SYBASE®
An SAP® Company

管理ガイド 第 2 巻

Replication Server® 15.7.1

ドキュメント ID： DC35494-01-1571-01
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

Replication Server の検証とモニタリング

Replication Server® の検証とモニタリングでは、エラー・ログのチェック、複写システムのコンポーネントの実行確認、システム・コンポーネントとシステム・プロセスのステータスのモニタリングについて説明します。

複写システムには、データ・サーバと Replication Server があります。また、異機種データ・サーバ用の Replication Agent も組み込むことができます。Adaptive Server の Replication Agent は RepAgent であり、これは Adaptive Server スレッドの 1 つです。

注意： 異機種データ・サーバ用の Replication Agent を使用している場合は、Replication Agent のトラブルシューティングの方法について、使用しているデータ・サーバ用の Replication Agent に関するマニュアルを参照してください。

完全に動作している複写システムでは、すべてのデータ・サーバ、Replication Server、Replication Agent、それらの内部スレッドとその他のコンポーネントが実行されています。複写システムで実行できる基本的なトラブルシューティング作業には、次のものがあります。

- エラー・ログでステータスやエラー・メッセージをチェックする
- システム・サーバにログインして、すべてのスレッドが機能しており、ルートとコネクションが所定の通りであり、interfaces ファイル情報が正しいことをチェックする
- Replication Server とそのスレッドをモニタし、パーティション・スレッショルド・レベルをチェックする

Replication Server のモニタリングとトラブルシューティングの詳細については、『Replication Server トラブルシューティング・ガイド』を参照してください。

複写システムのログ・ファイルを使ったエラーのチェック

Replication Server は、内部エラーを含むステータス・メッセージとエラー・メッセージを Replication Server のエラー・ログ・ファイルに記録します。

現在のログ・ファイルへのパスを表示するには、**admin log_name** コマンドを使用します。ログ・ファイルのデフォルト名は、`repserver.log` です。このデフォルト名は、**repserver** に **-E** オプションを指定して実行して、新しいログ・ファイル名を指定すると変更できます。

これらのコマンドの詳細については、『Replication Server リファレンス・マニュアル』の「Replication Server コマンド」を参照してください。

内部エラーとは、Replication Server で実行できるアクションが、スタックのダンプと終了だけであるエラーのことです。Replication Server は、診断のために、その実行スタックのトレースをログに出力し、エラー発生時のステータス・レコードを残します。

エラー・ログ・ファイルのメッセージは、削除しないかぎり蓄積され続けます。このため、Replication Server の停止時にログ・ファイルをトランケートすることもできます。また、Replication Server のログ・ファイルをクローズして、**admin set_log_name** コマンドを使用して新しいログ・ファイルを開始することもできます。

Replication Server のログ・ファイルには、**create subscription** や **create route** などの非同期コマンドの実行中に生成されたメッセージが保存されます。これらのコマンドは、コマンド終了後も処理を継続します。非同期コマンドの実行中は、この手順の影響を受ける Replication Server のログ・ファイルに特に注意してください。

ログ・ファイルを使用できない場合、重要なエラー情報は、標準エラー出力ファイルに書き込まれます。このファイルは、端末で表示したり、ファイルにリダイレクトしたりできます。

複写システムの検証

複写定義やサブスクリプションを作成したり、システムで診断を実行したりする場合は、複写システム全体が動作しているかどうか確認します。

前提条件

複写システムが動作していることを確認する前に、停止しているスレッドがないことを確認してください。

手順

エラーが発生した場合は、システムの動作確認によって、スレッドやコンポーネントが実行されていない可能性、またはルートやコネクションが正しく設定されていない可能性を除外できます。

Replication Server スレッドが実行中であることを確認するには、**admin who_is_down** を実行します。このコマンドでは、実行されていないスレッドのみが表示されます。または、**admin who** を実行し、すべてのスレッドに関する情報を表示することもできます。

1. 複写システム・サーバと Replication Agent が実行中で使用できることを確認します。

   プライマリ・サイトで、次のサーバにログインします。

- プライマリ・データベースとその Replication Agent を備えたデータ・サーバ
  Adaptive Server を使用している場合は、Adaptive Server で **sp_help_rep_agent** を実行して、RepAgent スレッドのステータス情報を表示します。
- プライマリ・データベースを管理する Replication Server
- プライマリ Replication Server 用の RSSD (およびその Replication Agent)
  Adaptive Server を使用している場合は、Adaptive Server で **sp_help_rep_agent** を実行して、RepAgent スレッドのステータス情報を表示します。

レプリケート・サイトで、次のサーバにログインします。

- レプリケート・データベース、およびそれらのデータベースで要求ファンクションが実行されている場合は、その Replication Agent を備えたデータ・サーバ
  Adaptive Server を使用している場合は、Adaptive Server で **sp_help_rep_agent** を実行して、RepAgent スレッドのステータス情報を表示します。
- レプリケート・データベースを管理する Replication Server
- レプリケート Replication Server 用の RSSD (およびその Replication Agent)
  Adaptive Server を使用している場合は、Adaptive Server で **sp_help_rep_agent** を実行して、RepAgent スレッドのステータス情報を表示します。

2. Replication Server で **admin show_connections** コマンドを使用して、以下のルートとコネクションが所定どおりであることを確認します。
   - プライマリ Replication Server から各レプリケート Replication Server へのルート
   - プライマリ Replication Server とプライマリ・データベースとの間のデータベース・コネクション
   - レプリケート Replication Server からプライマリ Replication Server へのルート (要求ファンクションが実行されるレプリケート・データベースがレプリケート Replication Server によって管理されている場合)
   - 各レプリケート Replication Server とそのレプリケート・データベースの間のデータベース・コネクション
3. interfaces ファイルのエントリの正確さを確認します。

   サブスクリプションを作成する場合は、プライマリ・データ・サーバのエントリが、レプリケート Replication Server 用の interfaces ファイル内に存在することを確認します アトミックまたはノンアトミック・マテリアライゼーションを使用している場合、レプリケート Replication Server は、プライマリ・データ・サーバへの直接コネクションを介して最初のローを取得します。
4. **admin who** を使用して、次の Replication Server スレッドが実行されているかどうかを確認します。
   - データ・サーバ・インタフェース (DSI)

- Replication Server インタフェース (RSI)
- ディストリビュータ (DIST)
- ステーブル・キュー・マネージャ (SQM)
- ステーブル・キュー・トランザクション (SQT) インタフェース
- RepAgent ユーザ

参照：
- 複写システムのスレッドのステータス表示 (10 ページ)

Replication Server のモニタリング

複写システムの動作中は、そのコンポーネントやプロセスをモニタすることが必要な場合があります。

次のような操作が必要な場合があります。

- 複写システム・サーバをモニタする。
- DSI、RSI、および他のスレッドのステータスをモニタする。
- システム情報コマンドを使用して、Replication Server のさまざまな情報を取得する方法

サーバ・ステータスの確認

サーバのステータスは、次のいくつかの方法によって確認できます。

- **isql** を使用して、各サーバにログインします。ログインが成功すれば、サーバは稼働しています。
- 各 Adaptive Server とその RepAgent スレッド、その他の Replication Agent (ある場合)、Replication Server にログインして、そのステータスを表示するスクリプトを作成する方法。スクリプト内のすべてのサーバが interfaces ファイルに含まれていることを確認してください。

ログインが失敗した場合、次のいずれかの問題が原因である可能性があります。

問題：間違った名前を入力したか、使用している interfaces ファイルにそのサーバのエントリがない。

```
DB-LIBRARY error:
  Server name not found in interface file.
```

問題：サーバは実行されているが、指定されたログイン名またはパスワードが間違っている。

```
DB-LIBRARY error:
  Login incorrect.
```

問題：サーバが実行されていない。

```
Operating-system error:
  Invalid argument
 DB-LIBRARY error:
  Unable to connect: Server is unavailable
  or does not exist.
```

問題：interfaces ファイルが見つからない。

```
Operating-system error:
  No such file or directory
 DB-LIBRARY error:
  Could not open interface file.
```

問題：interfaces ファイルは存在するが、ファイルにアクセスするためのパーミッションがない。

```
Operating-system error:
  Permission denied
 DB-LIBRARY error:
   Could not open interface file
```

ログインできないのにエラー・メッセージを受信しない場合は、サーバが処理を停止したことが考えられます。問題の特定に支援が必要な場合は、Sybase 製品の保守契約を結んでいるサポート・センタに問い合わせてください。

ステータスの視覚的なモニタリング

Replication Manager GUI を使用して、Replication Monitoring Services (RMS) のステータスをモニタします。Replication Manager は、RMS を介して環境内のサーバに接続します。

Replication Manager により、環境またはオブジェクトのステータスが図示されます。

環境のステータスとは、そのコンポーネントの状態のことです。オブジェクトのステータスには、現在の状態、および状態の理由のリストが含まれます。各オブジェクトのステータスは、オブジェクト・アイコン、親オブジェクトの [詳細] リスト、そのオブジェクトの [プロパティ] ダイアログ・ボックスに表示されます。サーバ、コネクション、ルート、キューのステータスをモニタできます。

『Replication Server 管理ガイド 第 1 巻』の「Sybase Central での Replication Server の管理」を参照してください。

複写システムのスレッドのステータス表示

関連する **admin who** コマンドまたはシステム・プロシージャを持つさまざまなタイプの現在の Replication Server スレッドに関する一般情報を表示します。

表 1 : Replication Server スレッドのモニタリング

Replication Server スレッド	コマンド
ディストリビュータ (DIST) ― SQT と SQM を使用して、インバウンド・キューからトランザクションを読み取る。	**admin who, dist**
データ・サーバ・インタフェース (DSI) ― データ・サーバにトランザクションを送信する。	**admin who, dsi**
REP AGENT USER – データ・サーバからのトランザクションが有効であることを確認し、そのトランザクションをインバウンド・キューに書き込む。	**admin who** **注意：** Adaptive Server で RepAgent スレッドのステータスを表示するには、**sp_who** または **sp_help_rep_agent** を使用する。
Replication Server インタフェース (RSI) ― 各送信先 Replication Server にログインし、ステーブル・キューから送信先サーバにコマンドを転送する。	**admin who, rsi**
ステーブル・キュー・マネージャ (SQM) ― Replication Server のステーブル・キューを管理する。	**admin who, sqm**
ステーブル・キュー・トランザクション・インタフェース (SQT) ― キュー内のトランザクションを読み取り、SQT リーダに渡す。	**admin who, sqt**

次を参照してください。

- トラブルシューティングのためのコマンド出力の説明については、『Replication Server トラブルシューティング・ガイド』
- 『Replication Server リファレンス・マニュアル』の「Replication Server コマンド」の「**admin who**」
- 『Replication Server リファレンス・マニュアル』の「Adaptive Server コマンドとシステム・プロシージャ」の「**sp_help_rep_agent**」
- 『Adaptive Server Enterprise』の「リファレンス・マニュアル：プロシージャ」にある「システム・プロシージャ」の「**sp_who**」

システム情報コマンドの使用

Replication Server には、**admin who** 以外にも、Replication Server のモニタリングに役立つ **admin** コマンドが用意されています。

各コマンドの詳細については、『Replication Server リファレンス・マニュアル』の「Replication Server コマンド」を参照してください。

表 2 : システム情報コマンドの概要

コマンド	説明
admin disk_space	Replication Server がアクセスするディスク・パーティションの使用率を表示する。
admin echo	ローカルの Replication Server が実行されているかどうかを調べる。
admin get_generation	リカバリ・オペレーションで使用されたプライマリ・データベースの生成番号を取得する。
admin health	Replication Server 全体のステータスを表示する。
admin log_name	現在のログ・ファイルのパスを表示する。
admin logical_status	ウォーム・スタンバイ・アプリケーションで使用される論理データベース・コネクションのステータスを表示する。
admin pid	Replication Server のプロセス ID を表示する。
admin quiesce_check	Replication Server のキューがクワイスされているかどうかを調べる。
admin quiesce_force_rsi	Replication Server がクワイス状態であるかどうかを調べる。また、アウトバウンド・メッセージを配信するように Replication Server に指示する。
admin rssd_name	RSSD のデータ・サーバとデータベースの名前を表示する。
admin security_property	Replication Server によってサポートされている、ネットワークベースのセキュリティ・システムのセキュリティ機能を表示する。
admin security_setting	特定のターゲット・サーバにおけるネットワークベースのセキュリティ設定を表示する。
admin set_log_name	Replication Server の既存のログ・ファイルをクローズし、新しいログ・ファイルをオープンする。
admin show_connections	Replication Server との間のすべてのコネクションとルートに関する情報を表示する。
admin show_function_classes	既存のファンクション文字列クラスとその親クラスの名前を表示して、継承のレベル番号を示す。

コマンド	説明
admin show_route_versions	Replication Server から始まるルートと、Replication Server で終了するルートのバージョン番号を表示する。
admin show_site_version	Replication Server のサイト・バージョンを表示する。
admin sqm_readers	インバウンド・キューを読み取っているスレッドに関する情報を表示する。
admin stats	Replication Server のカウンタに関する情報と統計を表示する。**admin statistics** の置き換えコマンド。
admin statistics, md	メッセージ配信とカウンタに関する統計を表示する。
admin statistics, mem	メモリ使用率に関する統計を表示する。
admin statistics, reset	メッセージ配信の統計を再設定する。
admin version	実行中の Replication Server のバージョン (ソフトウェア・バージョン) を表示する。
admin who	Replication Server 内のすべてのスレッドに関する情報を表示する。
admin who, dsi	データ・サーバに接続する DSI スレッドに関する情報を表示する。
admin who, rsi	他の Replication Server に接続する RSI スレッドに関する情報を表示する。
admin who, sqm	SQM によって管理されるすべてのキューに関する情報を表示する。
admin who, sqt	SQT によって管理されるすべてのキューに関する情報を表示する。
admin who_is_down	**admin who** を実行した場合と同じ情報を表示する。ただし、停止中のスレッドに関する情報のみを表示する。
admin who_is_up	**admin who** を実行した場合と同じ情報を表示する。ただし、実行中のスレッドに関する情報のみを表示する。

スレッショルド・レベルの設定と使用

パーティションが満杯に近くなったときに警告が表示されるように Replication Server を設定できます。

Replication Server のメッセージ受信量が送信量よりも多い場合、ステーブル・キューのパーティションが満杯になります。たとえば、プライマリ・サイトとレプリケート・サイトの間のネットワークが停止した場合、プライマリ・サイトの Replication Server は配信できないメッセージをキューに入れます。ネットワークの

サービスが再開すると、そのメッセージは配信できるようになり、その後、プライマリ Replication Server のパーティションから削除されます。

パーティションが満杯になると、送信側は Replication Server にメッセージを配信できなくなり、メッセージは、前のサイトのパーティションとプライマリ・データベースのトランザクション・ログにバックアップされます。

警告！ 状況が改善されないと、RepAgent がデータベース・ログのセカンダリ・トランケーション・ポイントを更新できなくなり、トランザクション・ログが満杯になります。このため、クライアントがプライマリ・データベースでトランザクションを実行できなくなります。

パーティションが満杯に近くなったときに警告が表示されるように Replication Server を設定するには、**configure replication server** を **sqm_warning_thr1**、**sqm_warning_thr2**、および **sqm_warning_thr_ind** と併用します。『Replication Server リファレンス・マニュアル』の「Replication Server コマンド」の「**configure replication server**」を参照してください。

パーティションのパーセンテージのモニタリング

Replication Server のパーティションのパーセンテージで変更をモニタリングするには、ログ・ファイルのメッセージを使用します。

Replication Server は 1MB のパーティション・セグメント単位で動作します。Replication Server によるパーティション・セグメントの割り付け、または割り付け解除では、必ず次のパーセンテージが計算されます。

- 使用中の合計パーティション・セグメントのパーセンテージ
- 影響を受けたステーブル・キューによって使用されている合計パーティション・セグメントのパーセンテージ

使用中のパーティション・セグメントのパーセンテージが **sqm_warning_thr1** または **sqm_warning_thr2** に指定されたパーセンテージを上回ると、メッセージがログ・ファイルに書き込まれます。

```
WARNING: Stable Storage Use is Above threshold percent
```

このメッセージが頻繁に表示される場合、Replication Server にパーティションを追加するか、キューが満杯になる原因となる障害が再発しないようにする必要があります。

一度上回ったパーセンテージが **sqm_warning_thr1** または **sqm_warning_thr2** に指定されたパーセンテージを下回ると、元の警告の原因となった状態が解消されたことを示すメッセージが書き込まれます。

```
WARNING CANCEL: Stable Storage Use is Below threshold percent
```

1 つのステーブル・キューによって使用されている合計領域のパーセンテージが **sqm_warning_thr_ind** に指定されたパーセンテージを上回ると、影響を受けたス

テーブル・キューによって使用されている合計パーティション・セグメントのパーセンテージに基づいて、警告メッセージがトリガされます。

```
WARNING: Stable Storage Use by queue name is Above threshold percent
```

この警告メッセージは、何らかの問題が発生して、特定のステーブル・キューが満杯になり、全パーティション領域のうち必要以上に多くの領域が使用されていることを知らせるものです。たとえば、ルートが一定時間サスペンドされたときに、そのステーブル・キューにより占有されるパーティションが増え続けると警告がトリガされます。

ステーブル・キューによって使用される合計パーティション領域のパーセンテージが **sqm_warning_thr_ind** のパーセンテージを下回ると、Replication Server によってキャンセル・メッセージが書き込まれます。

```
WARNING CANCEL: Stable Storage Use by queue name is Below threshold
percent.
```

データベース・オペレーションのカスタマイズ

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を作成および変更し、Adaptive Server以外のデータベース・サーバで複写定義を使用できるようにします。

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

Replication Server は、プライマリ・データベースからのコマンドを、insert、delete、select、begin transaction などのデータ・サーバのオペレーションを表す Replication Server ファンクションに変換します。これらのファンクションは、システム内のリモート Replication Server に分配されます。リモート Replication Server は、これらのオペレーションをリモート・データベースで実行します。

プライマリ Replication Server は、実際に複写データを更新するデータ・サーバのタイプに関係なく、同じ形式でファンクションを分配します。ファンクションは、データベースごとに固有ではありません。ファンクションには、オペレーションを実行するために必要なすべてのデータが含まれていますが、送信先データ・サーバでオペレーションを実行するために必要な構文がファンクションで指定されているのではありません。

