができます。複写ファンクションが送信元データベースで開始された場合は、送信先サイトに配布できるようになるまでステーブル・キューに格納されます。

Replication Server によるデータ分配

Replication Server はトランザクション (データ・コピーではなくインクリメンタル変更) とストアド・プロシージャの呼び出し (ストアド・プロシージャ自体ではありません) を複写するので、データの整合性を維持すると同時にパフォーマンスのよい分散データ環境を提供します。

Replication Server は次の機能によって、ネットワーク上でデータを分配します。

- アプリケーション開発者とシステム管理者に、データとストアド・プロシージャを複写するための、柔軟性のあるパブリッシュとサブスクライブ・モデルを提供する。
- ネットワーク上のトランザクションの整合性を維持しながら、複写トランザクションを管理する。

パブリッシュとサブスクライブ・モデル

Replication Server は、プライマリ・データベースとレプリケート・データベース間のデータの複写に、基本的なパブリッシュとサブスクライブ・モデルを使用しています。

Replication Server システムでは、送信元データベースのトランザクションは Replication Agent によって検出されて、ローカルの Replication Server に転送されます。その情報は LAN や WAN を介して送信先サイトの Replication Server に分配されます。これらの Replication Server は、リモート・クライアントの要求に従って、ターゲット・データベースを更新します。ネットワークやシステムのコンポーネントが失敗すると、配布処理中のデータは一時的にキューに格納されます。失敗したコンポーネントが動作可能になると、複写システムはデータのコピーの同期を取り直して、通常の複写をレジュームします。

プライマリ・データは、Replication Server が他のデータベースに複写するデータのソースです。ユーザは、他のサイトの Replication Server が「サブスクリプションを作成」する対象のプライマリ・サイトにデータを「パブリッシュ」します。はじめにプライマリ・データのロケーションを指定するための「複写定義」を作成します。複写定義ではテーブルの構造を記述し、テーブルのプライマリ・コピーを含むデータベースを指定します。管理しやすくするために、複写定義を「パブリケーション」にまとめることができます。

複写定義やパブリケーションを作成しただけでは、Replication Server がデータを複写するようにはなりません。他のデータベース内のデータを複写するように Replication Server に指示するための複写定義 (またはパブリケーション) に対するサ

ブスクリプションを作成する必要があります。サブスクリプションは SQL の **select** 文に似ています。**where** 句を追加して、必要なデータだけが複写されるように、ローカル・データベースに複写するテーブルのローを指定できます。

Replication Server のバージョン 11.5 からは、1 つのプライマリ・テーブルに対して複数の複写定義を持つことができます。レプリケート・テーブルでは、異なる複写定義のサブスクリプションを作成し、データの異なるビューを取得することができます。

複写定義またはパブリケーションのサブスクリプションを作成すると、Replication Server は、データのサブスクリプションを持つデータベースにトランザクションを複写します。

複写ファンクション

データベース間で Adaptive Server のストアド・プロシージャを非同期で複写すると、多数の変更を 1 つの「複写ファンクション」にカプセル化することによって、通常のデータ複写のパフォーマンスを向上できます。複写ファンクションはテーブル複写定義と関連付けられていないので、データを直接修正するかどうかに関係なくストアド・プロシージャを実行できます。

ストアド・プロシージャの呼び出しは、プライマリ・データベースからレプリケート・データベースに、またレプリケート・データベースからプライマリ・データベースに複写できます。

複写ファンクションを使用すると、別のデータベースでストアド・プロシージャを実行できます。次のことができます。

- サブスクリプションを作成するサイトに Adaptive Server ストアド・プロシージャの実行を複写する。
- データベースの実際の変更ではなく、ストアド・プロシージャの名前とパラメータだけを複写することによって、パフォーマンスを向上させる。

テーブルと同様に、複写ストアド・プロシージャは、「ファンクション複写定義」と呼ばれる複写定義とサブスクリプションを持つことができます。複写ストアド・プロシージャが実行されると、Replication Server はその名前と実行パラメータを、サブスクリプションを作成するサイトに渡し、そこで対応するストアド・プロシージャが実行されます。

ファンクション複写定義は、プライマリ・データ・サイトで作成します。Replication Server は、適用ファンクションと要求ファンクションをサポートします。

適用ファンクションと要求ファンクションは、プライマリ・データベースからレプリケート・データベースに複写されます。レプリケート・サイトでファンクション複写定義のサブスクリプションを作成し、プライマリ・データベースで複

写対象としてストアド・プロシージャにマークを付けます。適用ファンクションは、maint_user によってレプリケート・データベースで適用されますが、要求ファンクションはプライマリ・データベースでストアド・プロシージャを実行したのと同じユーザによって、レプリケート・データベースで適用されます。

参照：
- 適用ファンクション (46 ページ)
- 要求ファンクション (72 ページ)

トランザクション管理

Replication Server では、トランザクション処理サービスの提供をデータ・サーバに依存しています。分散データの一貫性を保証するために、データ・サーバは、原子性や一貫性といったトランザクション処理規則に従う必要があります。

プライマリ・データを格納するデータ・サーバは、分散データベース・システムで必要とされるほとんどの同時制御機能を提供します。トランザクションが、プライマリ・データを持つテーブルの更新に失敗すると、Replicatin Server は、そのトランザクションを他のサイトには分配しません。トランザクションによってプライマリ・データが更新されると、Replication Server は変更を分配し、障害が発生しないかぎり、データのサブスクリプションを作成したすべてのサイトで更新が正常に行われます。

Replication Server は、複写データの一貫性を維持するためにオプティミスティック同時実行制御を使用します。この制御は次のことを行います。

- 分散トランザクションの処理中にデータをロックしないので、データの可用性を高める。競合発生時には変更内容がロールバックされる。
- トランザクションを処理するのに必要なシステム・リソースが少なくてすむ。
- 分散トランザクションに関与するためにデータ・サーバが特別な分散トランザクション処理機能を持つ必要がない。

失敗した複写トランザクション
プライマリ・データを修正しても、他のサイトのデータのレプリケート・コピーを更新できないことがあります。

複写テーブルに対する更新が失敗する理由は次のとおりです。

- データ・サーバのメンテナンス・ユーザのログイン名が、レプリケート・データを更新するのに必要なパーミッションを持っていない。
- システムのリカバリが発生したため、データのレプリケート・バージョンとプライマリ・バージョンが矛盾している。
- クライアントが、プライマリ・バージョンを更新するのではなく、レプリケート・データを直接更新している。

- レプリケート・テーブルを格納するデータ・サーバがプライマリ・バージョンを格納するデータ・サーバによる要求を受け付けないよう制限されている。
- テーブルの複写コピーを格納するデータ・サーバが、データベース内の領域がないなどのシステムの失敗のために、トランザクションを拒否している。

トランザクションが失敗すると、Replication Server はデータ・サーバからエラーを受け取ります。データ・サーバのエラーは、Replication Server のエラー・アクションにマップされます。失敗したトランザクションに対するデフォルトのアクションは、Replication Server エラー・ログにメッセージ (データ・サーバによって返されるメッセージなど) を書き込んで、データベースのコネクションをサスペンドすることです。問題が解決されると、データベース・コネクションをレジュームし、Replication Server は失敗したトランザクションをリトライします。

Replication Server に失敗したトランザクションを例外ログ (Replication Server システム・データベースの 3 つのテーブル) に記録させて次のトランザクションの処理を続けさせることもできます。RSSD の説明については、「Replication Server」を参照してください。

例外ログを使用する場合は、レプリケート・データとプライマリ・データとの一貫性を確保するために例外ログに保存されているトランザクションを手動で解析する必要があります。場合によっては、この処理はインテリジェント・アプリケーション・プログラム内で、拒否されたトランザクションを処理するためのロジックをカプセル化することによって、自動化できます。

複数のデータ・サーバとデータベース内のデータを修正するトランザクション
複数のデータ・サーバ内のプライマリ・データを修正するトランザクションは、さらに別の同時制御を必要とすることがあります。トランザクション処理の必要条件は、トランザクション内のすべてのオペレーションが実行されるか、どれも実行されないかのいずれかです。トランザクションが 1 つのデータ・サーバで失敗した場合、そのトランザクションで更新された他のすべてのデータ・サーバでロールバックされなければなりません。

通常、各プライマリ・データベースに対して存在する Replication Agent は 1 つだけです。1 つのトランザクションが複数のプライマリ・データベースを更新する場合は、そのトランザクションは各プライマリ・データベースのそれぞれに対して、1 つずつの複数の独立したトランザクションとして複写されます。また、そのようなトランザクションを 1 つのストアド・プロシージャにカプセル化して、サブスクリプションを作成するサイトにアトミックな (分割できない) 単位として、渡すこともできます。

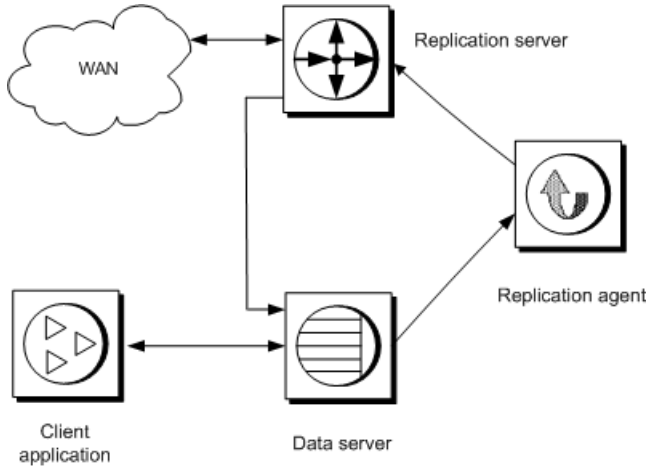
参照：
- Replication Server (12 ページ)

複写システムのコンポーネント

Replication Server は、既存のシステムとアプリケーションから複写システムを構築して、それを組織の成長と変化に応じて拡張することができるオープン・アーキテクチャを持っています。

WAN ベースで Replication Server を使用している分散データベース・システム内の 1 つの複写システム・サイトを示します。

図 1 : 複写システム



複写システム・ドメイン

「複写システム・ドメイン」とは、同じ ID サーバを使用するすべての複写システム・コンポーネントを指します。

次の制限事項のもとで、複数の複写システム・ドメインを設定できます。

- 異なるドメインに属する Replication Server 間では、データを交換できません。各ドメインは、相互に通信できない独立した複写システムとして扱います。異なるドメインに属する Replication Server 間にルートを作成することはできません。
- 1 つのデータベースを管理できるのは、1 つのドメイン内のただ 1 つの Replication Server のみです。どのデータベースも、ただ 1 つの ID サーバのドメイン内に存在します。つまり、異なるドメインから同じデータベースへのコネクションを複数作成することはできません。

Replication Server

各サイトの Replication Server は、ローカル・データ・サーバのデータ複写アクティビティを調整して、他のサイトの Replication Server とデータを交換します。

Replication Server は、次の処理を実行します。

- Replication Agent を介してデータベースからプライマリ・データ・トランザクションを受信して、それらをデータに対するサブスクリプションを持っているサイトに分配する。
- 他の Replication Server からトランザクションを受け取り、ローカル・データベースに適用する。

Replication Server のシステム・テーブルは、これらのタスクを実行するのに必要な情報を格納します。システム・テーブルには、複写定義やサブスクリプションなどの説明、Replication Server ユーザのセキュリティ・レコード、他のサイトのルート指定情報、ローカル・データベースのアクセス方法、およびその他の管理情報が格納されます。

Replication Server システム・テーブルは、RSSD (Replication Server システム・データベース) という Adaptive Server データベース、または ERSSD (Embedded Replication Server システム・データベース) という SQL Anywhere® データベースに格納されます。RSSD または ERSSD は、各 Replication Server に割り当てられます。RSSD を持つ Adaptive Server データ・サーバは、アプリケーション・データベースを格納することもできます。詳細については、『Replication Server 管理ガイド 第 1 巻』を参照してください。

Replication Server の情報を管理するには、複写コマンド言語 (RCL) または Sybase Central™ の Replication Manager プラグインを使用します。RCL コマンドは、SQL コマンドに似ており、Sybase の対話型 SQL ユーティリティである **isql** を使用して、Replication Server で実行できます。RCL の詳細については、『Replication Server リファレンス・マニュアル』を参照してください。Replication Manager と Replication Monitoring Services については、『Replication Server 管理ガイド 第 1 巻』および Replication Manager のオンライン・ヘルプを参照してください。

参照：

- 容量の計画 (125 ページ)

パーティションとステーブル・キュー

Replication Server は、失敗の後でもメッセージを配布できるようにメッセージをディスク上に格納します。

Replication Server をインストールするときに、Replication Server がディスク格納領域として使用する最初の「ディスク・パーティション」を割り付けます。

Replication Server をインストールした後で、パーティションを追加することもできます。

パーティションは、ロー・ディスク・デバイスまたはオペレーティング・システム・ファイルのどちらかです。UNIX オペレーティング・システムでは、ファイル I/O にバッファを使用するので、失敗の後にデータを完全にリカバリすることができない場合があります。そのようなシステムでは、パーティション用にオペレーティング・システム・ファイルを使用するのはテスト環境だけにしてください。運用環境では、ロー・ディスク・パーティションを使用してください。パーティションを追加する方法の詳細については、『Replication Server リファレンス・マニュアル』を参照してください。

Replication Server は、サービスを提供するルートとコネクションに対してそのディスク・パーティションから、「ステーブル・キュー」を割り付けます。メッセージは、少なくともそのメッセージの送信先で受信されたことが確認されるまでは、ステーブル・キューに保存されます。

Replication Server パーティションに割り付ける必要のあるディスク領域の量は、トランザクションのサイズとアプリケーションでのトランザクションの割合により異なります。ステーブル・キューは、データが複写システム内で転送されるときのバッファとして働きます。ネットワーク障害のためにリモート・サイトの Replication Server にアクセスできない場合は、プライマリ Replication Server はトランザクションをステーブル・キューに格納します。ディスク・パーティションに割り付ける領域が多いほど、Replication Server はデータを長い間キューに入れておくことができるため、プライマリ・データベースでオペレーションが中断することがなくなります。

ID サーバ

ID サーバは、複写システム内のすべての Replication Server とデータベースを登録している Replication Server です。

ID サーバは、次の場合に稼働している必要があります。

- Replication Server のインストール
- ルートを作成する場合
- データベース・コネクションを作成または削除する場合

上記の条件があるので、ID サーバは、複写システムをインストールするときに最初に起動する Replication Server になります。

ID サーバには、Replication Server が ID サーバと接続する時に使用する、Replication Server 用のログイン名が必要です。このログイン名は、**rs_init** 設定プログラムによって、複写システム内のすべての Replication Server の設定ファイルに記録されます。

複写環境

複写環境は、複写に関与する一連のサーバで構成されます。これには、データ・サーバ、Replication Agent、Replication Server、DirectConnect™ サーバが含まれます。複写環境に、複写システム・ドメイン内のすべてのサーバが含まれている必要はありません。

Replication Manager

Replication Manager (RM) は、Sybase Central のプラグインとしてインストールされます。RM は、複写環境の開発、管理、モニタリング用の管理ユーティリティです。

『Replication Server 管理ガイド 第 1 巻』を参照してください。

Replication Monitoring Services

Replication Monitoring Services (RMS) は、複写環境用の 3 層管理ソリューションの中間層として機能します。

RMS は、複写環境内のサーバとコンポーネントの状態をモニタし、問題をトラブルシューティングするための情報と、問題を解決するためのコマンドを提供します。『Replication Server 管理ガイド 第 1 巻』を参照してください。

データ・サーバ

データ・サーバは、プライマリ・データまたは複写データが保存されたデータベースを管理します。クライアント・アプリケーションは、データ・サーバを使用して、データの格納と取得を行い、クエリやトランザクションを処理します。Replication Server は、データベース・ユーザとしてログインして、データ・サーバ内の複写データを管理します。

Replication Server は、オープン・インタフェースによって異機種データ・サーバをサポートしています。データ格納用のどのシステムでも、必要な一連のデータ・オペレーションとトランザクション処理命令をサポートしていれば、データ・サーバとして使用できます。

参照：

- ASE 以外のデータ・サーバのサポート (21 ページ)

Replication Agent

Replication Agent は、プライマリ・データに加えられた変更を表すトランザクション・ログ情報を、他のデータベースに分配するためにデータ・サーバから Replication Server に転送します。

Replication Agent はデータベースのトランザクション・ログを読み込んで、複写テーブルと複写ストアド・プロシージャのログ・レコードを、そのデータベースを管理する Replication Server に転送します。Replication Server は、トランザクションを再構築して、データに対するサブスクリプションを持っているサイトに転送します。

Replication Agent は、プライマリ・データを含むデータベースごと、または複写ストアド・プロシージャが実行されるデータベースごとに必要です。複写データのコピーだけを含んでいて、複写ストアド・プロシージャを含んでいないデータベースには、Replication Agent は必要ありません。

RepAgent は Adaptive Server スレッドであり、別のプロセスではないため、またほとんどのシステム概要図には Adaptive Server が関与しており、Sybase サポート対象外のデータベースは関与していないため、Replication Agent は別のアイコンとして表示されていません。

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」は、データ・サーバにアクセスするプログラムです。Adaptive Server をデータ・サーバに使用している場合は、Open Client/Server™、Embedded SQL™、PowerBuilder®、または Sybase C/SI (Client/Server Interfaces™) と互換性のあるその他のフロントエンド開発ツールを使用して、クライアント・アプリケーションを作成することができます。

クライアント・アプリケーションは、プライマリ・データだけを更新するようにします。Replication Server は変更内容を他のサイトに分配します。データを修正しないクライアント・アプリケーションは、プライマリ・データと複写データを区別する必要はありません。

複写管理ソリューション

Replication Server は、異なる複写環境をサポートするための 2 層および 3 層からなる管理ソリューションを提供します。

2 層管理ソリューション

2 層管理ソリューションでは、RM は管理層経由で通信することなく、環境内のサーバに直接接続して複写環境を管理します。

この 2 層管理ソリューションにより、10 台未満のサーバから成る小規模で単純な構成の複写環境を管理できます。複写環境内のコンポーネントを作成、変更、削除できます。RM を使用すると、複写環境を管理できるほか、複写環境内のサーバと複写コンポーネントのステータスをモニタできます。

3 層管理ソリューション

3 層管理ソリューションでは、RM は RMS を使用して、大規模で複雑な複写環境を管理できます。RM は、RMS を介して環境内のサーバに接続します。

RMS は、複写環境のモニタリング機能を備えています。この 3 層管理ソリューションでは、RMS は複写環境内のサーバと他のコンポーネントのステータスをモニタします。RM には、RMS によって提供される情報を表示するクライアント・インタフェースが用意されています。

複写システムのコンポーネント

Replication Server、Replication Agent、Adaptive Server は、ネットワークを介して通信するために C/SI を使用します。さらに、Replication Server は、他の Replication Server やデータベースにメッセージを送信するためにルートとコネクションを使用します。

[interfaces ファイル]

データ・サーバ、Replication Server、Replication Agent などのサーバ・プログラムは、クライアント・アプリケーションや他のサーバ・プログラムで検出できるように interfaces ファイル (Windows の場合は `sql.ini`、UNIX の場合は `interfaces`) または LDAP (Lightweight Directory Access Protocol) サーバに登録されます。

通常は、各サイトごとに 1 つある interfaces ファイルがローカルおよびリモートの Replication Server とデータ・サーバのすべてのエントリを含んでいます。各サーバのエントリには、他のサーバとクライアント・プログラムの接続に必要なユニークな名前とネットワーク情報が含まれます。

interfaces ファイルの管理には、テキスト・エディタを使用します。LDAP サーバの詳細については、『Replication Server 管理ガイド 第 1 巻』を参照してください。

注意： Replication Server バージョン 12.0 以降で使用できるネットワークベースのセキュリティを使用している場合は、ネットワーク・セキュリティ・メカニズム (interfaces ファイル以外) のディレクトリ・サービスを使用して、Replication Server、Adaptive Server、ゲートウェイ・ソフトウェアを登録します。詳細については、ネットワークベースのセキュリティ・メカニズムに添付されているマニュアルを参照してください。ネットワークベースのセキュリティの管理方法の詳細については、『Replication Server 管理ガイド 第 1 巻』の「Replication Server のセキュリティ管理」を参照してください。

ルートとコネクション

ルートとコネクションを使用することによって、Replication Server は相互にメッセージを送信し、データベースにコマンドを送信できます。

「ルート」は、ある Replication Server から別の Replication Server に要求を送信する一方通行のメッセージ・ストリームです。「コネクション」は、Replication Server からデータベースへのメッセージ・ストリームです。Replication Server は「論理コネクション」を使用して、ウォーム・スタンバイ・アプリケーションでのアクティブ・データベースとスタンバイ・データベースを表します。

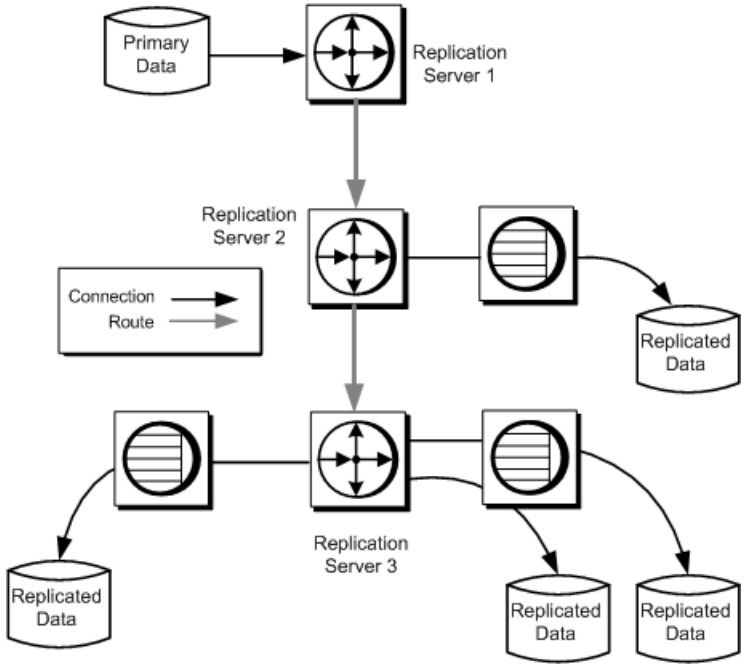
あるデータベースから別のデータベースにデータを複写するには、Replication Server がその送信元から送信先にデータを転送できるルートとコネクションを最初に設定する必要があります。

データベースを複写システムに追加すると、Sybase Central または **rs_init** がコネクションを作成します。Adaptive Server ではないデータベースにデータを複写する場合を除き、コネクションを直接作成する必要はありません。

複写システムに複数の Replication Server がある場合は、その間にルートを作成する必要があります。Replication Server が 1 つだけの場合は、ルートを作成する必要はありません。

次の図に、3 つの Replication Server、プライマリ・データを格納する 1 つのデータベース、複写データを格納する 4 つのデータベース間のコネクションとルートを示します。

図 2 : ルートとコネクション



プライマリ Replication Server からレプリケート Replication Server へのルートを作成すると、トランザクションはプライマリ・サーバからレプリケート・サーバに転送されます。

プライマリ・データベースを更新するためにレプリケート・データベースで複写ストアド・プロシージャを実行しようとする場合は、レプリケート Replication Server からプライマリ Replication Server へのルートを作成する必要もあります。

直接ルートと間接ルート

1 つのプライマリ Replication Server と多数のレプリケート Replication Server を持つ複写システムでは、プライマリ Replication Server での負荷を軽減するために間接ルートを使用できます。間接ルートを使用すると、Replication Server は 1 つの中間 Replication Server を経由して複数の送信先にメッセージを送信できます。

中間サイトを持つルートには、次のような重要な利点があります。

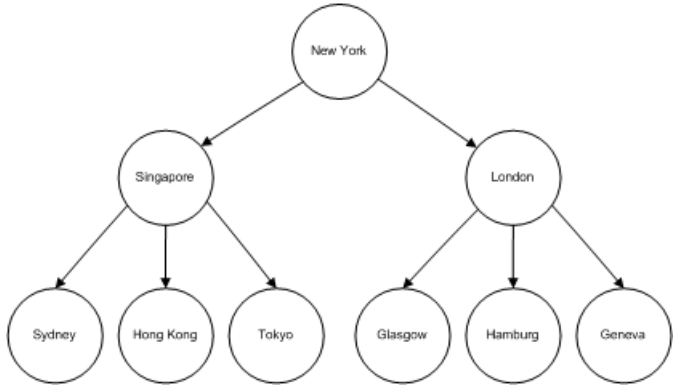
- WAN の通信量を軽減する。
  Replication Server は各中間サイトにメッセージのコピーを 1 つだけ分配します。中間サイトの Replication Server は、メッセージをそれぞれの出力キュー用に複製します。
- Replication Server の負荷を軽減する。

別々のコンピュータで稼働する追加 Replication Server は、処理の負荷を共有して、プライマリ・サイトでの Replication Server に必要とされる処理量を軽減します。

- フォールト・トレランス
  中間サイトに格納されるメッセージを使って、リモート・サイトでのパーティションの障害をリカバリできます。詳細については、『Replication Server 管理ガイド 第 1 巻』を参照してください。

次の図に、中間サイトを使用してメッセージの分配が処理される方法を示します。メッセージは、直接ルートを経て中間サイトへ転送されます。中間サイトからは、直接ルートを経てローカル・サイトへ転送されます。このルート指定の方法では、プライマリ・サイトは 8 つではなく 2 つのメッセージを送信します。

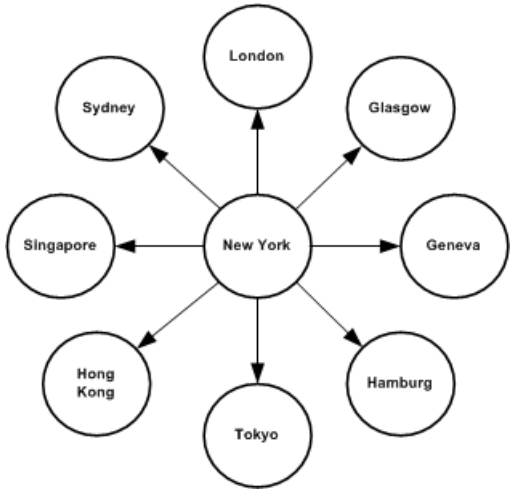
図 3 : 階層状構成のルート



中間サイトはプライマリ・サイトのメッセージ通信量を減少させますが、プライマリ・サーバとレプリケート・サーバでの更新の時間を増加させます。ルートは慎重に計画し、使用する中間サイトは必要な数だけに限定してください。

中間サイトを使用しない場合は、ルートは放射状に構成されます。

図 4 : 放射状構成の直接ルート



プライマリ・サイトでローが更新されると、プライマリ Replication Server は WAN を介して、ローに対するサブスクリプションを持っている各リモート・サイトにメッセージを送信します。たとえば、上の図では、ニューヨークは同一のデータを 8 つの異なるルートで送信できます。多数のサイトがあると、ネットワークは冗長なメッセージのためにすぐ過負荷になってしまいます。

階層状構成のルートを作成すると、コネクションとプライマリ・サイトから分配されるメッセージの数が減って、負荷のバランスを取ることができます。別々のコンピュータで稼働する追加 Replication Server は、処理の負荷を共有して、プライマリ・サイトでの Replication Server に必要とされる処理量を軽減します。

master データベースの複写

master データベースは、Adaptive Server のオペレーションを制御し、すべてのユーザ・データベースと関連データベース・デバイスについての情報を格納します。

master データベースは複写できます。ただし、複写されるのは、ログインと役割の管理に使用されるデータ定義言語 (DDL) とシステム・コマンドのみです。master データベースを複写しても、システム・テーブルのデータ、master データベース内の他のユーザ・テーブルのデータやプロシージャは複写されません。

送信元 Adaptive Server と送信先 Adaptive Server では、同じハードウェア・アーキテクチャのタイプ (32 ビット・バージョンと 64 ビット・バージョンは互換性があります)、また同じオペレーティング・システム (バージョンは異なっていてもかまいません) を使用している必要があります。

master データベースでサポートされている DDL とシステム・プロシージャのリストについては、『Replication Server 管理ガイド 第 2 巻』を参照してください。

Replication Server 12.0 以降で master データベースを複写する場合、ウォーム・スタンバイ、および Replication Server 12.6 以降の MultiSite Availability (MSA) を使用できます。プライマリ Adaptive Server またはアクティブ Adaptive Server は、Adaptive Server 15.0 ESD #2 以降である必要があります。

MSA で master データベースを複写する方法の詳細については、『Replication Server 管理ガイド 第 1 巻』を参照してください。ウォーム・スタンバイ環境で master データベースを複写する方法の詳細については、『Replication Server 管理ガイド 第 2 巻』を参照してください。

ASE 以外のデータ・サーバのサポート

Replication Server のオープン・アーキテクチャでは、複写システムで ASE 以外のデータ・サーバを使用することがサポートされています。

オープン・アーキテクチャの内容は次のとおりです。

- Sybase Enterprise Connect™ Data Access
- ExpressConnect for Oracle
- Replication Agent
- エラー・クラスとエラー処理アクション
- ファンクション、ファンクション文字列、ファンクション文字列クラス
- ユーザ定義データ型 (UDD) とデータ型変換
- 接続プロファイル

詳細については、『Replication Server 異機種間複写ガイド』および「Adaptive Server 以外のデータ・サーバへのデータの複写」を参照してください。

参照：

- Adaptive Server 以外のデータ・サーバへのデータの複写 (103 ページ)

Enterprise Connect Data Access (ECDA)

ECDA は、異機種データベース環境にあるデータへのアクセスを可能にするソフトウェア・アプリケーションと接続ツールを統合したセットです。

ECDA を使用すると、LAN ベースの Sybase 以外のデータ・ソースやメインフレームのデータ・ソースにアクセスできるようになります。ECDA は、企業内で透過的なデータ・アクセスを提供するコンポーネントで構成されます。アクティブにサポートされている ASE 以外のデータベースごとに、固有の ECDA コンポーネン

トが必要になります。『Replication Server Options 概要ガイド』を参照してください。

ExpressConnect for Oracle

Replication Server Options バージョン 15.5 以降で使用できる ExpressConnect for Oracle (ECO) は、Replication Server と レプリケート Oracle データ・サーバとの間で直接通信できるようにします。

ExpressConnect for Oracle では、ゲートウェイ・サーバのインストールと設定の必要がないため、複写システムのパフォーマンスを向上させ、複写システムの管理が煩雑にならないようにします。ECO の詳細については、『ExpressConnect for Oracle インストールおよび設定ガイド』を参照してください。

Replication Agent

Replication Agent は、プライマリ・データを格納する、または複写ファンクションを開始するすべてのデータベースに必要です。

Replication Agent は、データ・サーバのトランザクション・ログを読み込んでプライマリ・データに対する変更と複写ストアド・プロシージャの実行を検出します。トランザクション・ログは、コミットされたリカバリ可能なトランザクションのレコードを含んでいるので、プライマリ・データの修正についての情報の信頼できるソースを提供します。

データ・サーバ・エラーの処理

Replication Server は、データ・サーバによって返されるエラーとステータス・コードをユーザの指示に従って処理します。各ベンダのデータ・サーバは、それぞれ異なるエラー・コードのセットを持ちます。

RCL (複写コマンド言語) コマンドでは、次のようなことができます。

- エラー・クラスを作成して、データベースに対するエラー・コードのマッピングを一緒にしてグループ化する。
- **warn**、**retry_log**、**stop_replication** などのエラー・アクションをデータ・サーバのエラー・コードに割り当てる。
- エラー・クラスをデータベースに対応付ける。

注意： Replication Server 15.2 以降には、アクティブにサポートされるデータベースのために、事前にロードされたエラー・クラスとそれに対応するエラー・アクションが含まれます。『Replication Server 管理ガイド 第 1 巻』の「接続プロファイル」を参照してください。

参照：

- エラー・クラス (107 ページ)

ファンクション、ファンクション文字列、ファンクション文字列クラス

「ファンクション」は、**insert**、**delete**、および **begin transaction** などのデータ・サーバ・オペレーションを表す Replication Server オブジェクトです。

異機種データベース環境で動作するために、Replication Server はデータ・サーバ要求を他のサイトに分配するために使用するファンクションとデータベース・コマンドとを区別します。Replication Server はファンクション文字列を使用して、ファンクションをデータ・サーバ固有のコマンドに変換します。「ファンクション文字列」は、データ・サーバがトランザクションを制御命令またはデータ修正命令として解釈できるコマンドを、Replication Server が生成するために使用するテンプレートです。

「ファンクション文字列クラス」は、データベースで使用されるすべてのファンクション文字列の集合です。ファンクション文字列クラスは、Adaptive Server と DB2 データ・サーバ用に提供されます。トランザクション制御命令用のファンクション文字列は、各ファンクション文字列クラス用に 1 回だけ定義されます。ローの挿入、削除、更新のためのファンクション文字列は、データベース内の各複写テーブルに対して 1 回だけ定義されます。

ファンクション文字列には、カラムの値、プロシージャ・パラメータ、システムで定義されている情報、ユーザ定義の変数を表す変数 (疑問符 (?) で囲まれた識別子) を含むことができます。Replication Server は、この変数を実際の値に置き換えてから、ファンクション文字列をデータ・サーバに送信します。

ファンクション文字列を使うと、そのフォーマットに応じてリモート・プロシージャ・コール (RPC) または SQL 文などのデータベース・コマンドのいずれかを生成できます。RPC 形式のファンクション文字列には、データ・パラメータのリストが後に続くリモート・プロシージャ・コールが含まれます。埋め込み変数は、ランタイム値をパラメータに割り当てるために使用できます。Replication Server は RPC ファンクション文字列を解釈して、リモート・プロシージャ・コールを構築し、変数をランタイム・データ値で置き換えます。RPC は、Adaptive Server 内のデータ・サーバやストアド・プロシージャに対する Open Server™ ゲートウェイ内のレジスタード・プロシージャを実行できます。

言語形式のファンクション文字列は、データベース・コマンドをデータ・サーバに渡します。Replication Server は、埋め込み変数をランタイム・データ値で置き換える目的以外では、文字列を解釈しようとはしません。たとえば、いくつかのリレーショナル・データベース・サーバでは、SQL データベース言語を使用します。SQL コマンドは、言語ファンクション文字列として表される必要があります。