リモート Replication Server は、ファンクションを、それらが実行される送信先データ・サーバに固有のコマンドに変換します。ファンクション文字列には、データベース固有のファンクション実行命令が含まれます。データベースを管理するレプリケート Replication Server は、適切なファンクション文字列を使用して、データ・サーバに対する一連の命令にファンクションをマップします。たとえば、**rs_insert** ファンクションのファンクション文字列は、レプリケート・データベースに適用される実際の言語を指定します。

このようにファンクションとデータ・サーバ・コマンドが分けられているため、異機種データ・サーバ間で複写データを管理することができます。Replication Server では、ファンクション文字列をカスタマイズして SQL コマンドへの Replication Server ファンクションのマップ方法を指定できます。データ・サーバのオペレーションをカスタマイズする必要がある場合は、ファンクション文字列を作成できます。複写データ・アプリケーションは、送信先データベースでのオペレーションの実行方法を変更することによってカスタマイズします。

ファンクション文字列はファンクション文字列クラスにグループ化されており、データ・サーバに応じてコマンドへのファンクションのマッピングをグループ化

できます。Replication Server には、Adaptive Server Enterprise、Oracle、Microsoft SQL Server、IBM DB2 UDB などの各データベース用のファンクション文字列クラスが用意されています。また、新しい派生ファンクション文字列クラスを作成すると、特定のファンクション文字列をカスタマイズして、それ以外のすべてのファンクション文字列をこれらのクラス、または他のクラスから継承できます。さらに、まったく新しいクラスを作成して新しいファンクション文字列を作成することもできます。

リモート・データベースでストアド・プロシージャを実行できるように、複写ファンクション用のファンクション文字列を作成する必要がある場合もあります。複写ファンクションによっては、送信先データベースが使用するファンクション文字列クラスのファンクション文字列を、Replication Server が自動的に生成しない場合があります。このような複写ファンクションに対しては、ファンクション文字列を作成する必要があります。

ファンクション、ファンクション文字列、クラスの処理

ファンクションやファンクション文字列を使用して、データベースのオペレーションをカスタマイズする方法はいくつかあります。

次のことができます。

- 特定のタイプのデータベースで使用する新しいファンクション文字列クラスを作成し、一部またはすべてのファンクション文字列をカスタマイズします。
- アトミック・マテリアライゼーションの場合、レプリケート・データベース・コネクションではなく、プライマリ・データベース・コネクションに関連付けられているファンクション文字列クラスのファンクションを使用します。
- システム提供ファンクション文字列クラス **rs_sqlserver_function_class** のファンクション文字列を変更します。
- システム提供ファンクション文字列クラス **rs_default_function_class** のファンクション文字列を直接的または間接的に継承するファンクション文字列クラスを作成します。
- システムが提供するファンクション文字列クラスを ASE 以外のデータ・サーバ (**rs_iq_function_class**、**rs_db2_function_class**、**rs_mss_function_class**、または **rs_oracle_function_class**) に使用します。異機種データ型サポート (HDS) 機能を使用したデータ型変換の詳細については、『Replication Server 管理ガイド 第 1 巻』の「複写テーブルの管理」の「HDS を使用したデータ型の変換」を参照してください。

ファンクション、ファンクション文字列、クラスは、**isql** を使ってコマンド・ラインに入力する Sybase Central™ または RCL コマンドを使用して操作できます。

システム・ファンクションの詳細については、『Replication Server リファレンス・マニュアル』の「Replication Server システム・ファンクション」を参照してください。

参照：
- ファンクション文字列クラスの管理 (31 ページ)
- ファンクション文字列の管理 (37 ページ)

関数

Replication Server では、主に次の 2 つのタイプのファンクションを使用します。

Replication Server の主なタイプのファンクションには、次のものがあります。

- システム・ファンクション
- ユーザ定義ファンクション

必要に応じて、いずれかのタイプのファンクションのカスタム・ファンクション文字列を作成できます。

参照：
- ファンクション文字列の管理 (37 ページ)

システム・ファンクション

システム・ファンクションは、Replication Server によって提供されているファンクション文字列を使用するか、新しいデータベースを複写システムにインストールすると使用可能になるデータ・サーバのオペレーションを表します。

アプリケーションで必要な場合以外は、システム・ファンクションのファンクション文字列をカスタマイズする必要はありません。システム提供クラスによって、システム・ファンクションのファンクション文字列は自動的に生成されます。

システム・ファンクションには、次のものがあります。

- insert、update、delete、select、select with holdlock など、データ操作のオペレーションを表すファンクション。これらのシステム・ファンクションは、複写定義スコープを持ちます。
- トランザクション制御命令を表すファンクション。これらのファンクションには、begin transaction や commit transaction などのオペレーションがあります。これらのシステム・ファンクションは、ファンクション文字列クラス・スコープを持ちます。

参照：
- ファンクションのスコープ (18 ページ)
- システム・ファンクションの概要 (19 ページ)

ユーザ定義ファンクション

ユーザ定義ファンクションを使用すると、Replication Server を使用して複写システムのサイト間で複写ストアド・プロシージャを分配できます。

rs_default_function_class からファンクション文字列を直接的または間接的に継承するファンクション文字列クラスを使用しない場合は、ユーザ定義ファンクションのファンクション文字列を作成する必要があります。ユーザ定義ファンクションには、次のものがあります。

- ファンクション複写定義に関連付けられているストアド・プロシージャを複写する場合に使用されるファンクション。Replication Server では、ファンクション複写定義を作成するときに、自動的にこのタイプのユーザ定義ファンクションが作成されます。『Replication Server 管理ガイド 第 1 巻』の「複写ファンクションの管理」を参照してください。
- テーブル複写定義に関連付けられているストアド・プロシージャを複写する場合に使用されるファンクション。このタイプのユーザ定義ファンクションは、ユーザが作成して管理します。
  非同期プロシージャを使用して、テーブル複写定義に対応するストアド・プロシージャを複写できます。

ユーザ定義ファンクションは、ファンクション・スコープのタイプとして複写定義スコープを持ちます。

ユーザ定義ファンクションのファンクション文字列は、複写定義を作成したプライマリ Replication Server で作成する必要があります。ファンクション複写定義を使用している場合は、『Replication Server 管理ガイド 第 1 巻』の「複写ファンクションの管理」の「複写ファンクションの使用」を参照してください。

参照：

- 非同期プロシージャ (445 ページ)
- ファンクションのスコープ (18 ページ)

ファンクションのスコープ

ファンクションのスコープによって、ファンクションが適用するオブジェクト (複写定義またはファンクション文字列クラス) を定義します。

ファンクション文字列をカスタマイズする場所 (プライマリ Replication Server またはレプリケート Replication Server) を決定するには、ファンクションのスコープを把握する必要があります。

ファンクション文字列クラス・スコープ

ファンクション文字列クラス・スコープを持つファンクションは、そのクラスに対して一度定義されます。ファンクション文字列クラス・スコープを持つファンクションには、トランザクション制御命令 (**rs_begin**、**rs_commit**、**rs_marker** など)

を表し、データ操作を実行しないシステム・ファンクションがあります。ユーザ定義ファンクションのファンクション文字列は、クラス・スコープを持ちません。

ファンクション文字列クラス・スコープを持つファンクションのファンクション文字列は、ファンクション文字列クラスのプライマリ Replication Server でカスタマイズする必要があります。

複写定義スコープ
複写定義スコープを持つファンクションは、特定のテーブル複写定義やファンクション複写定義について一度定義されますが、複数のファンクション文字列を持つこともあります。

複写定義スコープを持つファンクションには、次のものがあります。

- データ操作のオペレーションを実行するシステム・ファンクション (**rs_insert**、**rs_delete**、**rs_update**、**rs_select**、**rs_select_with_lock** に加え、`text` データ、`unitext` データ、`image` データの複写に使用する特殊なファンクションなど)。
- テーブル複写定義またはファンクション複写定義用のユーザ定義ファンクション。
  複写定義スコープを持つシステム・ファンクションは、複写定義が作成された Replication Server でカスタマイズする必要があります。複写定義スコープを持つユーザ定義ファンクションは、複写定義が作成された Replication Server でカスタマイズする必要があります。

すべてのシステム・ファンクションの詳しい資料については、『Replication Server リファレンス・マニュアル』の「Replication Server システム・ファンクション」を参照してください。

参照：
- ファンクション文字列クラスのプライマリ・サイト (34 ページ)
- ファンクション文字列クラス・スコープを持つシステム・ファンクション (20 ページ)
- 複写定義スコープを持つシステム・ファンクション (23 ページ)

システム・ファンクションの概要

Replication Server には、ファンクション文字列クラス・スコープと複写定義スコープがあるシステム・ファンクションが用意されています。

すべてのシステム・ファンクションの詳しい資料については、『Replication Server リファレンス・マニュアル』の「Replication Server システム・ファンクション」を参照してください。

ファンクション文字列クラス・スコープを持つシステム・ファンクション

Replication Server には、ファンクション文字列クラス・スコープがあるシステム・ファンクションがいくつか用意されています。

Replication Server では、複写システムをインストールすると、各システム提供クラスに対してデフォルトで生成されたファンクション文字列が用意されます。

ファンクションの中には、すべての Replication Server アプリケーションで必要なものと、ウォーム・スタンバイ・アプリケーション、並列 DSI スレッド、コーディネート・ダンプなどの特定の場合にのみ適用されるものがあります。

デフォルト (**rs_sqlserver_function_class**) 以外のファンクション文字列クラスを使用し、ファンクション文字列の継承を使用しない場合は、使用するシステム・ファンクションのうち、ファンクション文字列クラス・スコープを持つものに対して、それぞれファンクション文字列を作成する必要があります。

クラス・スコープを持つシステム・ファンクションのファンクション文字列は、ファンクション文字列クラスのプライマリ・サイトの Replication Server でカスタマイズします。ファンクション文字列クラスのプライマリ・サイトを 1 つの Replication Server から別の Replication Server に割り当てるまたは変更することが必要な場合もあります。

表 3 : ファンクション文字列クラス・スコープを持つシステム・ファンクション

ファンクション名	説明
rs_batch_start	**rs_begin** 文の他に、コマンドのバッチの始まりを示すマークを付けるために必要な SQL 文を指定する。
rs_batch_end	コマンドのバッチの終わりを示すマークを付けるために必要な SQL 文を指定する。このファンクション文字列は、**rs_batch_start** とともに使用される。
rs_begin	トランザクションを開始する。
rs_check_repl	テーブルが複写するようマーク付けされているかどうかチェックする。
rs_commit	トランザクションをコミットする。
rs_dumpdb	コーディネート・データベース・ダンプを開始する。
rs_dumptran	コーディネート・トランザクション・ダンプを開始する。
rs_get_charset	データ・サーバによって使用される文字セットを返す。
rs_get_lastcommit	`rs_lastcommit` システム・テーブルからローを取得する。
rs_get_sortorder	データ・サーバによって使用されるソート順を返す。

ファンクション名	説明
rs_get_thread_seq	`rs_threads` システム・テーブルの指定されたエントリの現在のシーケンス番号を返す。このファンクションは、並列 DSI を使用している場合にのみ実行される。
rs_get_thread_seq_noholdlock	**noholdlock** オプションを使用して、**rs_threads** システム・テーブルの指定されたエントリの現在のシーケンス番号を返す。このスレッドは、dsi_isolation_level が 3 の場合に使用される。
rs_initialize_threads	`rs_threads` システム・テーブルの各エントリのシーケンスを 0 に設定する。このファンクションは、並列 DSI を使用している場合にのみ実行される。
rs_marker	サブスクリプション・マテリアライゼーションの調整に役立つ。このファンクションは、その最初のパラメータを独立したコマンドとして Replication Server に渡す。
rs_non_blocking_commit	Replication Server の非ブロッキング・コミットと、レプリケート・データ・サーバの対応するファンクションを調整する。 Adaptive Server 15.0 以降では **set delayed_commit on** ファンクション文字列に、Oracle 10g v2 では **alter session set commit_write = nowait;** ファンクション文字列にマップする。その他すべての Sybase 以外のデータベースでは、**rs_non_blocking_commit** は null にマップする。 **dsi_non_blocking_commit** の値が 1 ～ 60 の場合、DSI がレプリケート・データ・サーバに接続するたびに実行される。**dsi_non_blocking_commit** の値が 0 の場合は、**rs_non_blocking_commit** は実行されない。
rs_non_blocking_commit_flush	**dsi_non_blocking_commit** が有効になっている場合に、データベース・トランザクションがディスクにフラッシュされるようにする。 Adaptive Server 15.0 以降と Oracle 10g v2 以降では、対応するファンクション文字列にマップする。その他すべての Sybase 以外のデータベースでは、**rs_non_blocking_commit_flush** は null にマップする。 **rs_non_blocking_commit_flush** は **dsi_non_blocking_commit** での間隔の指定に従って 1 ～ 60 分間隔で実行される。**rs_non_blocking_commit_flush** は、**dsi_non_blocking_commit** がゼロの場合、実行されない。
rs_raw_object_serialization	Java カラムを直列化されたデータとして複写する。
rs_repl_off	スタンバイ・データベース・コネクションに対して Adaptive Server の複写をオフに設定する。

ファンクション名	説明
rs_repl_on	スタンバイ・データベース・コネクションに対して Adaptive Server の複写をオンに設定する。
rs_rollback	トランザクションをロールバックする。
rs_set_ciphertext	**set ciphertext on** をオンにする。これにより、**rs_default_function_class** および **rs_sqlserver_function_class** の暗号化カラムの複写が可能になる。その他のすべてのクラスについては、このファンクションは null に設定される。
rs_set_isolation_level	レプリケート・データ・サーバにトランザクションの独立性レベルを渡す。
rs_set_dml_on_computed	コネクションが確立されたときにレプリケート・データベース DSI で適用される。Repulication Server はコマンド **set dml_on_computed "on"** を **use database** 文の後に発行する。
rs_set_proxy	ユーザのパーミッション、ログイン名、サーバ・ユーザ ID を想定する。
rs_set_quoted_identifiers	データ・サーバへの DSI コネクションを設定して、引用符付き識別子をコネクション経由で送信できるようにする。 前提条件：データ・サーバでの引用符付き識別子の使用を有効にするには、**dsi_quoted_identifier** が "on" に設定され、**rs_set_quoted_identifier** に必要なコマンドが含まれている必要がある。Adaptive Server と Microsoft SQL Server の場合、コマンドは **set quoted_identifiers on** に設定される。
rs_thread_check_lock	DSI エグゼキュータ・スレッドがレプリケート・データベース・プロセスをブロックするロックを保持しているかどうかを判別する。
rs_triggers_reset	スタンバイ・データベース・コネクションに対して Adaptive Server でトリガをオフに設定する。
rs_trunc_reset	ウォーム・スタンバイ・データベースのセカンダリ・トランケーション・ポイントを再設定する。このファンクションは、ウォーム・スタンバイ・データベース作成時、またはスタンバイ・データベースへの切り替え時にのみ実行される。
rs_trunc_set	ウォーム・スタンバイ・データベースのセカンダリ・トランケーション・ポイントを設定する。このファンクションは、ウォーム・スタンバイ・データベース作成時、またはスタンバイ・データベースへの切り替え時にのみ実行される。
rs_update_threads	`rs_threads` テーブルの指定されたエントリのシーケンス番号を更新する。このファンクションは、並列 DSI を使用している場合にのみ実行される。

ファンクション名	説明
rs_usedb	データベース・コンテキストを変更する。

参照：

- ファンクション文字列クラスのプライマリ・サイトの変更 (35 ページ)

複写定義スコープを持つシステム・ファンクション

Replication Server には、複写定義スコープがあるシステム・ファンクションがいくつか用意されています。

Replication Server では、複写定義が作成されると、各システム提供クラスに対してデフォルトで生成されたファンクション文字列が用意されます。

すべての Replication Server アプリケーションに必要なファンクションもあれば、`text` データ型、`unitext` データ型、`image` データ型の複写、並列 DSI スレッド、サブスクリプション・マテリアライゼーションやマテリアライゼーション解除の実行など、特定の場合に適用されるファンクションもあります。

複写定義スコープを持つシステム・ファンクションのファンクション文字列は、複写定義が作成された Replication Server でカスタマイズします。

表 4 : 複写定義スコープを持つシステム・ファンクション

ファンクション名	説明
rs_datarow_for_writetext	`text` カラム、`unitext` カラム、または `image` カラム (Transact-SQL® の **writetext** コマンドや CT-Library ファンクションまたは DB-Library ファンクションで更新) に関連付けられているデータ・ローのイメージを提供する。
rs_delete	テーブルのローを削除する。
rs_get_textptr	`text` カラム、`unitext` カラム、`image` カラム、`rawobject` カラムのテキスト・ポインタを取得する。
rs_insert	テーブルにローを挿入する。
rs_select	サブスクリプション・マテリアライゼーションまたはマテリアライゼーション解除のため、テーブルからローを取得する。
rs_select_with_lock	holdlock を使用してサブスクリプション・マテリアライゼーションまたはマテリアライゼーション解除のローを取得する。
rs_textptr_init	`text` カラム、`unitext` カラム、`image` カラム、`rawobject` カラムのテキスト・ポインタを割り付ける。

ファンクション名	説明
rs_truncate	テーブルをトランケートする。
rs_update	テーブルのローを更新する。
rs_writetext	`text` データ、`unitext` データ、`image` データ、`rawobject` データを変更する。

ファンクション文字列

ファンクション文字列には、データベースでファンクションを実行するための命令が含まれます。

これらの命令は、データベースによって異なります。たとえば、Sybase 以外のデータベースは、Adaptive Server データベースとは異なる命令を必要とし、異なるファンクション文字列を持つ場合があります。

ファンクション文字列には、言語とリモート・プロシージャ・コール (RPC) の 2 つのフォーマットがあります。言語フォーマットのファンクション文字列には、データ・サーバが解析する SQL 文などのコマンドが含まれます。RPC フォーマットのファンクション文字列には、Open Server™ ゲートウェイ・アプリケーションや Adaptive Server データベースでレジスタード・プロシージャを実行するリモート・プロシージャ・コールが含まれます。どちらのファンクション文字列フォーマットにも、データ値で置換できる変数を含めることができます。ファンクション文字列で使用されるフォーマットは、データ・サーバのタイプや Replication Server とデータ・サーバとの対話方法によって決まります。出力テンプレートを変更するとファンクション文字列をカスタマイズできます。

ファンクション文字列は、ファンクション文字列クラスにグループ化されます。各データベース・コネクションには、レプリケート・データベースのタイプに応じたファンクション文字列クラスを割り当てる必要があります。Replication Server には、アクティブにサポートされているすべてのデータ・サーバ用にデフォルトのファンクション文字列を生成するファンクション文字列クラスが用意されています。

複写システムを設定したりシステムにデータベースを追加したりする場合は、ファンクション文字列の必要条件を考慮し、ファンクション文字列クラスの使い方やファンクション文字列のカスタマイズの必要性を判断します。

参照：

- 出力テンプレート (39 ページ)
- ファンクション文字列クラス (26 ページ)
- ファンクション文字列の管理 (37 ページ)

入力テンプレートと出力テンプレート

ファンクション文字列はすべて、特定のデータ・サーバに対してファンクションを実行するときに、出力テンプレートを使用して送信先データベースに命令を出します。

rs_select ファンクションと **rs_select_with_lock** ファンクションのファンクション文字列は、入力テンプレートと出力テンプレートの両方を使用し、サブスクリプション・マテリアライゼーションとマテリアライゼーション解除を実行します。

ファンクション文字列は、関連する入力テンプレートと出力テンプレートを変更することによってカスタマイズします。**rs_select** と **rs_select_with_lock** 以外のファンクションのファンクション文字列をカスタマイズするには、出力テンプレートのみを変更します。ファンクション文字列の変更方法は、ファンクション文字列のフォーマット (言語または RPC) によって異なります。

参照：

- ファンクション文字列の入力テンプレートと出力テンプレート (38 ページ)

カスタマイズされたファンクション文字列を使用するアプリケーション

カスタム・ファンクション文字列作成には、いくつかのアプリケーションがあります。

- データ・サーバに送信されるコマンドをフォーマットするファンクション文字列の出力テンプレートを変更し、任意のネイティブ・データベース言語 (Transact-SQL 以外も含む) でオペレーションを実行する。
- 1 つのファンクション文字列が指定された同一の複写定義に対する複数のサブスクリプションのマテリアライゼーションまたはマテリアライゼーション解除を実行する。
- 既存のシステム・ファンクション文字列の出力テンプレートを次のように変更します。
  - 監査情報を記録する。
  - RPC を実行する。
  - 同一のデータベース内の複数のレプリケート・テーブルにデータを複写する。
  - プライマリ・テーブルとは異なる名前、カラム名、カラム順のレプリケート・テーブルにデータを複写する。
    レプリケート Replication Server のバージョンが 11.5 以降である場合、レプリケート・テーブルの関連情報を指定するようにカスタマイズした複写定義を作成すると、同じタスクをより簡単に実行できます。『Replication Server 管理ガイド 第 1 巻』の「複写テーブルの管理」にある「複写定義の作成」の「テーブルごとに複数の複写定義を作成する方法」を参照してください。

複数のファンクション文字列を持つシステム・ファンクション

複写定義スコープがある他のシステム・ファンクションの同じ複写定義に対しては、複数のファンクション文字列のインスタンスを作成できます。

クラス・スコープのシステム・ファンクションの場合、各ファンクションはそのクラス内のファンクション文字列にマップされます。複写定義スコープを持つシステム・ファンクション **rs_insert**、**rs_delete**、**rs_update** は、それぞれ各複写定義のクラス内のファンクション文字列にマップされます。

複写定義スコープを持つその他のシステム・ファンクション (**rs_select**、**rs_select_with_lock**、**rs_datarow_for_writetext**、**rs_get_textptr**、**rs_textptr_init**、および **rs_writetext**) の同じ複写定義に対しては、複数のファンクション文字列のインスタンスを作成できます。この場合、ファンクション文字列の各インスタンスに、異なる名前を指定する必要があります。複数のファンクション文字列を指定できるシステム・ファンクションは、次のとおりです。

- **rs_select** ファンクションと **rs_select_with_lock** ファンクション ― 同じ複写定義に複数のサブスクリプションがある場合のサブスクリプション・マテリアライゼーションおよびマテリアライゼーション解除で使用されます。ファンクション文字列の各インスタンスには、複写定義内でユニークな名前を指定できます。ファンクション文字列の各インスタンスは、複写定義のサブスクリプションの作成に使用される **where** 句に対応しています。
- **rs_datarow_for_writetext**、**rs_get_textptr**、**rs_textptr_init**、**rs_writetext** ファンクション ― ファンクション文字列の各インスタンス。複写定義に指定された `text` カラム、`unitext` カラム、または `image` カラムには、ファンクション文字列の各インスタンス名を指定する必要があります。

ファンクション文字列クラス

各ファンクション文字列は、ファンクション文字列クラスに属します。ファンクション文字列クラスは、タイプや必要条件が同じようなデータベースで使用するファンクション文字列をグループ化したものです。

Replication Server は、送信先データベースのデータ・サーバに応じたファンクション文字列クラスに各データベース・コネクションを割り当てます。

Replication Server は、割り当てられたファンクション文字列クラスのファンクション文字列を使用して、データベースにファンクションを適用します。ファンクション文字列クラスには、システム・ファンクションや任意のユーザ定義ファンクションのファンクション文字列が含まれます。

ファンクション文字列がすべてのデータ・サーバで実行できる場合、そのファンクション文字列クラスを複数のデータベースで使用できます。たとえば、

Adaptive Server が管理する複数のデータベースがあるシステムでは、すべてのデータベースに対して **rs_sqlserver_function_class** を使用できます。

ECDA を使用してさまざまなデータ・サーバにアクセスする場合には、ASE 以外のデータ・サーバ間で 1 つのファンクション文字列クラスを使用することもできます。

システム提供クラス

Replication Server には、システム提供クラスと呼ばれる、Replication Server がサポートしているデータ・サーバ用のデフォルトのファンクション文字列を格納するファンクション文字列クラスが用意されています。

- **rs_sqlserver_function_class** ― このクラスには、デフォルトの Adaptive Server のファンクション文字列が用意されています。**rs_sqlserver_function_class** のデフォルトのファンクション文字列は、**rs_default_function_class** のものと同じです。デフォルトでは、**rs_sqlserver_function_class** は、**rs_init** を使用して複写システムに追加する Adaptive Server データベースに割り当てられます。
  このクラスのファンクション文字列はカスタマイズできます。ただし、このクラスはファンクション文字列クラスの継承に関与できません。ほとんどの場合には、**rs_sqlserver_function_class** を直接使用せず、**rs_default_function_class** を親クラスとして指定する派生クラスを使用することをおすすめします。
- **rs_default_function_class** ― このクラスには、デフォルトの Adaptive Server のファンクション文字列が用意されています。**rs_sqlserver_function_class** と **rs_default_function_class** のデフォルトのファンクション文字列は同じです。
  このクラスのファンクション文字列はカスタマイズできません。ただし、このクラスはファンクション文字列クラスの継承に関与できます。ほとんどの場合には、**rs_default_function_class** を直接使用せず、**rs_default_function_class** を親クラスとして指定する派生クラスを使用することをおすすめします。

  注意： システム提供ファンクション文字列クラス **rs_default_function_class** と **rs_sqlserver_function_class** には、**rs_dumpdb** と **rs_dumptran** 以外のすべてのシステム・ファンクションのデフォルトのファンクション文字列が含まれています。これらのファンクション用のファンクション文字列を使用する必要がある場合、派生クラスまたは **rs_sqlserver_function_class** でそのファンクション文字列を作成する必要があります。
- **rs_db2_function_class** ― このクラスには、DB2 固有のファンクション文字列が用意されています。『Replication Server 管理ガイド 第 1 巻』の「複写テーブルの管理」の「HDS を使用したデータ型の変換」で「クラス・レベル変換の作成」を参照してください。

DB2 ファンクション文字列が必要なほとんどの場合で、**rs_db2_function_class** を直接使用するのではなく、このクラスを親クラスとして指定する派生クラスを使用することがすすめられます。

- **rs_iq_function_class** – このクラスには、Sybase® IQ のファンクション文字列が用意されています。『Replication Server 管理ガイド 第 1 巻』の「複写テーブルの管理」の「HDS を使用したデータ型の変換」で「クラス・レベル変換の作成」を参照してください。
- **rs_msss_function_class** — このクラスには、Microsoft SQL Server のファンクション文字列が用意されています。『Replication Server 管理ガイド 第 1 巻』の「複写テーブルの管理」の「HDS を使用したデータ型の変換」で「クラス・レベル変換の作成」を参照してください。
- **rs_oracle_function_class** — このクラスには、Oracle のファンクション文字列が用意されています。『Replication Server 管理ガイド 第 1 巻』の「複写テーブルの管理」の「HDS を使用したデータ型の変換」で「クラス・レベル変換の作成」を参照してください。

参照：

- ファンクション文字列作成のガイドライン (47 ページ)
- ファンクション文字列クラス・スコープを持つシステム・ファンクション (20 ページ)

ファンクション文字列の継承

クラスの関係を作成することで、クラス間でファンクション文字列定義を共有することを「ファンクション文字列継承」と呼びます。

通常はファンクション文字列継承を使用し、特別な場合にシステム提供クラスから継承すると、複写システム管理者による管理やアップグレードに役立ちます。システム提供クラスから継承したクラスを使用する場合、カスタマイズするファンクション文字列のみを変更して、それ以外のすべてを継承します。

システム提供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必ず新しいファンクション文字列を追加してください。

親クラスからファンクション文字列を継承するクラスを「派生クラス」と呼びます。派生クラスがファンクション文字列を継承するクラスを、派生クラスの「親クラス」と呼びます。通常、特定のファンクション文字列をカスタマイズし、その他すべてのファンクション文字列を親クラスから継承する場合に、派生クラスを作成します。

親クラスからファンクション文字列を継承しないクラスを「基本クラス」と呼びます。システム提供クラス **rs_default_function_class** と **rs_db2_function_class**、および親クラスからファンクション文字列を継承せずに追加で作成したクラスは、基

本クラスです。システム提供クラス **rs_iq_function_class**、**rs_msss_function_class**、および **rs_oracle_function_class** は、**rs_default_function_class** の派生クラスです。

親クラスは複数の派生クラスを持つことができますが、派生クラスは 1 つの親クラスしか持つことができません。また、派生クラスを 1 つ以上の派生クラスの親クラスとして使用することもできます。同じ基本クラスから分岐するすべての階層の派生クラス群を「クラス・ツリー」と呼びます。

システム提供クラス **rs_default_function_class** と **rs_db2_function_class** は、派生クラスの親クラスになることができます。ただし、他の親クラスの派生クラスになることはできません。

システム提供クラス **rs_sqlserver_function_class** は、親クラスになることも、派生クラスになることもできません。

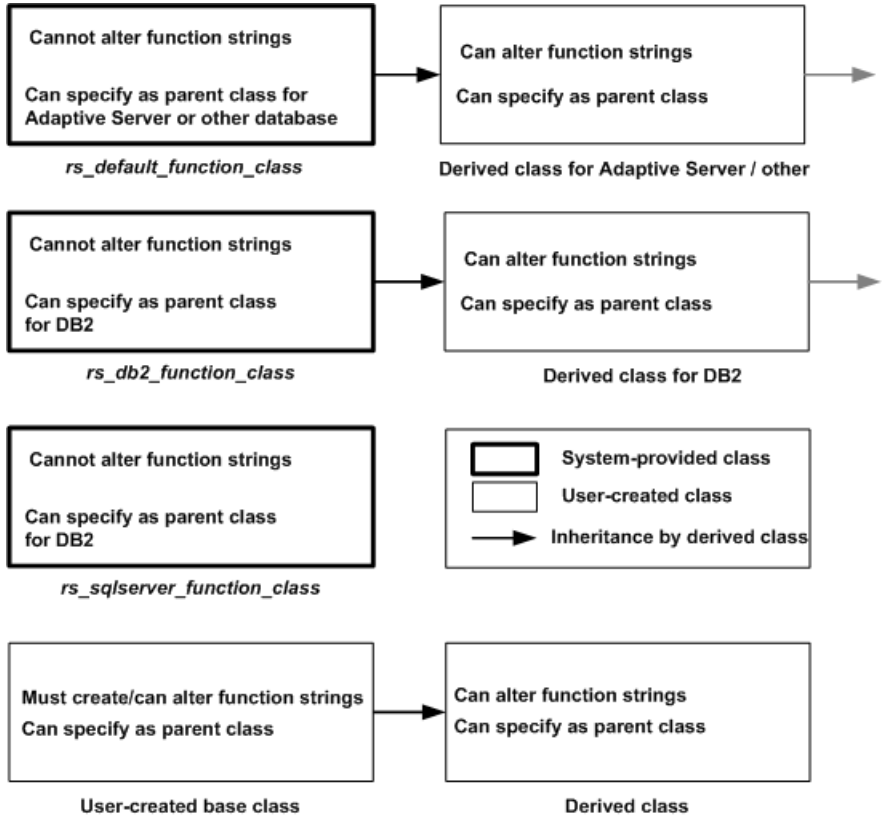
作成した基本クラスは、派生クラスになるように修正することも、派生クラスの親クラスとして指定することもできます。派生クラスは、異なる親クラスからファンクション文字列を継承するように修正することも、親クラスから切り離して基本クラスにすることもできます。

作成したすべての基本クラスに対して、そのクラスが割り当てられた各データベースで Replication Server が呼び出すファンクションのファンクション文字列を用意する必要があります。システム・ファンクションのファンクション文字列の一部が欠落しているときにデータベースにファンクション文字列を割り当てる場合、Replication Server がファンクション文字列を適用しようとすると、DSI がエラーをレポートしてデータベース・コネクションをサスペンドします。

循環形のファンクション文字列継承関係は禁止されています。つまり、親クラスを修正して、親クラス自体のいずれかの派生クラスから、またはそれらのいずれかの派生クラスの派生クラスからファンクション文字列を継承させたりすることはできません。

次の図にファンクション文字列クラスの関係を示します。

図 1 : ファンクション文字列クラスの関係



混合バージョン・システムの制限

混合バージョン・システムの場合、ファンクション文字列継承に関与するクラスを使用できるのは、Replication Server バージョン 11.5 以降のみです。

プライマリ・サイトが Replication Server バージョン 11.0.x であるクラスは、ファンクション文字列継承に関与できません。このようなクラスを変更して派生クラスになるようにしたり、親クラスとして使用したりするには、そのクラスを Replication Server バージョン 11.5 以降のプライマリ・サイトに移動させる必要があります。その後、必要に応じてクラス関係を変更したり、そのクラスや派生クラスを Replication Server バージョン 11.5 以降が管理するコネクションに割り当てたりすることができます。

Replication Server バージョン 11.5 以降で作成した、ファンクション文字列継承に関与しない基本クラスは、複写システムの Replication Server によって管理されるコネクションに割り当てることができます。Replication Server バージョン 11.5 以降が管理するデータベースに割り当てられない基本クラスは、**move primary** コマンド

を使用して、Replication Server バージョン 11.0.x が管理するプライマリ・サイトに割り当てることができます。

Replication Server 間の互換性の詳細については、リリース・ノートを参照してください。

注意： Replication Server バージョン 11.0.x との互換性のため、引き続き **rs_sqlserver_function_class** のファンクション文字列をカスタマイズしなければならないことがあります。ただし、Replication Server バージョン 11.5 以降で管理されるデータベースでは、派生クラスのみでファンクション文字列を継承したりカスタマイズしたりすることをおすすめします。

ファンクション文字列クラスの管理

ファンクション文字列クラスの管理には、ファンクション文字列の作成、割り当て、削除などの作業が含まれます。

ファンクション文字列を作成またはカスタマイズする場合は、それが属するクラスを指定します。カスタマイズされたファンクション文字列を作成して使用する場合は、次のことができます。

- **rs_default_function_class**、**rs_db2_function_class**、または他の親クラスからファンクション文字列を継承する派生ファンクション文字列クラスを作成できます。その後、派生クラスで、上書きするファンクション文字列のみを作成できます。

  注意： Sybase 以外のデータ・サーバ用のファンクション文字列クラスは、変更、追加、削除ができません。
- 新しいファンクション文字列クラスや、すべてのファンクションに対するファンクション文字列を作成できます。
- **rs_sqlserver_function_class** のファンクション文字列をカスタマイズできます。

カスタマイズしたファンクション文字列を作成するには、これらの方法のうちどれを使用するかを事前に決定し、その方法に従ってクラスを設定します。通常は、クラス **rs_sqlserver_function_class** のファンクション文字列ではなく、派生クラスのファンクション文字列をカスタマイズすることをおすすめします。別のクラスからファンクション文字列を継承する派生ファンクション文字列クラスを作成して展開するには、Replication Server バージョン 11.5 以降を使用してください。

参照：
- ファンクション文字列の管理 (37 ページ)

ファンクション文字列クラスの作成

既存のクラスのファンクション文字列が特定のデータベース・コネクションの要件を満たさない場合に、既存のクラスのファンクション文字列をカスタマイズで

きないときには、必要なファンクション文字列を作成するための新しいクラスを作成できます。

次のいずれかを行います。

- 既存の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を作成する。
- 他のクラスからのファンクション文字列を継承しない基本クラスを作成する。

1. **create function string class** を使用してファンクション文字列クラスを作成します。

   次のいずれかの構文のうち適切な構文を使用してください。
   - 派生クラスの作成
   - 基本クラスの作成

   新しいクラスの名前は、識別子の規則に従っている必要があります。『Replication Server リファレンス・マニュアル』の「トピック」の「識別子」を参照してください。
2. **create function string** を使用して新しいクラス用のファンクション文字列を作成します。
   - 派生クラスを作成する場合は、上書きするファンクション文字列のみを作成し、その他すべてのファンクション文字列は指定された親クラスから継承する必要があります。
   - システム提供クラスであるクラス **rs_default_function_class** には、**rs_dumpdb** ファンクションと **rs_dumptran** ファンクションのデフォルトのファンクション文字列は含まれていません。**rs_default_function_class** から継承する派生クラスでこれらのファンクション文字列が必要な場合は、作成する必要があります。
   - 基本クラスを作成する場合は、クラスに必要なファンクション文字列をすべて作成する必要があります。
3. 既存のデータベース・コネクション用の新しいファンクション文字列クラスを準備する場合は、コネクションをサスペンドしてから新しいクラスを使用する必要があります。