RPC ファンクション文字列は、Replication Server から送信されるネットワーク・パケットの方がコンパクトなので、言語ファンクション文字列よりも効率的な場合があります。

たとえば、 **create connection with using profile** 句を使用し、プロファイルの特定のバージョン v9_1 を使用して、DB2 レプリケート・データベースに対するコネクションを作成します。この例では、接続プロファイルによって提供されているコマンド・バッチのサイズが、このコマンドにより新しい値 16384 で上書きされます。

```
create connection to db2.subsys
using profile rs_ase_to_db2;v9_1
set username to db2_maint
set password to db2_maint_pwd
set dsi_cmd_batch_size to '16384'
```

注意： Replication Server 15.2 以降には、アクティブにサポートされるデータベースのために、ファンクション文字列が事前にロードされたファンクション文字列クラスが含まれます。『Replication Server 管理ガイド 第 1 巻』の「接続プロファイル」を参照してください。

Replication Server のセキュリティ

Replication Server のセキュリティには、**grant** と **revoke** コマンドを使用する、パスワードで保護されたログイン名とパーミッション・システムが含まれます。

Replication Server 12.0 以降は、ネットワークを介した安全なメッセージ転送を保証し、ユーザ認証を可能にするサードパーティのセキュリティ・サービスをサポートします。Replication Server 12.5 以降は、Advanced Security オプションによる SSL (Secure Socket Layer) ベースのセキュリティをサポートします。

ログイン名

各 Replication Server は、データ・サーバのログイン名とは異なるログイン名を使用します。クライアントの多くは、データ・サーバ・アプリケーションを使用して作業を行うので Replication Server ログイン名を必要としません。

Replication Server ログイン名

Replication Server をインストールすると、**rs_init** は他の Replication Server と Replication Agent が Replication Server へのログインに使用する Replication Server ログイン名を作成します。

複写システム管理者は、新しいユーザの追加やルートの変更など、複写データや複写システム・ファンクションの管理に使用する Replication Server ログイン名とパスワードを作成して管理します。パスワードは暗号化することができます。

データ・サーバのログイン名

データ・サーバのログイン名は、クライアント・アプリケーションからデータ・サーバに接続するために使用されます。データベース管理者は、データ・サーバのログイン・アカウントを作成して管理します。

複写されたテーブルのコピーに対するクライアントのアクセスも、データベース管理者によって管理されます。Sybase では、複写テーブルは読み込み専用にすることをおすすめしています。こうすることで、クライアントは複写データを表示できても、ローの挿入、削除、更新はできなくなります。

テーブルを修正するには、クライアントはプライマリ・データが格納されているデータ・サーバのログイン名と、プライマリ・データを更新するのに必要なパーミッションを持っている必要があります。

複写テーブルを修正するには、クライアントは、データに対するサブスクリプションを持っているレプリケート・データベースに対して、Replication Server が変更内容を分配できるように、プライマリ・データを修正する必要があります。テーブルを修正するには、クライアントはプライマリ・データが格納されているデータ・サーバのログイン名と、プライマリ・データを更新するのに必要なパーミッションを持っている必要があります。

データ・サーバのメンテナンス・ユーザのログイン名

Replication Server は複写テーブルを含む各ローカル・データ・サーバのデータベースに対してメンテナンス・ユーザ・ログイン名を使用します。Replication Server は、複写テーブルを管理するのにこのログインを使用します。

データベース管理者は、メンテナンス・ユーザ・ログイン名がデータベース内の複写テーブルを更新するのに必要なパーミッションを持っていることを確認する必要があります。

通常は、メンテナンス・ユーザによって適用されるトランザクションは、データベースから複写されないように Replication Agent によってフィルタされます。ただし、アプリケーションによっては、これらのトランザクションを複写する必要があります。

参照：

- 実装方式 (43 ページ)

パーミッション

クライアント向けに Replication Server のパーミッションの付与と取り消しを行う手順について説明します。

表 1 : Replication Server のパーミッション

パーミッション	機能
sa	受信者にシステム管理者機能を付与する。**sa** パーミッションを持つクライアントは、どの Replication Server コマンドでも実行できる。
create object	複写定義やサブスクリプションなどの Replication Server オブジェクトの作成、変更または削除を受信者に許可する。
primary subscribe	プライマリ・データベースにサブスクリプションを作成することはできるが、他のオブジェクトを作成することはできないパーミッションを受信者に付与する。リモート・サイトにサブスクリプションを作成するには、クライアントはレプリケート・データベースに **create object** パーミッションを、プライマリ・データベースに **create object** または **primary subscribe** パーミッションを、それぞれ必要とする。
connect source	このパーミッションは、Replication Agent で使用されるログイン名に対して必要である。Replication Agent に予約されている RCL コマンドの実行を受信者に許可する。

ネットワークベース・セキュリティ

サードパーティのネットワークベースのセキュリティ・メカニズムを使用して、ログイン時にユーザ認証を行うことができます。

認証は、ユーザが本人であることを確認するための処理です。ユーザはリモート・サーバに提示することのできるクレデンシャルを受け取るため、複写システムのコンポーネントに、1 つのログイン名で連続的にアクセスできます。

ネットワークベースのセキュリティ・メカニズムは、メッセージの機密保持や順序不整合検査などの、さまざまなデータ保護サービスも提供します。Replication Server はサービスを要求し、データを準備し、それを暗号化や検証のためにネットワーク・サーバに送信します。サービスが実行された後は、データは要求を行った Replication Server の分配用に返されます。

安全な経路が確立された後は、データはどちらの方向にも転送できます。経路のどちらの端も、同じセキュリティ・メカニズムをサポートして同じ設定にしなければなりません。セキュリティ・メカニズムは、ネットワーク・セキュリティを使用するすべてのマシンにインストールする必要があり、関与するすべてのマシンに Replication Server 12.0 以降をインストールしなければなりません。

複写システムでのネットワークベース・セキュリティについては、『Replication Server リファレンス・マニュアル』を参照してください。

Advanced Security オプション

Replication Server の Advanced Security オプションは、SSL (Secure Sockets Layer) のセッションベースのセキュリティを提供します。SSL はインターネット上で機密情報を安全に送信するための標準です。

SSL はいくつかの暗号化アルゴリズムが採用された、低負荷で管理が容易なセキュリティ・メカニズムです。SSL は高度なセキュリティが要求されるデータベース・コネクションやルートを対象としたプロトコルです。

Advanced Security オプションの使用方法については、『Replication Server 管理ガイド 第 1 巻』を参照してください。

概要

Replication Server の主な概念について簡単にまとめます。

- Replication Server は、テーブルのコピーをネットワーク上の異なるデータベース内に維持します。複写されたコピーは、速い応答と高い可用性という 2 つの利点をサイトのユーザに提供します。
- Replication Server 上に構築された複写システムは、Sybase ECDA を使用して、Oracle や Microsoft SQL Server などの異機種レプリケート・データ・サーバに接続します。
- Replication Server は、Sybase 以外のデータ・サーバを複写システムに含めることができるオープン・インタフェースを使用して設計されます。
- テーブルの 1 つのコピーはプライマリ・バージョンです。他のすべては、複写 (された) コピーです。
- サブスクリプションを使用すると、テーブルの複写コピーはテーブル内のローのサブセットを含むことができます。
- Replication Server のセキュリティは、ログイン名、パスワード、パーミッションで構成されます。Replication Server は、サードパーティのネットワークベースのセキュリティ・メカニズムやセキュリティ保護された通信チャネルもサポートします。
- Replication Server は障害が発生したときに処理を行うオプティミスティック同時制御を使用します。他の方法と比較して、オプティミスティック同時制御は、さらに高いデータ可用性を提供し、リソースの使用量も少なくなります。また異機種データ・サーバでも適切に動作します。

複写システムのアプリケーション・アーキテクチャ

複写システムの構造設計によって、複写環境の全体的な完成度が決まります。アプリケーション設計者およびユーザとして、Replication Server を実装する前に検討する必要のあるさまざまなオプションについて説明します。

アプリケーションのタイプ

構築する Replication Server アプリケーションのタイプを決定することは、使用する複写方式のタイプを決定することでもあります。

実装方式ごとに、複写のさまざまなケースについて考察します。

Replication Server は、次の基本的なアプリケーション・タイプをサポートします。

- 意思決定支援
- 分散オンライン・トランザクション処理 (OLTP)
- 要求ファンクションを使用するリモート OLTP
- ウォーム・スタンバイ

これらのアプリケーション・タイプはそれぞれ、プライマリ・データを更新する方法と複写システム内でプライマリ・データと複写データを分配する方法が異なります。

参照：

- アプリケーションにおける緩やかな一貫性の効果 (36 ページ)
- プライマリ・データの更新方法 (37 ページ)
- 実装方式 (43 ページ)

意思決定支援アプリケーション

意思決定支援クライアントと運用 OLTP (オンライン・トランザクション処理) クライアントは、データを異なる方法で使用します。意思決定支援クライアントは、順序一貫性を保証するためにテーブルに対するロックを保有する長いクエリを実行します。

それに対して OLTP クライアントは、速やかに完了する必要があり、意思決定支援クライアントのデータ・ロックによって発生するような遅延を許容できないトランザクションを実行します。この 2 つのタイプのクライアントは、複写テーブルの別々のコピーを維持すれば、互いに干渉することはありません。

Replication Server は、意思決定支援アプリケーションに関する処理負荷を、集中化されたオンライン・トランザクション処理アプリケーションからローカル・サーバに分散します。プライマリ・データベースはトランザクション処理を管理し、ローカル・サイトのレプリケート・データベースは意思決定支援クライアントからの情報の要求を処理します。データの別々の参照専用コピーを提供すれば、OLTP システムは妨害されずに継続することができます。

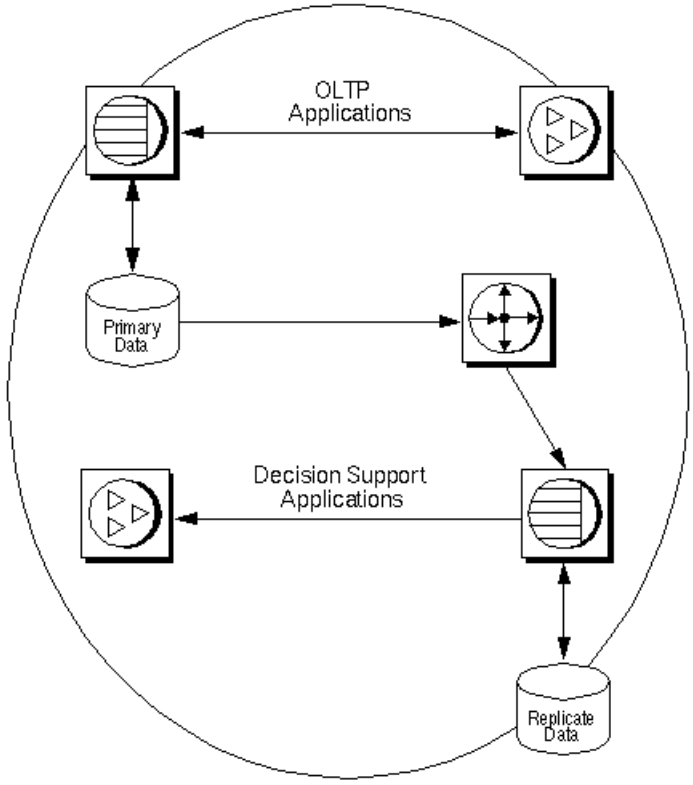
意思決定支援アプリケーションで使用されるプライマリ・データを含んでいるテーブルの複数のコピーは、ネットワーク上の 1 つのサイトまたは複数のサイトで維持できます。

1 つのサイトの複数コピー

同じサイトの異なるデータベースにテーブルを作成してそれぞれのサブスクリプションを作成することによって、テーブルの複数コピーのサブスクリプションを作成することができます。

OLTP クライアントと意思決定支援クライアントが同じ LAN 上にある場合は、1 つの Replication Server でプライマリ・データとレプリケート・データの両方を管理できます。

図 5 : 1 つの LAN での意思決定支援複写



最適なパフォーマンスを得るために、データベースは通常は異なるデータ・サーバによって管理されます。サブスクリプションは、各データベースに維持されるデータの異なるサブセットを要求できるので、複写コピーは同じものである必要はありません。

同じデータベースにテーブルのコピーを 2 つ持つ必要がある場合は、プライマリ・テーブルに対して複数の複写定義を使用できます。一方の複写定義がレプリケート・テーブル名として `publishers` を持ち、もう一方が `publishers2` を持つこともできます。複数の複写定義は、異なるカラム・サブセットを受信するために異なるレプリケートを必要とする場合にも役立ちます。

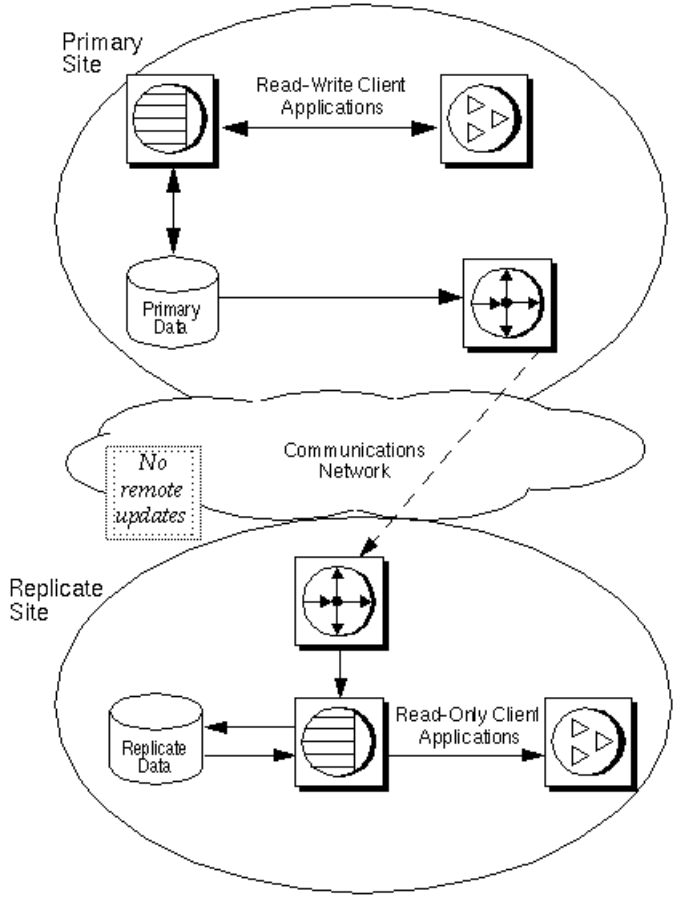
Adaptive Server データベース内の複数のテーブルを更新するための別の方法は、ストアド・プロシージャを使用するものです。ストアド・プロシージャ内に複数の更新コードを作成し、ストアド・プロシージャを実行するために Replication Server ファンクション文字列を書き込みます。複数のテーブルを更新するために複写ファンクションとストアド・プロシージャを使用することもできます。

参照：
- 複数の複写定義 (65 ページ)

ネットワーク上に分配される複数コピー

意思決定支援アプリケーションのテーブルのコピーが WAN 上で分配される場合、すべての更新はプライマリ・サイトで実行するアプリケーションによって実行され、データに対するサブスクリプションを持っているリモート・サイトに分配されます。

図 6 : マルチ LAN 意思決定支援レプリケート



このタイプのシステムでは、プライマリ・データを更新する集中化されたプライマリ・メンテナンス方法を使用します。リモート・サイトのクライアントは、プライマリ・データの複写定義またはパブリケーションに対してサブスクリプションを作成します。プライマリ・データの更新は行いません。

参照：

- 集中化されたプライマリ・メンテナンス (38 ページ)

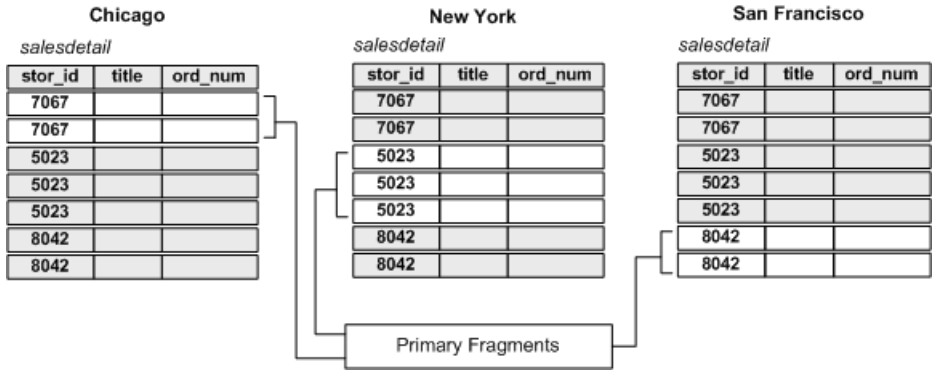
分散 OLTP アプリケーション

分散トランザクション処理アプリケーションによっては、集中化されたプライマリ・データを維持するものもあれば、プライマリ・データをレプリケート・サイト間に分散させているものもあります。

「プライマリ・フラグメント」は、一連のローのプライマリ・バージョンを保持するテーブルの水平方向セグメントです。更新は、まずプライマリ・バージョンに適用されてから、データの複写コピーを持つサイトに分配されます。

テーブルの一部を保持するように定義されているサイトは、複数のプライマリ・フラグメントを持ちます。たとえば、`salesdetail` テーブルは、シカゴ、ニューヨーク、サンフランシスコにプライマリ・フラグメントを持ちます。

図 7 : 複数のプライマリ・フラグメントを持つテーブル



1 つまたは複数のカラムで構成されるキーは、ローが属しているプライマリ・フラグメントを特定します。`salesdetail` テーブルのキーは `stor_id` カラムです。

- `stor_id` カラムに “7067” を持つローは、シカゴ・サイトのプライマリ・フラグメントに属します。
- `stor_id` カラムに “5023” を持つローは、ニューヨーク・サイトのプライマリ・フラグメントに属します。
- `stor_id` カラムに “8042” を持つローは、サンフランシスコ・サイトのプライマリ・フラグメントに属します。

複数のプライマリ・フラグメントに基づいた、次のような 3 つのアプリケーション・モデルがあります。

- 分散プライマリ・フラグメント ― 各サイトのテーブルには、プライマリ・データと複写データの両方が含まれます。プライマリ・バージョンに対する更新は、他のサイトに分配されます。プライマリではないデータに対する更新は、プライマリ・サイトから受信されます。
- コーポレート・ロールアップ ― リモート・サイトで管理されている複数のプライマリ・フラグメントが中央サイトの 1 つの集約レプリケート・テーブルに統合されます。
- 再分配コーポレート・ロールアップ ― このモデルは、統合テーブルが再分配される点を除けばコーポレート・ロールアップ・モデルと同じです。

これらのモデルの詳細については、「実装方式」のトピックを参照してください。

要求ファンクションを使用するリモート OLTP

複写ファンクションを使用すると、トランザクションをリモートで実行することができます。

リモート・サイトのクライアント・アプリケーションは、要求ファンクションを使用して、プライマリ・データを非同期に更新できます。クライアント・アプリケーションはプライマリ・サイトへのネットワーク・コネクションを必要としません。また要求は、プライマリ・サイトにアクセスできない場合でも、Replication Server によって受け入れられます。

要求ファンクションがプライマリ・データベース内でストアド・プロシージャを実行した後、Replication Server はプライマリ・データベースで行われた一部または全部のデータ変更を複写することができます。これらの変更は、レプリケート・データベースにデータ・ローまたは適用ファンクションとして送信できます。

ローカル更新アプリケーション

ローカル更新アプリケーションでは、リモート・サイトのクライアントは複写システムがプライマリ・サイトから更新内容を返す前に、入力したその内容を表示できます。

たとえば、カスタマ・アカウントがリモート・サイトで更新された場合、そのサイトのクライアントはプライマリ・サイトにアクセスできない場合でもトランザクションの結果を表示することができます。

ローカルの更新は、保留中の更新テーブルを使用することによって実行できます。各複写テーブルに対して、対応するローカル・テーブルは暫定的な更新 (プライマリ・サイトに送信はされているが、複写システムから返されていない更新) を含んでいます。クライアント・アプリケーションは、保留中のトランザクション・テーブルを更新すると同時に要求ファンクションをプライマリ・サイトに送信します。

プライマリ・コピーに対する更新が成功すると、トランザクションが開始されたサイトを含むリモート・サイトに分配されます。ファンクション文字列または複

写ストアド・プロシージャを作成して、複写テーブルを更新して保留中のテーブルからローカル更新を削除できます。これによって、クライアント・アプリケーションはどのトランザクションが確認されていてどれが保留中であるかを知ることができます。

参照：

- ローカル保留テーブルの使用例 (72 ページ)

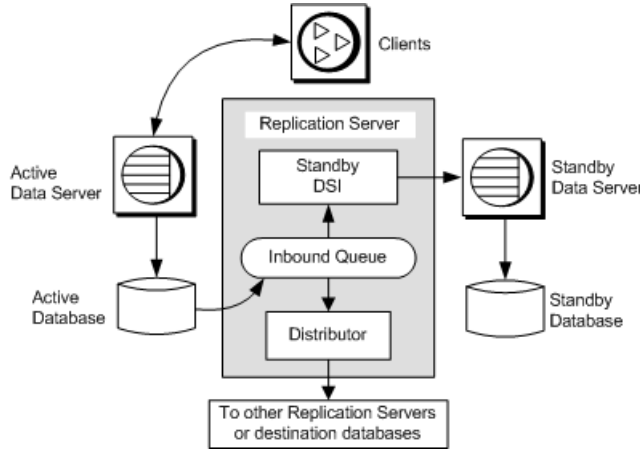
スタンバイ・アプリケーション

「ウォーム・スタンバイ・アプリケーション」は同じベンダによる 1 組のデータベースを管理し、その 1 つがもう一方のスタンバイとして機能します。

通常はクライアント・アプリケーションがアクティブ・データベースを更新するのに対して、Replication Server はスタンバイ・データベースをアクティブ・データベースのコピーとして管理します。Replication Server は、アクティブ・データベースのトランザクション・ログから取り出されたトランザクションを複写することによって、スタンバイ・データベースとアクティブ・データベースの一貫性を維持します。

アクティブ・データベースが失敗した場合、またはアクティブ・データベースかデータ・サーバでメンテナンスを実行する必要がある場合は、スタンバイ・データベースに切り替えて、クライアント・アプリケーションがほとんど中断されることなく作業をレジュームできます。

図 8 : ウォーム・スタンバイ・システム



ウォーム・スタンバイ・アプリケーションの 2 つのデータベースは、複写システム内では 1 つの論理データベースとして扱われます。アプリケーションによっては、この論理データベースは複写には関与しないか、複写システム内の他のデー

タベースに関してプライマリ・データベースまたはレプリケート・データベースであることができます。

Replication Server と Replication Agent 機能の一部には、ウォーム・スタンバイ・アプリケーションを明示的にサポートするものもあります。ウォーム・スタンバイ・アプリケーションの詳細については、『Replication Server 管理ガイド 第 2 巻』を参照してください。

アプリケーションにおける緩やかな一貫性の効果

レプリケート・データベース内のデータは、プライマリ・データベースのデータと「緩やかな一貫性」を持っています。

レプリケート・データは、プライマリ・データベースから複写環境の他の部分に更新を分配するのにかかる時間だけ、プライマリ・データより遅れが発生します。この遅延時間は、システムが正しく動作している場合は、数秒 (またはそれ以下) の値です。コンポーネントが失敗した場合、たとえば、ネットワーク・コネクションが一時的に失われた場合は、更新が数分、数時間、または数日にわたって遅れることがあります。そのため、遅延時間情報を使用して、複写環境のパフォーマンスおよび状態をモニタできます。

レプリケート・データはプライマリ・データより遅れることがありますが、トランザクション的にはプライマリ・データと一貫性があります。Replication Server は、プライマリ・データベースでコミットされた順にトランザクションをレプリケート・データベースに配布します。これによって、レプリケート・データがプライマリ・データと同じ順序に従って更新されることが保証されます。

緩やかな一貫性の重要性は、アプリケーションごとに、さらにはアプリケーション内部でも異なります。一部のアプリケーションには、特に準備がなくても平均的なシステム遅延時間や、時にはさらに長い遅延に耐えられるものもあります。遅延時間が大きくなると特別な処理を必要とするものや、特定のタイプのトランザクションについて特別な処理を必要とするものもあります。

参照：
- アプリケーションのタイプ (29 ページ)
- プライマリ・データの更新方法 (37 ページ)

上限値トランザクションでのリスク

上限値トランザクションと下限値トランザクションを区別することによって、データ複写のリスクを軽減することができます。

データの複写によって発生する遅れは、ビジネスによっては意思決定のリスクを大きくします。たとえば、現金の引き出しを承認するバンキング・アプリケーションは、顧客の残高が引き出しを行うのに十分な額かどうかを確認するために、利用可能な最新の口座情報を使用します。プライマリ・データベースで処理され

る引き出し情報がレプリケート・データベースに届いていない場合、レプリケート・データベースを使用するアプリケーションはその顧客の口座の残高を超える引き出しを承認する危険にさらされます。

リスクを軽減するために、バンキング・アプリケーションは上限値トランザクションと下限値トランザクションを区別することができます。たとえば、100 ドルの引き出しはローカル・レプリケート・データベース内の残高に基づいて承認するが、1,000 ドルの引き出しはプライマリ・データベースにログインして残高を確認してから承認するようにできます。

遅延

レプリケート・サイトに対する遅延の尺度によって、一部のトランザクションについてのリスクを制限します。遅延時間が小さいということは、プライマリ・データとレプリケート・データがほとんど一貫していることを示します。遅延時間が大きい場合は、プライマリ・データとレプリケート・データ間に大きな差がある可能性があることを示します。

アプリケーションでは、次のような目的に遅延時間の尺度を使用できます。

- 遅延時間が増大したときにクライアントが実行できるトランザクションを制限することによってリスクを制限する。たとえば、バンキング・アプリケーションでは、その承認方式に遅延時間を含めることがあります。遅延時間が 1 分未満であればローカル複写テーブル内の残高に基づいて 1,000 ドルまでの引き出しを承認します。ただし、遅延時間が 1 分を超える場合は、500 ドルを超える引き出しを承認するには、アプリケーションはプライマリ・データベースにログインします。
- クライアントに、データ複写のための「パフォーマンスの尺度」を提供する。クライアントは、遅延時間の見積りを助言として使用できます。たとえば、意思決定支援ユーザは、遅延時間が大きいことに気づいた場合、ローカル・データがプライマリ・データに対する遅れを取り戻すまで待機して、遅延時間が減少してからレプリケート・データに基づいて分析を実行するようにします。

遅延時間の尺度の詳細については、『Replication Server 管理ガイド 第 2 巻』の「パフォーマンス・チューニング」を参照してください。

プライマリ・データの更新方法

複写システムでは、データ・ローのプライマリ・コピーは最も確実なコピーです。プライマリ・データベースでコミットされた更新は信頼できるものであり、そのデータに対するサブスクリプションを持っているすべてのデータベースに分配されます。

Replication Server は、プライマリ・データベースでコミットされた後にトランザクションを分配します。レプリケート・データに加えられた変更は分配されないの

で、レプリケート・データベース内のデータはクライアントに対して読み込み専用にして、すべてのクライアント・トランザクションをプライマリ・データベースに送信するようにしてください。

Replication Server をベースとした複写システム内のプライマリ・データを更新するには、次の 4 つの方法があります。

- プライマリ・データのメンテナンスをプライマリ・サイトに集中化する。クライアントは、リモート・サイトからプライマリ・データを更新することはできません。
- リモート・サイトのクライアントが、ネットワーク・コネクションを介してプライマリ・データを更新する。
- リモート・サイトのクライアントが、要求ファンクションを使用してプライマリ・データを更新する。
- プライマリ・データのメンテナンスを、複数のプライマリ・サイトに分散する。結果として生じる競合を避けるか解決する必要があります。

参照：

- アプリケーションのタイプ (29 ページ)
- アプリケーションにおける緩やかな一貫性の効果 (36 ページ)

集中化されたプライマリ・メンテナンス

集中化されたプライマリ・メンテナンスの方法は、最も簡単で、リモート・クライアントにとって最も制限のあるものです。リモート・サイトのクライアント・アプリケーションは、レプリケート・データを参照専用に使用します。

このアーキテクチャは、意思決定支援アプリケーションが OLTP システムとは別に実行できる運用 OLTP システムのコピーを作成するために使用できます。

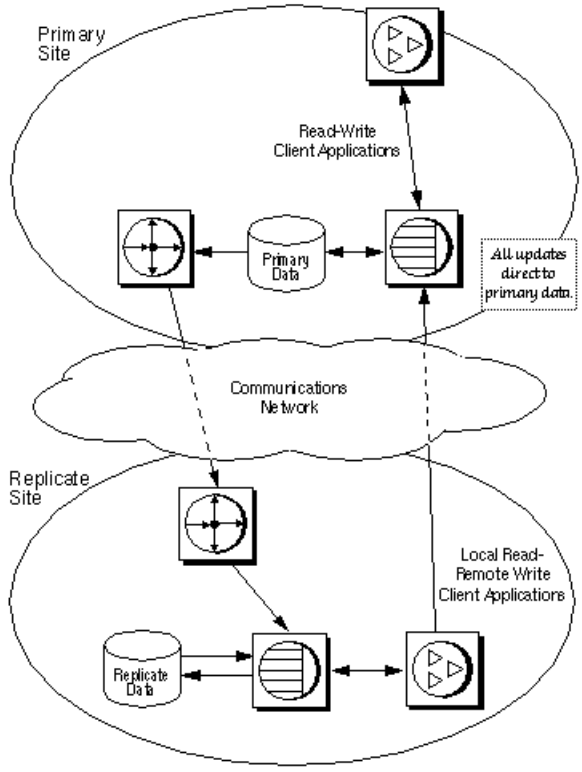
参照：

- 意思決定支援アプリケーション (29 ページ)

ネットワーク・コネクションを介したプライマリ・メンテナンス

アプリケーションによっては、リモート・サイトのクライアントがプライマリ・データを更新する必要があります。このための最も簡単な方法は、クライアントがネットワークを介してプライマリ・データ・サーバに直接接続することです。Replication Server は、プライマリ・サイトからリモート・サイトに通常の方法で更新を分配します。

図 9 : ネットワークを介したプライマリ・データ・メンテナンス



このアーキテクチャでは、WAN を介したクライアント・コネクションを使用します。これは、大量のデータがあり更新の程度が少ないアプリケーションの場合に役立ちます。更新は、プライマリ・データに対して直接実行され、データに対するサブスクリプションを持っているすべてのサイトに分配できます。リモート・サイトでは、意思決定支援アプリケーションにデータのローカル複写コピーを使用して、ネットワーク負荷を軽減できます。

複数プライマリに対する更新の競合の管理

複数のプライマリが存在する場合、リモート更新は、複数のリモート・サイトから同時に同じ情報を更新する要求が行われても、エラーが発生しないように設計する必要があります。

2 つの異なるサイトからの更新がプライマリ・サイトに適用されるときに競合すると、一方の更新は拒否され、更新が拒否されたサイトの複写データはプライマリ・データとの一貫性がなくなります。

複数のプライマリがある場合に、サイト間の同時実行性の競合を処理するには、次の手順に従います。

- 各ローに所有者がある場合 — サイト間の競合が起こらないようにアプリケーションを設計する。たとえば、あるサイトで実行される更新の対象となるローを、他のサイトのクライアントが更新できないローに制限します。これによって更新がプライマリ・サイトで競合しないことを保証します。
- 所有権の分割がない場合 – バージョン制御情報をファンクション文字列に追加して、競合を検出して処理できるようにする。

競合の起こらないアプリケーションの設計

異なるサイトからの更新の競合を処理します。

競合が起こらないようにアプリケーションとその環境を構成します。たとえば、複数の支店に分配される顧客口座情報を持つアプリケーションは、顧客の口座がある支店でないと口座を更新できないように設計できます。これによって、2 つのクライアントが異なるデータベース内で同時に同じ口座を更新できないようにします。

支店 ID などのロケーション・キーを複写テーブルのプライマリ・キーに含める方法もあります。各サイトがそのトランザクションのすべてについてユニークなロケーション・キーを使用すれば、サイト間の競合は起こりません。

バージョン制御による更新

バージョン制御による更新を使用すると、さまざまなデータ更新の競合を検出して解決できます。

バージョン制御による更新は、次のような場合に使用できます。

- 競合する更新を検出して解決する。
- 1 つのプライマリ・ソースがない場合に、複数のプライマリの競合を解決する。
- 1 つのプライマリに対して複数の要求を行う。

更新は、ローが更新されるたびに変更されるバージョン・カラムに基づいて受け入れまたは拒否されます。バージョン・カラムは、更新が行われるごとに増加する数値、タイムスタンプ、または一連のユニークな値で構成されるその他の値とすることができます。

ローを更新するには、アプリケーションはプライマリ・サイトでバージョン・カラムの最新の値を提示する必要があります。通常、その値は複写ストアド・プロシージャに対するパラメータとして提示されます。プライマリ・サイトのストアド・プロシージャは、バージョン・パラメータを調べて、競合を検出した場合は適切なアクションをとります。アプリケーションがトランザクションをロールバックすることを選択した場合は、そのトランザクションはトランザクション・ログに書き込まれます。

注意： バージョン制御を使用して更新の競合を管理する方法は、慎重なプランニングと設計を必要とします。複数のプライマリが存在する場合は、通常、テーブルの各ローに対して所有者を設定する方がはるかに簡単で効果的です。

実装方式

複写システムを実装する場合、さまざまなモデルや方式があります。Replication Server には、独自のアプリケーションに適応できるプロシージャやサンプル・スクリプトも含まれています。

モデルと方式の概要

データを複写する場合、さまざまなデータ複写モデルがあります。

次のモデルがあります。

- 基本プライマリ・コピー・モデル ― 集中プライマリ・データと分散レプリケート・データで構成される。
- 分散プライマリ・フラグメント・モデル ― 複写システムを使用して分配されるプライマリ・データとレプリケート・データの両方で構成される。
- コーポレート・ロールアップ ― 分散プライマリ・データと集中レプリケート・データで構成される。
- 再分配コーポレート・ロールアップ ― 更新がレプリケート・データベースに再分配される点を除けばコーポレート・ロールアップと同じ。
- ウォーム・スタンバイ・アプリケーション ― 2 つのデータベースの一方が他方のバックアップとして働き、論理単位として一緒に複写に関与する。