   『Replication Server 管理ガイド 第 1 巻』の「データベース・コネクションの管理」の「データベース・コネクションの変更」で、「データベース・コネクションのサスペンド」を参照してください。
4. 新しいクラスを割り当てるためのデータベース・コネクションを作成または変更します。
5. 既存のデータベース・コネクションを変更して新しいクラスを使用する場合は、コネクションをレジュームします。

   『Replication Server 管理ガイド 第 1 巻』の「データベース・コネクションの管理」の「データベース・コネクションの変更」で、「データベース・コネクションのサスペンド」を参照してください。

参照：
- ファンクション文字列の継承 (28 ページ)
- 派生クラスの作成 (33 ページ)
- 基本クラスの作成 (34 ページ)
- ファンクション文字列の作成 (45 ページ)
- システム提供クラス (27 ページ)
- データベースへのファンクション文字列クラスの割り当て (35 ページ)

派生クラスの作成

親クラスを指定して **create function string class** コマンドを使用し、親クラスのファンクション文字列を継承する派生ファンクション文字列クラスを作成します。

たとえば、親クラスのプライマリ・サイトで、次のように入力します。

```
create function string class
 sqlserver_derived_class
 set parent to rs_default_function_class
```

この例では、新しいクラス **sqlserver_derived_class** が、システム提供クラス **rs_default_function_class** からファンクション文字列を継承します。これにより、継承ファンクション文字列の一部を上書きするファンクション文字列を作成できます。

プライマリ・サイトで Replication Server バージョン 11.5 以降が実行されている任意の既存クラスを、親クラスとして指定できます。ただし、システム提供クラス **rs_sqlserver_function_class** は親クラスに指定できません。また、循環形の継承関係を作る親クラスも指定できません。

親クラスが **rs_default_function_class** または Sybase 以外のデータ・サーバ用のファンクション文字列クラスである場合、新しいクラスを使用する他の Replication Server へのルートが設定された任意の Replication Server で、このコマンドを入力できます。このサイトは、派生クラスと、その派生クラスから派生した新しいクラスすべてのプライマリ・サイトになります。

親クラスがユーザ作成のクラスである場合、親クラスのプライマリ・サイトである Replication Server でこのコマンドを入力します。このサイトは、親クラスから派生したすべてのクラスのプライマリ・サイトになります。

参照：
- ファンクション文字列の継承 (28 ページ)

基本クラスの作成

親クラスを指定せずに **create function string class** コマンドを使用して、親クラスのファンクション文字列を継承しないファンクション文字列クラスである基本ファンクション文字列クラスを作成します。

たとえば、次のように入力します。

```
create function string class base_class
```

この例では、新しいクラス **base_class** は、親クラスからファンクション文字列を継承しません。

このコマンドは、新しいクラスを使用する他の Replication Server へのルートが設定された任意の Replication Server で入力します。すると、このサイトが、そのクラスとそのクラスが親クラスとなるすべての派生クラスのプライマリ・サイトになります。

基本クラスは、派生クラスの親クラスとして使用することも、派生クラスになるように修正することもできます。

作成したすべての基本クラスに対して、そのクラスが割り当てられた各データベースで Replication Server が呼び出すファンクションのファンクション文字列を用意する必要があります。

基本クラスを作成して、それを派生クラスになるよう変更してから、実際にデータベース・コネクションで使用する場合は、すべてのファンクション文字列を作成する必要はありません。

ファンクション文字列クラスのプライマリ・サイト

ほとんどのファンクション文字列はレプリケート・データベースで実行されますが、**create function string class** コマンドは、ファンクション文字列クラスが使用されるすべてのサイトへのルートが設定された Replication Server (通常はプライマリ Replication Server) で実行します。

このコマンドを実行すると、その Replication Server がクラスのプライマリ・サイトとして指定されます。ファンクション文字列クラスは、その他の複写システムのデータとともに、ルートを介して複写されます。

クラス・スコープを持つファンクション文字列は、クラスのプライマリ・サイトだけで作成または変更ができます。複写定義スコープを持つファンクション文字列は、複写定義のプライマリ・サイトで作成または変更する必要があります。

デフォルトでは、クラス **rs_sqlserver_function_class** にはプライマリ・サイトが指定されていません。このクラスのクラス・スコープのファンクション文字列を変更するには、最初に Replication Server をクラスのプライマリ・サイトとして指定する必要があります。このファンクション文字列クラスのサイトを指定するには、プライマリ・サイトにする予定の Replication Server で次のコマンドを実行します。

```
create function string class rs_sqlserver_function_class
```

このコマンドを実行した後、**move primary** コマンドを使用してファンクション文字列クラスのプライマリ・サイトにさらに変更を行うことができます。

ファンクション文字列クラスのプライマリ・サイトの変更

ファンクション文字列クラスのプライマリ Replication Server を変更するには、**move primary** コマンドまたは Sybase Central を使用します。

たとえば、ファンクション文字列を新しいルート設定で分配できるように、プライマリ・サイトを別の Replication Server に変更しなければならない場合があります。新しいプライマリ・サイトには、ファンクション文字列クラスが使用されるすべての Replication Server へのルートが含まれていなければなりません。

基本クラスを移動すると、そのクラスから派生したすべてのクラスも移動します。

親クラスがデフォルトのファンクション文字列クラスではない場合、派生クラスのプライマリ・サイトは移動できません。

move primary は、ファンクション文字列クラスの新しいプライマリ・サイトとして指定する Replication Server で実行します。

たとえば、次のコマンドを実行すると、**sqlserver2_function_class** ファンクション文字列クラスのプライマリ・サイトが、コマンドを入力した SYDNEY_RS Replication Server に変更されます。

```
move primary of function string class
 sqlserver2_function_class
 to SYDNEY_RS
```

クラス **rs_sqlserver_function_class** にプライマリ・サイトが割り当てられていない場合、**move primary** を使用してプライマリ・サイトを割り当てることはできません。クラスのプライマリ・サイトを最初に指定するには、**create function string class** を使用します。

参照：

- ファンクション文字列クラスのプライマリ・サイト (34 ページ)

データベースへのファンクション文字列クラスの割り当て

ファンクション文字列クラスをデータベース・コネクションに割り当てるには、Sybase Central を使用するか、そのデータベースを管理する Replication Server で **create connection** コマンドまたは **alter connection** コマンドを実行します。

rs_init プログラムを使用してデータベース・コネクションを追加する場合、デフォルトではクラス **rs_sqlserver_function_class** がデータベースに割り当てられます。

データベースに割り当てられているファンクション文字列クラスを変更する前に、データベースへのコネクションをサスペンドする必要があります。**create connection** と **alter connection** の **set function string class** 句で、データベースで使用するファンクション文字列クラスの名前を指定します。

次のことを確認してから、ファンクション文字列クラスをデータベース・コネクションに割り当ててください。

- 指定するファンクション文字列クラスが Replication Server にすでに作成されていて使用可能なこと。
- 必要なファンクション文字列を、すべてこのクラスに作成すること。

注意： 接続プロファイルを使用してコネクションを作成する場合は、接続プロファイルによってファンクション文字列クラスが割り当てられます。

create connection コマンドと **alter connection** コマンドの使用方法、およびコネクション・プロファイルの使用方法については、『Replication Server 管理ガイド 第 1 巻』の「データベース・コネクションの管理」の「データベース・コネクションの作成」および『Replication Server 管理ガイド 第 1 巻』の「データベース・コネクションの管理」の「データベース・コネクションの変更」を参照してください。また、これらのコマンドについての説明は、『Replication Server リファレンス・マニュアル』の「Replication Server コマンド」も参照してください。

rs_init の詳細については、使用しているプラットフォームの『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。

新しいコネクションの作成例

次のコマンドを実行すると、TOKYO_DS データ・サーバによって管理される `pubs2` データベースへのコネクションが作成されます。

```
create connection to TOKYO_DS.pubs2
 set error class tokyo_error_class
 set function string class tokyo_func_class
 set username pubs2_maint
 set password pubs2_maint_pw
```

このコマンドは、**tokyo_func_class** ファンクション文字列クラスをデータベース・コネクションに割り当てます。

既存のコネクションの変更例

次のコマンドを実行すると、既存のデータベース・コネクションが変更され、別のファンクション文字列クラスが指定されます。

```
alter connection to TOKYO_DS.pubs2
 set function string class tokyo_func_class2
```

参照：
- ファンクション文字列の作成 (45 ページ)
- ファンクション文字列クラスの作成 (31 ページ)

ファンクション文字列クラスの削除

再度使用するかどうかわからない作成済みのファンクション文字列クラスは、**drop function string class** コマンドを使用すると複写システムから削除できます。

3 つのシステム提供クラスと現在の親クラスであるユーザ作成クラス以外の、任意のファンクション文字列クラスを削除できます。ファンクション文字列クラスを削除するには、事前にそのファンクション文字列クラスを使用するすべてのデータベース・コネクションを削除するか、別のクラスを使用するようにコネクションを変更する必要があります。

ファンクション文字列クラスを削除すると、そのクラスに定義されているすべてのファンクション文字列が削除され、そのクラスへのすべての参照が RSSD から削除されます。

たとえば、**tokyo_func_class** ファンクション文字列クラスとそのすべてのファンクション文字列を削除するには、次のコマンドを **isql** コマンド・ラインで実行します。

```
drop function string class tokyo_func_class
```

このコマンドは、クラスのプライマリ・サイトである Replication Server で入力します。

『Replication Server リファレンス・マニュアル』の「Replication Server コマンド」の「**drop function string class**」を参照してください。

ファンクション文字列の管理

各送信先 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 Adaptive Server などの送信先データ・サーバに適したコマンドに変換して送信します。

レプリケート Replication Server でこの変換を実行するコンポーネントである DSI スレッドの詳細については、『Replication Server 管理ガイド 第 1 巻』の「Replication Server の技術的概要」を参照してください。

コマンドの構文やパーミッションの詳細については、『Replication Server リファレンス・マニュアル』を参照してください。

ファンクション文字列とファンクション文字列クラス

ファンクション文字列をカスタマイズする必要がない場合、システム提供ファンクション文字列クラスの 1 つを使用して、デフォルトのファンクション文字列を提供することができます。文字列をクラス・スコープまたは複写定義スコープでカスタマイズする必要がある場合、システム提供クラス **rs_sqlserver_function_class** を使用して、ファンクション文字列をカスタマイズしたり、派生または基本ファンクション文字列クラスを作成したりする必要があります。

- ファンクションが実行されるデータベースのコネクションで、システム提供ファンクション文字列クラスや、**rs_default_function_class** または Sybase 以外のデータ・サーバ用のファンクション文字列クラスから直接的または間接的に継承する派生クラスが使用される場合、各システム・ファンクションとユーザ定義ファンクションに対して、デフォルトのファンクション文字列が提供されます。
- ファンクション文字列を継承しないユーザ作成の基本ファンクション文字列クラスや、そのようなクラスから継承する派生クラスがコネクションで使用される場合、各システム・ファンクションとユーザ定義ファンクションに対してファンクション文字列を作成する必要があります。ファンクション文字列を基本クラスのすべての派生クラスで使用できるようにするには、基本クラスでそれらのファンクション文字列を作成します。

参照：

- ファンクション文字列クラス (26 ページ)

ファンクション文字列の入力テンプレートと出力テンプレート

ファンクション文字列をカスタマイズするには、関連する入力テンプレートおよび出力テンプレートを変更します。

ファンクションによっては、ファンクション文字列に入力テンプレートと出力テンプレートの両方、または出力テンプレートのみが含まれるようにしたり、どちらのテンプレートも含まれないようにしたりできます。

- サブスクリプション・マテリアライゼーションで使用される **rs_select** ファンクションと **rs_select_with_lock** ファンクションの場合、Replication Server は、入力テンプレートを使用して、サブスクリプションの **where** 句に対応するファンクション文字列を特定します。
- Replication Server は、すべてのファンクションについて、出力テンプレートを使用して言語コマンドにファンクションをマップしたり、送信先データ・サーバで RPC 呼び出しを適用したりします。

入力テンプレートと出力テンプレートの条件

テンプレートを変更してファンクション文字列をカスタマイズする場合には、いくつかの条件があります。

その条件には、次のような条件が含まれます。

- ファンクション文字列の入力テンプレートと出力テンプレートの最大サイズは 64K バイトです。ファンクション文字列の入力テンプレートまたは出力テンプレート内の埋め込み変数にランタイム値を代入した結果が、64K を超えないようにします。
- ファンクション文字列の入力テンプレートと出力テンプレートは、一重引用符 (') で区切ります。
- ファンクション文字列変数は疑問符 (?) で囲みます。
- 変数名とその変更子は感嘆符 (!) で区切ります。

言語出力テンプレートを使用する場合には、他にも関連する条件があります。

参照：

- 出力テンプレート (39 ページ)

出力テンプレート

Replication Server は、出力テンプレートを使用して、データ・サーバに送信するコマンドのフォーマットを決定します。出力テンプレートを変更するとファンクション文字列をカスタマイズできます。

ほとんどの出力テンプレートでは、ファンクション文字列自体のフォーマットに応じて、言語、RPC または **none** のフォーマットを使用できます。

rs_writetext ファンクション文字列の出力テンプレートでは、RPC フォーマットに加え、**writetext** または **none** のいずれかのフォーマットを使用できますが、言語出力テンプレートは使用できません。

参照：

- ファンクション文字列 (24 ページ)
- text、unitext、image、rawobject データ型でのファンクション文字列の使用 (60 ページ)

言語出力テンプレート

言語出力テンプレートには、データ・サーバでコマンドとして解釈されるテキストが含まれています。

Replication Server は、出力テンプレートに埋め込まれた変数に値を代入し、結果の言語コマンドが処理されるよう、それらのコマンドをデータ・サーバに渡します。

Replication Server では、言語出力テンプレートの特定の文字が次のような特殊な方法で解釈されます。

- 2 つの連続した一重引用符文字 (') は、1 つの引用符として解釈される。
- 2 つの連続した疑問符 (??) は、1 つの疑問符として解釈される。
- 2 つの連続したセミコロン (;;) は、1 つのセミコロンとして解釈される。

Replication Server では、埋め込み変数の代入やこれらの特殊な解釈以外には、言語出力テンプレートの内容の解釈は行いません。

参照：
- ファンクション文字列の作成 (45 ページ)
- ファンクション文字列変数 (43 ページ)
- ファンクション文字列変数のフォーマット (45 ページ)

RPC 出力テンプレート

Replication Server は、言語出力テンプレートの場合とは異なり、RPC 出力テンプレートの内容を解釈します。

RPC 出力テンプレートは、Transact-SQL の **execute** コマンドのフォーマットで記述されています。Replication Server はこの出力テンプレートを解析して、Adaptive Server、Open Server ゲートウェイ、Open Server アプリケーションに送信するリモート・プロシージャ・コールを作成します。

RPC 出力テンプレートは、言語パーサを持たないゲートウェイや Open Server で使用すると効果的です。通常、RPC は、言語要求よりもコンパクトであり、データ・サーバによる解析を必要としないので、より効率的な場合もあります。そのため、データ・サーバが言語要求をサポートしている場合でも RPC を使用できます。

none パラメータを使用する出力テンプレート

出力コマンドのないクラス・レベル・ファンクション文字列およびテーブル・レベル・ファンクション文字列を特定するために、**none** パラメータを使用することで、ファンクション文字列を作成または変更するときにファンクション文字列の効率を向上させることができます。Replication Server はそれらのファンクション文字列をレプリケート・データベースで実行しません。

rs_writetext ファンクション文字列の出力テンプレート

Replication Server は、**rs_writetext** ファンクション文字列を作成するための 3 つの出力フォーマット (RPC、**none**、**writetext**) をサポートしています。**writetext** 出力テンプレートは、**rs_writetext** ファンクション文字列でのみ使用できます。

参照：

- text、unitext、image、rawobject データ型でのファンクション文字列の使用 (60 ページ)

入力テンプレート

入力テンプレートは、非バルク・マテリアライゼーションと **with purge** 指定のあるマテリアライゼーション解除でのみ使用されます。その場合、Replication Server は、選択されたテーブルから追加または削除するデータを選択します。

入力テンプレートを持つことができるファンクション文字列は、**rs_select** と **rs_select_with_lock** のみです。Replication Server は、マテリアライゼーション中またはマテリアライゼーション解除中にサブスクリプションで使用するファンクション文字列を、次のようにして決定します。

- サブスクリプションの複写定義を照合する。
- 入力テンプレートとサブスクリプションで使用されている **where** 句を照合する。

また、**rs_select** と **rs_select_with_lock** には、目的のマテリアライゼーションやマテリアライゼーション解除を実行する実際の select 文やその他のオペレーションを指定する出力テンプレートが含まれます。

システム提供クラスの場合、Replication Server は、複写定義の作成時に **rs_select** ファンクションと **rs_select_with_lock** ファンクション用のデフォルトのファンクション文字列を生成します。通常、これらのファンクション文字列は、複写定義に複数のサブスクリプションが存在する場合にのみカスタマイズする必要があります。

rs_select ファンクションと **rs_select_with_lock** ファンクションのファンクション文字列は、ほとんどの場合、マテリアライゼーションで使用します。同じ複写定義に対して複数のサブスクリプションを使用する場合、ファンクション文字列を作成してからサブスクリプションを作成します。サブスクリプション・マテリアライゼーションの詳細については、『Replication Server 管理ガイド 第 1 巻』の「サブスクリプションの管理」の「サブスクリプション・マテリアライゼーション・メソッド」を参照してください。

rs_select と **rs_select_with_lock** の各ファンクション文字列は、サブスクリプション・マテリアライゼーション解除にも使用されます。その場合には、サブスクリプションの作成に使用されたコマンドの **where** 句を使用します。これらのファンクションのファンクション文字列は、サブスクリプションを削除する前に存在していなければなりません。マテリアライゼーション解除の詳細については、『Replication Server 管理ガイド 第 1 巻』の「サブスクリプションの管理」の「サブスクリプション・コマンド」の「drop subscription コマンド」を参照してください。

入力テンプレートには、サブスクリプションの **where** 句内の定数から値が取得されるユーザ定義変数を含めることができます。その他のタイプのファンクション

文字列変数は、入力テンプレートで使用できません。同じファンクション文字列内の出力テンプレートからは、これらのユーザ定義変数を参照できます。

マテリアライゼーション・データを選択するように出力テンプレートをカスタマイズする必要がある場合、**rs_select** または **rs_select_with_lock** の各ファンクション文字列の入力テンプレートを省略できます。これにより、任意の **select** 文と一致するデフォルトのファンクション文字列が作成されます (**select** コマンドと一致するファンクション文字列の入力テンプレートが他にない場合)。

複写定義スコープを持つ他のファンクションと同様に、**rs_select** ファンクションと **rs_select_with_lock** ファンクションのファンクション文字列は、複写定義が作成されたプライマリ Replication Server で作成します。

ファンクション文字列を作成する場所の決定

ファンクション文字列を作成するクラスを決定します。

マテリアライゼーション用のファンクション文字列 **rs_select** と **rs_select_with_lock** は、マテリアライゼーション・データを選択しているプライマリ・データベースへのコネクションに割り当てられたファンクション文字列クラスに作成します。バルク・マテリアライゼーションを使用している場合は、マテリアライゼーション用のファンクション文字列 **rs_select** と **rs_select_with_lock** を作成する必要はありません。

マテリアライゼーション解除用のファンクション文字列 **rs_select** と **rs_select_with_lock** は、マテリアライゼーション解除するデータを選択しているレプリケート・データベースへのコネクションに割り当てられたファンクション文字列クラスに作成します。**without purge** オプションを指定し、**drop subscription** を使用してサブスクリプションを削除する場合、マテリアライゼーション解除用のファンクション文字列 **rs_select** と **rs_select_with_lock** は必要はありません。

rs_select ファンクション文字列の例

この例では、サイトが複写定義 **titles_rep** を介して、指定の出版社の書籍のタイトルにサブスクリプションを作成しています。`pubs2` データベースの `titles` テーブルの出版社のカラムと、出版社を識別するユーザ定義値を比較する入力テンプレートを持つ **rs_select** ファンクション文字列が必要です。

create function string コマンドを実行すると、出版社のカラム `pub_id` とユーザ定義変数 ?*pub_id*!*user*? を比較する入力テンプレートを持つファンクション文字列が作成されます。

入力テンプレートは、**where pub_id =** *constant* という書式の **where** 句を持つ任意のサブスクリプションと一致します。その結果、出力テンプレートが使用される場合、そのテンプレートには *constant* 値が含まれます。出力テンプレートにより、2つの異なるテーブルからマテリアライゼーション・データが選択されます。

```
create function string titles_rep.rs_select;pub_id
    for sqlserver2_function_class
scan 'select * from titles where pub_id =
    ?pub_id!user?'
output language
    'select * from titles where pub_id =
    ?pub_id!user?
    union
    select * from titles.pending where pub_id =
    ?pub_id!user?'
```

構文の詳細については、『Replication Server リファレンス・マニュアル』を参照してください。

参照：

- ファンクション文字列変数 (43 ページ)
- ファンクション文字列の作成 (45 ページ)

ファンクション文字列変数

ファンクション文字列の入力テンプレートまたは出力テンプレートに埋め込まれた変数は、さまざまなランタイム値のシンボリック・マーカとして使用できます。

変数は、カラム名、システム定義変数名、ユーザ定義ファンクションのパラメータ名、または入力テンプレートに定義されているユーザ定義変数を表します。この変数は、変数が割り当てられているのと同じデータ型の値を参照します。

ファンクション文字列変数は、次のように疑問符 (?) で囲まれます。

```
?variable!modifier?
```

変数の *modifier* 部分は、変数が表すデータ型を示します。変数名と変更子は、感嘆符 (!) で区切られます。

rs_truncate ファンクション文字列は、次の形式で、位置に基づくファンクション文字列変数を受け入れます。

```
?n!param?
```

ここで、*n* は 1 ～ 255 の数字であり、LTL でのファンクション・パラメータの位置を表します。LTL での **rs_truncate** の最初のパラメータは、`?1!param?` として、ファンクション文字列で表されます。位置に基づくファンクション文字列変数の場合、受け入れられる変更子は param のみです。

位置に基づく変数が指定された **rs_truncate** のサンプル・ファンクション文字列は、次のようになります。

```
truncate table publishers partition ?1!param?
```

参照：

- デフォルトのシステム変数 (58 ページ)

ファンクション文字列変数の変更子

Replication Server では、いくつかのファンクション文字列の変数変更子が認識されます。

表 5 : ファンクション文字列変数の変更子

変更子	説明
`new`、`new_raw`	Replication Server が挿入または更新しているローのカラムの新しい値への参照。
`old`、`old_raw`	Replication Server が挿入または更新しているローのカラムの古い値への参照。
`user`、`user_raw`	**rs_select** ファンクション文字列または **rs_select_with_lock** ファンクション文字列の入力テンプレートに定義されている変数への参照。
`sys`、`sys_raw`	システム定義変数への参照。
`param`、`param_raw`	ストアド・プロシージャ・パラメータへの参照。
`text_status`	`text_status` 値 (`text`、`unitext`、または `image` データの) への参照。有効な値は次のとおり。 • 0x000 – NULL 値を含むテキスト・フィールド。テキスト・ポインタは初期化されていない。 • 0x0002 – テキスト・ポインタは初期化されている。 • 0x0004 – 実テキスト・データが続く。 • 0x0008 – テキスト・データが複写されていないので、テキスト・データは続かない。 • 0x0010 – テキスト・データは複写されていないが、NULL 値を含む。

注意： ユーザ定義ファンクションのファンクション文字列は、**new** 変更子または **old** 変更子を使用しない場合があります。

ファンクション文字列の入力テンプレートまたは出力テンプレートで使用できるシステム定義変数のリストについては、『Replication Server リファレンス・マニュアル』の「Replication Server コマンド」の「create function string」を参照してください。

ファンクション文字列変数のフォーマット

Replication Server は、データ・サーバ・コマンドにファンクション文字列の出力テンプレートをマップする場合、Adaptive Server のフォーマットを使用して変数をフォーマットします。

ほとんどの変数 (*_raw* で終わる変更子を持つ特別な場合を除く) については、Replication Server はデータを次のようにフォーマットします。

- 文字と日付／時刻の値に出現する一重引用符に一重引用符を 1 つ追加する。
- 文字と日付／時刻の値を囲む一重引用符がない場合、一重引用符を追加する。
- money データ型の値に、適切な通貨記号 (ドル記号など) を追加する。
- バイナリ・データ型の値に “0x” プレフィクスを追加する。
- 円記号 (¥) と改行文字を組み合わせたものを、文字値内の円記号と改行文字の既存のインスタンス間に追加する。Adaptive Server は、円記号とその後ろの改行文字をひと続きの文字として扱うため、元の文字をそのまま残し、追加された一組の文字を削除する。

Replication Server は、*_raw* で終わる変更子を持つデータ型を上記の方法では変更しません。

ファンクション文字列の作成

ファンクション文字列クラスにファンクション文字列を追加する場合、またはターゲット・データベースには、**create function string** コマンドを使用します。

ファンクション文字列コマンドは、ファンクション文字列のプライマリ・サイトで入力してください。ファンクション文字列に応じたプライマリ・サイトは次のとおりです。

- 複写定義スコープを持つファンクション文字列の場合、プライマリ・サイトは複写定義が作成された Replication Server です。
- クラス・スコープを持つファンクション文字列の場合、プライマリ・サイトは、そのクラスのプライマリ・サイトである Replication Server です。派生クラスのプライマリ・サイトは、親クラスがシステム提供クラスの 1 つである場合を除いて、その親クラスと同じです。
- ターゲットスコープ・ファンクション文字列を持つ場合、プライマリ・サイトは、ターゲット・データベース (スタンバイ・データベースまたはレプリケート・データベース) を制御する Replication Server です。ターゲットスコープ・ファンクション文字列は、スタンバイ・データベースまたはレプリケート・データベースに対して作成され、ターゲットスコープ・ファンクション文字列はどのファンクション文字列クラスにも関連付けられません。

親クラスがシステムによって提供されない派生ファンクション文字列クラスを使用する場合は、特定のデータベース・コネクションに実際に割り当てられている

派生クラスではなく、親クラスでファンクション文字列をカスタマイズできます。これによって、その親クラスの追加されたすべての派生クラスで、カスタマイズされたファンクション文字列を使用できるようになります。

参照：

- ファンクション文字列クラスのプライマリ・サイト (34 ページ)

ターゲットスコープおよび複写定義スコープのファンクション文字列の比較

ターゲットスコープおよび複写定義スコープのファンクション文字列の間にはいくつか異なる点があります。

複写定義スコープ	ターゲットスコープ
複写定義スコープ・ファンクション文字列は、ファンクション文字列クラスに関連付けられている。	ターゲットスコープ・ファンクション文字列は、ターゲット (スタンバイまたはレプリケート) データベースに関連付けられている。
複写定義スコープのファンクション文字列を、複写定義を管理する Replication Server のファンクション文字列クラスの複写定義に対して作成する。	ターゲット・テーブルまたはストアド・プロシージャについて、スタンバイ・データベースまたはレプリケート・データベースを管理する Replication Servers のターゲットスコープ・ファンクション文字列を作成する。
Replication Server は、ファンクション文字列を複写定義に対して作成または変更したときに、ファンクション文字列のカラムまたはパラメータが有効かどうかを確認する。	Replication Server は、ファンクション文字列テンプレートを使用して DML コマンドを変換し、コマンドをスタンバイ・データベースまたはレプリケート・データベースに適用するまで、ファンクション文字列のカラムまたはパラメータが有効かどうかを確認しない。カラムまたはパラメータに関する情報が不正な場合は、DSI コネクションはシャットダウンする。ファンクション文字列を訂正してから、DSI コネクションを再開する。
複写定義スコープのファンクション文字列を作成する場合、Replication Server は一連のデフォルトのファンクション文字列を複写定義に対して作成する。	ターゲットスコープ・ファンクション文字列を作成する場合、Replication Server は指定するファンクション文字列のみを作成する。 ただし、**rs_get_textptr** と **rs_writetext** ファンクション文字列は共存するため例外である。**rs_writetext** のカスタム・ファンクション文字列がある場合は、**rs_get_textptr** のファンクション文字列も存在する。共に、カスタム・ファンクション文字列またはシステム作成のデフォルトのファンクション文字列として存在できる。

ファンクション文字列作成のガイドライン

ファンクション文字列作成には、いくつかのガイドラインがあります。

ファンクション文字列を作成する場合、ファンクション文字列クラスに関して次のガイドラインに従ってください。

- ファンクション文字列をカスタマイズする必要がある場合、システム提供クラス **rs_default_function_class** と **rs_db2_function_class** 以外の任意のクラスでカスタマイズできます。

  rs_db2_function_class、**rs_iq_function_class**、**rs_msss_function_class**、および **rs_oracle_function_class** の場合

  - **rs_begin** などのファンクション文字列クラス・スコープ・システム・ファンクションを使用して、カスタマイズされたクラスレベルのファンクション文字列を作成することはできません。
  - **rs_insert** などの複写定義スコープ・システム・ファンクションを使用すると、カスタマイズされたテーブルレベルのファンクション文字列を作成できます。
- ファンクション文字列クラスにプライマリ・サイトを割り当ててから、そのクラスのファンクション文字列を作成します。**create function string class** コマンドでプライマリ・サイトを割り当てるまで、システム提供クラス **rs_sqlserver_function_class** にはプライマリ・サイトはありません。
- ファンクション文字列クラスが新しい基本クラスである場合は、必要なすべてのシステム・ファンクションのファンクション文字列を作成してからそのクラスを使用します。

ファンクション文字列に関しては、次のガイドラインに従ってください。

- ファンクション文字列にオプションの名前を指定できます。**rs_select**、**rs_select_with_lock**、**rs_datarow_for_writetext**、**rs_get_textptr**、**rs_textptr_init**、**rs_writetext** ファンクションの場合、Replication Server はファンクション文字列名を使用してファンクション文字列をユニークに識別します。ファンクション文字列名は、完全に修飾するとユニークになります。
- **rs_select** ファンクション文字列または **rs_select_with_lock** ファンクション文字列に対して入力テンプレートが省略されている場合、Replication Server は一致するファンクション文字列を持たないすべてのサブスクリプションを照合します。
- 複写定義スコープを持つファンクションのファンクション文字列をカスタマイズする場合は、ファンクション文字列を作成してからサブスクリプションを作成します。
- ターゲットスコープ・ファンクション文字列には、ファンクション文字列クラスはありません。スタンバイ・データベースまたはレプリケート・データベースを制御する Replication Server で、**create function string** を使用してこれらの

ターゲット・データベースに対してターゲットスコープ・ファンクション文字列を作成します。

- スタンバイ・テーブルまたはレプリケート・テーブルのターゲットスコープ・ファンクション文字列の場合、有効なファンクションは、**rs_insert**、**rs_update**、**rs_delete**、**rs_truncate**、**rs_writetext**、**rs_datarow_for_writetext**、**rs_textptr_init**、および **rs_get_textptr** です。
- セミコロンで区切ることによって、言語出力テンプレートに複数のコマンドを含めることができます。
- ASE 以外のサーバのコマンドをバッチ処理できます。

  データベース・コネクションの **batch** パラメータがコマンドのバッチ処理を許可するように設定されていることを確認します。『Replication Server 管理ガイド 第 1 巻』の「データベース・コネクションの管理」の「データベース・コネクションの変更」の「物理コネクションに影響するパラメータの設定と変更」で、「1 つのコネクションに影響するパラメータの変更」を参照してください。
- Adaptive Server の構文を使用すると、ファンクション文字列の *constant* に null 値を指定できます。
- 出力コマンドのないクラス・レベル・ファンクション文字列およびテーブル・レベル・ファンクション文字列を特定するために、**none** パラメータを使用することで、ファンクション文字列を作成または変更するときにファンクション文字列の効率を向上させることができます。Replication Server はそれらのファンクション文字列をレプリケート・データベースで実行しません。

構文の詳細については、『Replication Server リファレンス・マニュアル』の「Replication Server コマンド」の「**create function string**」を参照してください。

参照：

- ファンクション文字列での複数のコマンドの定義 (53 ページ)
- ASE 以外のサーバのコマンドのバッチ処理 (54 ページ)

ファンクション文字列の作成例

例を使用してファンクション文字列を作成します。

rs_begin ファンクション文字列

次の例では、ストアド・プロシージャ **begin_xact** を実行することによってデータベースでトランザクションを開始する **rs_begin** ファンクションのファンクション文字列を作成します。

```
create function string rs_begin
 for gateway_func_class
 output rpc 'execute begin_xact'
```

rs_insert ファンクション文字列

次の例では、**publishers_rep** 複写定義を参照する **rs_insert** ファンクションのファンクション文字列を作成します。この複写定義は、プライマリ・テーブルへの **insert** の結果として、レプリケート・データベースで RPC を実行します。ストアド・プロシージャ **insert_publisher** は、レプリケート・データベースでのみ定義されます。

```
create function string publishers_rep.rs_insert
    for function class rs_sqlserver_function_class
    output rpc
    'execute insert_publisher
      @pub_id = ?pub_id!new?,
      @pub_name = ?pub_name!new?,
      @city = ?city!new?,
      @state = ?state!new?'
```

upd_bits ターゲットスコープ・ファンクション文字列

カスタマイズされたファンクション文字列 (この **upd_bits** ストアド・プロシージャは、NY_DS データ・サーバの `rdb1` ターゲット・データベースにあります) を作成します。ストアド・プロシージャのファンクションの名前はストアド・プロシージャと同じになります。

```
create function string upd_bits.upd_bits
     for database NY_DS.rdb1
     with overwrite
     output language
     'exec upd_bits
           @firstbit = ?firstbit!param?,
           @secondbit = ?secondbit!param?,
     @commit = ?comment!param?'
```

ファンクション文字列の変更

alter function string コマンドを実行すると、既存のファンクション文字列が置き換えられます。

alter function string の機能は、**create function string** と基本的に同じです。ただし、最初に **drop function string** コマンドを実行する点が異なります。ファンクション文字列が失われたことによりエラーが発生しないように、ファンクション文字列は 1 つのトランザクション内で削除されて再作成されます。

ファンクション文字列は、**alter function string** コマンドまたは **create function string** コマンドを使用して変更できます。**create function string** コマンドを使用してファンクション文字列を変更するには、ファンクション文字列クラス名の後ろにオプション句 **with overwrite** を指定します。このコマンドを実行すると、**alter function string** コマンドと同様に、既存のファンクション文字列が削除されて再作成されます。

alter function string コマンドを使用してファンクション文字列を変更するには、最初にファンクション文字列を作成します。

派生クラスでは、最初に **create function string** コマンドを使用して、親クラスから継承されたファンクション文字列を上書きします。派生クラスに対してファンクション文字列を明示的に作成していないと、その派生クラスのファンクション文字列を変更できません。

ファンクション文字列は、既存のファンクション文字列のプライマリ・サイトである Replication Server で変更します。ファンクションはそれぞれ次のようになります。

- 複写定義スコープ - 複写定義が指定されたプライマリ Replication Server でファンクション文字列を変更します。
- クラス・スコープ - ファンクション文字列クラスのプライマリ・サイトでファンクション文字列を変更します。派生クラスのプライマリ・サイトは、親クラスがシステム提供クラスの 1 つである場合を除いて、その親クラスと同じです。
- ターゲットスコープ - スタンバイ・データベースまたはレプリケート・データベースを制御する Replication Server でファンクション文字列を変更します。

rs_select や **rs_select_with_lock** など、複数のファンクション文字列のマッピングが可能なシステム・ファンクションの場合、**alter function string** 構文で完全なファンクション文字列名を指定します。Replication Server はこの名前を使用して、変更するファンクション文字列を判別します。

構文の詳細については、『Replication Server リファレンス・マニュアル』の「Replication Server コマンド」の「**create function string**」を参照してください。

参照：

- ファンクション文字列クラスのプライマリ・サイト (34 ページ)
- ファンクション文字列の作成 (45 ページ)