使用可能なその他のモデルの変化形と方式

- 複数の複写定義
- パブリケーション
- 要求ファンクション
- 保留テーブル
- マスタ／ディテール関係

構築するアプリケーションのタイプ、プライマリ・データを更新する方法、可能性のある更新の競合を管理する方法によって、複写アプリケーションを実装するために使用するモデルが決まります。

たとえば、意思決定支援アプリケーションや処理量の少ない分散 OLTP システムを実装するには、基本プライマリ・コピー・モデルを使用できます。意思決定支援アプリケーションを実装するには、プライマリ・データが集中化されているか分散化されているかによって、基本プライマリ・コピー・モデルまたは再分配コーポレート・ロールアップ・モデルのどちらかを使用します。分散 OLTP アプリケーションは、意思決定支援の必要性があるかどうかによって、コーポレー

ト・ロールアップあり、またはなしの分散プライマリ・フラグメント・モデルを使用して実装します。

参照：
- モデルの変化形と方式 (65 ページ)

基本プライマリ・コピー・モデル

基本プライマリ・コピー・モデルでは、プライマリ・データベースから送信先データベースにデータを複写できます。

処理量の少ないトランザクション処理アプリケーションは WAN を介して直接、または要求ファンクション (複写ストアド・プロシージャ) を通してプライマリ・データをリモートから更新できますが、このモデルが適しているのは意思決定支援アプリケーションです。その後、リモート・サイトから更新されたプライマリ・データをサブスクリプションを作成するサイトに複写できます。

基本プライマリ・コピー・モデルは、次のいずれかまたは全部を使用して実装できます。

- テーブル複写定義
- 適用ファンクション
- 要求ファンクション

テーブル複写定義

テーブル複写定義を使うと、プライマリ・ソースからデータを読み込み専用コピーとして複写できます。

1 つのプライマリ・テーブルに対しては 1 つまたは複数の複写定義を作成できますが、特定のレプリケート・テーブルがサブスクリプションを作成できるのは 1 つの複写定義に対してだけです。複数の複写定義の使い方の例については、「複数の複写定義」を参照してください。

複写定義を 1 つのパブリケーションに集め、パブリケーション・サブスクリプションを使用して一度にそれらすべてに対してサブスクリプションを作成することもできます。パブリケーションの使い方の例については、「パブリケーション」を参照してください。

基本プライマリ・コピー・モデルに従って複写する各テーブルについては、次の作業を行う必要があります。

- Replication Server 間にルートとコネクションを設定する。
- プライマリ・データベース内に複写元のテーブルを作成する。

- 送信先データベース内に複写先の 1 つまたは複数のテーブルを作成する。
- テーブルに対するインデックスを作成して適切なパーミッションを付与する。

プライマリ・サイトでの作業

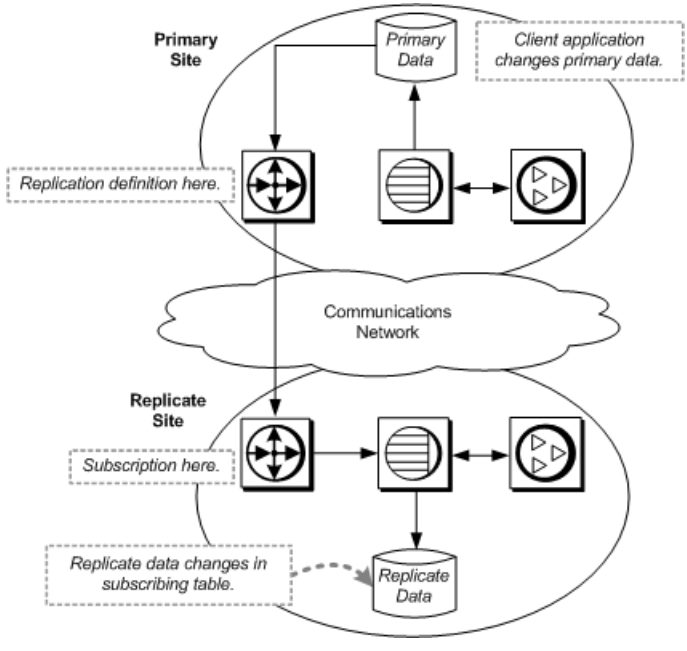
- **sp_setreptable** システム・プロシージャを使用してプライマリ・テーブルを複写するようマーク付けする。
- プライマリ Replication Server のテーブルに対する 1 つまたは複数の複写定義を作成する。

レプリケート・サイトでの作業
各レプリケート Replication Server に、テーブル複写定義に対するサブスクリプションを作成する。

基本プライマリ・コピー・モデルを設定する方法の詳細については、『Replication Server 管理ガイド 第 1 巻』を参照してください。

この図では、プライマリ・サイト (東京) のクライアント・アプリケーションがプライマリ・データベース内の `publishers` テーブルに変更を加えます。レプリケート・サイト (シドニー) では、`pub_id` が 1,000 以上のローについて、`publishers` テーブルがプライマリ `publishers` テーブルに対してサブスクリプションを作成します。

図 10 : テーブル複写定義を使用する基本プライマリ・コピー・モデル



テーブルを複写するようマーク付けする方法
次のスクリプトは、publishers テーブルを複写するようマーク付けします。

```
-- Execute this script at Tokyo data server
 -- Marks publishers for replication
 sp_setreptable publishers, 'true'
 go
 /* end of script */
```

複写定義
次のスクリプトは、publishers テーブルに対するテーブル複写定義をプライマリ Replication Server に作成します。

```
-- Execute this script at Tokyo Replication Server
-- Creates replication definition pubs_rep
create replication definition pubs_rep
with primary at TOKYO_DS.pubs2
with all tables named 'publishers'
(pub_id char(4),
 pub_name varchar(40),
 city varchar(20),
 state varchar(2))
primary key (pub_id)
go
/* end of script */
```

サブスクリプション
次のスクリプトは、プライマリ Replication Server で定義されている複写定義に対するサブスクリプションを作成します。

```
-- Execute this script at Sydney Replication Server
 -- Creates subscription pubs_sub
 Create subscription pubs_sub
 for pubs_rep
 with replicate at SYDNEY_DS.pubs2
 where pub_id >= 1000
 go
 /* end of script */
```

参照：
- パブリケーション (68 ページ)
- 複数の複写定義 (65 ページ)

適用ファンクション

ストアド・プロシージャ呼び出しをレプリケート・データを持つリモート・サイトに複写するために適用ファンクションを使用することができます。

プライマリ・データを複写するために適用ファンクションを使用すると次のことができます。

- WAN 上のネットワーク・トラフィックを減少させる。
- 適用ファンクションは高速に実行されるので、スループットを増大させて遅延時間を減少させる。
- よりモジュール化されたシステム設計を可能にする。

次の例では、プライマリ・サイト (東京) のクライアント・アプリケーションがユーザ・ストアド・プロシージャ **upd_publishers_pubs2** を実行して、プライマリ・データベースの `publishers` テーブルに変更を加えます。

upd_publishers_pubs2 を実行すると、ファンクション複写が呼び出されて、同じく **upd_publishers_pubs2** という名前の対応するストアド・プロシージャをレプリケート・データ・サーバ上で実行させます。

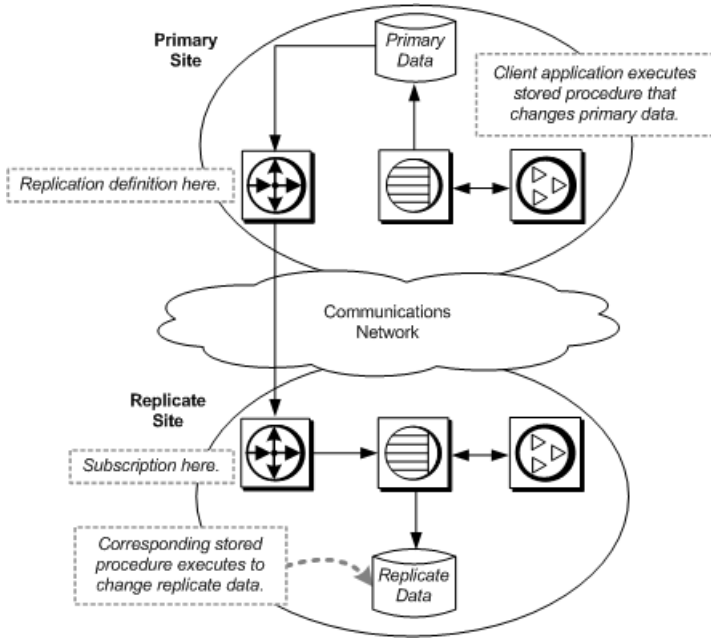
プライマリ・サイトでの作業

- プライマリ・データベースにユーザ・ストアド・プロシージャを作成する。
- **sp_setrepproc** を使用して、ユーザ・ストアド・プロシージャに複写ファンクションを配布するようにマーク付けする。
- 適切なユーザに適切なプロシージャ・パーミッションを付与する。
- プライマリ Replication Server で、ストアド・プロシージャのものと一致する名前、パラメータ、データ型を持つストアド・プロシージャ用のファンクション複写定義を作成する。複写するパラメータだけを指定することができます。

レプリケート・サイトでの作業

- プライマリ・データベースに作成されたのと同じパラメータ (またはそれらのパラメータのサブセット) とデータ型を持つストアド・プロシージャをレプリケート・データベース内に作成する。そのプロシージャに対する適切なパーミッションをメンテナンス・ユーザに付与します。
- レプリケート Replication Server に、ファンクション複写定義に対するサブスクリプションを作成する。

図 11 : 適用ファンクションを使用する基本プライマリ・コピー・モデル



ストアド・プロシージャ
次のスクリプトは、プライマリ・サイトとレプリケート・サイトに `publishers` テーブル用のストアド・プロシージャを作成します。

```
-- Execute this script at Tokyo and Sydney data servers
 -- Creates stored procedure upd_publishers_pubs2
 create procedure upd_publishers_pubs2
(@pub_id char(4),
@pub_name varchar(40),
 @city varchar(20),
 @state char(2))
 as
 update publishers
 set
    pub_name = @pub_name,
    city = @city,
    state = @state
 where
    pub_id = @pub_id
 go
 /* end of script */
```

ファンクション複写定義
次のスクリプトは、プライマリ Replication Server に `publishers` テーブル用の適用ファンクション複写定義を作成します。この複写定義は、プライマリ・データベース内のストアド・プロシージャと同じパラメータとデータ型を使用します。

```
-- Execute this script at Tokyo Replication Server
-- Creates replication definition
_upd_publishers_pubs2_repdef
-- create applied function replication definition
   upd_publishers_pubs2_repdef
with primary at TOKYO_DS.pubs2
with all functions named upd_publishers_pubs2
(@pub_id char(4),
 @pub_name varchar(40),
 @city varchar(20),
 @state char(2))
go
/* end of script */
```

サブスクリプション
ファンクション複写定義に対するサブスクリプションは、次の 2 つの方法のいずれかで作成できます。

- **create subscription** コマンドで非マテリアライゼーション・メソッドを使用する。
  この方法は、プライマリ・データがすでにレプリケートにロードされていて更新処理の実行中でない場合に使用します。
- **define subscription**、**activate subscription**、**validate subscription** コマンドでバルク・マテリアライゼーション・メソッドを使用する。

非マテリアライゼーション・メソッドの使用

次のスクリプトは、プライマリ Replication Server で定義されている複写定義用の非マテリアライゼーション・メソッドを使用して、レプリケート Replication Server にサブスクリプションを作成します。

```
-- Execute this script at Sydney Replication Server
-- Creates subscription using no-materialization
-- for upd_publishers_pubs2_repdef
create subscription upd_publishers_pubs2_sub
   for upd_publishers_pubs2_repdef
with replicate at SYDNEY_DS.pubs2
without materialization
go
/* end of script */
```

バルク・マテリアライゼーションの使用

次のスクリプトは、プライマリ Replication Server で定義されている複写定義に対するサブスクリプションをレプリケート Replication Server で定義し、アクティブにして、確定化します。

```
-- Execute this script at Sydney Replication Server
-- Creates subscription using bulk materialization
-- for upd_publishers_pubs2_repdef
define subscription upd_publishers_pubs2_sub
   for upd_publishers_pubs2_repdef
```

```
with replicate at SYDNEY_DS.pubs2
go
```

```
activate subscription upd_publishers_pubs2_sub
   for upd_publishers_pubs2_repdef
with replicate at SYDNEY_DS.pubs2
go
/* Load data. If updates are in progress,use activate
subscription with the "with suspension" clause and
resume connection after the load. */
```

```
validate subscription upd_publishers_pubs2_sub
   for upd_publishers_pubs2_repdef
with replicate at SYDNEY_DS.pubs2
go
/* end of script */
```

分散プライマリ・フラグメント・モデル

分散プライマリ・フラグメント・モデルでは、各サイトのテーブルには、プライマリ・データと複写データの両方が含まれます。

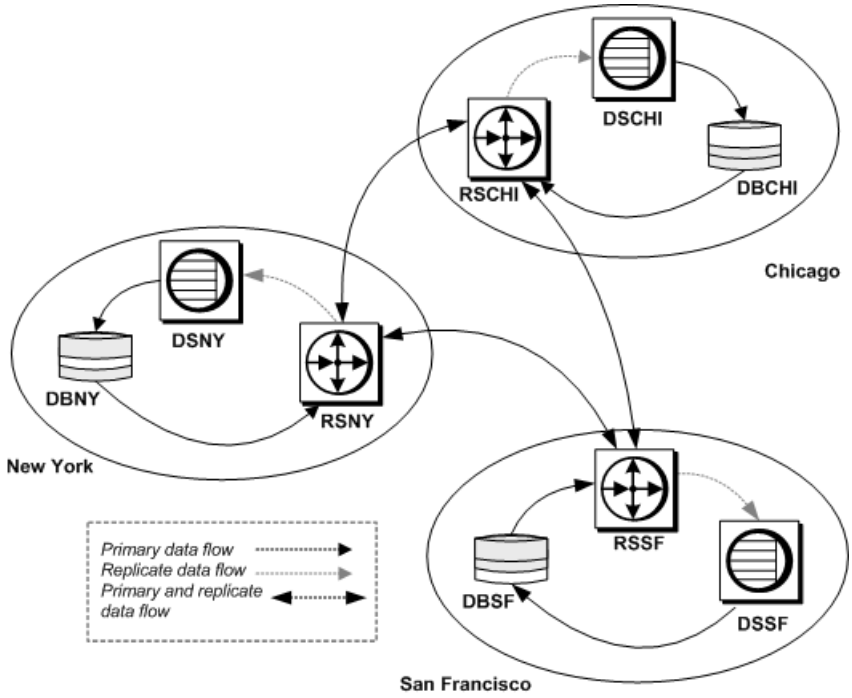
各サイトは「フラグメント」と呼ばれる特別なローのサブセットに対するプライマリ・サイトとして機能します。プライマリ・フラグメントに対する更新は、他のサイトに分配されます。非プライマリ・データに対する更新は、他のフラグメントのプライマリ・サイトから受信されます。

分散プライマリ・フラグメント・モデルを使用するアプリケーションは、プライマリ・データと複写データを含んでいる分散テーブルを持ちます。各サイトの Replication Server は、ローカル・プライマリ・データに行われた修正を他のサイトに分配して、他のサイトから受信した修正をローカルに複写されるデータに適用します。

分散プライマリ・フラグメント・モデルのテーブルを複写するには、各サイトで次の作業を実行する必要があります。

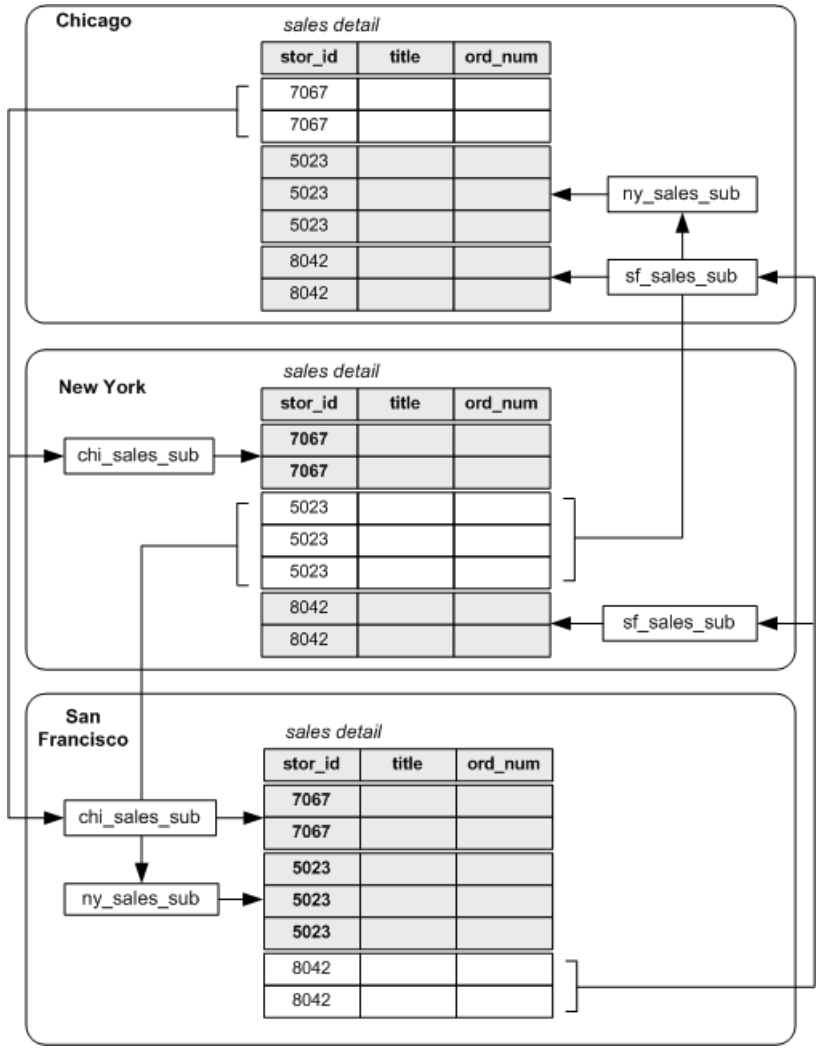
- 各データベースにテーブルを作成する。このテーブルは各データベース内で同じ構造を持つ必要があります。
- テーブルに対するインデックスを作成して適切なパーミッションを付与する。
- **sp_setreptable** システム・プロシージャを使用して、テーブルの複写を許可する。
- 各サイトのテーブルについて複写定義を作成する。
- 各サイトで、他のサイトの複写定義に対するサブスクリプションを作成する。サイト数が *n* であれば、*n*-1 の数のサブスクリプションを作成します。

図 12 : 分散プライマリ・フラグメント・モデル



次の図は、3 つのサイトに分散されたプライマリ・フラグメントを使用して設定された `salesdetail` テーブルを示しています。各サイトは、2つのサブスクリプションによって複写データを受信します。

図 13 : 3 つの分散プライマリ・フラグメントを持つテーブル



複写定義

サンプル・スクリプトを使用して、各サイトに `salesdetail` テーブル用の複写定義を作成します。

```
-- Execute this script at Chicago RSCHI.
-- Creates replication definition chi_sales.
create replication definition chi_sales_rep
with primary at DSCHI.DBCHI
with all tables named 'salesdetail'
```

```
   (stor_id char(4),
    ord_num varchar(20),
    title_id varchar6),
    qty smallint,
    discount float)
primary key (stor_id, ord_num)
searchable columns(stor_id, ord_num, title_id)
go
/* end of script */
```

```
-- Execute this script at New York RSNY.
-- Creates replication definition ny_sales.
create replication definition ny_sales_rep
with primary at DSNY.DBNY
with all tables named 'salesdetail'
   (stor_id char(4),
    ord_num varchar(20),
    title_id varchar6),
    qty smallint,
    discount float)
primary key (stor_id, ord_num)
searchable columns(stor_id, ord_num, title_id)
go
/* end of script */
```

```
-- Execute this script at San Francisco RSSF.
-- Creates replication definition sf_sales.
create replication definition sf_sales_rep
with primary at DSSF.DBSF
with all tables named 'salesdetail'
   (stor_id char(4),
    ord_num varchar(20),
    title_id varchar6),
    qty smallint,
    discount float)
primary key (stor_id, ord_num)
searchable columns(stor_id, ord_num, title_id)
go
/* end of script */
```

サブスクリプション

各サイトは、他の 2 つのサイトの複写定義に対するサブスクリプションを持ちます。

次のスクリプトは、このサブスクリプションを作成します。

```
-- Execute this script at Chicago RSCHI.
-- Creates subscriptions to ny_sales and sf_sales.
create subscription ny_sales_sub
   for ny_sales_rep
   with replicate at DSCHI.DBCHI
   where stor_id = '5023'
go
create subscription sf_sales_sub
   for sf_sales_rep
```

```
   with replicate at DSCHI.DBCHI
   where stor_id = '8042'
go
/* end of script */
```

```
-- Execute this script at New York RSNY.
-- Create subscriptions to chi_sales and sf_sales.
create subscription chi_sales_sub
   for chi_sales_sub
   with replicate at DSNY.DBNY
   where stor_id = '7067'
go
create subscription sf_sales_sub
   for sf_sales_rep
   with replicate at DSNY.DBNY
   where stor_id = '8042'
go
/* end of script */
```

```
-- Execute this script at San Francisco RSSF.
-- Creates subscriptions to chi_sales and ny_sales.
create subscription chi_sales_sub
   for chi_sales_rep
   with replicate at DSSF.DBSF
   where stor_id = '7067'
go
create subscription ny_sales_sub
   for ny_sales_rep
   with replicate at DSSF.DBSF
   where stor_id = '5023'
go
/* end of script */
```

コーポレート・ロールアップ

コーポレート・ロールアップ・モデルでは、リモート・サイトで管理されている複数のプライマリ・フラグメントが中央サイトの 1 つの集約レプリケート・テーブルに統合されます。

コーポレート・ロールアップ・モデルには、複数の分散プライマリ・フラグメント・モデルと 1 つの集中化された統合レプリケート・テーブルがあります。各プライマリ・サイトのテーブルには、そのサイトでプライマリであるデータだけが含まれます。これらのサイトにはデータは複写されません。コーポレート・ロールアップ・テーブルは、プライマリ・サイトからロールアップされたデータで構成されています。

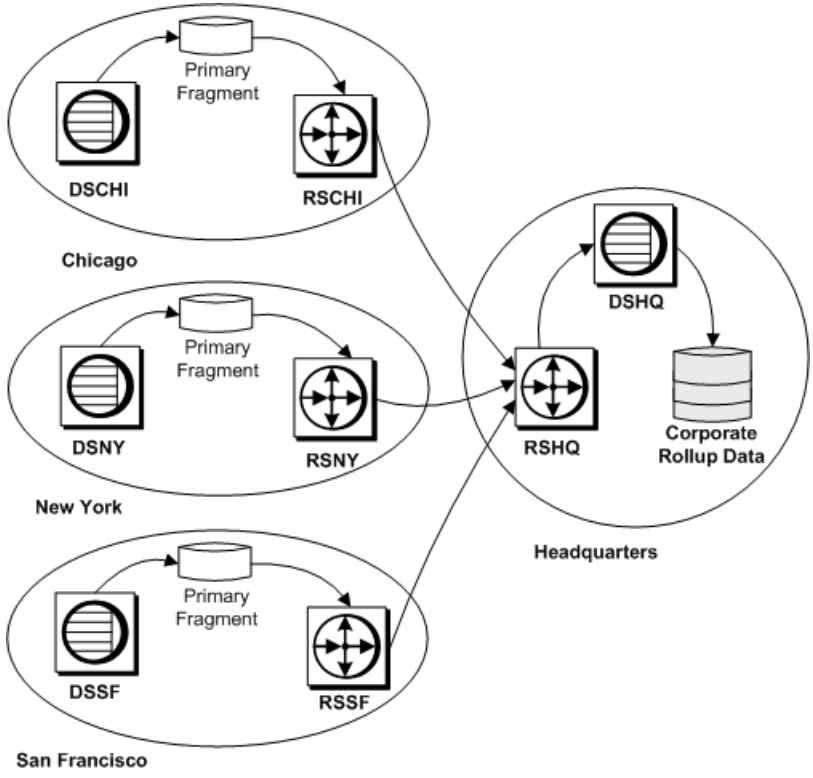
コーポレート・ロールアップ・モデルは、それぞれのプライマリ・サイトに別々の複写定義を必要とします。データが統合されるサイトは、各プライマリ・サイトの複写定義に対するサブスクリプションを持ちます。

Replication Agent はプライマリ・サイトには必要ですが、データは中央サイトからは複写されないので中央サイトには必要ありません。

分散プライマリ・フラグメントからコーポレート・ロールアップを作成するには、次の作業を実行する必要があります。

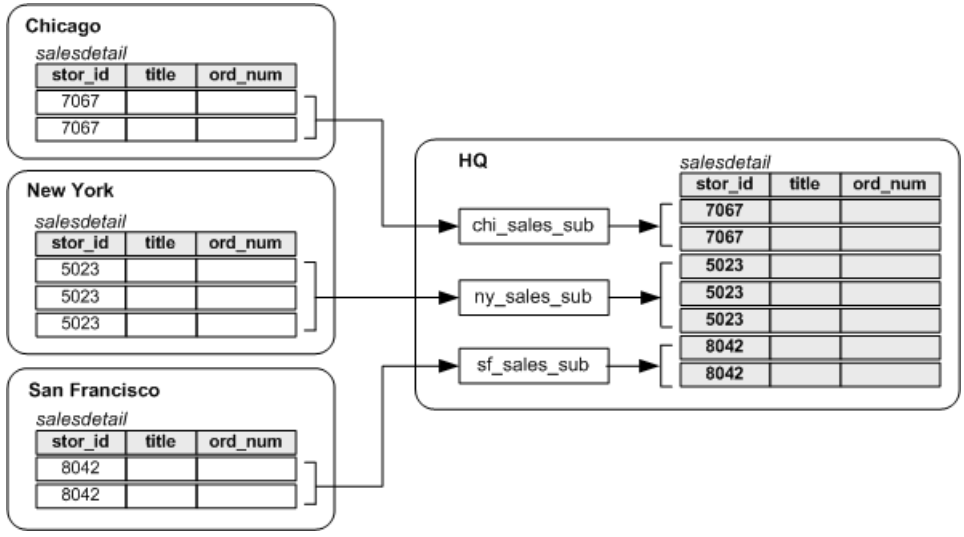
- 各プライマリ・データベースと中央サイトのデータベースにテーブルを作成する。これらのテーブルは、同じ構造と同じ名前を持つ必要があります。
- テーブルに対するインデックスを作成して適切なパーミッションを付与する。
- 各リモート・データベースで、**sp_setreptable** システム・プロシージャを使用して、テーブルに関する複写を許可する。
- 各リモート・サイトのテーブル用の複写定義を作成する。
- データが統合される本社サイトで、リモート・サイトの複写定義に対するサブスクリプションを作成する。

図 14 : 分散プライマリ・フラグメントを持つコーポレート・ロールアップ・モデル



次の図に、本社サイトにコーポレート・ロールアップを持つ salesdetail テーブルを示します。本社サイトは、3 つのサブスクリプションによってリモート・サイトからデータを受信します。

図 15 : 複数のプライマリ・フラグメントを持つテーブル



複写定義

サンプル・スクリプトを使用して、各プライマリ・サイトに salesdetail テーブル用の複写定義を作成します。

```
-- Execute this script at Chicago RSCHI.
-- Creates replication definition chi_sales.
create replication definition chi_sales_rep
   with primary at DSCHI.DBCHI
   with all tables named 'salesdetail'
   (stor_id char(4),
    ord_num varchar(20),
    title_id varchar6),
    qty smallint,
    discount float)
   primary key (stor_id, ord_num)
   searchable columns(stor_id, ord_num, title_id)
go
/* end of script */
```

```
-- Execute this script at New York RSNY.
-- Creates replication definition ny_sales.
create replication definition ny_sales_rep
   with primary at DSNY.DBNY
```

```
   with all tables named 'salesdetail'
   (stor_id char(4),
    ord_num varchar(20),
    title_id varchar6),
    qty smallint,
    discount float)
   primary key (stor_id, ord_num)
   searchable columns(stor_id, ord_num, title_id)
go
/* end of script */
```

```
-- Execute this script at San Francisco RSSF.
-- Creates replication definition sf_sales.
create replication definition sf_sales_rep
   with primary at DSSF.DBSF
   with all tables named 'salesdetail'
   (stor_id char(4),
    ord_num varchar(20),
    title_id varchar6),
    qty smallint,
    discount float)
   primary key (stor_id, ord_num)
   searchable columns(stor_id, ord_num, title_id)
go
/* end of script */
```

サブスクリプション

本社サイトは、3 つのプライマリ・サイトのそれぞれの複写定義に対するサブスクリプションを持っています。プライマリ・サイトにはサブスクリプションはありません。

次のスクリプトは、RSHQ Replication Server にサブスクリプションを作成します。

```
-- Execute this script at Headquarters RSHQ.
-- Creates subscriptions to chi_sales, ny_sales,
-- and sf_sales.
create subscription chi_sales_sub
   for chi_sales_rep
   with replicate at DSHQ.DBHQ
   where stor_id = '7067'
go
create subscription ny_sales_sub
   for ny_sales_rep
   with replicate at DSHQ.DBHQ
   where stor_id = '5023'
go
create subscription sf_sales_sub
   for sf_sales_rep
   with replicate at DSHQ.DBHQ
   where stor_id = '8042'
go
/* end of script */
```

再分配コーポレート・ロールアップ

再分配コーポレート・ロールアップ・モデルは、統合テーブルがリモート・サイトに再分配される点を除けばコーポレート・ロールアップ・モデルと同じです。

リモート・サイトに分配されるプライマリ・フラグメントは、中央サイトの統合テーブルにロールアップされます。フラグメントが統合されるサイトでは、RepAgent は統合テーブルをプライマリ・データであるかのように処理します。

統合テーブルは、複写定義で記述されます。他のサイトは、このテーブルに対するサブスクリプションを作成できます。

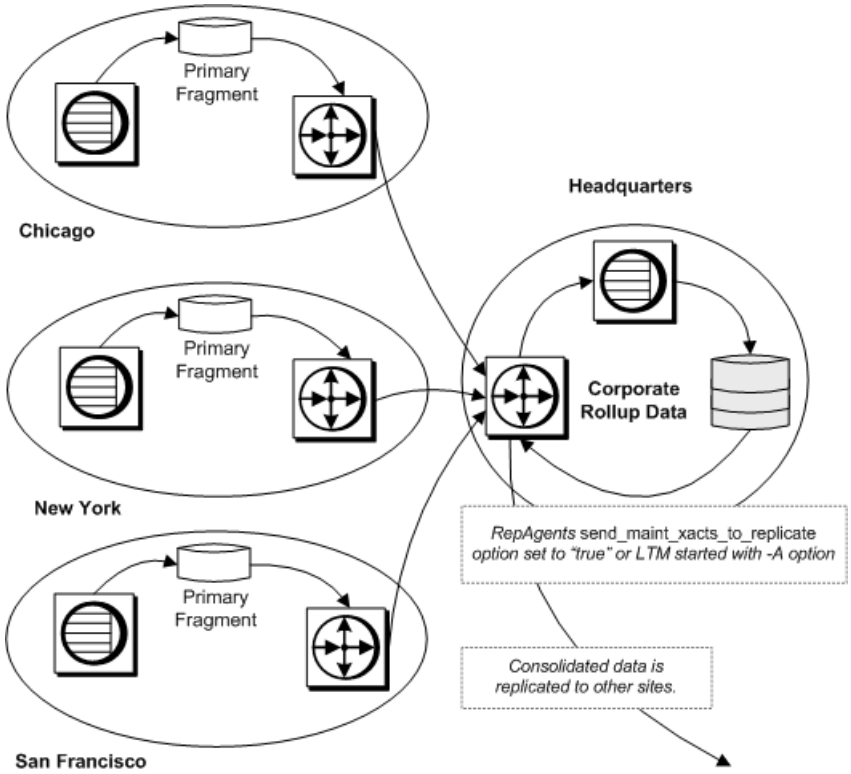
通常は、Adaptive Server の RepAgent はメンテナンス・ユーザによって行われる更新を除外するため、複写データがプライマリ・データとして再分配されることはありません。

RepAgent の **send_maint_xacts_to_replicate** オプションは、再分配コーポレート・ロールアップ・モデルに対して提供されます。**send_maint_xacts_to_replicate** を “true” に設定して RepAgent を起動すると、RepAgent はすべての更新を、あたかもクライアント・アプリケーションによって行われたかのように Replication Server に送信します。

再分配コーポレート・ロールアップ・モデルを使用する場合は、次のことを許可しないでください。

- プライマリ・サイトで、そのプライマリ・データに対するサブスクリプションを再作成する。再作成を許可すると、トランザクションがシステムの無限ループを発生させることがある。
- コーポレート・ロールアップ・テーブルをアプリケーションで更新する。すべての更新は、プライマリ・サイトから開始するようにする。

図 16 : 分散フラグメント・モデルでの再分配コーポレート・ロールアップ



再分配コーポレート・ロールアップ・モデルの設計は、次の点を除けばコーポレート・ロールアップ・モデルと同じです。

- DBHQ データベース用に本社サイトに RepAgent をインストールする必要がある。RepAgent は、メンテナンス・ユーザからログ・レコードを転送するように、**send_maint_xacts_to_replicate** option を設定して起動する必要がある。
- データはそのサイトから分配されるので、RepAgent が RSHQ RSSD に対して必要である。
- 複写定義を、本社サイトの `salesdetail` テーブル用に作成する必要がある。他のサイトは、この複写定義に対するサブスクリプションを作成することはできるが、プライマリ・サイトはそれ自身のプライマリ・データに対してサブスクリプションを作成できない。
- RSHQ Replication Server は、統合レプリケート・テーブルに対するサブスクリプションを作成する他のサイトへのルートを持つ必要がある。プライマリ・サイトがサブスクリプションを作成する場合は、RSHQ からそれらへのルートを作成する必要がある。