ファンクション文字列の削除

派生クラスでカスタマイズされたファンクション文字列を廃棄し、親クラスからファンクション文字列をリストアするには、ファンクション文字列を削除します。

drop function string コマンドを使用すると、ファンクション文字列クラス内のファンクション文字列、またはスタンバイ・データベースやレプリケート・データベースのファンクション文字列を削除できます。

警告！ ファンクション文字列を削除して再作成する場合は、**alter function string** を使用して既存のファンクション文字列を新しいファンクション文字列で置き換えてください。それ以外の方法でファンクション文字列を削除して再作成すると、ファンクション文字列が一時的に失われた状態になる可能性があります。ファン

クション文字列の削除と再作成との間に、このファンクション文字列を使用するトランザクションが発生すると、Replication Server がこのファンクション文字列を失われたものとして検出し、トランザクションが失敗します。

派生クラスからファンクション文字列を削除した場合は、親クラスからファンクション文字列をリストアします。

『Replication Server リファレンス・マニュアル』の「Replication Server コマンド」の「**drop function string**」を参照してください。

システム提供クラス **rs_sqlserver_function_class** から、カスタマイズされたファンクション文字列を削除することもできます。

削除してしまった複写定義スコープを持つファンクション文字列に対してデフォルトのファンクション文字列をリストアするには、**alter function string** コマンドを使用して、**output** 句を省略します。

参照：
- デフォルトのファンクション文字列のリストア (52 ページ)

ファンクション文字列の削除例

例を挙げてファンクション文字列の削除方法を説明します。

複写定義のファンクション文字列の削除

次のコマンドを実行すると、`publishers_rep` 複写定義の **rs_insert** ファンクション文字列が削除されます。この複写定義は、`sqlserver2_func_class` にあります。

```
drop function string
publishers_rep.rs_insert
for sqlserver2_func_class
```

複写定義のファンクション文字列のインスタンスの削除

次のコマンドを実行すると、**rs_select** ファンクションのファンクション文字列の `pub_id` インスタンス (`publishers_rep` 複写定義用) が削除されます。この複写定義は、クラス `derived_class` にあります。**rs_select_with_lock** ファンクションのファンクション文字列も同様の方法で削除します。

```
drop function string
publishers_rep.rs_select;pub_id
for derived_class
```

ファンクション文字列のファンクション文字列クラスからの削除

次のコマンドを実行すると、**rs_begin** ファンクション文字列が `gateway_func_class` ファンクション文字列クラスから削除されます。

```
drop function string rs_begin
for gateway_func_class
```

ターゲット・データベースのファンクション文字列の削除

次のコマンドを使用すると、ターゲット・ストアド・プロシージャのファンクション文字列をデータベース NY_DS.rdb1 から削除できます。

```
drop function string upd_bits.upd_bits
for database NY_DS.rdb1
```

デフォルトのファンクション文字列のリストア

複写定義スコープを持つシステム・ファンクションに対して Adaptive Server のデフォルトのファンクション文字列をリストアするには、**create function string** コマンドまたは **alter function string** コマンドで **output** 句を省略します。

ファンクション文字列クラス・スコープを持つシステム・ファンクションから出力テンプレートを省略することはできませんが、空のテンプレートを指定できます。

これらのコマンドの詳細については、『Replication Server リファレンス・マニュアル』の「Replication Server コマンド」を参照してください。

派生クラスを含むすべてのクラスにおいて、**output** 句を指定しないで **create function string** コマンドまたは **alter function string** コマンドを実行すると、システム提供クラス **rs_sqlserver_function_class** と **rs_default_function_class** にデフォルトで提供されているのと同じファンクション文字列がリストアされます。

この方法で解放したデフォルトのファンクション文字列定義は、クラスを割り当てたデータベースに適している場合とそうでない場合があります。この方法が最も適しているのは、カスタマイズされた **rs_sqlserver_function_class** を使用する場合、または Adaptive Server データベース用のその他のユーザ作成の基本クラスを使用する場合です。

派生クラスで、カスタマイズされたファンクション文字列を廃棄して親クラスからファンクション文字列をリストアする場合、ファンクション文字列を削除します。

ファンクション文字列の変更例

次のコマンドを実行すると、`publishers_rep` 複写定義のカスタマイズされた **rs_insert** ファンクション文字列がデフォルトのファンクション文字列に置き換えられます。

```
alter function string publishers_rep.rs_insert
for rs_sqlserver_function_class
```

派生クラスでのファンクション文字列の作成例

派生ファンクション文字列クラスでこの方法を使用すると、継承されたファンクション文字列を Adaptive Server のデフォルトのファンクション文字列で上書きできます。

次のコマンドを実行すると、publishers_rep 複写定義の継承された **rs_insert** ファンクション文字列がデフォルトのファンクション文字列に置き換えられます。

```
create function string publishers_rep.rs_insert
for derived_class
```

参照：

- ファンクション文字列の削除 (50 ページ)
- ファンクション文字列の変更 (49 ページ)
- ファンクション文字列の作成 (45 ページ)

出力テンプレートを使用した空のファンクション文字列の作成

output language 句に、空のファンクション文字列を 2 つの一重引用符を使用して指定すると、アクションを実行しない空のファンクション文字列を作成できます。

たとえば、次のコマンドを実行すると、publishers_rep 複写定義の **rs_insert** ファンクション文字列にアクションが定義されません。

```
alter function string publishers_rep.rs_insert
for derived_class
output none
```

参照：

- ファンクション文字列の変更 (49 ページ)

ファンクション文字列での複数のコマンドの定義

ファンクション文字列を使用すると、データベース・サーバのコマンドをバッチ処理できます。

言語出力テンプレートには、多数のコマンドを指定できます。Adaptive Server では、複数のコマンドをバッチ処理できます。この機能は他のデータ・サーバではほとんどサポートされていませんが、Replication Server では、コマンドをセミコロン (;) で区切ることにより、任意のデータ・サーバのファンクション文字列でコマンドをバッチにできます。

セミコロンがコマンド・セパレータとして解釈されないように、2 つの連続したセミコロン (;;) を使用します。

データ・サーバがコマンド・バッチをサポートしている場合、Replication Server は、必要に応じて、セミコロンを DSI コマンド・セパレータ文字 (**dsi_cmd_separator** 設定パラメータ) で置き換え、コマンドを 1 つのバッチで送信します。

データ・サーバがコマンド・バッチをサポートしていない場合、Replication Server はファンクション文字列の各コマンドを別々に送信します。

たとえば、次のファンクション文字列の出力テンプレートには 2 つのコマンドが指定されています。

```
create function string rs_commit
for sqlserver2_function_class
output language
'execute rs_update_lastcommit
    @origin = ?rs_origin!sys?,
    @origin_qid = ?rs_origin_qid!sys?,
    @secondary_qid = ?rs_secondary_qid!sys?;
    commit transaction'
```

Replication Server では、**alter connection** コマンドを使用してバッチのサポートを有効または無効にできます。

データベースでのコマンドのバッチ処理を許可するには **batch** を “on” に設定し、データ・サーバに個々のコマンドを送信するには “off” に設定します。

この例でバッチ処理を “on” に設定するには、次のように入力します。

```
alter connection to SYDNEY_DS.pubs2
    set batch to 'on'
```

バッチ処理を “off” に設定するには、次のように入力します。

```
alter connection to SYDNEY_DS.pubs2
    set batch to 'off'
```

ASE 以外のサーバのコマンドのバッチ処理

Replication Server では、ASE 以外のデータベース・サーバのコマンドをバッチ処理できます。これにより、パフォーマンスが向上する場合があります。

コマンドのバッチ処理をサポートするには、以下が必要です。

- 2 つのファンクション文字列 **rs_batch_start** と **rs_batch_end** を使用していること。
- 2 つのファンクション文字列の処理を制御するための DSI コネクション・パラメータを使用していること。

コマンドのバッチ処理をサポートするためのファンクション文字列

ASE 以外のサーバに対するコマンドのバッチ処理をサポートするには、**rs_batch_start** と **rs_batch_end** の 2 つのファンクション文字列を使用します。

これらのファンクション文字列には、コマンド・バッチの先頭と最後をマーク付けするために必要な SQL 変換が格納されます。ASE や、ファンクション文字列 **rs_begin** と **rs_commit** によって必要な機能がすでにサポートされている他のデータ・サーバでは、これらのファンクション文字列を使用する必要はありません。

コマンドのバッチ処理をサポートするためのコネクション設定

use_batch_markers DSI コネクション・パラメータを使用して、**rs_batch_start** ファンクション文字列および **rs_batch_end** ファンクション文字列の処理を制御します。

alter connection コマンドと **configure connection** を使用して、**use_batch_markers** を設定します。**use_batch_markers** が on に設定されている場合、**rs_batch_start** ファンクション文字列および **rs_batch_end** ファンクション文字列が実行されます。デフォルトは off です。

注意： **use_batch_markers** は、**rs_begin** ファンクション文字列に含まれていないコマンドのバッチの開始時と終了時に追加の SQL の送信を必要とするレプリケート・データ・サーバに対してのみ on に設定します。

処理の順序

バッチ・マーカのファンクション文字列を使用するようにコネクションが設定されると、Replication Server によって文がデータ・サーバに特定の順序で送信されます。

1. 最初に、**rs_begin** コマンドが、コマンドのバッチとは別に、またはそれらとグループ化されてレプリケート・データ・サーバに送信されます。これは、現在の機能と同様に、設定パラメータ **batch_begin** に基づいて行われます。
2. **rs_batch_start** コマンドは、**use_batch_markers** が true に設定されている場合にのみ処理および送信されます。
   rs_batch_start マーカは、バッチとして送信されるコマンドとグループ化されます。有効な **rs_begin** および **rs_batch_start** ファンクション文字列により、データ・サーバへの単一のトランザクションおよびバッチ処理されたトランザクションの処理が可能になります。
3. コマンドのバッチが、レプリケート・データ・サーバに送信されます。
   バッチのサイズは異なります。また、バッチの送信は、レプリケート・データ・サーバに対するコマンドのグループ化の終了およびフラッシュについての既存のルールに従います。これらのコマンドには、個々のコマンドの間にコマンド・セパレータが含まれます。
4. **rs_batch_end** は、コマンドのバッチの最後のコマンドです。**rs_batch_end** マーカは、設定パラメータ **use_batch_markers** が true に設定されている場合にのみ送信されます。
   rs_batch_start、コマンドのバッチ、および **rs_batch_end** は、**dsi_cmd_batch_size** などの制限によってコマンドがフラッシュされたときに複数のバッチが必要になると繰り返すことができます。
5. 最後の **rs_batch_end** コマンドが送信されると、**rs_commit** コマンドが、レプリケート・データ・サーバに送信されます。**rs_commit** は、現在のルールに従って処理されます。

DSI 設定

次に示すように、コマンドをバッチ処理するコネクションごとに、考慮の必要な DSI 設定パラメータがいくつかあります。

- **batch**
- **batch_begin**
- **use_batch_markers**

使用している ASE 以外のレプリケート・データ・サーバでコマンドのバッチ処理が許可されているかどうかを確認するには、『Replication Server 異機種間複写ガイド』を参照してください。

設定パラメータの使用方法については、『Replication Server 管理ガイド 第 1 巻』の「データベース・コネクションの管理」の「データベース・コネクションの変更」の「物理コネクションに影響するパラメータの設定と変更」の「物理データベース・コネクションに影響する設定パラメータ」、および『Replication Server リファレンス・マニュアル』の「Replication Server コマンド」の「alter connection」を参照してください。

言語出力テンプレートでの宣言文の使用

言語出力テンプレートでローカル変数を定義するための宣言文を指定するには、データベースに接続している Replication Server の **batch** 設定パラメータを “off” に設定します。

batch を “on” (Adaptive Server の場合のデフォルト) に設定すると、Replication Server がファンクション文字列の複数の呼び出しを 1 つのコマンド・バッチとしてデータ・サーバに送信できます。その結果、そのバッチには同じ変数の宣言が複数存在することになりますが、これは Adaptive Server では許可されません。

バッチ・モードを off にすると、Replication Server は各コマンドの応答を待ってから次のコマンドを送信するため、パフォーマンスが低下します。パフォーマンス要件が高くない場合は、**batch** を “off” に設定するとファンクション文字列内で宣言文を使用できます。反対に、パフォーマンスを向上させるためにバッチ・モードを使用する必要がある場合は、ストアド・プロシージャを実行するファンクション文字列の言語出力テンプレートを作成して、宣言文またはその他のコマンドを指定できます。

batch の詳細については、『Replication Server 管理ガイド 第 1 巻』の「データベース・コネクションの管理」の「データベース・コネクションの変更」の「物理コネクションに影響するパラメータの設定と変更」で、「物理データベース・コネクションに影響する設定パラメータ」を参照してください。

ファンクション関連情報の表示

Replication Server の **admin** コマンドまたは Adaptive Server のストアド・プロシージャを使用すると、使用している複写システムの既存のファンクション文字列およびクラスに関する情報を取得できます。

admin コマンドの詳細については、『Replication Server リファレンス・マニュアル』の「Replication Server コマンド」を参照してください。

admin コマンドを使用した情報の取得

Replication Server の **admin** コマンドのいずれかを使用すると、Replication Server システムで使用されるファンクション文字列クラス名を表示できます。

既存のファンクション文字列クラスとその親クラスの名前を表示するには、**admin show_function_classes** を使用します。これは、クラスの継承レベルも示します。レベル 0 は **rs_default_function_class** または **rs_db2_function_class** などの基本クラス、レベル 1 は基本クラスから継承する派生クラスなどのようになっています。

次に例を示します。

```
       admin show_function_classes
```

```
          Class                   ParentClass                   Level
          --------                ------------                  -----
           sql_derived_class        rs_default_function_class      1
           rs_db2_derived_class     rs_db2_function_class          1
           rs_db2_function_class                                   0
...
```

『Replication Server リファレンス・マニュアル』の「Replication Server コマンド」の「**admin show_function_classes**」を参照してください。

ストアド・プロシージャを使用した情報の取得

Replication Server の RSSD でストアド・プロシージャを使用すると、使用しているシステムの既存のファンクション、ファンクション文字列、ファンクション文字列クラスに関する情報を取得できます。

詳細については、『Replication Server リファレンス・マニュアル』の「RSSD ストアド・プロシージャ」を参照してください。

rs_helpfunc

rs_helpfunc は、Replication Server または特定のテーブルやファンクション複写定義の、システム・ファンクションとユーザ定義ファンクションに関する情報を表示します。構文は次のとおりです。

```
rs_helpfunc [replication_definition [, function_name]]
```

rs_helpfstring

rs_helpfstring は、複写定義に関連したファンクションのパラメータとファンクション文字列テキストを表示します。構文は次のとおりです。

```
rs_helpfstring replication_definition
    [, function_name]
```

rs_helpobjfstring

rs_helpobjfstring は、スタンバイ、レプリケート・テーブル、またはストアド・プロシージャに関連したファンクション文字列のパラメータとファンクション文字列テキストを表示します。構文は次のとおりです。

```
rs_helpobjfstring data_server, database, [owner.]object [,function]
```

rs_helpclass

rs_helpclass は、すべてのファンクション文字列クラスとエラー・クラス、およびそれらのプライマリ Replication Server をリストします。構文は次のとおりです。

```
rs_helpclass [class_name]
```

rs_helpclassfstring

rs_helpclassfstring は、クラス・スコープのファンクションのファンクション文字列情報を表示します。構文は次のとおりです。

```
rs_helpclassfstring class_name [, function_name]
```

デフォルトのシステム変数

デフォルトのシステム変数 *rs_default_fs* を使用すると、ファンクション文字列を拡張およびカスタマイズできます。

- 複写定義スコープを持つファンクション文字列を拡張して、監査または追跡などのための追加のコマンドを指定する。
- **rs_update** ファンクション文字列と **rs_delete** ファンクション文字列をカスタマイズし、**replicate minimal columns** オプションを引き続き複写定義で使用できるようにする。

注意： *rs_default_fs* システム変数が指定されたファンクション文字列は、Adaptive Server または Adaptive Server の構文を受け入れるデータ・サーバでのみ適用できます。それ以外のサーバで使用すると、エラーが発生します。

ファンクション文字列のシステム変数の詳細なリストについては、『Replication Server リファレンス・マニュアル』の「Replication Server コマンド」の「**create function string**」を参照してください。

デフォルト・ファンクション文字列の拡張

デフォルトのファンクション文字列の動作を拡張する方法として、複写定義スコープ (テーブルまたはファンクション) を持つすべてのファンクション文字列で *rs_default_fs* システム変数を使用できます。

rs_default_fs システム変数を使用すると、デフォルトのファンクション文字列の機能を変更しないで追加のコマンドを指定したい場合に、入力の手間が省けます。たとえば、コマンドを追加して、監査または追跡のためにデフォルトのファンクション文字列の機能を拡張できます。

言語出力テンプレートに追加するコマンドは、*rs_default_fs* システム変数の前後いずれかに指定できます。これらのコマンドは、レプリケート・テーブルへのローの複写方法に影響する場合と影響しない場合があります。

次の例は、*rs_default_fs* システム変数を **create function string** コマンド (または **alter function string** コマンド) で使用して、更新が発生したことを確認する方法を示しています。

```
create function string replication_definition.rs_update
  for function_string_class
  output language '?rs_default_fs!sys?;
if (@@rowcount = 0)
  begin
  raiserror 99999 "No rows updated!"
  end'
```

この例では、言語出力テンプレートに埋め込まれた *rs_default_fs* システム変数がデフォルトの **rs_update** ファンクション文字列の機能を維持し、出力テンプレートがローの更新をチェックします。ローが更新されていない場合、エラーが発生します。

この例では、システム変数に続くコマンドは、レプリケート・サイトでのローの複写方法には影響しません。*rs_default_fs* システム変数は、検証または監査のために類似の追加コマンドで使用できます。

replicate minimal columns 句の使用

rs_update ファンクション文字列と **rs_delete** ファンクション文字列をカスタマイズし、**replicate minimal columns** 句の使用を継続します。

複写定義に **replicate minimal columns** 句を指定した場合、通常、**rs_update**、**rs_delete**、**rs_get_textptr**、**rs_textptr_init**、または **rs_datarow_for_writetext** システム・ファンクションにデフォルト以外のファンクション文字列を作成できません。

rs_update ファンクションと **rs_delete** ファンクションのデフォルト以外のファンクション文字列を作成するには、*rs_default_fs* システム変数を **create function string** コマンドまたは **alter function string** コマンドの言語出力のテンプレートに埋め込み、最少カラム・オプションの使用を継続します。

最少カラム・オプションを使用する複写定義の **rs_update** ファンクション文字列または **rs_delete** ファンクション文字列では、*rs_default_fs* システム変数を含めて、キー・カラム以外の値にアクセスする変数は使用できません。このようなファンクション文字列を作成するときに、プライマリ・テーブルで修正されるカラムを事前に知ることができないためです。ただし、キー・カラムの値にアクセスする変数は指定できます。

replicate minimal columns 句の詳細については、『Replication Server リファレンス・マニュアル』の「Replication Server コマンド」の「**create replication definition**」を参照してください。

text、unitext、image、rawobject データ型でのファンクション文字列の使用

`text` データ型、`unitext` データ型、`image` データ型、`rawobject` データ型をサポートする環境では、出力テンプレートのフォーマットの **writetext** または **none** を使用して、**rs_writetext** ファンクションのファンクション文字列をカスタマイズできます。

『Replication Server リファレンス・マニュアル』の「Replication Server システム・ファンクション」の「**rs_writetext**」を参照してください。

Replication Server バージョン 11.5 以降では、ファンクション文字列の代わりに複数の複写定義を使用できます。『Replication Server 管理ガイド 第 1 巻』の「複写テーブルの管理」の「複写定義の作成」で、「テーブルごとに複数の複写定義を作成する方法」を参照してください。

rs_writetext ファンクション文字列に対する writetext 出力テンプレートの使用

rs_writetext ファンクション文字列の **writetext** 出力テンプレート・オプションは、Client-Library™ 関数 **ct_send_data** を使用して `text`、`unitext`、`image`、または `rawobject` のカラム値を更新するように Replication Server に指示します。

このオプションは、レプリケート・データベースでの `text`、`unitext`、`image`、または `rawobject` カラムのロギング動作を指定します。

writetext 出力テンプレートは、次のオプションをサポートします。

- **use primary log** — プライマリ・データベースにロギング・オプションが指定されている場合、レプリケート・データベースでデータのログを記録する。
- **with log** — レプリケート・データベースのトランザクション・ログにデータのログを記録する。
- **no log** — レプリケート・データベースのトランザクション・ログにデータのログを取らない。

rs_writetext ファンクション文字列のための none 出力テンプレートの使用

rs_writetext ファンクション文字列の **none** 出力テンプレート・オプションは、Replication Server に `text`、`unitext`、または `image` カラム値を複写しないように指示します。これは、異機種環境での `text`、`unitext`、および `image` カラムの使用に対する柔軟性を与えるためです。

異機種間複写と text、unitext、image、および rawobject データ

ASE 以外のデータ・サーバから Adaptive Server データベースに `text`、`unitext`、`image`、`rawobject` データを複写するには、複写定義に `text`、`unitext`、`image`、`rawobject` データを指定して Adaptive Server データベースにサブスクリプションを作成できるようにする必要があります。

ただし、レプリケート・データ・サーバが他の外部データ・サーバか他の Adaptive Server であるかどうかにかかわらず、`text`、`unitext`、`image`、`rawobject` データを他のレプリケート・データ・サーバに複写したくない場合があります。

none 出力テンプレート・オプションを使用して **rs_writetext** ファンクション文字列をカスタマイズすると、レプリケート・サイトでより小さなテーブルにオペレーションをマップしたり、レプリケート・サイトに対して `text`、`unitext`、`image`、または `rawobject` オペレーションを実行しないように **rs_writetext** ファンクション文字列に指示したりできます。

1 つの **rs_writetext** ファンクション文字列が、複写定義の text、unitext、image、および rawobject のカラムごとに存在します。特定の text、unitext、image、または rawobject カラムを複写したくない場合は、そのカラムの **rs_writetext** ファンクション文字列をカスタマイズします。次の例に示すように、**create** コマンドまたは **alter function string** コマンドにカラム名を指定します。**rs_insert** ファンクション文字列のカスタマイズが必要な場合もあります。

例
ここでは、複写定義で text、unitext、image、または rawobject カラムに null 値を使用できず、レプリケート・サイトで特定の text、unitext、image、または rawobject カラムが必要ないものと想定します。

プライマリ・サイトでこれらのカラムに挿入が発生した場合、レプリケート・サイトで不要な text、unitext、image、または rawobject カラムの **rs_writetext** ファンクション文字列をカスタマイズする必要があります。また、複写定義の **rs_insert** ファンクション文字列もカスタマイズする必要があります。

たとえば、プライマリ・テーブル foo があるとします。

```
foo (int a, b text not null, c image not null)
```

foo で次の挿入を実行します。

```
insert foo values (1, "111111", 0x11111111)
```

デフォルトでは、Replication Server は **rs_insert** を次の形式に変換し、DSI スレッドによってレプリケート・テーブル foo に適用されるようにします。

```
insert foo (a, b, c) values (1, "", "")
```

DSI スレッドは、次を呼び出します。

- text データをカラム b に挿入する **ct_send_data**
- image データをカラム c に挿入する **ct_send_data**

text カラム b と image カラム c では null 値が許可されていないため、レプリケート・テーブルにカラム b またはカラム c がない場合、DSI スレッドは停止します。

レプリケート・テーブルにカラム a とカラム b のみがある場合、次のように、カラム c の **rs_writetext** ファンクションをカスタマイズして、**output none** を使用する必要があります。

```
alter function string foo_repdef.rs_writetext;c
    for rs_sqlserver_function_class
    output none
```

カラム名 (この例では c) を上記のように指定して、そのカラムの **rs_writetext** ファンクション文字列を変更します。

レプリケート・テーブルにカラム a とカラム b のみがある場合、次のように複写定義の **rs_insert** ファンクション文字列もカスタマイズして、カラム c には挿入しないようにする必要があります。

```
alter function string foo_repdef.rs_insert
    for rs_sqlserver_function_class
    output language
    'insert foo (a, b) values (?a!new?, "")'
```

複写定義でカラム c に null 値を許可するように指定されている場合は、**rs_insert** をカスタマイズする必要はありません。デフォルトでは、**rs_insert** は、null 値を許可する text カラム、unitext カラム、または image カラムには影響しません。

ウォーム・スタンバイ・アプリケーションの管理

プライマリ・データベースまたはアクティブ・データベースと 1 つのスタンバイ・データベースの 2 つのデータベース間でウォーム・スタンバイ・アプリケーションを設定、構成、およびモニタします。

プライマリ・データベースへの変更は、ウォーム・スタンバイ・データベースに直接コピーされます。送信するデータを変更する、またはデータに条件を設定するには、テーブル複写定義およびファンクション複写定義を追加する必要があります。

Replication Server では、Adaptive Server および Oracle データベース用のウォーム・スタンバイ・アプリケーションの設定と管理がサポートされています。2 つの Oracle データベース間のウォーム・スタンバイ・アプリケーションの設定方法の詳細については、『Replication Server 異機種間複写ガイド』の「Oracle に対する異機種ウォーム・スタンバイ」を参照してください。

また、MSA (Multi-Site Availability) を使用して、Adaptive Server データベース間にウォーム・スタンバイ・アプリケーションを設定することもできます。MSA によって、複数のスタンバイ・データベースとレプリケート・データベースへの複写が有効になります。データベース全体を複写するのか、または指定のテーブル、トランザクション、ファンクション、システム・ストアド・プロシージャ、データ定義言語 (DDL) を複写するのかを選択することができます。『Replication Server 管理ガイド 第 1 巻』の「MSA を使用した複写オブジェクトの管理」を参照してください。

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。

アクティブ・データベースで障害が発生した場合、またはアクティブ・データベースやデータ・サーバをメンテナンス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ると、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処理を再開できます。

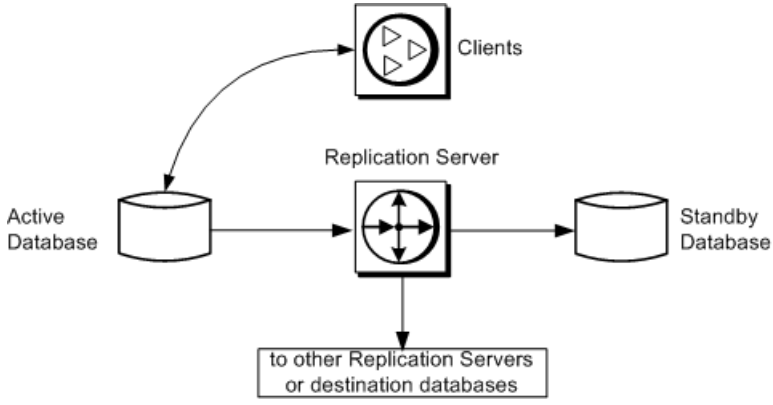
スタンバイ・データベースが常にアクティブ・データベースとの一貫性を保つようにするために、Replication Server では、アクティブ・データベースのトランザクション・ログから取得したトランザクション情報が再生成されます。複写定義は、スタンバイ・データベースへの複写を容易にしますが、必須ではありません。スタンバイ・データベースへのデータの複写には、サブスクリプションは必要ありません。

ウォーム・スタンバイの動作

ウォーム・スタンバイがどのように動作するのかについて説明します。

この図は、ウォーム・スタンバイ・アプリケーションを使用した通常の処理の例を示しています。

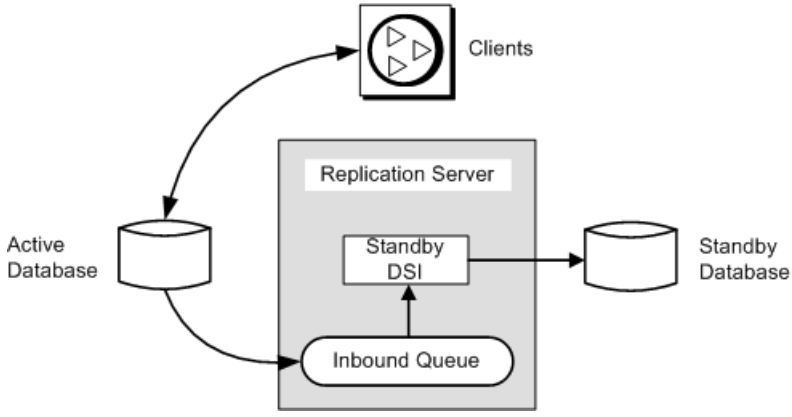
図 2 : ウォーム・スタンバイ・アプリケーション — 通常の処理



このウォーム・スタンバイ・アプリケーションでは、次の処理が実行されます。

- クライアント・アプリケーションが、アクティブ・データベースでトランザクションを実行する。
- アクティブ・データベースの RepAgent が、トランザクション・ログからトランザクションを取得して Replication Server に転送する。
- Replication Server が、スタンバイ・データベースでトランザクションを実行する。
- 場合によっては、Replication Server は送信先データベースやリモート Replication Server にもトランザクションをコピーする。

図 3 : ウォーム・スタンバイ・アプリケーションの例 ― 切り替え前



この図は、ウォーム・スタンバイ・アプリケーションのコンポーネントとプロセスの詳細を示しています。

参照 :

- 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前 (105 ページ)

ウォーム・スタンバイ・アプリケーションでのデータベース・コネクション

ウォーム・スタンバイ・アプリケーションでは、Replication Server から 1 つの論理データベースへのコネクションとして、アクティブ・データベースとスタンバイ・データベースが複写システムに示されます。

複写システム管理者は、この「論理コネクション」を作成して、アクティブ・データベースとスタンバイ・データベースの両方に対する 1 つの記号名を設定します。

したがって、ウォーム・スタンバイ・アプリケーションには、Replication Server からの次のデータベース・コネクションが関係します。

- 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
- アクティブ・データベースとスタンバイ・データベースに対する論理コネクション

Replication Server は、論理コネクションを現在のアクティブ・データベースにマップして、トランザクションをアクティブ・データベースからスタンバイ・データベースにコピーします。

『Replication Server 管理ガイド 第 1 巻』の「データベース・コネクションの管理」を参照してください。

複写のパフォーマンスを向上させるには、ウォーム・スタンバイ環境内で代替コネクションと代替論理コネクションを作成します。

参照：

- ウォーム・スタンバイ環境での複数のレプリケーション・パス (325 ページ)
- ASE ウォーム・スタンバイ・データベースの設定 (87 ページ)

プライマリ・データベースとレプリケート・データベース、およびウォーム・スタンバイ・アプリケーション

論理データベースが、プライマリ・データベースまたはレプリケート・データベースとして機能する場合もあります。

多くの Replication Server アプリケーションでは、次のことが当てはまります。

- プライマリ・データベースは、複写定義とサブスクリプションを使用して他のデータベースにコピーされるデータの送信元である。
- レプリケート・データベースは、プライマリ・データベースからデータを受信する。

Replication Server は、論理データベースをその他のデータベースと同じように扱います。アプリケーションによっては、ウォーム・スタンバイ・アプリケーション内の論理データベースが、複写に関与しないデータベースとして動作する場合があります。この場合、この論理データベースは、純粋にウォーム・スタンバイのバックアップとして存在します。また、論理データベースが、プライマリ・データベースまたはレプリケート・データベースとして動作する場合もあります。

データベースの関係の比較
通常、データベースは「プライマリ」または「レプリケート」として定義されますが、ウォーム・スタンバイ・アプリケーションの場合、「アクティブ」または「スタンバイ」としても定義されます。

表 6 : アクティブ・データベースおよびスタンバイ・データベースと、プライマリ・データベースおよびレプリケート・データベース

アクティブ・データベースおよびスタンバイ・データベース	プライマリ・データベースおよびレプリケート・データベース
アクティブ・データベースとスタンバイ・データベースは、同じ Replication Server で管理する必要がある。	プライマリ・データベースとレプリケート・データベースは、同じ Replication Server でも異なる Replication Server でも管理できる。

アクティブ・データベースおよびスタンバイ・データベース	プライマリ・データベースおよびレプリケート・データベース
アクティブ・データベースとスタンバイ・データベースは、Adaptive Server データベースでなければならない。	ウォーム・スタンバイ・アプリケーションに関与する場合以外は、プライマリ・データベースとレプリケート・データベースは Adaptive Server データベースでなくてもよい。
アクティブ・データベースには、1 つのスタンバイ・データベースがある。 情報は、常にアクティブ・データベースからスタンバイ・データベースにコピーされる。	プライマリ・データベースは、1 つ以上のレプリケート・データベースを持つことができる。 データベースの中には、プライマリ・データとコピー・データの両方が存在するものがある。
複写定義の使用は任意である。サブスクリプションは使用されない。	プライマリ・データベースからレプリケート・データベースへの複写に、複写定義とサブスクリプションが必要である。
スタンバイ・データベースへのコネクションでは、ファンクション文字列クラス **rs_default_function_class** を使用する。 このクラスのファンクション文字列はカスタマイズできない。	レプリケート・データベースへのコネクションでは、ファンクション文字列をカスタマイズできるファンクション文字列クラスを使用できる。たとえば、**rs_default_function_class** からファンクション文字列を継承する派生クラスを使用できる。
アクティブ・データベースとスタンバイ・データベースの役割を切り替えることができる。	プライマリ・データベースとレプリケート・データベースの役割は切り替えることができない。
通常、クライアント・アプリケーションはアクティブ・データベースに接続する(ただし、スタンバイ・データベースで読み込み専用オペレーションを実行できる)。 Replication Server をスタンバイ・データベースに切り替える場合に、クライアント・アプリケーションを切り替えるメカニズムがない。	クライアント・アプリケーションは、プライマリ・データベースまたはレプリケート・データベースのいずれかに接続できる。直接修正できるのはプライマリ・データのみである。 通常、クライアント・アプリケーションは、プライマリ・データベースとレプリケート・データベースを切り替える必要はない。

アクティブ・データベースおよびスタンバイ・データベース	プライマリ・データベースおよびレプリケート・データベース
アクティブ・データベースの RepAgent が、メンテナンス・ユーザのトランザクションなど、複写テーブルのすべてのトランザクションを Replication Server に送信し、Replication Server はスタンバイ・データベースでそのトランザクションを再生成する。 送信先データベース用のウォーム・スタンバイ・アプリケーションでは、通常、メンテナンス・ユーザがアクティブ・データベースのトランザクションを実行する。	ほとんどのアプリケーションで、RepAgent は送信先データベースで再生成されるメンテナンス・ユーザのトランザクションを Replication Server に送信しない。 通常、メンテナンス・ユーザはプライマリ・データベースでトランザクションを実行しない。

参照：

- 複写を使用するウォーム・スタンバイ・アプリケーション (124 ページ)

ウォーム・スタンバイの条件と制限

Replication Server のすべてのウォーム・スタンバイ・アプリケーションに対して適用されるいくつかの条件と制限があります。

- Adaptive Server などのウォーム・スタンバイ・アプリケーションをサポートするデータ・サーバを使用する必要があります。
- 1 つの Replication Server で、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を管理します。アクティブ・データベースとスタンバイ・データベースはいずれも、Adaptive Server データベースにする必要があります。2 つの Oracle データベース間のウォーム・スタンバイ・アプリケーションの設定方法の詳細については、『Replication Server 異機種間複写ガイド』の「Oracle に対する異機種ウォーム・スタンバイ」を参照してください。
- RSSD に対しては、スタンバイ・データベースを作成できません。Adaptive Server 15.0 ESD #2 以降など、Adaptive Server が master データベースの複写をサポートする場合にのみ、`master` データベースに対してスタンバイ・データベースを作成できます。
- Replication Server は、クライアント・アプリケーションをスタンバイ・データベースに切り替えません。
- アクティブ・データベースとスタンバイ・データベースは、それぞれ別のマシンの Adaptive Server で管理します。アクティブ・データベースとスタンバイ・

データベースを、同じデータ・サーバまたはハードウェア・リソース上で管理すると、ウォーム・スタンバイ機能の利点が損なわれます。

- Adaptive Server では重複するローを含むテーブルを許容しますが、アクティブ・データベースとスタンバイ・データベースのテーブルは、各ローのプライマリ・キー・カラムにユニークな値を持つ必要があります。
- 抽象プランのコマンドとプロシージャは、次の場合を除いて複写されます。
  - **create plan** の **and set** *@plan_id* 句は複写されません。たとえば、このコマンドは次のようには複写されません。