ウォーム・スタンバイ・アプリケーション

ウォーム・スタンバイ・アプリケーションでは、Replication Server は同じベンダからの 1 組のデータベースを管理します。この組は、一方がもう一方のバックアップとして機能します。

通常、クライアント・アプリケーションはアクティブ・データベースを更新するのに対して、Replication Server は他のデータベースをアクティブ・データベースのスタンバイ・コピーとして管理します。アクティブ・データベースに障害が発生した場合、またはアクティブなデータ・サーバやアクティブ・データベースに対してメンテナンスを実行する必要がある場合は、クライアント・アプリケーションをほとんど中断させることなく、スタンバイ・データベースに切り替えたり元に戻したりすることができます。

ウォーム・スタンバイ・アプリケーションで、以下を作成します。

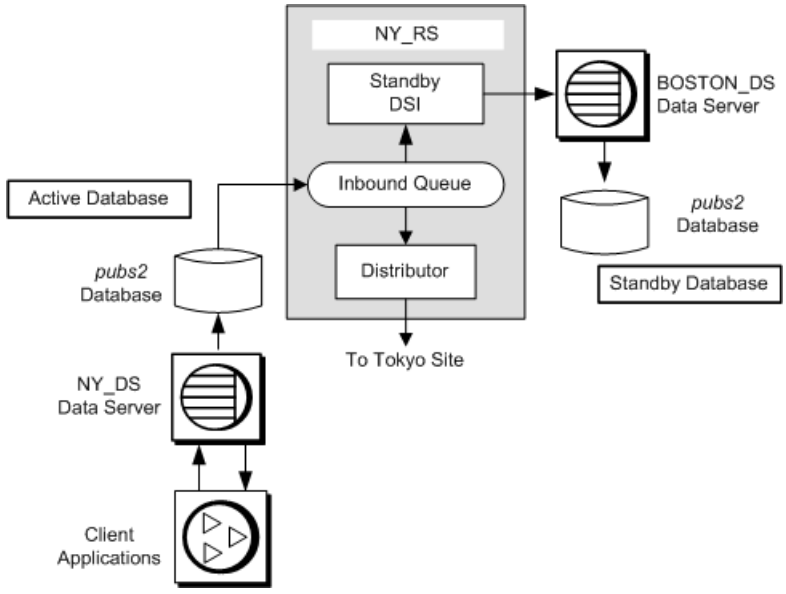
- Replication Server が現在アクティブなデータベースにマップする論理コネクション
- アクティブ・データベースに対する物理コネクション
- スタンバイ・データベースに対する物理コネクション

ウォーム・スタンバイ・アプリケーションの論理データベースは、複写システム内の他のデータベースに関して次のいずれかとして機能します。

- 複写に参加しないデータベース
- プライマリ・データベース
- レプリケート・データベース

次の図は、NY_DS データ・サーバの `pubs2` データベースに対して BOSTON_DS データ・サーバで稼働するウォーム・スタンバイ・アプリケーションを示します。このデータベースは TOKYO_DS に複写されます。

図 17 : ウォーム・スタンバイ・システム



この例の場合は、pubs2 データベースはプライマリ・データベースとして動作します。スタンバイ・データベースが作成されるプライマリ・データベース pubs2 を、「アクティブ・データベース」と呼びます。

ウォーム・スタンバイ・アプリケーションの設定

アクティブな Adaptive Server データベースがすでに設定されている例を使用して、アクティブ・データベースのウォーム・スタンバイ・アプリケーションの設定方法について説明します。

処理を行う前に、『Replication Server 管理ガイド 第 1 巻』内のウォーム・スタンバイ・アプリケーションについての説明を参照してください。Oracle 用のウォーム・スタンバイ・アプリケーションの設定方法については、『異機種間複写ガイド』を参照してください。

1. **sp_reptostandby** を使用して、スタンバイ・データベースに複写するようアクティブ・データベース全体をマーク付けします。

   sp_reptostandby は、DML (データ操作言語) とサポートされる DDL (データ定義言語) のコマンドおよびストアド・プロシージャの複写を可能にします。詳細については、『Replication Server 管理ガイド 第 2 巻』の「ウォーム・スタンバイ・アプリケーションの管理」を参照してください。
2. **sp_config_rep_agent** ストアド・プロシージャに **send_warm_standby_xacts** オプションを使用して RepAgent を再設定し、RepAgent を再起動します。

3. アクティブ・データベースのメンテナンス・ユーザに **replication_role** を付与します。
4. アクティブ・データ・サーバで、スタンバイ・データベースのメンテナンス・ユーザをアクティブ・データベースに追加して、**replication_role** を新しいメンテナンス・ユーザに付与します。この手順によって、データベースがロードされた後にメンテナンス・ユーザ ID がスタンバイ・データベースに存在することが保証されます (手順 8)。
5. ウォーム・スタンバイ・データベースを管理するための Replication Server にログインして、**create logical connection** コマンドを使用して、アクティブ・データベース用に論理コネクションを作成します。論理コネクションの名前は、アクティブ・データベースの名前と同じでなければなりません。

   注意： アクティブなデータベース・コネクションを作成する前に論理コネクションを作成する場合は、論理コネクションとアクティブ・データベースには異なる名前を使用してください。
6. スタンバイ・データ・サーバに、アクティブ・データベースと同じサイズのスタンバイ・データベースを作成します。
7. Sybase Central または **rs_init** を使用して、スタンバイ・データベース・コネクションを作成します。詳細については、Replication Server オンライン・ヘルプと各プラットフォーム用の『Replication Server インストール・ガイド』および『Replication Server 設定ガイド』を参照してください。

   コネクションが作成された後に、Replication Server にログインし、**admin logical_status** コマンドを使用して、新しいコネクションが「アクティブ」であることを確認します。
8. **dump** と **load** コマンドに、**rs_init** の “dump marker” オプションを指定しないで使用して、スタンバイ・データベースを初期化します。または、**bcp** を使用することもできます。詳細については、『Replication Server 管理ガイド 第 2 巻』を参照してください。
   a) Replication Server で、アクティブなデータベース・コネクションをサスペンドします。

      注意： アクティブ・データベースをサスペンドできない場合は、**dump** と **load** コマンドに **rs_init** の “dump marker” オプションを指定して使用してください。
   b) アクティブな Adaptive Server で、アクティブ・データベースをダンプします。
   c) アクティブ・データベースのダンプをスタンバイ・データベースにロードします。
   d) スタンバイ Adaptive Server で、スタンバイ・データベースをオンラインにします。

9. Replication Server で **resume connection** コマンドを使用して、アクティブ・データベースとスタンバイ・データベースへのコネクションをレジュームします。

   admin logical_status コマンドを使用して、論理ステータスを調べます。アクティブ・データベースとスタンバイ・データベースの両方が「アクティブ」としてマーク付けされないかぎり、継続しないでください。
10. 修正がアクティブ・データベースからスタンバイ・データベースに対して行われることを確認します。

    isql を使用して、アクティブ・データベースのレコードを更新してからスタンバイ・データベースの更新を確認します。

スタンバイ・データベースへの切り替え

アクティブ・データベースとスタンバイ・データベースの切り替え

アクティブ・データベースからスタンバイ・データベースに切り替える場合は、切り替え中にクライアント・アプリケーションが、トランザクションを実行したりアクティブ・データベースを更新したりできないようにする必要があります。切り替えが完了した後は、クライアントは新しいアクティブ・データベースに接続してその作業を継続することができます。

切り替えが必要かどうかの判断

- アクティブ・データ・サーバに一時的な障害が発生した場合は、切り替えは必要ありません。一時的な障害とは、特別なリカバリ手順を行わなくても Adaptive Server の再起動時にリカバリされる障害のことをいいます。
- アクティブ・データベースが長い間使用できない状態になる場合は切り替えを行います。

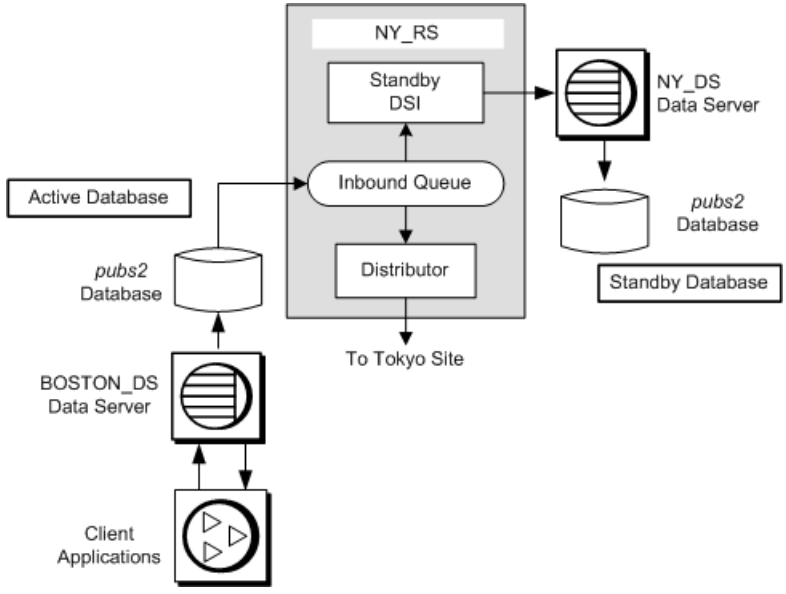
アクティブ・データベースとスタンバイ・データベースを切り替えるには、**switch active** コマンドを使用する必要があります。

1. Replication Server で、**switch active** を使用して、スタンバイ・データベースに処理を切り替える。
2. 切り替えの進行状況をモニタする。スタンバイ・コネクションがアクティブになって、以前のアクティブ・コネクションがサスペンドされれば、切り替えは完了です。
   a) Replication Server で、**admin_logical_status** コマンドを使用して論理ステータスを調べる。
   b) 切り替えの進行状況を調べるには、Replication Server のエラー・ログ内の最後のいくつかのエントリを調べる。
3. 新しいアクティブ・データベース用に RepAgent を起動する。
4. 以前のアクティブ・データベースで行う必要のあることを調べる。次のことができます。

- データベースを新しいスタンバイ・データベースとしてオンラインにして、Replication Server が新しいトランザクションを適用できるようにコネクションをレジュームします。
- **drop connection** コマンドを使用して、データベース・コネクションを削除する。新しいスタンバイ・データベースとして後から再度追加することもできます。
- 新しいスタンバイ・データベースが元のアクティブ・データベースでない場合は、新しいアクティブ・データベースの新しいウォーム・スタンバイ・データベース・コネクションを作成する。

5. **isql** を使用して、新しいアクティブ・データベース内のレコードを更新してから、新しいスタンバイ・データベースの更新を確認する。

図 18 : 切り替え後のウォーム・スタンバイ・システム



クライアントを新しいデータベースに切り替える方法

アクティブ・データベースからスタンバイ・データベースに切り替えても、クライアント・アプリケーションが新しいアクティブ・データ・サーバとデータベースに切り替えられることはありません。

クライアントの切り替えを処理するための方法を工夫する必要があります。たとえば次のようにします。

- 2 つの interfaces ファイルを使用して、一方をクライアント・アプリケーション用に、もう一方を Replication Server 用に設定する。切り替えのときに、クライ

アントの interfaces ファイルが新しいアクティブ・サーバを指すように修正します。
- クライアント・アプリケーションで使用できるように、データ・サーバの記号名を使用して interfaces ファイル・エントリを作成する。切り替えのときに、記号名に対応するアドレス情報を修正します。
- 中間的な Open Server などのメカニズムを使用して、クライアント・アプリケーションのデータ・サーバ・コネクションを現在のアクティブ・データ・サーバに自動的にマップする。

『Replication Server 管理ガイド 第 2 巻』を参照してください。

モデルの変化形と方式

モデルの変化形やその他の方式は、複写システム設計を実装する際に役立ちます。

モデルの変化形と方式の一部を紹介します。

- 複数の複写定義 ― プライマリ・テーブル用に複数の複写定義を使用して、異なるテーブル名、カラム・セット、カラム名を指定することによって、サブスクリプションを作成するレプリケート・テーブルに対するプライマリ・テーブルのさまざまなビューを提供する方式です。
- パブリケーション ― ユーザが 1 つのサブスクリプションを使用して、複写定義セットに対してサブスクリプションを作成できる方式です。
- 保留テーブル ― 更新結果がレプリケート・サイトに返される前にユーザがプライマリ・データに対する更新の結果を表示できる要求ファンクションを使用する方式です。
- マスタ／ディテール関係の実装 ― 適切なサブスクリプションのマイグレーションを保証するために要求ファンクションとストアド・プロシージャを使用します。

複数の複写定義

1 つのプライマリ・テーブルに対して複数の複写定義を作成することができます。各複写定義は、異なるテーブル名、カラム・セット、カラム名を指定することによって、サブスクリプションを作成するレプリケート・テーブルのそれぞれに対してプライマリ・テーブルの異なるビューを提供できます。

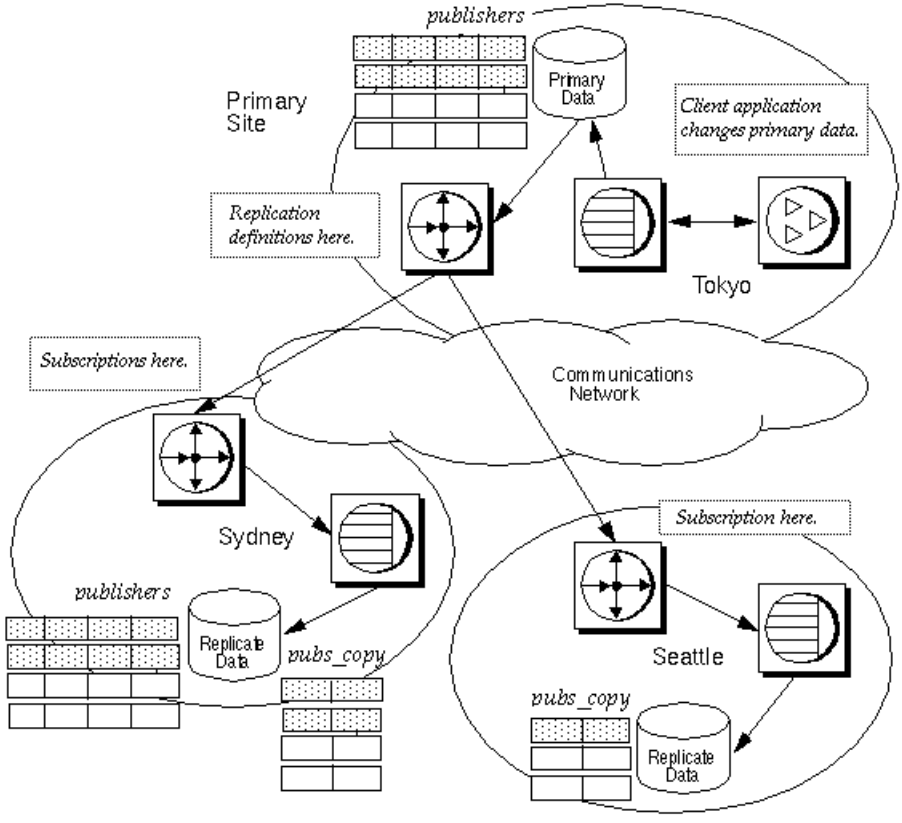
注意： 1 つのプライマリ・テーブルに対して複数の複写定義を作成でき、1 つのレプリケート・テーブルは複数のテーブル複写定義に対してサブスクリプションを作成できます。ただし、1 つのレプリケート・テーブルがサブスクリプションを作成できる複写定義は、各プライマリ・テーブルにつき 1 つだけです。

複数の複写定義を使用するシステムを設定するには、「テーブル複写定義」の手順に従い、必要に応じて複数の複写定義とそれぞれに対して 1 つのサブスクリプ

ションを作成します。複数の複写定義を使用するということは、プライマリ・コピー・モデルの変化形を使用するということです。

この図では、プライマリ・サイト (東京) のクライアント・アプリケーションがプライマリ・データベース内の publishers テーブルに変更を加えます。レプリケート・サイト (シドニー) では、publishers テーブルはテーブル全体に対してサブスクリプションを作成し、pubs_copy テーブルは pub_id カラムと pub_name カラムに対してのみサブスクリプションを作成します。別のレプリケート・サイト (シアトル) では、pubs_copy テーブルは pub_id カラムと pub_name カラムに対してサブスクリプションを作成します。このときの pub_id は 1,000 以上の値です。

図 19 : 複数の複写定義



参照：
- テーブル複写定義 (44 ページ)

複写定義
サンプル・スクリプトを使用して、プライマリ Replication Server に `publishers` テーブル用のテーブル複写定義を作成します。

各複写定義は、プライマリ・テーブルの異なるビューを記述します。

```
-- Execute this script at Tokyo Replication Server
-- Creates replication definitions pubs_rep and
-- pubs_copy_rep
create replication definition pubs_rep
with primary at TOKYO_DS.pubs2
with all tables named 'publishers'
(pub_id char(4),
 pub_name varchar(40),
 city varchar(20),
 state varchar(2))
primary key (pub_id)
go

create replication definition pubs_copy_rep
with primary at TOKYO_DS.pubs2
with primary table named 'publishers'
with replicate table named 'pubs_copy'
(pub_id char(4),
 pub_name varchar(40))
primary key (pub_id)
go
/* end of script */
```

サブスクリプション
プライマリ Replication Server で定義されている複写定義に対するサブスクリプションを作成します。

次のスクリプトは、プライマリ Replication Server で定義された複写定義に対するサブスクリプションを作成します。

```
-- Execute this script at Sydney Replication Server
 -- Creates subscription pubs_sub and pubs_copy_rep
 Create subscription pubs_sub
 for pubs_rep
 with replicate at SYDNEY_DS.pubs2
 go

 create subscription pubs_copy_sub
 for pubs_copy_rep
 with replicate at SYDNEY_DS.pubs2
 go
 /* end of script */
```

```
-- Execute this script at Seattle Replication Server
 -- Creates subscription pubs_copy_sub
 create subscription pubs_copy_sub
 for publ_copy_rep
 with replicate at SEATTLE_DS.pubs2
 where pub_id >= 1000
 go
 /* end of script */
```

パブリケーション

パブリケーションは、テーブルやストアド・プロシージャに対する複写定義をまとめて、1 つのグループとしてサブスクリプションを作成するために使用します。パブリケーションを使用することは別のモデルということではなく、どのモデルでも使用できるクループ化の方法を提供するものです。パブリケーションを使用すると、テーブルとプロシージャのセットに対する 1 つのパブリケーション・サブスクリプションのステータスをモニタできます。

パブリケーションを使用する場合は、次のオブジェクトを作成して管理します。

- アーティクル – テーブルやファンクション複写定義をパブリケーションに置くことのできるストアド・プロシージャやテーブルのための複写定義を拡張したものです。
- パブリケーション — 同じプライマリ・データベースからのアーティクルの集合です。
- パブリケーション・サブスクリプション — パブリケーションに対するサブスクリプションです。パブリケーション・サブスクリプションを作成すると、Replication Server は、パブリケーションの各アーティクルのサブスクリプションを作成します。

次の手順は、パブリケーションを使用してデータを複写するときの手順をまとめたものです。

プライマリ・サイトでの作業

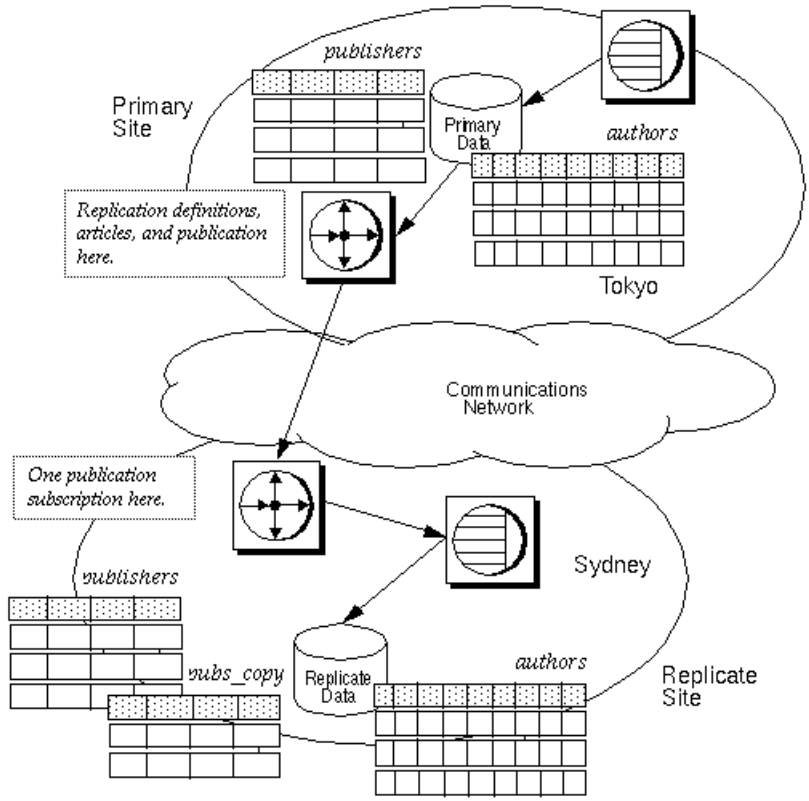
1. 複写定義を作成、または選択してパブリケーションに含める。
2. **create publication** コマンドを使用して、パブリケーションを作成する。
3. **create article** コマンドを使用して、選択した複写定義を参照するアーティクルを作成する。
4. **validate publication** コマンドを使用して、パブリケーションを確定化する。

レプリケート・サイトでの作業

create subscription コマンドを使用して、パブリケーションに対するサブスクリプションを作成する。

次の図では、2 つのアーティクルによって参照されるテーブル複写定義 pubs_rep と、1 つのアーティクルによって参照されるファンクション複写定義が、パブリケーション pubs2_pub にまとめられています。

図 20 : パブリケーション



ストアド・プロシージャ

サンプル・スクリプトを使用して、pubs2 データベース内の authors テーブルを更新するストアド・プロシージャ **update_ authors_pubs2** を作成します。

プライマリ・サイトとレプリケート・サイトに同じプロシージャを作成します。

```
-- Execute this script at the Tokyo and Sydney
-- data servers
-- Creates the stored procedure update_authors_pubs2
create procedure upd_authors_pubs2
(@au_id id,
```

```
 au_lname varchar(40),
 au_fname varchar(20),
 phone char(12),
 address varchar(12),
 city varchar(20),
 state char(2),
 country varchar(12),
 postalcode char(10))
as
 update authors
 set
   au_lname = @varchar(40),
   au_fname = @varchar(20),
   phone = @char(12),
   address = @varchar(12),
   city = @varchar(20),
   state = @char(2),
   country = @varchar(12),
   postalcode = @char(10)
 where au_id = @au_id
go
/* end of script */
```

ファンクション複写定義

サンプル・スクリプトを使用して、プライマリ・サイトに適用ファンクション複写定義を作成します。

```
-- Execute this script at Tokyo Replication Server
-- Creates the applied function replication definition
-- upd_authors_rep_repdef
create applied function replication definition
upd_authors_rep
with primary at TOKYO_DS.pubs2
with all functions named upd_authors_pubs2
(@au_id id,
 au_lname varchar(40),
 au_fname varchar(20),
 phone char(12),
 address varchar(12),
 city varchar(20),
 state har(2),
 country varchar(12),
 postalcode char(10))
go
 /* end of script */
```

テーブル複写定義

サンプル・スクリプトを使用して、プライマリ Replication Server に `publishers` テーブル用のテーブル複写定義を作成します。

```
-- Execute this script at Tokyo Replication Server
-- Creates replication definitions pubs_rep
```

```
create replication definition pubs_rep
with primary at TOKYO_DS.pubs2
with all tables named 'publishers'
(pub_id char(4),
 pub_name varchar(40),
 city varchar(20),
 state varchar(2))
primary key (pub_id)
go
/* end of script */
```

パブリケーション

サンプル・スクリプトを使用して、プライマリ Replication Server にパブリケーション pubs2_pub を作成します。

```
-- Execute this script at Tokyo Replication Server
 -- Creates publication pubs_pub
 create publication pubs2_pub
 with primary at TOKYO_DS.pubs2
 go
 /* end of script */
```

アーティクル

サンプル・スクリプトを使用して、プライマリ Replication Server にパブリケーション pubs2_pub のアーティクルを作成します。

次のサンプル・スクリプトは、複写定義 pubs_rep 用に 2 つのアーティクルを作成します。

```
-- Execute this script at Tokyo Replication Server
-- Creates articles upd_authors_art, pubs_art, and
-- pubs_copy_art
create article upd_authors_art
 for pubs2_pub
with primary at TOKYO_DS.pubs2
with replication definition upd_authors_rep
go
```

```
create article pubs_art
 for pubs2_pub
with primary at TOKYO_DS.pubs2
with replication definition pubs_rep
go
```

```
create article pubs_copy_art
 for pubs2_pub
with primary at TOKYO_DS.pubs2
 with replication definition pubs_rep
 where pub_id >= 1000
 go
 /* end of script */
```

確定化

サンプル・スクリプトを使用して、パブリケーション pubs2_pub のステータスを “valid” に変更します。

```
-- Execute this script at Tokyo Replication Server
-- Validates the publication pubs2_pub
validate publication pubs2_pub
with primary at TOKYO_DS.pubs2
go
/* end of script */
```

サブスクリプション

サンプル・スクリプトを使用して、パブリケーション pubs2_pub 用のサブスクリプション pubs2_pub_sub を作成します。

このスクリプトが実行されると、Replication Server は upd_authors_art、pubs_art、pubs_copy_art 用のアーティクル・サブスクリプションを作成します。

```
-- Execute this script at Sydney Replication Server
-- Creates publication subscription pubs2_pub_sub
create subscription pubs2_pub_sub
 for publication pubs2_pub
with primary at TOKYO_DS.pubs2}
with replicate at SF.pubs2
without materialization
go
/* end of script */
```

要求ファンクション

要求ファンクションを使用すると、プライマリ・データベース・ユーザがレプリケート・データに対してストアド・プロシージャを呼び出せるようにすることができます。

ローカル保留テーブルの使用例

保留テーブルを使用すると、リモート・サイトのクライアントは、中央データを更新して、中央サイトから返される前にリモート・サイトでの更新を表示することができます。このモデルは、ローカル更新アプリケーションを実装するために使用します。

この方式では、リモート・サイトのクライアント・アプリケーションは、要求ファンクションを使用して、中央サイトのデータを更新するユーザ・ストアド・プロシージャを実行します。中央データに対する変更は、適用ファンクションによってリモート・サイトに複写されます。ローカル保留テーブルは、複写システムが更新結果を返す前にリモート・サイトで保留されている更新を、リモート・サイトのクライアントが表示できるようにします。

クライアント・アプリケーションが、リモート・データ・サーバでユーザ・ストアド・プロシージャを実行する場合は、次のようになります。

- 関連するストアド・プロシージャを実行して、プライマリ・サイトのデータを更新する。
- これらの更新結果をローカル保留テーブルに入れる。

中央データベースで更新が成功すると、トランザクションが開始されたサイトを含めて、リモート・サイトに分配されます。リモート・サイトでは、ストアド・プロシージャは複写テーブルを更新して保留テーブルから対応する更新を削除します。

適用ファンクション、要求ファンクション、ローカル保留テーブルを使用するには、次の作業を完了させる必要があります。

リモート・サイトでの作業

- リモート・データベースに保留テーブルを作成する。適切なパーミッションを付与します。
- 要求ファンクションを開始するユーザ・ストアド・プロシージャをリモート・データベースに作成して保留テーブルにデータの更新を挿入する。
- **sp_setrepproc** を使用して、ユーザ・ストアド・プロシージャに複写ファンクションを配布するようにマーク付けする。
- プロシージャ・パーミッションを適切なユーザに付与する。
- リモート・テーブルを更新するユーザ・ストアド・プロシージャをリモート・データベースに作成して、保留テーブルから対応する更新を削除する。適切なパーミッションをメンテナンス・ユーザに付与します。
- 要求ファンクションに対する要求ファンクション複写定義を作成する。
- 中央サイトで作成された適用ファンクション複写定義に対するサブスクリプションを作成する。

プライマリ・サイトでの作業

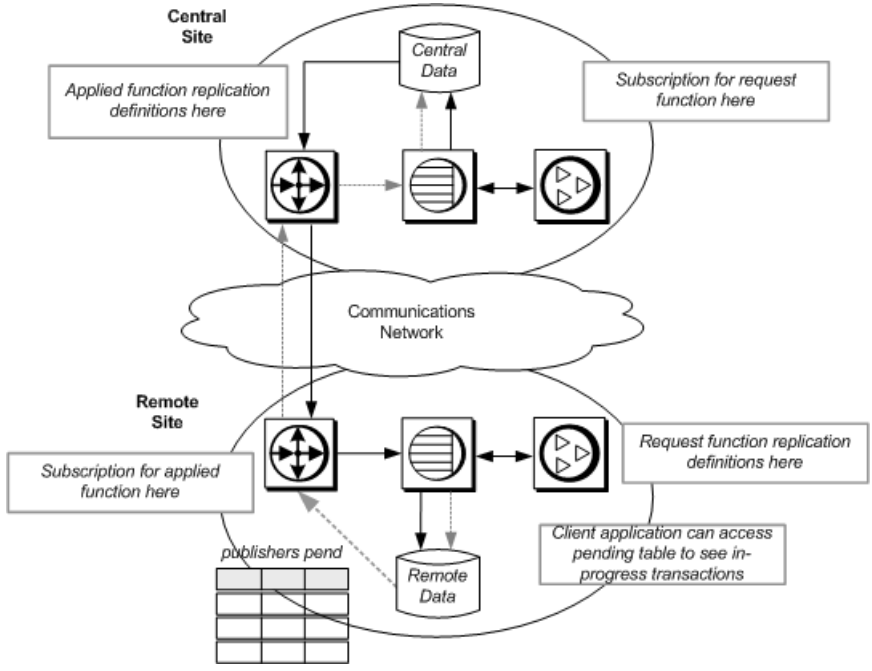
- 中央データを修正するストアド・プロシージャを作成する。
- 適用ファンクションに対する適用ファンクション複写定義を作成する。
- 要求ファンクション複写定義に対するサブスクリプションを作成する。

次の例では、リモート・サイト (シドニー) のクライアント・アプリケーションはストアド・プロシージャ **upd_publishers_pubs2_req** を実行して、`publishers_pend` テーブルに値を挿入し、対応するストアド・プロシージャ **upd_publishers_pubs2** を中央サイト (東京) で実行させます。中央サイトで **upd_publishers_pubs2** を実行すると、ストアド・プロシージャ **upd_publishers_pubs** がリモート・サイトで実行され、`publishers` テーブルを更新して、`publishers_pend` テーブルから対応する情報を削除します。

次の図に、適用ファンクション、要求ファンクション、ローカル保留テーブルを使用する場合のデータの流れを示します。グレーの矢印は、要求ファンクション

の配布の流れを示します。黒い矢印は、適用ファンクションの配布の流れを示します。

図 21 : 要求ファンクションとローカル保留テーブル



保留テーブル

サンプル・スクリプトを使用して、リモート・データベースに保留テーブルを作成します。

```
-- Execute this script at Sydney data server
-- Creates local pending table
create table publishers_pend
(pub_id char(4) not null,
 pub_name varchar(40) null,
 city varchar(20) null,
 statechar(2) null)
go
/* end of script */
```

ストアド・プロシージャ

サンプル・スクリプトを使用して、中央サイト (東京) にストアド・プロシージャ **upd_publisher_pubs2** を作成します。

```
-- Execute this script at Tokoyo data server
-- Creates stored procedure
create procedure upd_publishers_pubs2
```

```
(@pub_id char(4),
 @pub_name varchar(40),
 @city varchar(20),
 @state char(2))
as
   insert into publishers
   values (@pub_id, @pub_name, @city, @state)
go
/* end of script */
```

次のスクリプトは、リモート・サイト (シドニー) に **upd_publishers_pub2_req** ストアド・プロシージャを作成します。**insert into** 句は、publishers_pend テーブルに値を挿入します。

```
-- Execute this script at Sydney data server
-- Creates stored procedure
create procedure upd_publishers_pubs2_req
(@pub_id char(4),
 @pub_name varchar(40),
 @city varchar(20),
 @state char(2))
as
    insert into publishers_pend
    values (@pub_id, @pub_name, @city, @state)
go
/* end of script */
```

次のスクリプトは、リモート・サイト (シドニー) に対する **upd_publishers_pubs2** プロシージャを作成します。これは、publishers テーブルを更新して publishers_pend テーブルから対応する情報を削除します。

```
-- Execute this script at Sydney data server
-- Creates stored procedure upd_publishers_pubs2
 create procedure upd_publishers_pubs2
 (@pub_id char(4),
 @pub_name varchar(40),
 @city varchar(20),
 @state char(2))
 as
 update publishers
 set
    pub_name = @pub_name,
    city = @city,
    state = @state
 where
    pub_id = @pub_id
 delete from publishers_pend
 where
    pub_id = @pub_id
 go
 /* end of script */
```

ファンクション複写定義

サンプル・スクリプトを使用して、中央 Replication Server (東京) に適用ファンクション複写定義を作成します。

```
-- Execute this script at Tokyo Replication Server
-- Creates replication definition
create applied function replication definition
   upd_publishers_pubs2
with primary at TOKYO_DS.pubs2
(@pub_id char(4),
 @pub_name varchar(40),
 @city varchar(20),
 @state char(2))
go
/* end of script */
```

次のスクリプトは、リモート Replication Server (シドニー) に要求ファンクション複写定義を作成します。

```
-- Execute this script at Sydney Replication Server
-- Creates replication definition
create request function replication definition
   upd_publishers_pubs2_req
with primary at SYDNEY_DS.pubs2
with primary function named upd_publishers_pubs2_req
with replicate function named upd_publishers_pubs2
(@pub_id char(4),
 @pub_name varchar(40),
 @city varchar(20),
 @state char(2))
go
/* end of script */
```

サブスクリプション

サンプル・スクリプトを使用して、中央 Replication Server で定義されている適用ファンクション複写定義用の非マテリアライゼーション・メソッドを使用して、リモート Replication Server にサブスクリプションを作成します。

```
-- Execute this script at Sydney Replication Server
-- Creates subscription using no-materialization
for upd_publishers_pubs2
create subscription upd_publishers_pubs2_sub
for upd_publishers_pubs2
with replicate at SYDNEY_DS.pubs2
without materialization
go
/* end of script */
```

次のスクリプトは、リモート Replication Server で定義されている要求ファンクション複写定義用の非マテリアライゼーション・メソッドを使用して、中央 Replication Server にサブスクリプションを作成します。

```
-- Execute this script at Tokoyo Replication Server
-- Creates subscription using no-materialization
for upd_publishers_pubs2_req
create subscription upd_publishers_pubs2_req_sub
for upd_publishers_pubs2_req
with replicate at TOKYO_DS.pubs2
without materialization
go
/* end of script */
```

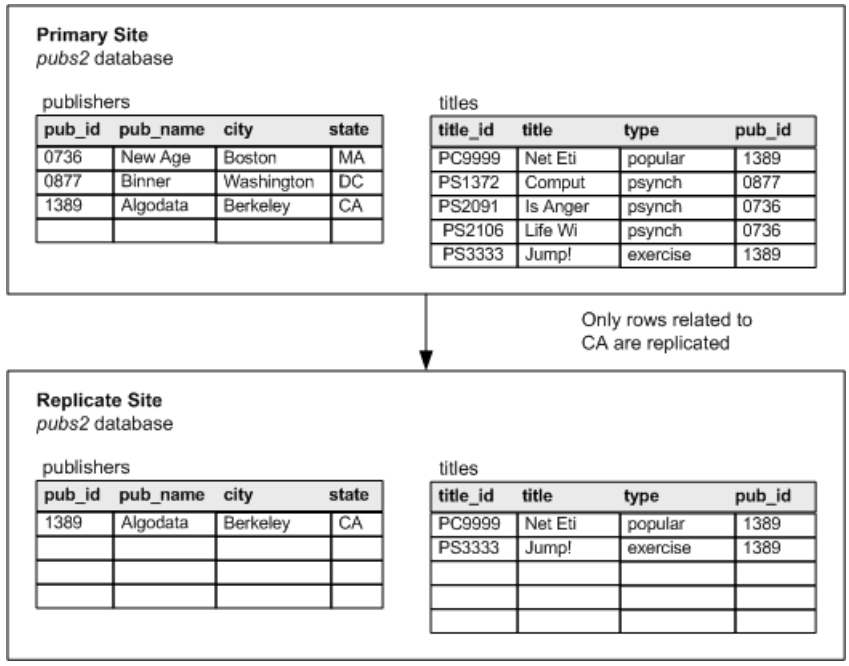
マスタ／ディテール関係の実装

適用ファンクションを使用すると、選択したデータだけをリモート・サイトに複写して、ネットワーク・トラフィックを軽減させることができます。

マスタ／ディテール関係を実装するには、マスタ・テーブルとディテール・テーブルに対する選択的なサブスクリプションをサポートするために適用ファンクションを使用します。この例では、次のようになります。

- `publishers` テーブルと `titles` テーブルは、プライマリ・サイトとレプリケート・サイトの `pubs2` データベースに存在します。
- NY_DS はプライマリ・サイトのデータ・サーバで、SF_DS はレプリケート・サイトのデータ・サーバです。

図 22 : マスタ／ディテール関係で使用されるサンプル・テーブル



プライマリ・サイトにはすべてのレコードが含まれますが、レプリケート・サイトが必要とするのはカリフォルニア (CA) 州に関連するレコードだけです。`publishers` と `titles` レコードだけが、`state` カラムを基に複写される必要があります。ただし、`state` カラムを含んでいるのは `publishers` テーブルだけです。

`titles` テーブルに `state` カラムを追加すると、システムに冗長性が加わります。2 つ目のさらに効率的なソリューションは、マスタ・テーブルとディテール・テーブルに対する更新をストアド・プロシージャによって結び付けてから、適用ファンクションを使用してストアド・プロシージャを複写するものです。選択的なサブスクリプションを管理するためのロジックは、ストアド・プロシージャ内に含まれます。

たとえば、プライマリ・サイトで、出版社の州が NY から CA に変更された場合は、その出版社のレコードはレプリケート・サイトに挿入される必要があります。サブスクリプション内のローを変化させる更新の結果としてレプリケート・ローを挿入または削除することを、「サブスクリプション・マイグレーション」と呼びます。

正しくサブスクリプション・マイグレーションを行うには、実際に更新を実行するストアド・プロシージャを制御する「上位レベル」のストアド・プロシージャのセットに対するサブスクリプションが必要です。それは、プライマリ・サイト

からレプリケート・サイトに複写される上位レベルのストアド・プロシージャに対する呼び出しです。

`state` カラムの変更を処理するには、新しい州か以前の州のどちらかが CA である場合に、レプリケート・サイトが更新に対してサブスクリプションを作成しなければなりません。

レプリケート Replication Server で選択的置換を有効にするには、プライマリ・サイトとレプリケート・サイトで次の手順を実行します。

プライマリ・サイトとレプリケート・サイトでの作業

- `publishers` テーブルと `titles` テーブルにレコードを挿入するストアド・プロシージャと、両方の挿入プロシージャの実行を制御する上位レベルのストアド・プロシージャを作成する。
- `publishers` テーブルと `titles` テーブルからレコードを削除するストアド・プロシージャと、両方の削除プロシージャの実行を制御する上位レベルのストアド・プロシージャを作成する。
- `publishers` テーブルと `titles` テーブル内のレコードを更新するストアド・プロシージャと、両方の更新プロシージャの実行を制御する上位レベルのストアド・プロシージャを作成する。
- すべての上位レベル・ストアド・プロシージャに、適切なパーミッションを付与する。

プライマリ・サイトでの作業

- **sp_setrepproc** を使用して、各上位レベル・ストアド・プロシージャを複写するようマーク付けする。
- 各上位レベル・ストアド・プロシージャに対するファンクション複写定義を作成する。

レプリケート・サイトでの作業
ファンクション複写定義に対するサブスクリプションを作成する。

insert 句を持つストアド・プロシージャ
サンプル・スクリプトを使用して、**ins_publishers** と **ins_titles** 挿入ストアド・プロシージャおよび上位レベル・ストアド・プロシージャである **ins_pub_title** を作成します。

挿入プロシージャはプライマリ・サイトとレプリケート・サイトで同じものです。挿入プロシージャを制御する上位レベル・ストアド・プロシージャと挿入プロシージャは、次のロジックに従います。

- 出版社レコードは、タイトル ID がないときにだけ挿入される。
- タイトル・レコードは、出版社レコードが存在するときにだけ挿入される。

```
-- Execute this script at NY and SF data servers
-- Creates stored procedure
 create procedure ins_publishers
 (@pub_id char(4), @pub_name varchar(40)=null,
 city varchar(20)=null, @state char(2)=null)
 as
     insert publishers values (@pub_id,
     @pub_name, @city, @state)
 /* end of script */
```

```
-- Execute this script at NY and SF data servers
 -- Creates stored procedure
 create procedure ins_titles
 (@title_id tid, @title varchar(80), @type char(12),
 @pub_id char(4)=null, @price money=null, @advance money=null,
 @total_sales int=null, @notes varchar(200)=null,
 @pubdate datetime, @contract bit)
 as
 if not exists (select pub_id from publishers
         where pub_id=@pub_id)
     raiserror 20001 "** FATAL ERROR: Invalid publishers id **"
 else
     insert titles values (@title_id, @title, @type, @pub_id,
     @price,    @advance, @total_sales, @notes, @pubdate, @contract)
 /* end of script */
```

```
-- Execute this script at NY and SF data servers
 -- Creates stored procedure
 create procedure ins_pub_title
 (@pub_id char(4), @pub_name varchar(40)=null,
 @city varchar(20)=null, @state char(2),
 @title_id tid=null, @title varchar(80)=null, @type char(12)=null,
 @price money=null, @advance money=null,
 @total_sales int=null, @notes varchar(200)=null,
 @pubdate datetime=null, @contract bit)
 as
 begin
     if @pub_name != null
     exec ins_publishers @pub_id, @pub_name, @city, @state
     if @title_id != null
     exec ins_titles @title_id, @title, @type, @pub_id, @price,
     @advance, @total_sales, @notes, @pubdate, @contract
 end/*
 end of script */
```

delete 句を持つストアド・プロシージャ

サンプル・スクリプトを使用して、**del_publishers** と **del_titles** ストアド・プロシージャおよび上位レベル・ストアド・プロシージャ **del_pub_title** を作成します。

削除プロシージャは、プライマリ・サイトとレプリケート・サイトで同じものです。削除プロシージャを制御する上位レベル・ストアド・プロシージャと削除プロシージャは、次のロジックに従います。

- レコードが削除されると、依存するすべての子レコードも削除される。

- 出版社レコードは、タイトル・レコードが存在する場合は削除されない。

```
-- Execute this script at NY and SF data servers
 -- Creates stored procedure
 create procedure del_publishers
 (@pub_id char(4))
 as
 begin
 if exists (select * from titles where pub_id=@pub_id)
     raiserror 20005 "**FATAL ERROR: Existing titles**"
 else
     delete from publishers where pub_id=@pub_id
 end
 /* end of script */
```

```
-- Execute this script at NY and SF data servers
 -- Creates stored procedure /
 create procedure del_titles
 (@title_id tid, @pub_id char(4)=null)
 as
 if @pub_id=null
     delete from titles where title_id=@title_id
 else
     delete from titles where pub_id=@pub_id
 end
 /* end of script */
```

```
-- Execute this script at NY and SF data servers
 -- Creates stored procedure
 create procedure del_pub_title
 (@pub_id char(4), @state char(2), @title_id tid=null)
 as
 begin
     if @title_id != null
         begin
             exec del_titles @title_id
             return
         end
     if @pub_id != null
         begin
             exec del_titles @title_id, @pub_id
             exec del_publishers @pub_id
         end
 end
 /* end of script */
```

update 句を持つストアド・プロシージャ

サンプル・スクリプトを使用して、**upd_publishers** と **upd_titles** ストアド・プロシージャ、および **upd_publishers** と **upd_titles** の実行を制御する上位レベル・ストアド・プロシージャ **upd_pub_title** を作成します。更新プロシージャは、プライマリ・サイトとレプリケート・サイトで異なります。

プライマリ・サイトでの作業

更新プロシージャは、次のロジックに従います。

- 不定の pub_id に関してエラーを生成する。
- タイトルが存在しない場合は挿入する。

upd_pub_title ストアド・プロシージャには、マイグレートするローに対してサブスクリプションを作成するための複写を可能にする追加カラム old_state があります。

```
-- Execute this script at NY data servers
 -- Creates stored procedure
 create procedure upd_publishers
 (@pub_id char(4), @pub_name varchar(40),
 @city varchar(20), @state char(2))
 as
 if not exists
     (select * from publishers where pub_id=@pub_id)
     raiserror 20005 "**FATAL ERROR: Unknown publishers id**"
 else
     update publishers set
     pub_name=@pub_name,
     city=@city,
     state=@state
     where pub_id = @pub_id
 end
 /* end of script */
```

```
-- Execute this script at NY data servers
 -- Creates stored procedure
 create procedure upd_titles
 (@title_id tid, @title varchar(80), @type char(12),
 @pub_id char(4)=null, @price money=null, @advance money=null,
 @total_sales int=null, @notes varchar(200)=null,
 @pubdate datetime, @contract bit)
 as
 if not exists
     (select * from titles where title_id=@title_id)
     raiserror 20005 "**FATAL ERROR: Unknown title id**"
 else
     update titles set
     title=@title,
     type=@type,
     pub_id=@pub_id,
     price=@price,
     advance=@advance,
     total_sales=@total_sales,
     notes=@notes,
     pubdate=@pubdate,
     contract=@contract
     where title_id = @title_id
 end
 /* end of script */
```

```
-- Execute this script at NY data server
 -- Creates stored procedure
 create procedure upd_pub_title
 (@pub_id char(4), @pub_name varchar(40)=null,
 @city varchar(20)=null, @state char(2)=null,
```

```
@title_id tid=null, @title varchar(80)=null, @type char(12)=null,
@price money=null, @advance money=null,
@total_sales int=null, @notes varchar(200)=null,
@pubdate datetime=null, @contract bit, @old_state char(2))
as
begin
if not exists (select * from publishers where pub_id=@pub_id)
    raiserror 20005 "**FATAL ERROR: Unknown publishers id**"
else
    exec upd_publishers @pub_id, @pub_name, @city, @state
if @title_id != null
begin
    if not exists
        (select * from titles where title_id=@title_id)
        exec ins_titles @title_id, @title, @type, @pub_id,
            @price, @advance, @total_sales, @notes, @pubdate,
            @contract
    else
    exec upd_titles @title_id, @title, @type, @pub_id, @price,
        @advance, @total_sales, @notes, @pubdate, @contract
end
end
/* end of script */
```

レプリケート・サイトでの作業
更新プロシージャは、次のロジックに従います。

- 不定の pub_id に関してエラーを生成する。
- タイトルが存在しない場合は挿入する。
- 次のテーブルに示すように正しい更新マイグレーションを実装する。

表 2 : レプリケート・サイト (CA) のマイグレーションの方式

以前の州	新しい州	必要な更新処理
CA	CA	publishers テーブルと titles テーブルを通常どおり更新する。
CA	NY	出版社を削除して、出版社に関連するすべてのタイトルを順番に削除する。
NY	CA	新しい出版社とタイトルがあれば挿入する。

次のスクリプトは、**upd_publishers** と **upd_titles** ストアド・プロシージャ、および **upd_publishers** と **upd_titles** の実行を制御する管理用ストアド・プロシージャ **upd_pub_title** を作成します。

```
-- Execute this script at SF data servers
 -- Creates stored procedure
 create procedure upd_publishers
 (@pub_id char(4), @pub_name varchar(40),
```

```
@city varchar(20), @state char(2))
as
if not exists
    (select * from publishers where pub_id=@pub_id)
    raiserror 20005 "**FATAL ERROR: Unknown publishers id**"
else
    update publishers set
    pub_name=@pub_name,
    city=@city,
    state=@state
    where pub_id = @pub_id
end
/* end of script */
```

```
-- Execute this script at SF data servers
 -- Creates stored procedure
create procedure upd_titles
(@title_id tid, @title varchar(80), @type char(12),
@pub_id char(4)=null, @price money=null, @advance money=null,
@total_sales int=null, @notes varchar(200)=null,
@pubdate datetime, @contract bit)
as
if not exists
    (select * from titles where title_id=@title_id)
    exec ins_titles @title_id, @title, @type, @pub_id,
        @price, @advance, @total_sales, @notes, @pubdate,
        @contract
else
    update titles set
    title=@title,
    type=@type,
    pub_id=@pub_id,
    price=@price,
    advance=@advance,
    total_sales=@total_sales,
    notes=@notes,
    pubdate=@pubdate,
    contract=@contract
    where title_id = @title_id
end
/* end of script */
```

```
-- Execute this script at SF data servers
 -- Creates stored procedure
create procedure upd_pub_title
(@pub_id char(4), @pub_name varchar(40)=null,
@city varchar(20)=null, @state char(2),
@title_id tid=null, @title varchar(80)=null, @type char(12)=null,
@price money=null, @advance money=null,
@total_sales int=null, @notes varchar(200)=null,
@pubdate datetime=null,@contract bit, @old_state char(2))
as
    declare @rep_state char (2)
begin
    select @rep_state=state from publishers
    where pub_id=@pub_id
```

```
    if @old_state = @state
    begin
        exec upd_publishers @pub_id, @pub_name,@city, @state
        if @title_id != null
            exec upd_titles @title_id, @title, @type,
                @pub_id, @price,@advance, @total_sales,
                @notes, @pubdate, @contract
    end
    else if @rep_state = @old_state
        begin
            exec del_titles @title_id, @pub_id
            exec del_publishers @pub_id
        end
    else if @rep_state = null
        begin
            exec ins_publishers @pub_id, @pub_name, @city,
                @state
            if @title_id != null
                exec ins_titles @title_id, @title, @type,
                    @pub_id,@price, @advance, @total_sales,
                    @notes, @pubdate,@contract
        end
end
/* end of script */
```

ファンクション複写定義

サンプル・スクリプトを使用して、ins_pub_title、del_pub_title、upd_pub_title に対する適用ファンクション複写定義を、プライマリ Replication Server に作成します。

挿入と削除の場合は state だけがサーチャブル・カラムであり、更新の場合は old_state もサーチャブル・カラムです。

```
-- Execute this script at NY data servers
-- Creates replication definition ins_pub_title
create applied function replication definition ins_pub_title
with primary at MIAMI_DS.pubs2
(@pub_id char(4),
 @pub_name varchar(40),
 @city varchar(20),
 @state char(2),
 @title_id varchar(6),
 @title varchar(80),
 @type char(12),
 @price money,
 @advance money,
 @total_sales int,
 @notes varchar(200),
 @pubdate datetime,@contract bit)
searchable parameters (@state)
go
/* end of script */
```

```
-- Execute this script at NY data servers
-- Creates replication definition upd_pub_title
create applied function replication definition upd_pub_title
with primary at MIAMI_DS.pubs2
(@pub_id char(4),
 @pub_name varchar(40),
 @city varchar(20),
 @state char(2),
 @title_id varchar(6),
 @title varchar(80),
 @type char(12),
 @price money,
 @advance money,
 @total_sales int,
 @notes varchar(200),
 @pubdate datetime,
 @contract bit,
 @old_state char(2))
searchable parameters (@state, @old_state)
go
/* end of script */
```

サブスクリプション

非マテリアライゼーション・メソッドを使用して、レプリケート Replication Server にサブスクリプションを作成します。この方法は、レプリケート・サイトにデータをロードする必要がないときに使用します。

サブスクリプション・マイグレーションを正しく実行するには、`upd_pub_title` 用に 2 つのサブスクリプションを作成する必要があります。

```
-- Execute this script at SF data servers
 -- Creates subscription for del_pub_title, ins_pub_title,
        and upd_pub_title
 create subscription del_pub_title_sub
 for del_pub_title
 with replicate at SF_DS.pubs2
 where @state='CA'
 without materialization
 go
```

```
create subscription ins_pub_title_sub
 for ins_pub_title
 with replicate at SF_DS.pubs2
 where @state='CA'
 without materialization
 go
```

```
create subscription upd_pub_title_sub1
 for upd_pub_title
 with replicate at SF_DS.pubs2
 where @state='CA'
 without materialization
 go
```

```
create subscription upd_pub_title_sub2
 for upd_pub_title
 with replicate at SF_DS.pubs2
 where @old_state ='CA'
 without materialization
 go
 /* end of script */
```

バックアップおよびリカバリの計画

Replication Server には、システム・コンポーネントで障害が発生した場合に、プライマリ・サイトとレプリケート・サイトの整合性を回復するさまざまなツールとメソッドが含まれています。

データ・ロスに対する保護

Replication Server は、さまざまなハードウェア・コンポーネントとソフトウェア・コンポーネントから構成される分散データベース・システムで動作します。このようなコンポーネントには、Adaptive Server、異機種データ・サーバ、Replication Agent、LAN、WAN、クライアント・アプリケーション・プログラムなどがあります。

Replication Server を含め、これらのコンポーネントは、障害を発生させる可能性を持っています。Replication Server は、このことを考慮して設計された、フォールト・トレラントなシステムを採用しています。障害が発生しても、ほとんどの場合、修復されるのを待つだけで処理を続行できます。障害が発生したために Replication Server または Replication Agent を再起動しなければならない場合でも、起動プロセス中にデータが消失したり重複することなく、複写がレジュームされます。

複写システムにおいても、データ・ロスに対する保護対策の重要性は集中型データベース・システムの場合と変わりません。どちらの場合も、データの正規のバージョンは 1 つだけです。そのデータを保護するために、徹底的な対策検討と十分なリソースが必要です。

複写システムでは、プライマリ・データを保護しておけば、どのような事態が起こってもレプリケート・データを必ず回復できます。レプリケート・データが失われても、そのデータのサブスクリプションを再作成するだけで復元できます。

すでに複写されているトランザクションがプライマリ・サイトで失われると (プライマリ・データベースが前のダンプにロールバックされた場合など)、プライマリ・データと複写データの一貫性が損なわれることがあります。

複写システムのバックアップとリカバリには、予防処置と修復処置があります。

予防処置

- ウォーム・スタンバイ
- ハードウェア・データのミラーリング (ホット・スタンバイ)
- より長いセーブ・インターバル

- コーディネート・ダンプ

リカバリ処置

- サブスクリプションの初期化
- サブスクリプション調整ユーティリティ (**rs_subcmp**)
- データベース・リカバリ
- コーディネート・ダンプのリストア
- データベース再同期化

予防処置

Replication Server には、複写システムのデータを保護するためのさまざまな予防処置が提供されています。

スタンバイ・アプリケーション

プライマリ・データの別々 (スタンバイ) のコピーを管理することによって、複写システムのデータを保護します。

これには次の 2 つの方法があります。

- ウォーム・スタンバイ・アプリケーション – データベースとそのバックアップ・コピーとして機能する同じベンダのデータベースで構成される 1 組のデータベースです。クライアント・アプリケーションはアクティブ・データベースを更新します。また、Replication Server はサポートされているオペレーションをアクティブ・データベースにコピーすることによって、スタンバイ・データベースを管理します。
- ハードウェア・データのミラーリング — ホット・スタンバイ・アプリケーションの一形式です。追加のハードウェアを使用する場合でも、データ・ミラーリングは、プライマリ・データについてのすべてのオペレーションを再生成することによって、プライマリ・データの正確なコピーを管理します。

参照：

- セーブ・インターバル (93 ページ)
- コーディネート・ダンプ (93 ページ)

方法の比較

ホット・スタンバイ・アプリケーションとウォーム・スタンバイ・アプリケーションの違いを考慮してください。

ホット・スタンバイ・アプリケーションでは、スタンバイ・データベースをサービス内に置くことができますが、その場合にクライアント・アプリケーションに対する割り込みはなく、トランザクションが失われることもありません。ホッ

ト・スタンバイ・データベースは、アクティブ・データベースでコミットされたトランザクションがスタンバイ・データベースでもコミットされることを保証します。両方のデータベースが起動する場合、アクティブ・データベースとスタンバイ・データベースは同期していて、ホット・スタンバイ・データベースはすぐに使用できるように準備されています。

また、Replication Server が管理するウォーム・スタンバイ・アプリケーションの場合は、次のようになります。

- データ・ミラーリング・アプリケーションを使用できない環境で、特に、必要なハードウェアを使用できないときに使用できます。
- スタンバイ・データベースが稼働していないときでも、コミットされたトランザクションをアクティブ・データベースに格納できるので、いくつかのホット・スタンバイ・アプリケーションよりも一時的なネットワーク障害に対して強くなっています。
- アクティブ・データベースは、データベースが同期していることを確認する必要がないので、アクティブ・データベースでのオーバヘッドを最小限に抑えます。

しかし、同じく Replication Server が管理するウォーム・スタンバイ・アプリケーションの場合は、次のようになります。

- スタンバイ・データベースに切り替えるときに、何らかのクライアント・アプリケーションの割り込みを要求します。
- アクティブ・データベースでコミットされた最も最近行われたトランザクションを、スタンバイ・データベースで実行していないことがあります。

ウォーム・スタンバイ

ウォーム・スタンバイ・アプリケーションは、データベースとそのバックアップ・コピーとして機能するデータベースで構成される 1 組のデータベースです。クライアント・アプリケーションは「アクティブ・データベース」を更新し、Replication Server はアクティブ・データベースのコピーとして「スタンバイ・データベース」を管理します。

Adaptive Server データベース用の Replication Server のウォーム・スタンバイ・アプリケーションについては、『Replication Server 管理ガイド 第 2 巻』の「ウォーム・スタンバイ・アプリケーションの管理」を参照してください。

注意： Replication Server バージョン 12.0 以降では、Adaptive Server Enterprise バージョン 12.0 の Sybase フェールオーバ機能をサポートしています。フェールオーバ・サポートはウォーム・スタンバイの代わりにはなりません。ウォーム・スタンバイ・アプリケーションでは、データベースのコピーを保持しますが、フェールオーバ・サポートでは、異なるマシンから同じデータベースにアクセスします。Replication Server からウォーム・スタンバイ・データベースへのコネクションは同じ働きをします。

Replication Server におけるフェールオーバ・サポートの動作の詳細については、『Replication Server 管理ガイド 第 2 巻』の「Sybase フェールオーバをサポートする

ための複写システムの設定」および『Replication Server 管理ガイド 第 2 巻』の「付録 D Sun Cluster 2.2 の 高可用性」を参照してください。

ハードウェア・データのミラーリング

データの高可用性を確実にするには、複写システムで最も重要なデータをミラーリングすることです。ミラーリングを行うと、I/O 操作を二重化できるので、同じデータのコピーを 2 つ維持することになります。

現在動作しているメディアが故障した場合、すぐにスタンバイがオンラインになります。ミラーリングによって、トランザクションが失われる危険性はほとんどなくなります。

次に、ミラーリングの対象を効果の大きな順に説明します。

1. プライマリ・データベースのトランザクション・ログ
   トランザクション・ログには、テープにダンプされていないトランザクションが格納されます。プライマリ・トランザクション・ログが失われた場合には、トランザクションを再発行しなければなりません。
2. プライマリ・データベース
   データベースは、前回のデータベース・ダンプとそれ以降のトランザクション・ダンプを再ロードすればリカバリできます。ただし、プライマリ・データが格納されたデータベースをリカバリする場合には、システム全体に複写されているデータもリカバリまたは初期化しなければなりません。システムの広範囲にわたるダウン時間は、OLTP システムにとっては致命的となります。プライマリ・データをミラーリングすることで、このような緊急事態の発生を防止できます。
3. Replication Server のステーブル・キュー
   Replication Server は、ステーブル・キューと呼ばれる蓄積転送ディスク・キューにトランザクションを格納します。このキューは、**create partition** コマンドで Replication Server に割り当てられたディスク・パーティションに割り付けられます。

注意： **create partition** は Replication Server でパーティションを使用できるようにするためのもので、既存の **add partition** コマンドに取って替わるものです。**add partition** は、下位互換性を確保するために引き続きサポートされます。この 2 つのコマンドの構文と使用法は同じである。『Replication Server リファレンス・マニュアル』の「Replication Server コマンド」の「**create partition**」を参照してください。

ステーブル・キューに格納されたデータは、プライマリ・データベースのトランザクション・ログにも存在するので、予備のデータともいえます。ただし、ステーブル・キューが失われると、Replication Server はトランザクションをレプリケート・サイトに送信できなくなります。その結果、レプリケート・サイ

トのサブスクリプションを再度初期化する必要があります。ディスク・パーティションをミラーリングすることで、ステーブル・キューを保護し、レプリケート・データベースのダウン時間を最小にとどめることができます。

4. Replication Server システム・データベース (RSSD)

   最後に行われたバックアップ以降に複写定義、サブスクリプション、ルート、またはファンクションやエラー・クラスなどのデータが修正されている場合、RSSD の障害からのリカバリは複雑な処理になる可能性があります。

   『Replication Server 管理ガイド 第 2 巻』の「複写システム・リカバリ」を参照してください。

   RSSD をミラーリングすることによって、システム・データのロスを防ぐことができ、複雑なリカバリ処理を行う必要がなくなります。RSSD をミラーリングしない場合は、システム・データを変更する RCL オペレーションを行ったら必ず RSSD をバックアップしてください。

セーブ・インターバル

Replication Server が送信先に配信した後にステーブル・キュー・メッセージを一定時間保管するように、ある Replication Server から別の Replication Server へのルートや、Replication Server からデータベースへのコネクションを設定することができます。このようなステーブル・キューの保管期間をセーブ・インターバルといいます。

セーブ・インターバルを設定することで、メッセージのバックログを送信元に作成できます。パーティションで障害が発生しても、セーブ・インターバルの間に修復されれば、複写システムは障害の影響を受けません。送信先のステーブル・キューで障害が発生した場合には、ステーブル・キューを再構築し、送信元の Replication Server がバックログされたメッセージを再送します。

詳細については、『Replication Server 管理ガイド 第 2 巻』を参照してください。

コーディネート・ダンプ

バックアップをリストアしてデータベースをリカバリする場合、他のサイトにある影響を受けるデータベース内の複写データもプライマリ・データと一貫性があるようにする必要があります。Replication Server では、分散システム内のすべてのサイトでデータベース・ダンプとトランザクション・ダンプを連動させることができます。

データベース・ダンプやトランザクション・ダンプは、プライマリ・データベースから開始されます。Replication Agent は、ログからダンプ・レコードを取り出した後、Replication Server に渡し、ダンプ要求がレプリケート・サイトに配信されるようにします。これによって、すべてのデータを特定の一貫性ポイントまでリストアすることができます。

コーディネート・ダンプは、プライマリ・データと複写データのどちらか一方だけが格納されたデータベースだけで実行できます。また、コーディネート・ダンプはプライマリ・データベースから開始されます。

コーディネート・ダンプの作成方法については、『Replication Server 管理ガイド第 2 巻』を参照してください。

参照：

- スタンバイ・アプリケーション (90 ページ)
- セーブ・インターバル (93 ページ)

リカバリ処置

Replication Server では、複写システムのコンポーネント障害によって失われたデータをリカバリするさまざまなオプションが提供されています。

サブスクリプションの再作成

プライマリ・バージョンのデータのすべての矛盾を解決します。リモート・サイトで障害からリカバリする方法の 1 つに、サブスクリプションの再作成があります。

サブスクリプションを再作成する方法は、小さな複写テーブルにとって最も適しています。大きなサブスクリプションやプライマリ・データの障害には、より堅牢なリカバリ方法が必要です。

サブスクリプション調整ユーティリティ (rs_subcmp)

rs_subcmp ユーティリティは、レプリケート・テーブルに失われたロー、孤立したロー、または矛盾したローがないかどうかをチェックし、不整合を解決します。

あまり重要でない不整合を解決する場合には、コーディネート・ロードなどの混乱をともなうリカバリ処理を行うよりも、**rs_subcmp** を使用する方が妥当です。『Replication Server リファレンス・マニュアル』の「**rs_subcmp**」を参照してください。

データベース・リカバリ

プライマリ・データベースで障害が発生した場合には、データベースとトランザクション・ログが損傷していなければ、コミットされたトランザクションをすべてリカバリできます。データベースまたはトランザクション・ログが損傷していて、自動リカバリができないときは、データベース・ダンプとトランザクション・ダンプをロードしてデータベースを既知の状態に復元し、最後のトランザクション・ダンプ後に実行されたトランザクションを再送信する必要があります。

Replication Server と Replication Agent をリカバリ・モードで実行すると、再ロードしたダンプからトランザクションを再実行できます。これによってプライマリ・データベース内のすべてのトランザクションを複写し、トランザクションの重複を防ぐことができます。ダンプからのプライマリ・データベースのリカバリ方法の詳細については、『Replication Server 管理ガイド 第 2 巻』を参照してください。

コーディネート・ダンプのリストア

コーディネート・ダンプによって作成されたデータベース・ダンプまたはトランザクション・ダンプをリストアすると、プライマリ・データと複写データが最新の整合状態に戻ります。

『Replication Server 管理ガイド 第 2 巻』を参照してください。

データベース再同期化

データベース再同期化を使用すると、データや整合性を損なうことなく、またプライマリ・データベースのクワイスを強制しないで、レプリケート・データベースをマテリアライズして複写をレジュームできます。

データベース再同期化は、信頼されたソースから取得したデータ・ダンプを同期先のデータベースに適用することをベースとしています。

Adaptive Server のデータベース再同期のコマンド、パラメータ、プロシージャおよびシナリオの説明については、『Replication Server 管理ガイド 第 2 巻』の「レプリケート・データベースを再同期する」を参照してください。Oracle データベースを再同期するには、『Replication Server 異機種間複写ガイド』の「Oracle レプリケート・データベースの再同期」および Replication Agent のマニュアルを参照してください。

Replication Agent の概要

Replication Agent は、プライマリ・データベースのトランザクション・ログから情報を取得し、プライマリ Replication Server に対応するようにフォーマットする Replication Server クライアントです。RepAgent は、Adaptive Server 用の Replication Agent コンポーネントです。

Replication Agent は、プライマリ・データに対する変更だけ検出し、それ以外のデータに対する変更を無視します。Replication Agent は RCL (Replication Control Language) のサブセットであるログ転送言語 (LTL: Log Transfer Language) を使用して、プライマリ・データに対する変更をプライマリ Replication Server に送信します。プライマリ Replication Server はその情報をレプリケート・データベースに分配します。

Replication Agent のコネクション・ステータスは、Replication Server の Replication Agent スレッドのステータスおよびプライマリ・データベースからデータを抽出している Replication Agent プロセスのステータスから派生します。Replication Agent スレッドまたは Replication Agent プロセスのいずれかが実行を停止すると、RMS は Suspended のステータスを返します。コンポーネントのいずれかが稼働していない場合、RMS はその説明も返します。

Sybase では、IBM DB2 Universal Database、Microsoft SQL Server、Oracle など、ASE 以外のその他のデータ・サーバ用の Replication Agent コンポーネントを提供しています。Replication Server でアクティブにサポートされているデータベースについては、『Replication Server 異機種間複写ガイド』と Replication Server Options のマニュアルを参照してください。

Replication Agent for DB2 は、OS/390 ベースの DB2 Universal Database 用の Replication Agent コンポーネントです。Sybase Replication Agent は、DB2 Universal Database (UNIX および Windows プラットフォーム)、Microsoft SQL Server、および Oracle 用の Replication Agent コンポーネントです。

RepAgent は 1 つの Adaptive Server スレッドです。Replication Agent for DB2 は、OS/390 ホスト上に存在する独立したプロセスです。Sybase Replication Agent は、UNIX または Windows ホスト上に常駐する独立したアプリケーションです。

Replication Agent のパフォーマンス

1. Replication Server にログインします。
2. **connect source** コマンドを送信してセッションをログ転送元として特定し、トランザクション情報が転送されるデータベースを指定します。

3. Replication Server からデータベースのメンテナンス・ユーザ名を取得します。**send_maint_xacts_to_replicate** 設定パラメータまたは **send_warm_standby_xacts** 設定パラメータが **true** に設定されていないかぎり、RepAgent はメンテナンス・ユーザが実行する操作をフィルタします。
4. Replication Server にデータベースのセカンダリ・トランケーション・ポイントを要求します。Replication Server は、「*オリジン・キュー ID*」と呼ばれる値を返します。RepAgent は、この値を使用してトランザクション・ログ内でトランザクション処理の転送を開始する位置を見つけます。Replication Server は、この位置までの処理をすでに受け取っています。
5. トランザクション・ログから、レコードを (セカンダリ・トランケーション・ポイントに続くレコードを起点として) 取得し、情報を LTL コマンドにフォーマットします。

Replication Agent トランザクション・ログ

一部の Replication Agent は、プライマリ・データベースのネイティブ・トランザクション・ログにアクセスして、トランザクションの複写に必要な情報を取得できないことがあります。

ネイティブ・トランザクション・ログが使用できない場合、Replication Agent は、専用のトランザクション・ログを使用して、複写に必要なプライマリ・データベース内のトランザクションを収集したり記録したりします。このような Replication Agent は、プライマリ・データベースで実行される SQL スクリプトを生成して、Replication Agent のトランザクション・ログを作成します。

ネイティブ・トランザクション・ログは、RepAgent スレッド (Adaptive Server 対応)、Replication Agent for DB2、Replication Agent for DB2 Universal Database (UNIX および Windows プラットフォーム上) によって使用されます。Sybase Replication Agent (Microsoft SQL Server、Oracle 対応) では、プライマリ・データベースで作成、維持されているトランザクション・ログを使用します。Sybase Replication Agent トランザクション・ログは、プライマリ・データベースのデータベース・オブジェクト (トリガ、テーブル、プロシージャ) から構成されています。

『Replication Agent 管理ガイド第 1 巻』を参照してください。

Replication Agent 製品

Replication Agent 製品により、Sybase 以外のデータベース・サーバを Sybase 複写システムのプライマリ・データベース・サーバとして使用できるようになるため、Replication Server の機能を拡張することができます。

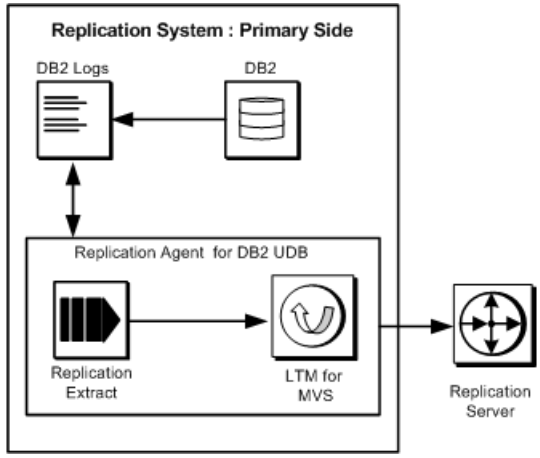
Sybase では、Sybase 以外のデータベースのために次の Replication Agent 製品を用意しています。

- Replication Agent for DB2
- Sybase Replication Agent

Replication Agent for DB2

Replication Agent for DB2 は、OS/390 メインフレーム・プラットフォーム上の DB2 プライマリ・データベース内のデータベース・トランザクションを収集して、Replication Server に送信します。

図 23 : Replication Agent for DB2 でのデータの流れ



Replication Agent for DB2 は、複写システムで次のように機能します。

- プライマリ・データベースは DB2 で、OS/390 においてサブシステムとして実行される。トランザクション・ログは DB2 ログ。
- Replication Agent for DB2 は、Replication Extract というログ抽出機能を提供する。これは、DB2 ログを読み込んで、複写するようにマーク付けされているテーブルに関連するアクティブな DB2 ログ・エントリとアーカイブ・ログ・エントリを取得します。

- 複写するようマーク付けされたデータを LTM for MVS が Replication Extract から受け取り、TCP/IP 通信プロトコルを使用して Replication Server に転送する。
- その後、Replication Server は、レプリケート・データベースに変更を適用する。

DB2 トランザクション・ログ

DB2 データベース・サーバは、DB2 テーブル内のローが変更されると、その変更をログに記録します。トランザクション・ログに書き込まれる情報には、変更前後のデータのコピーが含まれます。

DB2 では、これらのレコードは undo レコードおよび redo レコードとして使用されます。制御レコードは、コミットおよびアボート用に書き込まれます。これらのレコードは、コミットおよびロールバックに変換されます。

DB2 のログは、一連のデータ・セットで構成されます。Replication Extract は、これらのログ・データ・セットを使用して DB2 データの変更内容を識別します。DB2 は、変更レコードが生成されるとそれらをアクティブなログに書き込むため、Replication Extract はログ・レコードが書き込まれた直後にそれらを処理できます。

Sybase Replication Agent

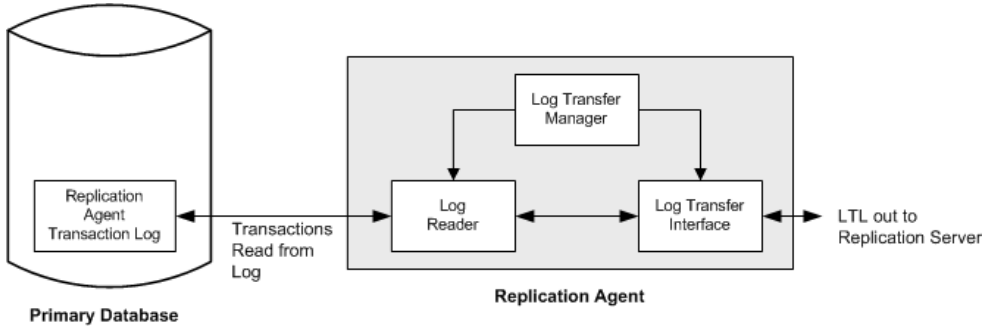
Sybase Replication Agent は、複写システムのコンポーネントで、DB2 Universal Database (UNIX および Windows プラットフォーム)、Microsoft SQL Server、または Oracle プライマリ・データベース内のトランザクションを収集し、Replication Server に転送します。

Sybase Replication Agent for DB2 Universal Database は、ネイティブ DB2 トランザクション・ログを使用して、複写するトランザクション・データを取得します。

Sybase Replication Agent (Microsoft SQL Server および Oracle 対応) は、独自のトランザクション・ログを作成して、複写するトランザクション (または、プロシージャの呼び出し) を記録します。Replication Agent の Log Reader コンポーネントは、トランザクション・ログを読み込んで、プライマリ・データベースからトランザクションを取得します。

プライマリ・データベースからトランザクション・データを取得すると、Replication Agent の Log Transfer Interface (LTI) コンポーネントは、トランザクションと「変更セット」の結果データを処理し、LTL 出力を生成します。Replication Server はこれを使用して、サブスクリプションを作成しているレプリケート・データベースにトランザクションを分配します。

図 24 : Sybase Replication Agent のデータの流れ



Sybase Replication Agent は、プライマリ Replication Server の Replication Server システム・データベース (RSSD) に格納されていた情報を使用して、最も効率的な LTL を生成するための複写トランザクションの処理方法を決定します。

Replication Agent から LTL を受信すると、プライマリ Replication Server は、直接またはレプリケート Replication Server を介して複写トランザクションをレプリケート・データベースに送信します。レプリケート Replication Server は、複写データをレプリケート・データベースのネイティブ言語に変換してから、レプリケート・データベース・サーバに送信します。レプリケート・データベースが複写トランザクションを正常に処理すると、レプリケート・データベースはプライマリ・データベースと同期されます。

Sybase Replication Agent は、プライマリ・データベース・サーバ、プライマリ Replication Server、およびその他の複写システムのコンポーネントとは独立したスタンドアロンのアプリケーションとして実行されます。

Sybase Replication Agent は、プライマリ・データベースまたは複写システムのその他のコンポーネントと同じホスト・マシン上、あるいは複写システムのコンポーネントとは別のマシン上のどちらに存在していてもかまいません。

Replication Agent の通信方法

Replication Agent は、Sybase JDBC ドライバ (jConnect™ for JDBC™) の 1 つのインスタンスを使用して、Open Client/Server アプリケーション (プライマリ Replication Server とその RSSD を含む) へのすべてのコネクションを管理します。

プライマリ・データベース・サーバの場合、Sybase Replication Agent はプライマリ・データベース用の JDBC ドライバに接続します。

トランザクションを複写する場合、Replication Agent はプライマリ・データベースとプライマリ Replication Server の両方との接続を維持します。また、Replication Agent は、時々プライマリ Replication Server の RSSD に接続して、複写定義データを取得します。

Java の実装

Sybase Replication Agent コンポーネントは、Java プログラミング言語で実装されています。このため、Sybase Replication Agent を実行するには、Replication Agent ホスト・マシンとして機能する任意のコンピュータに Java Runtime Environment (JRE) がインストールされている必要があります。

Adaptive Server 以外のデータ・サーバへのデータの複写

Sybase では、Open Server ゲートウェイ・アプリケーション製品をいくつか提供しています。この製品を使用して、レプリケート・データベース用の Sybase 以外のデータベース・サーバにアクセスできます。

ASE 以外のデータ・サーバを含む Replication Server のインタフェース

Replication Server は、データ・サーバに要求を送信することによって、データベースに格納されたレプリケート・データを更新します。Replication Server は、Adaptive Server をサポートしています。データベースを Adaptive Server 以外のデータ・サーバで管理する場合は、Replication Server 用のインタフェースを用意してください。

このインタフェースには、次のものが含まれます。

- Replication Server から命令を受け取り、それをデータ・サーバに適用する Sybase Enterprise Connect™ Data Access (ECDA) または ExpressConnect for Oracle。
- Replication Server がゲートウェイにログインするために使用できるメンテナンス・アカウント (ECDA など)。
- データベースで使用するファンクション文字列クラス。このクラスのファンクション文字列は、データ・サーバに対する命令のフォーマット方法を指定します。
- データ・サーバがゲートウェイを介して Replication Server に返すエラーを処理するためのエラー・クラスとエラー・アクション。
- 複写データを含む各データベースの `rs_lastcommit` テーブル。Replication Server は、このテーブルを使用して、データベースで正常にコミットされたトランザクションを追跡します。
- 最後のトランザクションを各送信元プライマリ・データベースから取り出す **rs_get_lastcommit** ファンクション呼び出し。

Replication Server の ASE 以外のデータ・サーバのサポート機能では、各種データベース・サーバをアクティブにサポートするための、外部データ・サーバ用インタフェースのコンポーネントを提供しています。この機能では、ファンクション文字列クラス、エラー・クラスとアクション、ユーザ定義データ型に加え、レプリケート・データベースに必要なテーブルとプロシージャを作成する接続プロファイルを用意しています。

ASE 以外のサポート機能の詳細については、『Replication Server 管理ガイド 第 1 巻』と、使用しているプラットフォームの『Replication Server 設定ガイド』を参照してください。Replication Server でアクティブにサポートされているデータベースについては、『Replication Server 異機種間複写ガイド』と Replication Server Options のマニュアルを参照してください。

Sybase データベース・ゲートウェイ製品

Sybase Enterprise Connect Data AccessUS BUG-Sybase Enterprise Connect Data AccessConnect は、Sybase のミドルウェアであり、クライアントと企業データ・ソース間のコネクティビィティを提供します。ECDA を使用すると、Sybase の複写システムに Sybase 以外のレプリケート・データベースを簡単に統合できます。

ECDA を使用すると、データベース・サーバに直接接続できます。ECDA の DirectConnect コンポーネントは Open Server ゲートウェイとして動作します。その際、DirectConnect サーバは、Replication Server が使用する Open Client/Server プロトコルを、Sybase 以外のレプリケート・データベースが使用するネイティブな通信プロトコルに置き換えて変換します。

ECDA を使用すると、以下と接続できます。

- Microsoft SQL Server
- OS/390 (UDB および DB2)
- Oracle
- ODBC アクセス可能なデータ・ソース

Replication Server Options のマニュアルを参照。

ExpressConnect for Oracle

ExpressConnect for Oracle (ECO) は、Oracle を複写するための Replication Server 15.5 以降によってロードされるライブラリです。

ECO には、次のような利点があります。

- 起動、モニタリング、または管理のために、個別のサーバ・プロセスは必要ありません。
- Replication Server と ECO は同一のプロセス内で実行されるため、これらの間で SSL は不要であり、ECDA for Oracle のグローバル設定パラメータでこれまでに扱われていた設定内容を設定するための要件もありません。
- サーバ接続は Replication Server から **create connection** コマンドと **alter connection** コマンドを使用して設定されるため、ECDA for Oracle の **connect_string** 設定で

個別に同等の設定をする必要はありません。『Replication Server リファレンス・マニュアル』を参照してください。

- **text_chunksize**、**autocommit**、**array_size** など、ECDA for Oracle と同等の特定の設定も設定する必要がありません。これらの設定は Replication Server によって (Replication Agent 入力に基づく場合もあります) 自動的に決められて ECO と通信します。

ECO には、次に示すように ECDA for Oracle と類似した機能が備えられています。

- データ型変換のセットが同じ。
- Sybase データと Oracle データ間での言語および charset の変換。ECO では、`map.cfg` ファイルを使用して設定される。
- ASE プライマリ・データベース内の空の文字列が Oracle レプリケート・データベースに複写されると、Oracle で、1 つまたは複数 (カラムが `varchar` と `fixed char width` のいずれのデータ型であるかによって異なる) の空白で構成される文字列値になる。

ExpressConnect for Oracle でロケーションの透過性を確立するには、`tnsnames.ora` ファイルのみが必要です。ECDA for Oracle のように `interfaces` ファイルは必要ではありません。コネクションの設定のためには、`tnsnames.ora` ファイルで定義されるサービス名を指定する必要があります。

『ExpressConnect for Oracle インストールおよび設定ガイド』を参照してください。

メンテナンス・ユーザ

Replication Server は、メンテナンス・ユーザとしてログインします。メンテナンス・ユーザのアカウントは、データベースの **create connection** に指定します。

ゲートウェイは、同じログイン名でも異なるログイン名でもデータ・サーバにログインできます。ログイン名は、レプリケート・データを変更するのに必要なパーミッションを持っていなければなりません。

ファンクション文字列クラス

データベースを管理する Replication Server には、ファンクション文字列クラスが必要です。Replication Server では、Adaptive Server 用のファンクション文字列を提供しています。また、Replication Server では、ASE 以外のデータ・サーバのサポート機能とともに、アクティブにサポートされているすべてのデータ・サーバ用にファンクション文字列クラスを提供しています。

Replication Server でアクティブにサポートされていない ASE 以外のデータ・サーバに複写する場合は、そのデータ・サーバ用のファンクション文字列クラスを作成してください。次のいずれかの方法を使用できます。

- システムが提供するクラスからファンクション文字列を継承するファンクション文字列クラスを作成します。
- ユーザ自身ですべてのファンクション文字列を作成します。

Replication Server は、そのファンクションに与えられたファンクション文字列にランタイム値をマップすることによって構成したコマンドをゲートウェイに送信します。ファンクション文字列がどのように書かれているかとデータ・サーバの稼働条件によって、ゲートウェイはそのコマンドを直接データ・サーバに渡すか、または何らかの方法でそのコマンドを処理してからデータ・サーバに要求を送信することができます。アクティブにサポートされているデータ・サーバについては、Replication Server Options のマニュアルを参照してください。

注意： Replication Server 15.2 以降には、アクティブにサポートされるデータベースのために、ファンクション文字列クラスが事前にロードされた接続プロファイルが含まれます。『Replication Server 管理ガイド 第 1 巻』の「接続プロファイル」を参照してください。

データベース・ゲートウェイが処理する必要のある Replication Server システム・ファンクションのリストについては、『Replication Server 管理ガイド 第 2 巻』の「データベース・オペレーションのカスタマイズ」を参照してください。

継承を使用したファンクション文字列クラスの作成

Replication Server では、ユーザはファンクション文字列の継承と呼ばれるメカニズムを使用してクラス間の関係を作成することによって、ファンクション文字列クラス間でファンクション文字列定義を共有できます。

システムが提供するクラス `rs_default_function_class` と `rs_db2_function_class` は、親クラスからファンクション文字列を継承する派生クラスの親クラスとして使用できます。派生クラスを作成することによって、データ・サーバに対していくつかのファンクション文字列をカスタマイズする一方、親クラスのその他のファンクション文字列をそのまま使用できます。

create function string class コマンドを使用して、親クラス `rs_default_class` または `rs_db2_function_class` から、親クラスを継承する派生クラスを作成します。必要な場合のみ、カスタマイズされたファンクション文字列を作成します。

注意： `rs_db2_function_class` では、`text` データや `image` データの複写ができません。DB2 データベースの `text` データや `image` データを複写できるようにするには、ゲートウェイを介し RPC メソッドを使用して **rs_writetext** ファンクション文字列をカスタマイズしてください。

ファンクション文字列の継承の詳細については、『Replication Server 管理ガイド第 2 巻』の「データベース・オペレーションのカスタマイズ」を参照してください。

独自のファンクション文字列クラスの作成

システムが提供するクラスを継承しないクラスを使用する場合は、ユーザ自身ですべてのファンクション文字列を作成して、新しいテーブルまたはファンクション複写定義を作成するときには、必ず新しいファンクション文字列を追加してください。

create function string class コマンドを使用して新しいファンクション文字列クラスを作成してから、そのクラスのスコープに入るすべてのファンクションに対してファンクション文字列を作成します。

データベース内で複写するテーブルごとにファンクション文字列 **rs_insert**、**rs_update**、および **rs_delete** を作成します。

`text` データ型または `image` データ型のカラムを複写する場合は、`text` カラムまたは `image` カラムごとにファンクション文字列 **rs_datarow_for_writetext**、**rs_get_textptr**、**rs_textptr_init**、**rs_writetext** を作成します。ファンクション文字列名には、複写定義の `text` または `image` カラム名を使用する必要があります。

rs_select および **rs_select_with_lock** ファンクション文字列は、データベースに複写定義用のプライマリ・データがある場合のみ必要になります。

エラー・クラス

エラー・クラスは、Replication Server がゲートウェイから返されたエラーを処理する方法を決定します。ゲートウェイのエラー・クラスを作成するには、**create error class** コマンドを使用します。

Replication Server API を使用して、データ・サーバのエラーに対するエラー処理を設定できます。データベースに対してエラー・クラスを作成し、データ・サーバが返す各エラーへの対応を指定できます。

注意： Replication Server 15.2 以降には、アクティブにサポートされるデータベースのために、エラー・クラスが事前にロードされた接続プロファイルが含まれます。『Replication Server 管理ガイド第 1 巻』の「接続プロファイル」および『Replication Server 管理ガイド第 2 巻』の「エラーと例外の処理」の「デフォルトの ASE 以外のためのエラー・クラス」を参照してください。

ゲートウェイから返されたエラーへの対応方法を Replication Server に指示するには、**assign action** コマンドを使用します。

表 3 : データ・サーバ・エラーに対する Replication Server のアクション

アクション	説明
ignore	コマンドが成功し、処理すべきエラーまたは警告状態がないとみなす。このアクションは、正常に実行されたことを示すリターン・ステータスに使用できる。
warn	警告メッセージを記録する。ただし、トランザクションのロールバックや実行の割り込みは行わない。
retry_log	トランザクションをロールバックし、リトライする。リトライの回数を設定するには、**configure connection** を使用する。リトライの上限を超えてもエラーが継続する場合には、トランザクションを例外ログに書き込み、次のトランザクションを実行する。
log	現在のトランザクションをロールバックし、例外ログに書き込む。次に、後続のトランザクションを実行する。
retry_stop	トランザクションをロールバックし、リトライする。リトライの回数を設定するには、**configure connection** を使用する。リトライ後もエラーが継続する場合は、データベースの複写をサスペンドする。
stop_replication	現在のトランザクションをロールバックし、データベースの複写をサスペンドする。**suspend connection** コマンドを実行した場合と同じで、これがデフォルトのアクションとなる。 このアクションはデータベースの複写処理を完全に停止させるため、データベース・コネクションを停止せずに処理できるデータ・サーバ・エラーには、他のアクションを割り当てること。

デフォルトのエラー・アクションは、**stop_replication** です。あるエラーに stop_replication 以外のアクションを割り当てなければ、そのエラーが発生したときに Replication Server はゲートウェイとのコネクションを停止します。

create error class と **assign action** の詳細については、『Replication Server リファレンス・マニュアル』を参照してください。

rs_lastcommit テーブル

rs_lastcommit テーブルの各ローは、プライマリ・データベースからデータベースに配信され、コミットされた最後のトランザクションを示します。Replication Server は、この情報によってすべてのトランザクションが配信されたことを確認します。

トランザクションがコミットされる前に、**rs_commit** ファンクション文字列を使用して `rs_lastcommit` テーブルを更新してください。これにより、Replication

Server がデータベースにトランザクションをコミットするたびに、テーブルが必ず更新されます。

Replication Server は、データベースのメンテナンス・ユーザとして、rs_lastcommit テーブルを管理します。メンテナンス・ユーザが、rs_lastcommit テーブルに必要なすべてのパーミッションを持っていることを確認してください。

表 4 : rs_lastcommit テーブルの構造

カラム名	データ型	説明
origin	int	トランザクションを開始したデータベースをユニークに識別する整数で、Replication Server によって指定される。
origin_qid	binary(36)	トランザクションのコミット・レコードのオリジン・キュー ID。
secondary_qid	binary(36)	サブスクリプション・マテリアライゼーション・プロセスで使用されるステーブル・キューのキュー ID。
origin_time	datetime	(オプションのカラム) トランザクションのオリジン時間
dest_commit_time	datetime	(オプションのカラム) トランザクションが送信先でコミットされた時間

origin カラムは、テーブルに対するユニーク・キーです。このデータベースにデータが複写されるプライマリ・データベースごとに 1 つのローがあります。

データベースでコーディネート・ダンプを使用する場合は、rs_lastcommit テーブルをファンクション文字列の **rs_dumpdb** と **rs_dumptran** で更新します。

Adaptive Server データベースの場合、ファンクション文字列 **rs_commit**、**rs_dumpdb**、および **rs_dumptran** によってストアド・プロシージャ **rs_update_lastcommit** を実行し、rs_lastcommit テーブルを更新します。rs_update_lastcommit ストアド・プロシージャのテキストは、次のとおりです。

```
/* Create a procedure to update the
 ** rs_lastcommit table. */
 create procedure rs_update_lastcommit
     @origin int,
     @origin_qid binary(36),
     @secondary_qid binary(36),
     @origin_time datetime
 as
     update rs_lastcommit
         set origin_qid = @origin_qid,
         secondary_qid = @secondary_qid,
```

```
        origin_time = @origin_time,
        commit_time = getdate()
    where origin = @origin
    if (@@rowcount = 0)
    begin
        insert rs_lastcommit (origin,
            origin_qid, secondary_qid,
            origin_time, commit_time,
            pad1, pad2, pad3, pad4,
            pad5, pad6, pad7, pad8)
        values (@origin, @origin_qid,
            @secondary_qid,@origin_time,
            getdate(), 0x00, 0x00, 0x00,
            0x00, 0x00, 0x00, 0x00, 0x00)
    end
go
```

注意： Replication Server 15.2 以降では、接続プロファイルを使用して最初にコネクションを作成すると、アクティブにサポートされている ASE 以外のデータ・サーバに対して `rs_lastcommit` テーブルが作成されます。

詳細については、『Replication Server リファレンス・マニュアル』の **rs_commit** ファンクション、**rs_dumpdb** ファンクション、**rs_dumptran** ファンクションの各項の説明を参照してください。

rs_get_lastcommit ファンクション

Replication Server は、**rs_get_lastcommit** ファンクション呼び出しをゲートウェイに送信し、最後にコミットされたトランザクションを各送信元プライマリ・データベースから取得します。

rs_get_lastcommit のファンクション文字列は、次のような簡単な **select** 文を実行します。

```
select origin, origin_qid, secondary_qid
     from rs_lastcommit
```

注意： **rs_get_lastcommit** ファンクションは、アクティブにサポートされている ASE 以外のデータ・サーバの接続プロファイルで事前に定義されています。

国際的な複写システムの設計

Replication Server と Replication Manager (RM)、複写システムの管理に使用する Sybase Central プラグインは、いずれも国際環境をサポートしています。

Replication Server および Replication Manager (RM) は、次のことを行います。

- 各国語へのメッセージのローカライゼーション
- Replication Server サイト間での文字セット変換機能を備えた、Sybase がサポートするすべての文字セットに対するサポート
- バイナリ・ソート順以外のソート順のサポート

国際環境に対応する複写システムを設計するときには、言語設定、文字セット設定、およびソート順設定がシステムに与える影響について理解しておいてください。Replication Server と RM では、言語、文字セット、ソート順について、きわめて柔軟性の高い設定ができます。

メッセージ言語

Replication Server を使用すると、いくつかの言語でメッセージをエラー・ログやクライアントに出力できます。

ただし、メッセージ言語は文字セットと互換性がなければなりません。すべての Sybase 文字セットと互換性がある英語は、デフォルトの言語です。サポートされる言語のリストについては、使用しているプラットフォームの『Replication Server 設定ガイド』を参照してください。

メッセージの言語を設定すると、Replication Server、Adaptive Server、その他のデータ・サーバを含め、複写システム内のサーバ・プログラムごとに指定された言語でメッセージがエラー・ログに出力されます。しかし、サーバによっては、クライアントへのメッセージをクライアント側で指定された言語で出力することもあります。

たとえば、Adaptive Server はクライアント (Replication Server) の言語設定をチェックし、その言語でメッセージを出力します。RepAgent、Adaptive Server スレッドもクライアントの言語でメッセージを返します。

しかし、Replication Server はクライアントの言語設定をチェックせず、それ自体の言語でメッセージを返します。このため、サーバによって言語の設定が異なっていると、エラー・ログに複数の言語メッセージが記録されます。

注意： エラー・ログに複数の言語メッセージが記録されると混乱するので、サイト内のすべてのサーバとクライアントで同じ言語を設定してください。

Replication Server メッセージ言語を変更する

Replication Server メッセージ言語を設定します。

注意： RepAgent は自動的に Replication Server の言語でメッセージを返すため、RepAgent 用に言語パラメータを設定する必要はありません。

1. Replication Server を停止します。
2. テキスト・エディタを使用して、Replication Server 設定ファイルの **RS_language** の値を変更します。
3. Replication Server を再起動します。

文字セット

Replication Server は、Sybase がサポートするすべての文字セットをサポートし、プライマリ・サイトとレプリケート・サイトの間でデータと識別子の文字セットを変換します。

しかし、文字セットの変換は互換性のある文字セットの間でしかできません。たとえば、シングルバイト文字セットのデータはマルチバイト文字セットには変換できません。文字セットの互換性の詳細については、『Adaptive Server Enterprise システム管理ガイド 第 1 巻』を参照してください。

あるサーバで選択した文字セットは、そのサーバがサポートする言語、サーバが動作するハードウェア、オペレーティング・システム、連動するシステムなどの影響を受けます。

Sybase がサポートする文字セットには、次のような特徴があります。

- Sybase がサポートする文字セットは、すべて 7 ビット ASCII 文字セットのスーパーセットである。
- いくつかの文字セットはまったく互換性がない。これらの文字セットは、7 ビット ASCII 文字を持たず、互いに共通した文字がありません。
- 互換性のある文字セットにも、共通しない文字がいくつかある。これは、2 つの文字セットがまったく同じ文字で構成されているわけではないからです。

デフォルトの文字セットを変更するには、「文字セットおよびソート順の変更」を参照してください。Replication Server 設定ファイルの **RS_charset** パラメータの値を変更するだけでデフォルトの文字セットを変更できますが、この変更によって複写データが破壊されないように、所定のタスク手順に正確に従ってください。

文字セットの変換

文字セットは送信先 Replication Server で変換されます。Replication Server インタフェース (RSI) 用に作成されたメッセージには、送信元 Replication Server の文字

セット名が含まれています。送信先の Replication Server は、この文字セット名をチェックし、文字データと識別子を送信先 Replication Server の文字セットに変換します。

Replication Server は、文字セットを変換するときに、両方の文字セットに互換性があるかどうかをチェックします。互換性がない場合は変換しません。互換性のある 2 つの文字セット間に共通しない文字が存在する場合、各文字は、認識できない文字として疑問符 (?) で置き換えられます。

Replication Server では、“?” に置き換えられるか、文字セット変換例外が発生するかは、コンテキストによって決まります。たとえば、複写する文字に対応する文字が変換先の文字セットにない場合には、その文字の代わりに “?” が使用されます。これに対し、設定ファイルのパラメータを変換するときに、変換先の文字セットに含まれない文字が見つかった場合には、Replication Server はエラー・メッセージを出力し、停止します。

文字セットを変換するかどうかは、**dsi_charset_convert** 設定パラメータで指定できます。『Replication Server リファレンス・マニュアル』の「**configure connection**」を参照してください。

参照：

- サブスクリプション (115 ページ)
- 文字セットまたはソート順を変更する (121 ページ)

Unicode UTF-8 および UTF-16 のサポート

Replication Server では、デフォルトの文字セットとして Unicode UTF-8 をサポートします。また、UTF-16 でコード化された `unichar`、`univarchar`、`unitext` という 3 つの Unicode データ型もサポートします。Unicode を使用すると、1 つのデータ・サーバ内で、さまざまな言語グループに属するさまざまな言語を混合して使用できます。

『Adaptive Server Enterprise システム管理ガイド 第 1 巻』を参照してください。

UTF-8

UTF-8 (UCS 変形フォーマット、8 ビット形式) は国際化文字セットであり、650 以上の言語をサポートします。UTF-8 は、Unicode 標準の可変長コード化方式で、8 ビット・シーケンスを使用します。

UTF-8 では、ASCII コード値のすべて (0 ～ 127)、およびその他多くの言語の値をサポートします。サロゲート以外のコード値は、1 ～ 3 バイトで表現されます。BMP (Basic Multilingual Plane) の範囲外のコード値では、サロゲート・ペアおよび 4 バイトが必要です。

Adaptive Server、Oracle、IBM UDB、Microsoft SQL Server の各データ・サーバでは、UTF-8 をサポートします。

UTF-16

UTF-16 (UCS 変形フォーマット、16 ビット形式) は、Unicode 標準の固定長コード化方式であり、16 ビット・シーケンスを使用します。ここで使用する文字は、すべて 2 バイト長です。UTF-8 の場合と同様に、BMP の範囲外のコード値は、4 バイトを必要とするサロゲート・ペアを使用して表現します。

Replication Server でも Adaptive Server でも、次の 3 つの文字データ型の値を UTF-16 を使用してコード化します。

- `unichar` – 固定幅の Unicode 文字データ型。
- `univarchar` – 可変幅の Unicode 文字データ型。
- `unitext` – ASE 15.0 および Replication Server 15.0 で導入された可変幅の Unicode ラージ・オブジェクト・データ型。`unitext` は、最大 1,073,741,823 個の Unicode 文字 (2,147,483,647 バイト相当) を格納できる。

稼働条件

Unicode UTF-8 デフォルト文字セット、またはデータ型 `unichar` および `univarchar` を使用するには、Replication Server バージョン 12.5 以降を実行してサイト・バージョンも 12.5 以降に設定してください。

`unitext` データ型を完全にサポートするには、プライマリ Replication Server とレプリケート Replication Server のサイト・バージョンおよびルート・バージョンが 15.0 以降であり、LTL バージョンが 700 である必要があります。connect-source で LTL バージョンが 700 未満である場合、RepAgent およびその他の Sybase Replication Agent は、`unitext` カラムを `image` に変換します。

文字セットを使用する際のガイドライン

所定の Replication Server サイトのすべてのサーバで同じ文字セットを使用し、複写システム内のすべての Replication Server で互換性のある文字セットを使用することを強くおすすめします。

文字セットの変換の問題を最小限に抑えるには、次の方法を使用します。

- 7 ビット ASCII 文字 (可能な場合) をすべてのデータ・サーバ、データ、およびオブジェクト名に使用してください。
  データとオブジェクト名がすべて 7 ビット ASCII 文字からなる場合、またはすべてのデータ・サーバと Replication Server が同じ文字セットを使用する場合には、文字セットの変換で問題が発生することはありません。
- シングルバイト・サーバとマルチバイト・サーバの間でデータを複写する必要がある場合は、文字データとオブジェクト名を 7 ビット ASCII 文字に制限し

て、データが破壊されないようにしてください。このようにしないと問題が発生する可能性があります。たとえば、Replication Server では、サーバ名に使用できる文字を 7 ビット ASCII 文字だけに制限していませんが、Adaptive Server または Connectivity Libraries には制限があります。

- 互換性のある別の種類の文字セット (ISO_1 と CP850 など) を使用するサーバ間でデータを複写する場合には、オブジェクト名と文字データの両方の文字セットに共通しない 8 ビット文字を使用しないでください。

ソート順、

Replication Server では、ソート順 (照合順) によって、文字データと識別子を比較および並べ替える方法が決まります。Replication Server では、バイナリ・ソート以外のソート順を含む、Sybase がサポートするすべてのソート順がサポートされます。バイナリ・ソート以外のソート順は、ヨーロッパ言語での文字データと識別子を正しくソートするために必要です。

デフォルトまたは Unicode のソート順を変更するには、「文字セットおよびソート順の変更」を参照してください。Replication Server 設定ファイルの **RS_sortorder** パラメータの値を変更するだけでデフォルトのソート順を変更できますが、この変更によって複写データが破壊されないように、所定の手順に従ってください。

注意： データと識別子が複写システムにわたって一貫して順序付けられていることを確認するには、すべての Replication Server コンポーネントを同じソート順で設定してください。

参照：

- 文字セットまたはソート順を変更する (121 ページ)

サブスクリプション

ソート順と文字セットは、サブスクリプションが有効性を保つために、どのサーバでも一貫性を持っていることが必要です。

サブスクリプションは、次のような場合にデータの比較を行います。

- マテリアライゼーション中：プライマリ・データ・サーバ
- 分析中：プライマリ Replication Server
- 初期化と削除中：レプリケート Replication Server
- 削除中：レプリケート・データ・サーバ

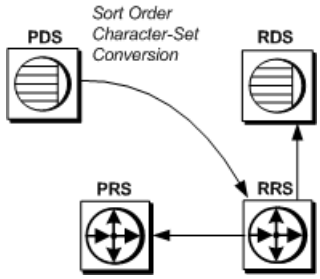
参照：

- 文字セットまたはソート順を変更する (121 ページ)

サブスクリプション・マテリアライゼーション

マテリアライゼーションとは、サブスクリプションを作成してアクティブ化し、プライマリ・データベースからレプリケート・データベースにデータをコピーすることによって、レプリケート・データベースを初期化することです。

図 25 : サブスクリプション・マテリアライゼーション



サブスクリプション・マテリアライゼーション

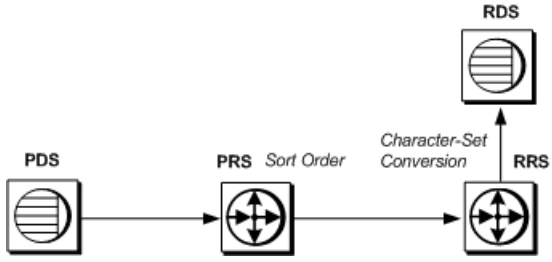
- レプリケート Replication Server がプライマリ・データ・サーバにログインし、**select** 文を発行してプライマリ・データを取得する。
- プライマリ・データ・サーバがすべての文字データをレプリケート Replication Server の文字セットに変換する。この変換を行うためには、両方のサーバの文字セットが異なる場合、レプリケート Replication Server の文字セットをプライマリ・データ・サーバにインストールしておく必要があります。
- レプリケート Replication Server は、データをレプリケート・データ・サーバに挿入する。

注意： バルク・マテリアライゼーションでは、ユーザが選択したメカニズムを使用し、複写システムの外部でサブスクリプションを初期化します。したがって、プライマリ・データ・サーバで初期データが正しいソート順に基づいて選択され、文字データが必要に応じてレプリケート・データ・サーバの文字セットに変換されるように設定してください。

サブスクリプション分析

サブスクリプションは、複写システムで完全に設定される前にいくつかの段階を経過します。

図 26 : サブスクリプション分析



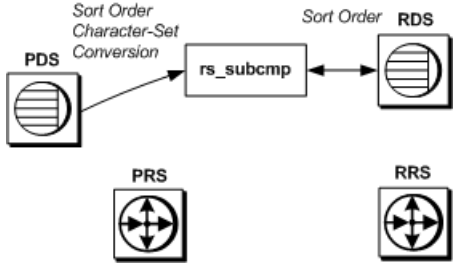
サブスクリプション分析では、次のような処理が行われます。

- Adaptive Server の RepAgent スレッドが、更新のログをスキャンする。
- プライマリ Replication Server のソート順を使用して、どのローがサブスクリプションの指定に適合するかを決定する。また、プライマリ Replication Server は、使用している文字セットの名前を RSI メッセージに追加する。
- レプリケート Replication Server が必要に応じてデータを固有の文字セットに変換し、更新をレプリケート・データ・サーバに適用する。

サブスクリプションの調整

サブスクリプションの調整は、レプリケート・テーブルの不整合をリカバリし、不一致を修正します。

図 27 : サブスクリプションの調整



サブスクリプションの調整では、次のような処理が行われます。

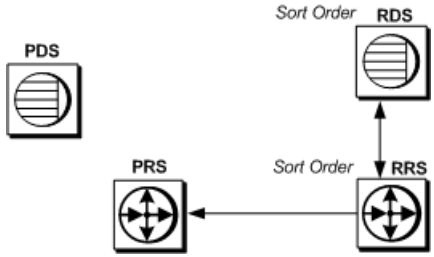
- **rs_subcmp** がレプリケート・データ・サーバの文字セットを使用し、プライマリ・データ・サーバとレプリケート・データ・サーバに接続する。

- プライマリ・データ・サーバが文字データをすべてレプリケート・データ・サーバの文字セットに変換する (**rs_subcmp** のオペレーションはすべてレプリケート・データ・サーバの文字セットを使用して実行される)。この変換を実行するには、両方のサーバの文字セットが異なる場合、レプリケート Replication Server の文字セットをプライマリ・データ・サーバにインストールしておく必要があります。
- **rs_subcmp** によって、両方のデータ・サーバに **select** 文が送られる。両方のデータ・サーバのソート順が同じでなければなりません。

マテリアライゼーション解除

マテリアライゼーション解除は、レプリケート Replication Server でサブスクリプションを削除するコマンドを実行した場合を指します。

図 28 : サブスクリプション・マテリアライゼーション解除



サブスクリプションのマテリアライゼーション解除では、次のような処理が行われます。

- レプリケート Replication Server がレプリケート・データ・サーバからデータを選択し、マテリアライゼーション解除キューを作成する。このとき、ローはレプリケート・データ・サーバのソート順に基づいて選択されます。
- レプリケート Replication Server のソート順に基づいて、他のサブスクリプションに属するローを除外する。
- レプリケート Replication Server が残ったローをレプリケート・データベースから削除する。

Unicode ソート順

Unicode のソート順は、Replication Server のソート順とは異なるので、個別に設定する必要があります。

Unicode のソート順を設定するには、テキスト・エディタを使用して、次の行を Replication Server 設定ファイルに追加します。

```
RS_unicode_sort_order=unicode_sort_order
```

データ・サーバへのコネクションをサスペンドして Replication Server をシャットダウンしてから、Unicode ソート順を変更するようにしてください。

現在使用しているソート順を変更するには、「文字セットおよびソート順の変更」で説明する手順に従ってください。

表 5 : サポートされている Unicode ソート順

名前	説明
defaultml	UTF-16 デフォルト ML 順
altnoacc	CP850 alt、アクセント記号の区別なし
altdict	cp850 alt、小文字優先
altnocsp	CP850 alt、大文字小文字の優先指定なし
scandict	CP850 スカンジナビア語辞書順
scannocp	CP850 スカンジナビア語、大文字小文字の優先指定なし
binary	(デフォルト) UTF-16 バイナリ
dict	Latin-1 英語辞書順
nocase	Latin-1 英語、大文字小文字の区別なし
nocasep	Latin-1 英語、大文字小文字の優先指定なし
noaccent	Latin-1 英語、アクセント記号の区別なし
espdict	Latin-1 スペイン語辞書
espnocs	Latin-1 スペイン語、大文字／小文字の区別なし
espnoac	Latin-1 スペイン語、アクセント記号の区別なし
rusnocs	8859-5 ロシア語、大文字小文字の区別なし
cyrnocs	8859-5 キリル文字、大文字小文字の区別なし
elldict	8859-7 ギリシア語辞書順
hundict	8859-2 ハンガリー語辞書順
hunnoac	8859-2 ハンガリー語、アクセント記号の区別なし
hunnocs	8859-2 ハンガリー語、大文字小文字の区別なし
turknoac	8859-9 トルコ語、アクセント記号の区別なし
turknocs	8859-9 トルコ語、大文字小文字の区別なし

名前	説明
thaidict	CP874 タイ語辞書順
utf8bin	UTF-16 ソート順 (UTF-8 ソート順と一致)

また、**rs_subcmp** に Unicode ソート順を指定することもできます。『Replication Server リファレンス・マニュアル』の「**rs_subcmp**」を参照してください。

参照：
- 文字セットまたはソート順を変更する (121 ページ)

文字セットとソート順

Adaptive Server の文字セットまたはソート順を変更する場合は、サーバにおける複写および別の Adaptive Server 上に存在している、関連するすべての RSSD を管理するすべての Replication Server の文字セットまたはソート順も変更する必要があります。

Adaptive Server のソート順を変更する場合は、レプリケート Replication Server およびレプリケート・データ・サーバのソート順も変更して、サブスクリプションに対して、一貫した処理が実行されるようにしてください。

文字セットまたはソート順を変更すると、サブスクリプションのセマンティックが変わる可能性があります。ソート順の変更は、明らかな影響をもたらします。たとえば、サブスクリプションに “where last_name = MacGregor” という句が指定されている場合、ソート順を dict から binary に変更すると、“MacGregor” の指定はソート処理に対して何の意味も持たなくなります。

プライマリ・データベースとレプリケート・データベースの同期
文字セットまたはソート順を変更したら、プライマリ・データベースとレプリケート・データベースが再度同期していることを確かめてください。次のいずれかの方法を使用することをおすすめします。

- 文字セットまたはソート順を変更した後で、**rs_subcmp** を使用する。
- 文字セットまたはソート順を変更する前にすべてのサブスクリプションを消去し、次に、文字セットまたはソート順を変更した後で、すべてのサブスクリプションを再マテリアライズする。

注意： 文字句が含まれているサブスクリプションがある場合は、消去してから再マテリアライズする方法を使用してください。この方法でのみ、文字句が指定されたサブスクリプションの再同期が保証されます。

文字セットまたはソート順を変更する

Replication Server で文字セットまたはソート順を変更します。

複写元が Adaptive Server であるすべての複写トランザクションは、文字セットまたはソート順を変更する前に、レプリケート・データ・サーバに受信されるようにしてください。この処理では、文字セットまたはソート順を変更しても、データが破壊されることはありません。

1. 関連する Replication Server のすべての RSSD を含む、プライマリ Adaptive Server に関連している、Replication Server および Adaptive Server をすべて特定します。

   文字セットを変更する場合、Replication Server ドメインに属するすべての Adaptive Server の文字セットを検索して、それらの文字セットも変更する必要があるかどうかを確認します。Sybase では、同じドメイン内のサーバ用に、代替文字セットをサポートしていますが、ユーザにはかなりの影響があります。

   ソート順を変更する場合、Sybase では、Replication Server ドメインに属するすべてのデータ・サーバで、同じソート順を使用することをおすすめします。これによって、複写システム全体で、データと識別子の順序の一貫性が維持されます。
2. プライマリの更新をすべてクワイス状態にして、その更新が Replication Server によって処理されていたことを確認してください。

   文字セットを変更する場合は、Adaptive Server 内に 1 ページ分の空のトランザクションを十分に確保できることを確認します。これは、Adaptive Server トランザクション・ログが空になったときに (手順 9 参照)、以前に使用していた文字セットのデータが存在しなくなるようにするためです。
3. 関連するすべての Replication Server をクワイス状態にします。
4. 関連するすべての Replication Server、RepAgent、Replication Agent をシャットダウンします。
5. Replication Server および (必要に応じて) Replication Agent の設定ファイルで、文字セットとソート順を変更します。
6. 関連するすべての Adaptive Server のデフォルトの文字セットとソート順を変更するための手順に従います。詳細については、『Adaptive Server Enterprise システム管理ガイド 第 1 巻』の「文字セット、ソート順、言語の設定」を参照してください。
7. 関連するすべての Adaptive Server をシャットダウンします。
8. 関連する Adaptive Server をシングルユーザ・モードで起動します。シングルユーザ・モードで起動しないと、プライマリ・データベースとレプリケート・データベースで、アクティビティが発生する可能性があります。

9. 関連するすべての Adaptive Server からセカンダリ・トランケーション・ポイントを削除します。この手順によって、RepAgent が Replication Server にまだ転送していないログ・レコードを、Adaptive Server がトランケートできるようになります。Adaptive Server から、次のように入力します。

   ```
   dbcc settrunc('ltm', 'ignore')
   ```

10. トランザクション・ログをトランケートします。Adaptive Server から、次のように入力します。

    ```
    dump transaction db_name with truncate_only
    ```

11. セカンダリ・トランケーション・ポイントを再設定します。Adaptive Server から、次のように入力します。

    ```
    dbcc settrunc('ltm', 'valid')
    ```

12. プライマリ・データベースのロケータ値をゼロ (0) に再設定します。この手順を実行すると、Replication Server は、Adaptive Server から新しいセカンダリ・トランケーション・ポイントを取得して、ロケータをその値に設定します。Adaptive Server から、次のように入力します。

    ```
    rs_zeroltm data_server, db_name
    ```

13. Adaptive Server のシャットダウンおよび再起動をノーマル・モードで実行します。
14. 関連する Replication Server を再起動します。
15. RepAgent (または Replication Agent) が Replication Server に再接続できるように、ログ転送をレジュームします。Replication Server から、次のように入力します。

    ```
    resume log transfer from data_server.db_name
    ```

16. RepAgent を起動します。
17. 複写を再開します。

文字セットを変更すると文字幅も変更される

文字セットの変更によって文字幅も変更される場合は、Replication Server が管理している全データベースのストアド・プロシージャ・メッセージを再ロードします。

ストアド・プロシージャ・メッセージは、UNIX の場合は `$SYBASE/$SYBASE_REP/scripts/rsspmsg1.sql` および `rsspmsg2.sql` に、Windows の場合は `%SYBASE%¥%SYBASE_REP%¥scripts¥rsspmsg1.sql` および `rsspmsg2.sql` にあります。

UNIX の場合

シングルバイトからマルチバイトに文字セットを変更する場合、次のように入力します。

```
isql -Uuser_name -Ppassword -Srssd_name -Jeucjis
< $SYBASE/$SYBASE_REP/scripts/rsspmsg2.sql
```

マルチバイトからシングルバイトに文字セットを変更する場合、次のように入力します。

```
isql -Uuser_name -Ppassword -Srssd_name -Jiso_1
< $SYBASE/$SYBASE_REP/scripts/rsspmsg1.sql
```

Windows の場合

シングルバイトからマルチバイトに文字セットを変更する場合、次のように入力します。

```
isql -Uuser_name -Ppassword -Srssd_name -Jeucjis
< %SYBASE%¥%SYBASE_REP%¥scripts¥rsspmsg2.sql
```

マルチバイトからシングルバイトに文字セットを変更する場合、次のように入力します。

```
isql -Uuser_name -Ppassword -Srssd_name -Jiso_1
< %SYBASE%¥%SYBASE_REP%¥scripts¥rsspmsg1.sql
```

UTF-8 に文字セットを変更する場合、`rsspmsg1.sql` と `rsspmsg2.sql` の両方をインストールします。

概要

複写設計について説明したトピックをまとめます。

- Replication Server がエラー・ログまたはクライアントに出力するメッセージの言語を、英語、フランス語、ドイツ語、日本語のいずれかに設定できます。デフォルト言語は英語です。
- 複写サイトでは、すべてのサーバを同じ言語に設定することをおすすめします。
- Replication Server は、Sybase がサポートするすべての文字セットとソート順をサポートします。バイナリ・ソート以外のソート順や Unicode UTF-8 文字セットも含まれます。
- Replication Server は、プライマリ Replication Server とレプリケート Replication Server の間、またはプライマリ・データベースとレプリケート・データベースの間で、データの文字セットを変換します。
- 複写システムのすべてのサーバで同じ文字セットを使用し、システム内のすべての Replication Server で互換性のある文字セットを使用することをおすすめします。
- ソート順は、サブスクリプションの処理中に重要な役割を果たします。サブスクリプションが有効性を保つためには、ソート順がどのサーバでも一貫性を持っていることが必要です。

容量の計画

複写システムに必要な CPU、メモリ、ディスク、ネットワーク・リソースを見積もります。

複写システムは、Replication Server、Replication Agent (RepAgent またはその他の Replication Agent)、Replication Manager (RM)、Replication Monitoring Services (RMS)、データ・サーバから構成されます。

警告！ Adaptive Server のバージョンを変更した場合は、新しいバージョンの Adaptive Server のトランザクション・ログに必要な容量に基づいて容量を計画し直す必要があります。

すべての容量の計画では、Adaptive Server が 2KB のページ・サイズを使用していると仮定しています。これより大きいページ・サイズを使用している場合は、それに応じて必要な容量を計算し直す必要があります。

Replication Server の稼働条件

Replication Server の稼働条件は次のとおりです。

Replication Server の必要最低限の稼働条件は、次のとおりです。

- Replication Server からのルートを設定する場合は、Replication Server システム・データベース (RSSD) 用の 1 つの Replication Agent。
- ステーブル・キュー用の 1 つ以上の 20MB ロー・パーティションまたはオペレーティング・システム・ファイル。
- RSSD 用の Adaptive Server または ERSSD 用の SQL Anywhere データ・サーバ (SA)。

  RSSD 用の Adaptive Server は、次の条件を満たしている必要があります。
  - RSSD データベース・ディレクトリ用の空きディスク領域が 10MB 以上
  - RSSD トランザクション・ログ・ディレクトリ用の空きディスク領域が 10MB

  ERSSD として使用する SQL Anywhere データ・サーバは、次の条件を満たしている必要があります。
  - ERSSD データベース・ディレクトリ用の空きディスク領域が 5MB 以上
  - ERSSD トランザクション・ログ・ディレクトリ用の空きディスク領域が 3MB 以上
  - ERSSD バックアップ・ディレクトリ用の空きディスク領域が 12MB

- RSSD には、Adaptive Server ユーザが必要とするユーザ・コネクションに加え、20 のユーザ・コネクションが必要。Replication Server を起動すると、複数のスレッドが同時に RSSD を読み込もうとします。こうした要求に応えるためには、20 のユーザ・コネクションを増やす必要があります。
- 複写システムの各 RM および各 RMS に対して 1 つの RSSD ユーザ・コネクションが必要。また、各 RM プロセスおよび各 RMS プロセスに対して、各データ・サーバごとに 1 つのユーザ・コネクションが必要です。
- プライマリ・データが格納されたデータベースごとに、2 つのユーザ・コネクション。
- 複写専用のデータベースごとに 1 つのユーザ・コネクション。
- Replication Server 実行プログラム、すべての Replication Agent、データ・メモリ、スタック・メモリ用として 512MB 以上の RAM (RepAgent は Adaptive Server スレッドなので、Replication Server のメモリは必要ありません)。

Replication Server プライマリ・データベースの稼働条件

Replication Server プライマリ・データベースの稼働条件を指定します。

Replication Server で、プライマリ・データベースごとに必要な条件は次のとおりです。

- Adaptive Server データベース用の RepAgent スレッド、または Sybase 以外のデータベース用のその他の Replication Agent
- 1 つのインバウンド・ステーブル・キュー
- 1 つのアウトバウンド・ステーブル・キュー
- データ・サーバ・インタフェース (DSI) 用のデータ・サーバとの 1 つのコネクション

Adaptive Server の互換性の条件については、リリース・ノートを参照してください。

注意： Replication Agent を Sybase 以外のデータ・ソースで使用している場合、互換性の条件については、適切な Replication Agent のマニュアルを参照してください。

Replication Server レプリケート・データベースの稼働条件

Replication Server レプリケート・データベースの稼働条件を指定します。

Replication Server には、レプリケート・データベースごとに次のものが必要です。

- 1 つのアウトバウンド・ステーブル・キュー
- DSI 用のデータ・サーバとの 1 つのコネクション

Replication Server ルートの稼働条件

Replication Server ルートの稼働条件を指定します。

Replication Server には、他の Replication Server への直接ルートごとに次のものが必要です。

- 1 つのアウトバウンド・ステーブル・キュー

データ量 (キュー・ディスク領域の条件)

複写システムに必要なリソースの量を見積もる際に最も重要な要素は、複写されるデータの量と更新率です。

データ量を正確に計算するためには、複写システムについて次の事項を調べる必要があります。

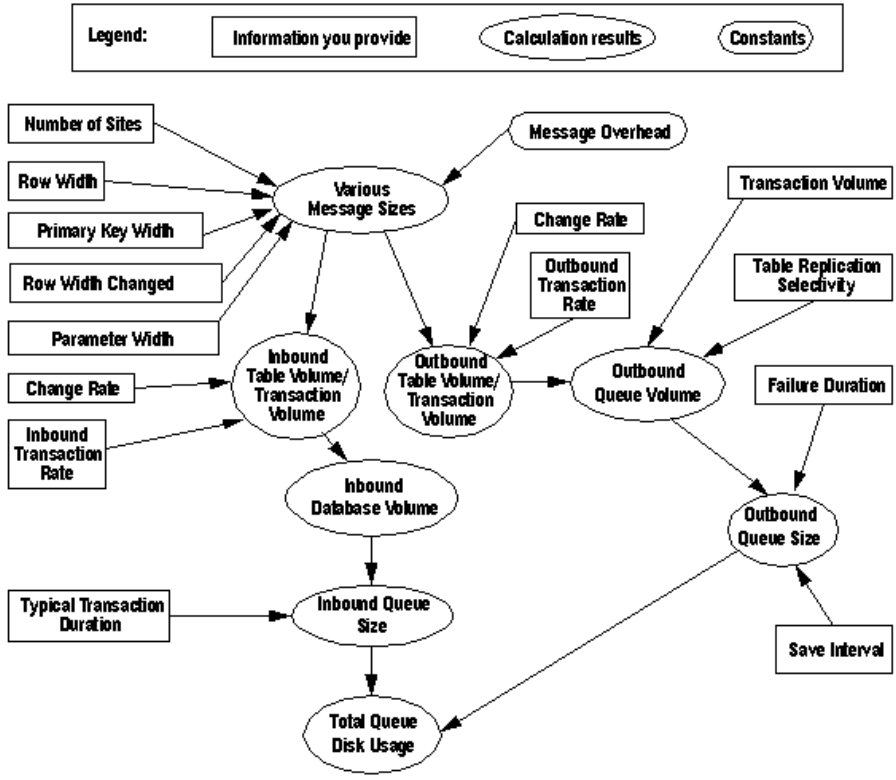
- サイト数
- 複写テーブルのロー幅
- 複写ファンクションのパラメータの幅
- 1 秒あたりのデータ変更回数
- 平均的なトランザクションの実行時間
- 複写テーブルの複写率 (テーブル全体のデータに対するキューから複写されるデータの割合)
- 転送先が使用不可能な場合にトランザクションをキューに保存しておく時間

必要なキューのサイズを計算するための式があります。RSSD のプロシージャ **rs_fillcaptable** と **rs_capacity** でもキューの見積もりサイズを算出できます。これらのストアド・プロシージャについては、『Replication Server リファレンス・マニュアル』を参照してください。

ディスク・キュー・サイズの計算の概要

データ量とキュー・ディスクの使用率を計算するときの順序と流れをグラフィカルに表現します。

図 29 : キュー・ディスク使用率の計算



変更率 (メッセージ数)

変更率とは、一定時間内にテーブルに加えられる変更 (**挿入**、**削除**、**更新**) の最大数をいいます。この値は各テーブルごとに異なります。

容量を見積もるときには、実際に記録された最大変更率よりも大きい値を使用してください。

注意： 変更率は、「単位時間あたりのオペレーション数」で表します (毎秒 5 回の更新など)。変更率を計算するときには、常に同じ単位時間を使用してください。

1 秒あたりの更新回数を測定する場合には、秒を基準としてすべての時間計算を行ってください。

変更量 (バイト数)

変更量とは、テーブル内で一定時間内に変更されるデータの量をいいます。この値は、各テーブルの変更率とデータ・サイズによって変化します。

テーブル・データ量の計算

各メッセージのサイズに 1 秒間に複写されるメッセージの数を乗算したものが複写されるデータの量となります。

各変更オペレーションのメッセージと各複写ファンクションのメッセージのサイズが明らかになれば、すべてのテーブルと複写ファンクションのメッセージ・サイズを足すことで、データベース全体で生成されるメッセージの総量を計算できます。

上限値に基づくテーブル・データ量の計算

テーブル・データ量とデータベースのデータ量はインバウンド・キューとアウトバウンド・キューについて、別々に計算してください。または、アウトバウンド・キューのデータ量だけを計算し、それを基にインバウンド・キューとアウトバウンド・キュー両方のデータ量を概算してください。

テーブル・データ量の上限を求めるには、テーブルの更新率に最大メッセージ・サイズを乗算します。次の式から、テーブル・データ量の上限が求められます。

```
InboundTableVolume upper = (Max[InboundMsgSize] *
    ChangeRate)
```

```
OutboundTableVolume upper = (Max[OutboundMsgSize] *
    ChangeRate
```

構文の説明は次のとおりです。

- *Max[InboundMsgSize]* と *Max[OutboundMsgSize]* は、最大のインバウント・メッセージとアウトバウンド・メッセージのバイト数です (通常は、パラメータ数が最大のメッセージのサイズ)。
- *ChangeRate* は、一定時間あたりの最大変更回数です。

値の合計によるテーブル・データ量の計算

正確なデータ量を計算するには、メッセージ・タイプごとにデータ量をすべて足します。

InboundTableVolume の計算は、次のとおりです。

```
InboundTableVolume =
     (InboundMsgSizeupdate * ChangeRateupdate) +
     (InboundMsgSizeinsert * ChangeRateinsert) +
     (InboundMsgSizedelete * ChangeRatedelete) +
```

```
     (InboundMsgSizefunction1 * ChangeRatefunction1) +
     (InboundMsgSizefunction2 * ChangeRatefunction2) +
     ...
```

OutboundTableVolume の計算は、次のとおりです。

```
OutboundTableVolume =
     (OutboundMsgSizeupdate * ChangeRateupdate) +
     (OutboundMsgSizeinsert * ChangeRateinsert) +
     (OutboundMsgSizedelete * ChangeRatedelete) +
     (OutboundMsgSizefunction1 * ChangeRatefunction1) +
     (OutboundMsgSizefunction2 * ChangeRatefunction2) +
     ...
```

構文の説明は次のとおりです。

- *ChangeRate* は、一定時間あたりの最大変更回数です。*ChangeRate* $_{update}$ はデータの更新、*ChangeRate* $_{insert}$ はデータの挿入などです。
- *InboundMsgSize* と *OutboundMsgSize* は、各種のインバウンド・メッセージとアウトバウンド・メッセージの最大サイズです。

参照：

- メッセージ・サイズ (141 ページ)

トランザクション・データ量

TransactionVolume とは、開始レコードとコミット・レコードによって生成されるデータの量です。

InboundTransactionVolume の式は、次のとおりです。

```
InboundTransactionVolume = (MsgSizecommit + MsgSizebegin) *
InboundTransactionRate
```

構文の説明は次のとおりです。

- *MsgSize* $_{begin}$ + *MsgSize* $_{commit}$ は、トランザクションあたり 450 バイトとします。
- *InboundTransactionRate* は、プライマリ・データベース内で一定時間内に実行されるトランザクションの総数です。この数には、すべてのトランザクション、つまり複写テーブルを更新するトランザクションと更新しないトランザクションが含まれます。1 つまたは複数のテーブルに対する 1 つまたは複数の変更を行うトランザクションも 1 つと数えます。

OutboundTransactionVolume の式は、次のとおりです。

```
OutboundTransactionVolume = (MsgSizecommit + MsgSizebegin) *
OutboundTransactionRate
```

構文の説明は次のとおりです。

- *MsgSize* $_{begin}$ + *MsgSize* $_{commit}$ は、トランザクションあたり 450 バイトとします。

- *OutboundTransactionRate* は、一定時間内に特定のアウトバウンド・キューから複写されるトランザクションの総数です。1 つまたは複数のテーブルに対する 1 つまたは複数の変更を行うトランザクションも 1 つと数えます。

  OutboundTransactionRate は、次の 2 つの数によって決まります。
  - 実際に複写データを更新するトランザクションの数
  - それらのうち、特定のアウトバウンド・キューから複写されるトランザクションの数

  たとえば、*InboundTransactionRate* の値が 1 秒あたり 50 トランザクションで、その半数が複写テーブルを更新し、その更新の 20% がキュー 1 からの複写によるものである場合、キュー 1 の *OutboundTransactionRate* は 1 秒あたり 5 トランザクションとなります (50*0.5*0.2)。

  トランザクションが複写率 (replication selectivity) のそれぞれ異なる多数の複写テーブルを更新する場合、*OutboundTransactionRate* の計算は複雑になります。このような場合には、*InboundTransactionRate* の値を *OutboundTransactionRate* の上限値と見なし、*OutboundTransactionRate* の代わりに、そのまま計算式で使用してかまいません。

  データベースのすべてのトランザクションがアウトバウンド・キューから複写される場合には、*OutboundTransactionRate* の値は *InboundTransactionRate* とまったく同じになります。

データベースのデータ量

データベースのデータ量の上限の概算は、各テーブルの最大メッセージ・サイズを足し合わせることで計算できます。

```
InboundDatabaseVolume upper = sum(InboundTableVolume upper) +
InboundTransVolume
```

InboundDatabaseVolume をより正確に特定するには、「上限値に基づくテーブル・データ量の計算」で計算したテーブルのデータ量を一緒に合計します。

```
InboundDatabaseVolume = sum(InboundTableVolume) +
InboundTransactionVolume
```

参照：

- データベース入力データ量 (138 ページ)
- 上限値に基づくテーブル・データ量の計算 (129 ページ)

インバウンド・キュー・サイズ

インバウンド・キューには、プライマリ・データベースのすべての複写テーブルに対する更新が保存されます。これらの更新は、最も長いトランザクションが実行されている間、インバウンド・キューに保持されます。キューのサイズはバイト数で表します。

インバウンド・キューの平均サイズを計算する式は、次のとおりです。

```
InboundQueueSizetypical = InboundDatabaseVolume *
TransactionDurationtypical
```

インバウンド・キューの最大サイズを計算する式は、次のとおりです。

```
InboundQueueSizelongest = InboundDatabaseVolume *
TransactionDurationlongest
```

構文の説明は次のとおりです。

- *InboundDatabaseVolume* は、「データベースのデータ量」で説明した式で計算されたメッセージ・データ量です。
  特に厳密な計算が必要でなければ、*OutboundDatabaseVolume* の値で *InboundQueueSize* を計算できます。
- *TransactionDuration* typical は、平均的なトランザクションの秒数を表します。
- *TransactionDuration* longest は、最も長いトランザクションの秒数を表します。

実際には、インバウンド・キューの最大サイズは計算値より多少大きくなります。これは、長いトランザクションの読み取り中に、新しいトランザクションがログに追加されるためです。最も長いトランザクションの時間は概算なので、値を決定するときには、トランザクションをステーブル・キューから読み取り、処理するための短い時間を見積もり、加算してください。

また、メッセージが Replication Server に転送される速度が遅い場合には、キューのサイズが大きくなります。Replication Server は、メッセージを 16K の固定ブロック単位でステーブル・キューに書き込みます。遅延時間を短くするために、Replication Server はブロックが満杯にならなくても 1 秒ごとに書き込みます (**init_sqm_write_delay** 設定パラメータを使用して、この遅延を 1 秒以外の値に変更できます)。いくつかのメッセージが同時に到着した場合は、それらがすべて 1 つのブロックに書き込まれますが、間隔をあけて到着すると、それぞれのメッセージが別々のブロックを占有することになります。

注意： 新しいバージョンの Adaptive Server では、トランザクション・ログに必要なディスク領域が増えることがあるため、Replication Server ステーブル・キューに必要なディスク領域も、それに応じて増えることがあります。

参照：

- インバウンド・キュー・サイズの計算例 (138 ページ)

アウトバウンド・キューのデータ量

アウトバウンド・キューは、データを他の Replication Server またはデータ・サーバに送ります。データ量の計算では、データの転送先を直接送信先サイトと呼んでいます。

直接送信先サイトに複写されるデータベースごとに、アウトバウンド・キューからテーブル・データ量の一部が転送されます。キューから複写されるテーブルの割合を複写率と呼びます。

すべてのテーブルについて、最高複写率を 1 (100%) と仮定して、*OutboundQueueVolume* の上限値を計算します。次に、キューから複写されるすべてのテーブルの *OutboundQueueVolume* を合計します。

```
OutboundQueueVolumeupper= sum(OutboundTableVolumes) +
OutboundTransactionVolume
```

たとえば、2 つのテーブルがあり、それぞれの *OutboundTableVolumes* が 1 秒あたり 20K と 10K で、*OutboundTransactionVolume* がいずれも 1 秒あたり 1K であるとします。これらのテーブルがアウトバウンド・キューから複写される場合、*OutboundQueueVolume* の値は、次のようになります。

```
20K/Sec + 10K/Sec + 1K/Sec = 31K/Sec
```

より正確なキューのデータ量を求めるには、テーブルの複写率を計算に含めます。式は次のようになります。

```
OutboundQueueVolume = sum(OutboundTableVolume *
ReplicationSelectivity) + OutboundTransactionVolume
```

たとえば、前の例で、1 つ目のテーブルは全体の 50% だけが複写され、2 つ目のテーブルは 80% 複写されたとすると、*OutboundQueueVolume* の値は次のようになります。

```
(20K/Sec*0.5) + (10K/Sec*0.8) + 1K/Sec = 19K/Sec
```

参照：

- アウトバウンド・キューのデータ量の計算例 (139 ページ)

障害時間

直接送信先サイトが使用できなくなった場合、キューはそのサイト宛のメッセージを保管していなければなりません。ここに示した計算式を使ってシステムの容量を見積もると、あるキューのサイズを計算するのに使用した障害時間の値によって、そのキューが耐えられるダウン時間の長さが決まります。

セーブ・インターバル

セーブ・インターバルとは、データを削除せずにアウトバウンド・キューに残しておく時間をいいます。またセーブ・インターバルは、ディスクの障害またはノード・ロスからのリカバリに使用します。

configure connection コマンドを使用して、データベース・コネクションにセーブ・インターバルを指定します。**configure route** コマンドを使用して、他の Replication Server へのルートにセーブ・インターバルを指定します。詳細については、『Replication Server リファレンス・マニュアル』を参照してください。

アウトバウンド・キュー・サイズ

アウトバウンド・キュー・サイズを計算します。

アウトバウンド・キューのサイズは、次に示す式で計算します。

```
OutboundQueueSize = OutboundQueueVolume * (FailureDuration +
SaveInterval)
```

構文の説明は次のとおりです。

- *OutboundQueueVolume* は、一定時間内にアウトバウンド・キューに保存されるデータのバイト数です。
- *FailureDuration* は、サイトが動作不能となったときにキューがデータを保持しているべき最大時間です。
- *SaveInterval* は、メッセージが削除されるまでにキューに保存される時間です。

あるアウトバウンド・キューの *OutboundQueueVolume* を 1 秒あたり 18K とし、*SaveInterval* を 1 時間とし、送信先サイトのダウンに耐えられる時間を 1 時間とする場合、そのキューのサイズは次の計算式で求められるように 130MB でなければなりません。

```
OutboundQueueSize =
 18K/sec * (60min + 60min) =
 0.018MB/sec * 120min * 60sec/min = 130MB
```

変更率とテーブル複写率に基づいてキューに書き込まれるデータが、1 秒間に 16K のブロックを満杯にしない場合には、実際の値はこの計算値より小さくなります。これは、Replication Server が 1 秒ごとに 1 つのブロックを満杯かどうかにかかわらず書き込むためです。キューの使用効率が最悪になるのは、短いトランザクションを 1 秒以上の間隔をあけて実行する場合です。Sybase の統計では、ほとんどのブロックが 50% ～ 95% の使用率にとどまっています。

参照：

- アウトバウンド・キュー・サイズ (139 ページ)

キュー・ディスクの総使用量

ディスク領域の総量を計算します。

キューの使用効率が最悪となる場合に必要なディスク領域の総量は、次のような式で計算します。

```
Sum[InboundQueueSize] + Sum[OutboundQueueSize]
```

構文の説明は次のとおりです。

- *Sum[InboundQueueSize]* は、各データベースのインバウンド・キュー・サイズの合計です。
- *Sum[OutboundQueueSize]* は、各直接送信先サイトのアウトバウンド・キュー・サイズの合計です。

その他の注意事項

システムの容量を見積もるときには、ここに記載の事項も考慮してください。

- ネットワーク障害によって、いずれかのリモート・サイトが使用できなくなった場合、同じネットワーク上の他のサイトも使用できなくなる可能性が高く、多くのキューで使用率が同時に増加します。したがって、すべてのキューが設定した障害時間に同時に耐えられるように、十分なディスク領域を割り付けることが必要です。
- キューが満杯に近づいた場合、スタンバイのディスク領域を用意していれば、いつでもパーティションを追加できます。
- すべてのキューが満杯になり、インバウンド・キューがメッセージを受け付けられなくなると、プライマリ・データベースがログをトランケートできなくなります。プライマリ・データベースがログをトランケートできるように手動で変更すると、ログからトランケートされた更新がレプリケート・データベースに配信されません。

キュー・サイズの計算例

ディスク・キュー・サイズの計算式を使用した一連のサンプル計算を示します。

計算例で使用するパラメータ

ディスク・キュー・サイズの計算式を使用して生成された計算値を示します。

次の計算例で使用する複写システム例を、次のように仮定します。

- `DB1`. データベースに `T1` テーブルと `T2` テーブルがある。
- `DB2`. データベースに `T3` テーブルがある。
- これら 3 つのテーブルは、S1、S2、S3 という 3 つのサイトに複写される。
- すべてのメッセージがすべてのサイトに複写されるわけではない。

表 6 : テーブル・パラメータ

データベース	テーブル	カラム数	カラム変更	変更率 (回数/秒)	ロー幅 (バイト)	ロー幅変更	サイト数
DB1	T1	10	5	20	200	100	3
DB1	T2	10	5	10	400	200	2
DB2	T3	120	100	2	1500	1000	2

表 7 : サイト・パラメータ (テーブルの複写率)

テーブル	S1 への更新複写率	S2 への更新複写率	S3 への更新複写率
T1	10%	40%	40%
T2	40%	80%	0%
T3	20%	0%	20%

表 8 : トランザクション率

トランザクション率	*DB1*	*DB2*	S1	S2	S3
インバウンド	20/秒	20/秒	-	-	-
アウトバウンド	-	-	5/秒	10/秒	8/秒

メッセージ・サイズの計算例

メッセージ・サイズの式を使用したサンプル計算です。

次の計算例では、*RowWidthChanged* 計算式 (最少カラムを除く) を使用します。

参照 :

- メッセージ・サイズ (141 ページ)

テーブル更新メッセージ・サイズの計算

メッセージ・サイズの上限を概算する計算例です。

次の計算式を使用します。

```
InboundMsgSizeupdate = InboundMsgOverhead +
  ColOverhead + RowWidth + RowWidthChanged
```

```
OutboundMsgSizeupdate = OutboundMsgOverhead +
  RowWidth + RowWidthChanged + (NumSites*8)
```

各サイトの更新メッセージ・サイズは、次のように計算できます。

```
InboundMsgSizeT1 = 380 + 450 + 200 + 100 = 1130
  bytes
```

```
InboundMsgSizeT2 = 380 + 450 + 400 + 200 = 1430
  bytes
```

```
InboundMsgSizeT3 = 380 + 6600 + 1500 + 1000 = 9480
  bytes
```

```
OutboundMsgSizeT1 = 200 + 200 + 100 + 24 = 524 bytes
```

```
OutboundMsgSizeT2 = 200 + 400 + 200 + 16 = 816 bytes
```

```
OutboundMsgSizeT3 = 200 + 1500 + 1000 + 16 = 2716
  bytes
```

begin と commit

begin と commit のメッセージのサイズを計算するには、次のような式を使用して値を求めます。

```
MsgSizebegin = 250
```

```
MsgSizecommit = 200
```

変更率

変更率とは、テーブルで一定時間内に行われるデータ変更の最大数をいいます。変更率は複写システムのテーブルごとに異なります。

テーブル・データ量の計算例

テーブル T1、T2、T3 のデータ量を計算します。

分かりやすくするために、最悪の場合を想定し、すべての変更が SQL 文の **update** とその begin/commit ブロックによるものと仮定します。

InboundTableVolume の計算には、次に示す上限値を使用した計算式を使用します。

```
InboundTableVolume = (Max[InboundMsgSize]*ChangeRate)
```

テーブル T1、T2、T3 の入力データ量は次のようになります。

```
InboundTableVolumeT1 = 1130*20 = 23K/sec
```

```
InboundTableVolumeT2 = 1430*10 = 14K/sec
```

```
InboundTableVolumeT3 = 9480*2 = 19K/sec
```

OutboundTableVolume の計算には、次に示す上限値を使用した計算式を使用します。

```
OutboundTableVolume = (Max[OutboundMsgSize]* ChangeRate)
```

テーブル T1、T2、および T3 の出力データ量は次のようになります。

```
OutboundTableVolumeT1 = 524*20 = 10K/sec
```

```
OutboundTableVolumeT2 = 816*10 = 8K/sec
```

```
OutboundTableVolumeT3 = 2716*2 = 5K/sec
```

構文の説明は次のとおりです。

- *InboundMsgSize* と *OutboundMsgSize* には、「テーブル更新メッセージ・サイズの計算」に示した計算値を使用しました。
- *ChangeRate* には、表 6 : テーブル・パラメータ (135 ページ)に示した値を使用しました。

RSSD およびログ・サイズの計算例

Replication Server は、それぞれ固有のインバウンド・キューに RSSD を持っています。この RSSD もアウトバウンド・キューのサイズに関係します。

ただし、RSSD の使用率はきわめて低く、トランザクションがいずれも小さいため、RSSD が必要とするディスク領域は、下記のようにごく少量であると考えてかまいません。

- インバウンド・キュー = 2MB
- アウトバウンド・キュー = なし

データベース入力データ量

データベース入力データ量を計算する例を示します。

計算された *InboundTableVolumes* の値に基づき、上限値を使用した計算式を使用して *InboundDatabaseVolume* の値を計算できます。

```
InboundDataBaseVolume = sum(InboundTableVolume) +
InboundTransactionVolume
```

構文の説明は次のとおりです。

- *TransactionVolume* は、*Message* $_{\text{begin}}$ と *Message* $_{\text{commit}}$ の合計 (250 バイト + 200 バイト) に *TransactionRate* の値を乗算して得られる値です。
- 各データベースのトランザクション率には、表 8 : トランザクション率 (136 ページ)に示した値を使用します。

したがって、*InboundDatabaseVolumes* は次のように計算できます。

```
InboundDatabaseVolumeDB1 = 23K + 14K +
 (450 bytes/tran * 20 tran/sec) = 46K/sec
```

```
InboundDatabaseVolumeDB2 = 19K +
 (450 bytes/tran * 20 tran/sec) = 28K/sec
```

インバウンド・キュー・サイズの計算例

インバウンド・キュー・サイズを計算する例を示します。

InboundDatabaseVolumes の値に基づき、次のような式を使用して DB1 および DB2 のインバウンド・キューの最大サイズを計算できます。

```
InboundQueueSize = InboundDatabaseVolume * TransactionDuration
```

構文の説明は次のとおりです。

- *Inbound DatabaseVolume* には、計算されたメッセージ量を使用します。

- *TransactionDuration* は、最も長いトランザクションの秒数です。この計算例では、DB1 の *TransactionDuration* を 10 分とし、DB2 の *TransactionDuration* を 5 分とします。

InboundQueue のサイズは、次のように計算できます。

```
InboundQueueDB1 = 46K/sec * 600sec = 28MB
```

```
InboundQueueDB2 = 28K/sec * 300sec = 8MB
```

アウトバウンド・キューのデータ量の計算例

アウトバウンド・キュー・サイズの計算例を示します。

テーブル T1、T2、T3 の *OutboundQueueVolume* を計算します。複写率はデータベースごとではなくテーブルごとに設定されるので、アウトバウンド・キューから複写される個々のテーブルについて、アウトバウンド・キューのサイズを計算しなければなりません。式は次のようになります。

```
OutboundQueueVolume = sum(OutboundTableVolume *
ReplicationSelectivity) + OutboundTransactionVolume
```

構文の説明は次のとおりです。

- *OutboundTableVolume* は、「テーブル・データ量の計算」トピックで行った計算例から求められます。
- *ReplicationSelectivity* の値には、表 7 : サイト・パラメータ (テーブルの複写率) (136 ページ)に示した値を使用します。
- *TransactionVolume* は、*Message* $_{\text{begin}}$ と *Message* $_{\text{commit}}$ の合計 (250 バイト + 200 バイト) に *TransactionRate* の値を乗算して得られるサイズです。

サイト 1 (S1) の *OutboundQueueVolume* を計算する式は次のとおりです。

```
OutboundQueueVolumeS1 = OutboundTransactionVolume +
(TableVolumeT1*ReplicationSelectivityT1,S1) +
(TableVolumeT2*ReplicationSelectivityT2,S1) +
(TableVolumeT3*ReplicationSelectivityT3,S1)
```

3 つのサイトそれぞれの *OutboundQueueVolumes* は次のようになります。

```
Site1 = (450*5) + (10K/sec*0.1) + (8K/sec*0.4) + (5K/sec*0.2) = 7K/
sec
```

```
Site2 = (450*10) + (10K/sec*0.4) + (8K/sec*0.8) + (5K/sec*0) = 15K/
sec
```

```
Site3 = (450*8) + (10K/sec*0.4) + (8K/sec*0) + (5K/sec*0.2) = 9K/sec
```

アウトバウンド・キュー・サイズ

アウトバウンド・キュー・サイズの計算方法を示します。

アウトバウンド・キューのサイズは、次に示す式で計算します。

```
OutboundQueueSize = OutboundQueueVolume * (FailureDuration +
SaveInterval)
```

構文の説明は次のとおりです。

- *OutboundQueueVolume* は、一定時間内にアウトバウンド・キューに保存されるデータのバイト数です。
- *FailureDuration* は、サイトが動作不能となったときにキューがデータを保持しているべき最大時間です。この計算例では、障害時間が 4 時間 (14,400 秒) に設定されていると仮定します。
- *SaveInterval* は、サイズを計算するキューに設定された時間です。この計算例では、セーブ・インターバルが 2 時間 (7,200 秒) に設定されていると仮定します。
- *FailureDuration* + *SaveInterval* の合計は 21,600 秒となります。

3 つのサイトそれぞれの *OutboundQueueSizes* のサイズは次のようになります。

```
Site1 = 7K/sec * 21,600sec = 151MB
```

```
Site2 = 15K/sec * 21,600sec = 324MB
```

```
Site3 = 9K/sec * 21,600sec = 194MB
```

キュー・ディスク総使用量の計算例

キュー・ディスク総使用量の計算例を示します。

最悪の障害時 (4 時間) にすべてのキューを維持するために必要なディスク領域の総量を計算するには、次のような式を使用します。

```
Sum(InboundQueueSizes) + Sum(OutboundQueueSizes)
```

構文の説明は次のとおりです。

- *InboundQueueSizes* には、「インバウンド・キュー・サイズの計算例」で計算した 2 つのキュー・サイズを使用します。
- *OutboundQueueSizes* には、前の例で計算した 3 つのアウトバウンド・キューのサイズを使用します。

RSSD インバウンド・キューに割り当てる 2MB を含めた最悪の障害時に必要なディスク領域の総量は、次のように計算できます。

```
2MB + 28MB + 8MB + 151MB + 324MB + 194MB = 707MB
```

最悪の事態を想定して十分なディスク領域を割り付けてください。セーブ・インターバル機能 (メッセージが次のサイトに配信された後もアウトバウンド・キューのデータをトランケートしない) を使用する場合は、トランザクション活動率が最高に達してもキューが耐えられるように、必ず十分な量の領域を割り付けてください。セーブ・インターバル機能を使用しない場合は、通常の動作状態となり、キューの使用量は極めて少なくなります (1 キューあたり 1MB か 2MB)。

ここに示した計算例では、すべてのアウトバウンド・キューが同じ障害時間に耐えられなければならないと仮定しました。このような前提がどの環境にも当てはまるわけではありません。一般的には、WAN 接続の方がローカル接続より長い障害時間に耐える必要があります。

参照：
- インバウンド・キュー・サイズの計算例 (138 ページ)

メッセージ・サイズ

Replication Server は、データベースに対する変更を ASCII フォーマットのメッセージとして配信します。複写システムから送出されるメッセージのほとんどは、削除、挿入、更新、ファンクションの実行に対応します。

テーブルごとに、挿入メッセージと削除メッセージのサイズと更新メッセージのサイズは異なります。ファンクションにもそれぞれに固有のメッセージ・サイズがあります。メッセージ・サイズはバイト数で表されます。

変更の種類 (**挿入**、**削除**、**更新**) が同じでも、メッセージがインバウンド・キューにあるかアウトバウンド・キューにあるかによってメッセージ・サイズが異なります。

テーブルの更新、挿入、削除、ファンクション、begin/commit の組み合わせを通知するメッセージのサイズをバイト単位で計算する際に使用できる式があります。

テーブルの更新
メッセージ・サイズのおおよその上限を計算するには、次のような式を使用します。

```
InboundMsgSizeupdate = InboundMsgOverhead +
  ColOverhead + (RowWidth*2)
```

```
OutboundMsgSizeupdate = OutboundMsgOverhead +
  (RowWidth*2) + (NumSites*8)
```

より正確な見積もりを出すには、*RowWidthChanged* を計算に加えます。

```
InboundMsgSizeupdate = InboundMsgOverhead +
  ColOverhead + RowWidth + RowWidthChanged
```

```
OutboundMsgSizeupdate = OutboundMsgOverhead +
  RowWidth + RowWidthChanged + (NumSites*8)
```

最少カラム機能を使用する場合、次の式を使用してメッセージのサイズを計算します。

```
InboundMsgSizeupdate = InboundMsgOverhead +
  ColOverhead + (RowWidthChanged*2) +
  PrimaryKeyWidth
```

```
OutboundMsgSizeupdate = OutboundMsgOverhead +
  (RowWidthChanged*2) + PrimaryKeyWidth +
  (NumSites*8)
```

テーブルの挿入
テーブル挿入のメッセージ・サイズを計算します。最少カラム機能を使用する場合にもこの式を使用できます。

```
InboundMsgSizeinsert = InboundMsgOverhead +
  ColOverhead + RowWidth
```

```
OutboundMsgSizeinsert = OutboundMsgOverhead +
  RowWidth + (NumSites*8)
```

テーブルの削除
最少カラム機能を使用しない場合、テーブルの削除を通知するメッセージのサイズを計算するには、次のような式を使用します。

```
InboundMsgSizeinsert = InboundMsgOverhead +
  ColOverhead + RowWidth
```

```
OutboundMsgSizeinsert = OutboundMsgOverhead +
  RowWidth + (NumSites*8)
```

最少カラム機能を使用する場合、テーブルの削除を通知するメッセージのサイズを計算するには、次のような式を使用します。

```
InboundMsgSizedelete = InboundMsgOverhead +
  ColOverhead + PrimaryKeyWidth
```

```
OutboundMsgSizedelete = OutboundMsgOverhead +
  PrimaryKeyWidth + (NumSites*8)
```

ファンクション
ファンクションのメッセージ・サイズを計算します。

```
InboundMsgSizefunction = InboundMsgOverhead +
  ParameterWidth +(RowWidth*2)
```

```
OutboundMsgSizefunction = OutboundMsgOverhead +
  ParameterWidth + (NumSites*8)
```

複写ファンクションには、上記のインバウンド・メッセージ・サイズの計算式にある *RowWidth* は必要ありません。これは、複写ファンクションの更新前イメージと更新後イメージがインバウンド・キューに送信されないからです。

begin と commit
begin と commit を通知するメッセージのサイズを計算します。

```
InboundMsgSizebegin = OutboundMsgSizebegin = 250
```

```
InboundMsgSizecommit = OutboundMsgSizecommit = 200
```

begin/commit ブロックに対応するメッセージ・サイズは、合計 450 バイトです。トランザクションが一般に多くのデータ変更を行う場合は、begin/commit メッセージをメッセージ・サイズの見積もりから除外してもかまいません。こうしたトラ

ンザクションの begin/commit メッセージのサイズがメッセージの合計サイズに占める割合は、ごくわずかだからです。

式の要素
データ量を計算する際のコンポーネント定義です。

- *InboundMsgOverhead* の値は 380 バイトとします。各メッセージは、トランザクション ID や重複を検出したシーケンス番号などの要素からなります。
- *OutboundMsgOverhead* の値は 200 バイトとします。各メッセージは、トランザクション ID や重複を検出したシーケンス番号などの要素からなります。
- *ColOverhead* (インバウンド・キューだけで使用) の値はカラムあたり 30 バイトとします。
  最少カラムを使用しない更新オペレーション
  ```
  (NumColumns+NumColumnsChanged) * 30
  ```
  最少カラムを使用した更新オペレーション
  ```
  ((NumColumnsChanged*2)+NumPrimaryKeyColumns) * 30
  ```
  挿入オペレーション (最少カラムを使用してもしなくても同じ)
  ```
  NumColumns * 30
  ```
  ```
  For a delete operation without minimal columns:
  ```
  ```
  NumColumns * 30
  ```
  最少カラムを使用した削除オペレーション
  ```
  NumPrimaryKeyColumns * 30
  ```
- *RowWidth* は、テーブル・カラムの ASCII 表現のサイズです。例：`char(10)` カラムであれば 10 バイト、`binary(10)` カラムであれば 22 バイトとなります。
  テーブル更新の場合は、*RowWidth* の値を 2 倍にします。これは、ローの更新前イメージと更新後イメージの両方がレプリケート・サイトに配信されるためです。
- *RowWidthChanged* は、ロー内の変更されたカラムの幅です。たとえば、ローに 10 のカラムがあり、*RowWidth* の合計が 200 バイトであるとします。このうち半数のカラムが変更されるとすると、*RowWidthChanged* の値はおよそ 100 バイトになります。
- *NumSites* は、メッセージが送信されるサイトの数です。この値が意味を持つのは、短いメッセージが多くのサイトに配信される場合だけです。サイト数を使用する式から省略してもかまいません。この式では、各サイトの ID は 8 バイトの長さなので、*Numsites* は 8 を掛けた値になります。
- *ParameterWidth* は、ファンクション・パラメータの ASCII 表現のサイズです。

- *Begin/Commit Pair* の組み合わせは、トランザクション開始メッセージのヘッダとコミット・トレーラを合わせたサイズです。450 バイトとします。

参照：
- テーブル更新メッセージ・サイズの計算 (136 ページ)

その他の必要なディスク領域

複写システムを構成する Replication Server、Replication Agent、データ・サーバに必要なディスク領域について詳しく説明します。

ステーブル・キュー

Replication Server をインストールするときに、Replication Server が「ステーブル・キュー」を設定するために使用するディスク・パーティションを設定します。

参照：
- キュー・ディスクの総使用量 (134 ページ)

RSSD

RSSD のディスク領域の最小要件を指定します。

RSSD には、データ用に 10MB 以上、ログ用に 10MB 以上を割り付けてください。これらのデフォルト値は、比較的小さい複写システム用に設定されています。数百個の複写定義と数千個のサブスクリプションに十分な領域を確保するには、データ領域とログ領域を 12MB に増やします。

エラーおよび拒否されたトランザクションも RSSD に保存されます。システム管理者は、これらのエラーおよび拒否されたトランザクションが格納されるテーブル (`rs_exceptscmd`、`rs_exceptshdr`、および `rs_exceptslas`) を定期的にトランケートする必要があります。障害の診断を行うためにステーブル・キューを RSSD にダンプした場合は、不要になったダンプ・テーブルからトランケートしてください。

ERSSD

ERSSD のディスク領域の最小要件を指定します。

Embedded RSSD (ERSSD) には、データ用に 5MB 以上、ログ用に 3MB 以上、バックアップ用に 12MB 以上を割り付けてください。これらのデフォルト値は、比較的小さい複写システム用に設定されています。数百個の複写定義と数千個のサブスクリプションに十分な領域を確保するには、データ領域とログ領域をそれぞれ 25MB にしてください。

エラーおよび拒否されたトランザクションも ERSSD に保存されます。システム管理者は、これらのエラーおよび拒否されたトランザクションが格納されるテーブルを定期的にトランケートする必要があります。障害の診断を行うためにステーブル・キューを ERSSD にダンプした場合は、不要になったダンプ・テーブルからトランケートしてください。

ログ

ログ・ファイルに必要な領域の最小要件を指定します。

Replication Server は、情報、エラー・メッセージ、トレース出力をエラー・ログ・ファイルに書き込みます。こうしたログ・ファイルには、1MB ～ 2MB のディスク領域を割り付けてください。Sybase 製品の保守契約を結んでいるサポート・センタからトレース・フラグをオンに設定するように指示された場合は、ログ用のディスク領域を増やす必要があります。

RepAgent は、Adaptive Server エラー・ログ・ファイルにメッセージを書き込みます。

メモリ使用状況

複写システムでは、Replication Server は個別のプロセスです。RepAgent は 1 つの Adaptive Server スレッドです。

Replication Server のメモリ要件

Replication Server のメモリ要件を指定します。

一般的なガイドラインを示すために、Replication Server を Sun SPARCstation で使用する場合を想定して、必要なメモリを見積もります。

- 新しくインストールした Replication Server は、データとスタック用に約 7MB を使用します。
- 各 DSI コネクションが約 500K を使用します。MD メモリ (**md_sqm_write_request_limit** 設定パラメータ) を大きい値に設定した場合は、このメモリ量も増加します。
- 各 RepAgent コネクションが 500K を使用します。ステーブル・キュー・トランザクション・メモリ (**sqt_max_cache_size** 設定パラメータ) を大きく設定した場合は、このメモリ量も増加します。
- 数千のサブスクリプションがあるか、ファンクション文字列キャッシュのサイズを大きくした場合には、それに見合う量のメモリを増設する必要があります。
- サブスクリプション・ルール用のメモリは、サブスクリプションで参照されているカラムの数とルール数のファンクションです。平均的なサブスクリプショ

ン 1 つにつき、80 バイト未満とサブスクリプション文全体のバイト数を加えた量のメモリが必要です。
- ファンクション文字列キャッシュのサイズは 200K まで増やすことができます。
- システム・テーブル・キャッシュのサイズは定義されたオブジェクトの個数によって決まります。複写定義は、そのカラム数の約 250 倍に当たる量のメモリを使用します。このメモリ量は、カラム数の多い複写定義が多数ある場合には、必要メモリを見積もるうえで重要な要素となります。

RepAgent のメモリ要件

オーバヘッド、スキーマ・キャッシュ、トランザクション・キャッシュ、text および image キャッシュに必要な RepAgent のメモリ要件を指定します。

RepAgent が使用するメモリのほとんどは、割り付けられた AdaptiveServer プロシージャ・キャッシュ (共有メモリ) です。

サーバのメモリの増やし方については、『Adaptive Server Enterprise システム管理ガイド 第 2 巻』を参照してください。

オーバヘッド

Adaptive Server は、5,612 バイトを各データベースのログ転送用に割り付けます。

RepAgent が有効の場合、Adaptive Server は起動時に 2,332 バイトの追加メモリを各 RepAgent に割り付けます。

スキーマ・キャッシュ

スキーマ・キャッシュに使用されるメモリの量は、複写されるオブジェクト (テーブルとストアド・プロシージャ) と記述子 (カラムとパラメータ) の数によって変わります。オブジェクトと記述子それぞれ 1 つにつき 128 バイト必要です。

起動時に Adaptive Server は 8K のメモリを割り付けますが、この値は 64 個分のオブジェクトまたは記述子に対して十分なメモリの量です。その後、メモリは 2K ずつのまとまりで割り付けられます。また Adaptive Server は、起動時にハッシュ・テーブルに対して 2,048 バイトを割り付けます。

「最低使用頻度」(LRU) メカニズムは、最近参照されていないオブジェクトまたは識別子をメモリから削除することによって、スキーマ・キャッシュ・サイズを管理できる状態にしています。したがって、RepAgent はすべての複写オブジェクトについて記述するための、十分なメモリを必要とはしません。RepAgent が最低限必要とするのは、1 つの複写オブジェクトについて記述するのに十分なメモリだけです。

トランザクション・キャッシュ

RepAgent はオープン・トランザクション 1 つにつき 256 バイトを要求します。トランザクション・キャッシュ・メモリは、2K ずつのまとまりで割り付けられます。トランザクションがコミットされたりアボートされたりする場合、新しいメモリが割り付けられるまでの間使用する必要があるプールには、空のメモリがあります。

RepAgent は、最大数のオープン・トランザクション用のメモリを要求します。十分なメモリを使用できない場合、RepAgent は停止します。

Adaptive Server は、起動時にハッシュ・テーブルに対して 2,048 バイトを割り付けます。

text および image キャッシュ

text および image のキャッシュ・サイズを指定します。

`text` キャッシュと `image` キャッシュは、`text` データと `image` データのサイズに影響されません。反対に、使用されるメモリの量は、`text` データと `image` データが入っている複写テーブルの数と、`text` データと `image` データが入っているテーブル内のカラムの数によって変わります。`text` データと `image` データが入っている複写テーブル 1 つにつき 170 バイト必要です。また、複写カラム 1 つにつき 52 バイト必要です。

`text` キャッシュ・メモリと `image` キャッシュ・メモリは、2K ずつのまとまりで割り付けられます。メモリが割り付けられるのは、`text` データと `image` データが入っている複写テーブルが存在している場合に限ります。

`text` キャッシュと `image` キャッシュは空きメモリ・プールを使用し、すべての `text` データと `image` データに対して十分なメモリを要求します。

その他のメモリ

Replication Server のその他のメモリ要件を指定します。

- RepAgent が有効である場合、Adaptive Server は RepAgent 用に追加のプロセス記述子を 1 つ割り付けます。
- RepAgent は Client-Library を使用して、Replication Server に接続します。**ct-lib** は必要に応じて直接メモリを割り付けます (動的メモリ)。

CPU 使用率

Replication Server は、マルチプロセッサ・プラットフォームでもシングルプロセッサ・プラットフォームでも動作します。Replication Server のマルチスレッド・アーキテクチャは、両方のハードウェア構成をサポートします。

Replication Server の構成で対称型マルチプロセッサ (SMP) 機能をオフにすると、Replication Server のスレッドは順番に実行されます。サーバ全体の相互排他ロック (mutex) によって逐次スレッドの実行が強制されるため、異なるプロセッサでスレッドが同時に実行されることはありません。

Replication Server の構成で SMP 機能をオンにすると、Replication Server のスレッドを並列に実行できるため、パフォーマンスと効率を高めることができます。サーバ全体の mutex は無効になり、個々のスレッドは、グローバル・データ、サーバ・コード、システム・ルーチンの安全を保障するためにスレッド管理技術を組み合わせて使用します。

マルチプロセッサ・マシン上で SMP を有効にするには、**configure replication server** を **smp_enable** オプションとともに使用します。例：

```
configure replication server set smp_enable to 'on'
```

Replication Server のマルチプロセッサ・サポートは、Open Server のマルチプロセッサ・サポートをベースとしています。つまり、単一のプロセスで複数のスレッドが実行されます。Replication Server は、UNIX プラットフォームでは POSIX スレッド・ライブラリ、Windows プラットフォームでは WIN32 スレッド・ライブラリを使用します。Open Server のマルチプロセッサ・マシンに対するサポートの詳細については、『Open Server Server-Library/C リファレンス・マニュアル』を参照してください。

ネットワークの稼働条件

Replication Server のネットワークの稼働条件を指定します。

アウトバウンド・キューへのデータ挿入速度を算出すると、必要なネットワーク・スループットについても検討することができます。Replication Server には、メッセージがキューに追加される速度より速く、キューから送り出せるだけのネットワーク・スループットが必要です。ネットワーク・メッセージは、アウトバウンド・キューに書き込まれるメッセージよりわずかながら大きいのが普通です。

キューが一気に満杯になり、ネットワークの帯域幅が狭いためにメッセージの送出がはかどらないということがあります。このような問題もキュー・サイズを見

積もるときには考慮に入れ、キューがコネクションや送信先サイトの障害だけでなくデータの急速な流入にも対処できるようにすることが必要です。さらに、別々の負荷を処理する場合には、モニタ・データとカウンタ・データを収集して、キュー・サイズを正確に計算できるようにします。

追加の説明や情報の入手

Sybase Getting Started CD、製品マニュアル Web サイト、オンライン・ヘルプを利用すると、この製品リリースについて詳しく知ることができます。

- Getting Started CD (またはダウンロード) – PDF フォーマットのリリース・ノートとインストール・ガイド、その他のマニュアルや更新情報が収録されています。
- Sybase 製品マニュアル Web サイト (http://sybooks.sybase.com/) にある製品マニュアルは、Sybase マニュアルのオンライン版であり、標準の Web ブラウザを使用してアクセスできます。マニュアルはオンラインで参照することも PDF としてダウンロードすることもできます。この Web サイトには、製品マニュアルの他に、EBFs/Maintenance、Technical Documents、Case Management、Solved Cases、Community Forums/Newsgroups、その他のリソースへのリンクも用意されています。
- 製品のオンライン・ヘルプ (利用可能な場合)

PDF 形式のドキュメントを表示または印刷するには、Adobe の Web サイトから無償でダウンロードできる Adobe Acrobat Reader が必要です。

注意： 製品リリース後に追加された製品またはマニュアルについての重要な情報を記載したさらに新しいリリース・ノートを製品マニュアル Web サイトから入手できることがあります。

サポート・センタ

Sybase 製品に関するサポートを得ることができます。

組織でこの製品の保守契約を購入している場合は、サポート・センタとの連絡担当者が指定されています。マニュアルだけでは解決できない問題があった場合には、担当の方を通して Sybase 製品のサポート・センタまでご連絡ください。

Sybase EBF と Maintenance レポートのダウンロード

EBF と Maintenance レポートは、Sybase Web サイトからダウンロードしてください。

1. Web ブラウザで http://www.sybase.com/support を指定します。

2. メニュー・バーまたはスライド式メニューの [Support (サポート)] で [EBFs/Maintenance (EBF/メンテナンス)] を選択します。
3. ユーザ名とパスワードの入力が 求められたら、MySybase のユーザ名とパスワードを入力します。
4. (オプション) [Display (表示)] ドロップダウン・リストからフィルタを指定し、期間を指定して、[Go (実行)] をクリックします。
5. 製品を選択します。

   鍵のアイコンは、「Authorized Support Contact」として登録されていないため、一部の EBF/Maintenance リリースを ダウンロードする権限がないことを示しています。未登録ではあるが、Sybase 担当者またはサポート・センタから有効な情報を得ている場合は、[My Account (マイ・アカウント)] をクリックして、「Technical Support Contact」役割を MySybase プロファイルに追加します。
6. EBF/Maintenance レポートを表示するには [Info] アイコンをクリックします。ソフトウェアをダウンロードするには製品の説明をクリックします。

Sybase 製品およびコンポーネントの動作確認

動作確認レポートは、特定のプラットフォームでの Sybase 製品のパフォーマンスを検証します。

動作確認に関する最新情報は次のページにあります。

- パートナー製品の動作確認については、http://www.sybase.com/detail_list?id=9784 にアクセスします。
- プラットフォームの動作確認については、http://certification.sybase.com/ucr/search.do にアクセスします。

MySybase プロファイルの作成

MySybase は無料サービスです。このサービスを使用すると、Sybase Web ページの表示方法を自分専用にカスタマイズできます。

1. http://www.sybase.com/mysybase を開きます。
2. [Register Now (今すぐ登録)] をクリックします。

アクセシビリティ機能

アクセシビリティ機能を使用すると、身体障害者を含むすべてのユーザーが電子情報に確実にアクセスできます。

Sybase 製品のマニュアルには、アクセシビリティを重視した HTML 版もあります。

オンライン・マニュアルは、スクリーン・リーダーで読み上げる、または画面を拡大表示するなどの方法により、視覚障害を持つユーザがその内容を理解できるよう配慮されています。

Sybase の HTML マニュアルは、米国のリハビリテーション法第 508 条のアクセシビリティ規定に準拠していることがテストにより確認されています。第 508 条に準拠しているマニュアルは通常、World Wide Web Consortium (W3C) の Web サイト用ガイドラインなど、米国以外のアクセシビリティ・ガイドラインにも準拠しています。

注意： アクセシビリティ・ツールを効率的に使用するには、設定が必要な場合もあります。一部のスクリーン・リーダーは、テキストの大文字と小文字を区別して発音します。たとえば、すべて大文字のテキスト (ALL UPPERCASE TEXT など) はイニシャルで発音し、大文字と小文字の混在したテキスト (Mixed Case Text など) は単語として発音します。構文規則を発音するようにツールを設定すると便利かもしれません。詳細については、ツールのマニュアルを参照してください。

Sybase のアクセシビリティに対する取り組みについては、Sybase Accessibility サイト (http://www.sybase.com/products/accessibility) を参照してください。このサイトには、第 508 条と W3C 標準に関する情報へのリンクもあります。

製品マニュアルには、アクセシビリティ機能に関する追加情報も記載されています。

索引

数字

A

B

C

D

E

G

I

J

L

M

O

P

R

S

T

U

V

W

あ

い

う

お

か

き

く

け

こ

さ

し

す

せ

そ

ひ

ふ

へ