    ```
    create plan "select avg(price)
    from titles" "(t_scan titles)
    into dev_plans and set @plan_id
    ```

    これは、次のように複写されます。

    ```
    create plan "select avg(price)
    from titles" "(t_scan titles)
    into dev_plans
    ```

  - プラン ID を引数とする抽象プランのプロシージャ (**sp_drop_qplan**、**sp_copy_qplan**、**sp_set_qplan**) は複写されません。
  - **set plan** コマンドは複写されません。
- 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・スタンバイでは、データベースのコピーを保持しますが、Sybase フェールオーバでは、異なるマシンから同じデータベースにアクセスします。フェールオーバ・サポートは、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に対しても同じ働きをします。

  Adaptive Server Enterprise マニュアル・セットの「高可用性システムにおける Sybase フェールオーバの使用」を参照してください。
- ダンプを使用してアクティブ・データベース上でマーカを有効にした場合、スタンバイ・データベースの再構築にプラットフォーム間の **dump** および **load** は使用できません。ダンプ・マーカは、Replication Agent によって再構築中のデータベースに送信される必要があります。プラットフォーム間でのダンプとロードの実行中にアクティブ・データベースからダンプを取得するときには、アクティブ・データベースはシングル・ユーザ・モードである必要があります。

参照：

- アクティブ・データ・サーバを使用するようにクライアントを設定する (116 ページ)
- 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 (382 ページ)
- プラットフォーム間でのダンプとロード (95 ページ)

スタンバイ・データベースを管理するためのファンクション文字列

Replication Server は、スタンバイ・データベースへのコネクションであるスタンバイ DSI に対して、システム提供ファンクション文字列クラス **rs_default_function_class** を使用します。

Replication Server は、このクラスに対してデフォルトのファンクション文字列を生成します。クラス **rs_default_function_class** のファンクション文字列は、カスタマイズできません。

ウォーム・スタンバイの複写情報

Replication Server では、いくつかの方法でスタンバイ・データベースへの複写を有効にできます。Replication Server がスタンバイ・データベースにコピーする情報のレベルとタイプは、選択する方法によって異なります。

次の 2 つの方法のいずれかを選択してください。

- **sp_reptostandby** システム・プロシージャを使用して、スタンバイ・データベースに複写するようデータベース全体にマーク付けします。**sp_reptostandby** によって、データ操作言語 (DML) コマンドとサポートされているデータ定義言語 (DDL) コマンドおよびシステム・プロシージャのセットの複写が有効になります。
  - **insert**、**update**、**delete**、**truncate table** などの DML コマンドは、ユーザ・テーブルのデータを変更します。
  - DDL コマンドとシステム・プロシージャは、データベースのスキーマまたは構造を変更します。

  sp_reptostandby によって、データベースに格納されているシステム・テーブルを変更する DDL コマンドとプロシージャの複写ができるようになります。DDL コマンドを使用すると、テーブルやビューなどのデータベース・オブジェクトを作成、変更、削除できます。サポートされている DDL システム・プロシージャは、データベース・オブジェクトの情報に影響を与えます。これらのコマンドやシステム・プロシージャは、元のユーザによってスタンバイ・データベースで実行されます。
- **sp_reptostandby** を使用しない場合は、**sp_setreptable** を実行して、個々のユーザ・テーブルに複写するようマーク付けできます。このプロシージャによって、マーク付けしたテーブルに対する DML オペレーションの複写が有効になります。

オプションで、スタンバイ・データベースに複写するユーザ・ストアド・プロシージャを Replication Server に指示することもできます。

- **sp_setrepproc** システム・プロシージャを使ってユーザ・ストアド・プロシージャをマーク付けすることによって、そのストアド・プロシージャの実行をスタンバイ・データベースにコピーできます。通常は、ファンクション複写定義に関連付けられているストアド・プロシージャのみがスタンバイ・データベースに複写されます。

Oracle のウォーム・スタンバイで複写される情報の詳細については、『Replication Server 異機種間複写ガイド』の「Oracle に対する異機種ウォーム・スタンバイ」を参照してください。

参照：

- sp_setrepproc を使用したユーザ・ストアド・プロシージャのコピー (82 ページ)

複写方法の比較

sp_reptostandby と **sp_setreptable** を比較して、それぞれがどのようにスタンバイ・データベースに情報をコピーするかを説明します。

表 7 : テーブルの複写方法の比較

sp_reptostandby	sp_setreptable
すべてのユーザ・テーブルをスタンバイ・データベースにコピーする。	ユーザが、スタンバイ・データベースにコピーするユーザ・テーブルを選択する。
DML コマンドおよびサポートされている DDL コマンドとシステム・プロシージャの複写を許可する。	マーク付けしたテーブルで実行される DML コマンドの複写を許可する。 **注意：** **isql** セッションでサポートされている DDL 操作を強制的に複写できます。
DML オペレーションと DDL オペレーションをレプリケート・データベースにコピーしない。 ウォーム・スタンバイ・アプリケーションがデータをレプリケート・データベースにもコピーする場合、**sp_setreptable** を使用してレプリケート・データベースにコピーするテーブルをマーク付けする必要がある。	DML オペレーションをスタンバイ・データベースとレプリケート・データベースにコピーする。

sp_reptostandby	sp_setreptable
truncate table コマンドの実行をスタンバイ・データベースにコピーする。サブスクリプションは必要ない。 **注意：** **alter logical connection** コマンドを使用して、**truncate table** のスタンバイ・データベースへの複写を有効または無効にできます。	Adaptive Server データベースを使用している場合、**truncate table** の実行をスタンバイ・データベースにコピーする。サブスクリプションは必要ない。
Replication Server は、テーブル名とテーブル所有者情報を使用して、スタンバイ・データベースでテーブルを識別する。	ウォーム・スタンバイに複写するようテーブルにマーク付けするときに **owner_on** キーワードを指定した場合、Replication Server はテーブル名とテーブル所有者情報を使用して、スタンバイ・データベースでテーブルを識別する。 ウォーム・スタンバイに複写するようテーブルにマーク付けするときに **owner_off** キーワードを指定した場合、Replication Server はテーブル名と “dbo”を使用して、スタンバイ・データベースでテーブルを識別する。
デフォルトでは、`text`、`unitext`、`image`、および `rawobject` の各カラムは、変更された場合にのみスタンバイ・データベースにコピーされる。 **sp_reptostandby** と **sp_setreptable** を使用してデータベース・テーブルをマーク付けすると、`text`、`unitext`、`image`、および `rawobject` の各データは別の方法で処理される。	デフォルトでは、`text`、`unitext`、`image` カラムは、常にスタンバイ・データベースにコピーされる。 **sp_setreptable** を使用して複写ステータスを設定した場合、`text`、`unitext`、`image`、および `rawobject` の各カラムの複写ステータスは、**always_replicate**、**replicate_if_changed**、または **do_not_replicate** でマーク付けされたように処理される。
アクティブ・データベースとスタンバイ・データベースが同じ場合に使用する最も簡単な方法である。	

参照：

- スタンバイ・データベースへの DDL コマンドの強制的な複写 (86 ページ)
- スタンバイ・データベースへのトランケート・テーブルの複写 (121 ページ)
- ウォーム・スタンバイ・アプリケーションでの text、unitext、image、rawobject データの複写 (83 ページ)
- サポートされている DDL コマンドとシステム・プロシージャ (76 ページ)

sp_reptostandby を使用した複写の有効化

sp_reptostandby は、DML コマンドとすべてのユーザ・テーブルに対してサポートされている DDL コマンドをスタンバイ・データベースにコピーするために使用します。

DML コマンドと DDL コマンドの複写を有効にするには、アクティブ・データベースを管理する Adaptive Server で **sp_reptostandby** を次のように実行します。

```
sp_reptostandby dbname, [[, 'L1' | 'ALL' | 'NONE' ] [, use_index]]
```

ここで、*dbname* はアクティブ・データベースの名前であり、キーワード **L1**、**all**、および **none** は複写サポートのレベルを設定します。

L1 は、Adaptive Server バージョン 12.5 がサポートする複写レベルを表します。

スキーマ複写サポートが常に最高のレベルになるようにするには、**all** キーワードを使用します。たとえば、スキーマ複写サポートのレベルを最新の Adaptive Server バージョンのレベルに設定するには、Adaptive Server にログインして、**isql** プロンプトで次のコマンドを実行します。

```
sp_reptostandby dbname, 'all'
```

これで、データベースが、上位レベルの複写サポートを備えた最新の Adaptive Server バージョンにアップグレードされる場合は、そのバージョンのすべての新機能が自動的に有効になります。

DDL コマンドまたはシステム・プロシージャにパスワード情報が含まれている場合、パスワード情報は、送信元 Adaptive Server のシステム・テーブルに格納されている暗号化テキストのパスワード値を使用して複写環境を介して送信されます。

『Replication Server リファレンス・マニュアル』の「Adaptive Server コマンドとシステム・プロシージャ」の「**sp_reptostandby**」を参照してください。

sp_reptostandby 使用時の制限および条件

ウォーム・スタンバイ・アプリケーションを設定し、**sp_reptostandby** を使用して複写を有効にする場合は、次の制限および条件を考慮してください。

- アクティブ・データベースとスタンバイ・データベースがいずれも Adaptive Server によって管理されていて、RepAgent がサポートされている必要があります。両方のデータベースのディスクの割り付け、セグメント名、ロールは同じでなければなりません。『Adaptive Server Enterprise システム管理ガイド』を参照してください。
- アクティブ・データベース名が、スタンバイ・サーバに存在している必要があります。存在しない場合、そのデータベース名が指定されたコマンドまたはプロシージャの複写は失敗します。

- Replication Server は、ローカル変数が指定された DDL コマンドの複写をサポートしません。これらのコマンドには、サイト固有の情報を明示的に定義する必要があります。
- ログイン情報は、スタンバイ・データベースに複写されません。サーバ・ユーザの ID を一致させ、送信先 Replication Server にログイン情報を追加してください。
- 次のコマンドは、スタンバイ・データベースにコピーされません。
  - **選択**
  - **update statistics**
  - **sp_dboption**、**sp_configure** などのデータベース・オプションまたは設定オプション
- **set proxy** がプライマリ Adaptive Server で実行されると、Replication Server は DDL コマンドの複写をサポートしなくなり、次のようにエラー 5517 が返されます。

```
A REQUEST transaction to database '...' failed because the
transaction    owner's password is missing. This prevents the
preservation of transaction ownership.
```

参照：
- サーバ・ユーザの ID を一致させる (99 ページ)

サポートされている DDL コマンドとシステム・プロシージャ

sp_reptostandby を使用して複写を有効にしたときに Replication Server がスタンバイ・データベースで再生成する DDL コマンド、Transact-SQL コマンド、Adaptive Server システム・プロシージャです。

Adaptive Server 12.5 以降で複写がサポートされるコマンドとストアド・プロシージャについては、アスタリスク (*) が付いています。

サポートされている DDL コマンドは、次のとおりです。

- **alter encryption key**
- **alter key**
- **alter login**
- **alter login profile**
- **alter...modify owner** – Replication Server は、所有者の異なるテーブルを別のテーブルとして扱います。**alter...modify owner** を使用して Adaptive Server でレプリケートされたテーブルの所有者を変更するには、該当するテーブル複写定義も変更する必要があります。『Replication Server 管理ガイド 第 1 巻』の「複写テーブルの管理」の「Modify Replication Definitions」の「Alter Replication Definitions」の「Changes You Can Make to the Replication Definition」の「Changing Table Owner」を参照してください。

- **alter table**
- **create default**
- **create encryption key**
- **create function**
- **create index**
- **create key**
- **create login**
- **create login profile**
- **create plan***
- **create procedure**
- **create rule**
- **create schema***
- **create table**
- **create trigger**
- **create view**
- **drop default**
- **drop function**
- **drop login**
- **drop login profile**
- **drop index**
- **drop procedure**
- **drop rule**
- **drop table**
- **drop trigger**
- **drop view**
- **grant**
- **installjava*** – **installjava** の複写は、MSA 環境ではサポートされていません。
- **remove java***
- **revoke**

サポートされているシステム・プロシージャは、次のとおりです。

- **sp_add_qpgroup***
- **sp_addalias**
- **sp_addgroup**
- **sp_addmessage**
- **sp_addtype**
- **sp_adduser**
- **sp_bindefault**
- **sp_bindmsg**

- **sp_bindrule**
- **sp_cachestrategy**
- **sp_changegroup**
- **sp_chgattribute**
- **sp_commonkey**
- **sp_config_rep_agent**
- **sp_drop_all_qplans***
- **sp_drop_qpgroup***
- **sp_dropalias**
- **sp_dropgroup**
- **sp_dropkey**
- **sp_dropmessage**
- **sp_droptype**
- **sp_dropuser**
- **sp_encryption**
- **sp_export_qpgroup***
- **sp_foreignkey**
- **sp_hidetext**
- **sp_import_qpgroup***
- **sp_primarykey**
- **sp_procxmode**
- **sp_recompile**
- **sp_rename**
- **sp_rename_qpgroup***
- **sp_replication_path**
- **sp_setrepcol**
- **sp_setrepdefmode**
- **sp_setrepproc**
- **sp_setreplicate**
- **sp_setreptable**
- **sp_unbindefault**
- **sp_unbindmsg**
- **sp_unbindrule**

master データベースの複写でサポートされている DDL コマンド・セットとシステム・プロシージャは、ユーザ・データベースの複写でサポートされているセットとは異なります。

データベースが master データベースの場合、次の DDL コマンドがサポートされています。

- **alter role**
- **create role**
- **drop role**
- **grant role**
- **revoke role**

データベースが master データベースの場合、次のシステム・プロシージャがサポートされています。

- **sp_addexternlogin**
- **sp_addlogin**
- **sp_addremotelogin**
- **sp_addserver**
- **sp_defaultdb**
- **sp_defaultlanguage**
- **sp_displaylevel**
- **sp_dropexternlogin**
- **sp_droplogin**
- **sp_dropremotelogin**
- **sp_dropserver**
- **sp_locklogin**
- **sp_maplogin**
- **sp_modifylogin**
- **sp_password**
- **sp_passwordpolicy** – **allow password downgrade** を除くすべてのオプションで複写されます。
- **sp_role**

alter table の複写：制限事項

Adaptive Server が **alter table ... add** *column_name* **default** ... 文を実行すると、サーバは `objid` を使用してデフォルト値の制約を作成します。

Replication Server がこの文を複写した後で、スタンバイ Adaptive Server が、異なる `objid` を使用して同じ制約を作成します。

あとで **alter table ... drop constraint ...** を使用して、制約がプライマリで削除された場合、objid が同じでないため、文をウォーム・スタンバイで実行することはできません。

プライマリ・データベースおよびスタンバイ・データベースの両方で制約を削除するには、次の方法のいずれかを実行します。

- ```
  alter table table_name
          ...
          replace column_name default null
  ```
- ```
  alter table table_name
          ...
          drop constraint constraint_name
  ```

  この文によって、DSI は停止します。対応する `objid` を使用して、スタンバイ・データベースで同じコマンドを実行してから、DSI へのコネクションをレジュームし、トランザクションを省略します。

master データベースの複写：制限事項

ユーザ・テーブルおよびユーザ・ストアド・プロシージャは master データベースからは複写されません。

master データベースを複写する場合、master データベースで次のシステム・プロシージャを実行する必要があります。

- **sp_addlogin**
- **sp_defaultdb**
- **sp_defaultlanguage**
- **sp_displaylevel**
- **sp_droplogin**
- **sp_locklogin**
- **sp_modifylogin**

使用されているデータベースが master データベースである場合、送信元 Adaptive Server およびターゲット Adaptive Server の両方で master データベースの複写機能をサポートしている必要があります。

データベースが master データベースである場合、送信元 Adaptive Server とターゲット Adaptive Server の両方で、同じハードウェア・アーキテクチャ・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)、および同じオペレーティング・システム (異なるバージョンでも互換性はあります) を使用している必要があります。

複写の無効化

データとスキーマの複写を無効にするには、**none** オプションを指定して **sp_reptostandby** を使用します。

Adaptive Server にログインし、**isql** プロンプトで次のコマンドを入力します。

```
sp_reptostandby dbname,'none'
```

複写を無効にすると、Adaptive Server は排他モードですべてのユーザ・テーブルをロックして、各テーブルの情報を保存します。データベースに大量のユーザ・テーブルがある場合、この処理に時間がかかることがあります。

このプロシージャは、ウォーム・スタンバイ・アプリケーションを無効にしている場合にのみ使用してください。

注意： 現在の **isql** セッションに対してのみ複写を無効にするには、**set replication** コマンドを使用します。

また、データベースが `text`、`unitext`、`image`、`rawobject` カラムのインデックスを使用して複写するようにマーク付けされている場合、**sp_reptostandby** *dbname,* **'none'** は、明示的に複写対象としてマーク付けされていないテーブルの複写のインデックスも削除します。

参照：

- 現在の isql セッションでの複写の変更 (85 ページ)

sp_setreptable を使用した複写の有効化

レプリケート・データベースまたはレプリケート・データベースとスタンバイ・データベースに複写するよう個々のテーブルをマーク付けするには、**sp_setreptable** を使用します。

Replication Server は、これらのテーブル上の DML オペレーションをスタンバイ・データベースとレプリケート・データベースにコピーします。

sp_setreptable は、次の場合に、スタンバイ・データベースに複写するようテーブルをマーク付けするために使用します。

- Adaptive Server データベースを使用する場合
- **sp_reptostandby** を使用しないことにした場合

sp_setreptable を使用すると、アクティブ・データベースとスタンバイ・データベースとの間のデータの一貫性が保持されますが、スキーマの一貫性は保持されません。**sp_setreptable** は通常、サポートされている DDL コマンドとプロシージャをスタンバイ・データベースにコピーしません。ただし、**set replication** コマンドを使用して、現在の **isql** セッションに対して DDL コマンドを強制的に複写することはできます。

データベースが master データベースである場合、ユーザ・テーブルは複写されません。

参照：

- 現在の isql セッションでの複写の変更 (85 ページ)

sp_setrepproc を使用したユーザ・ストアド・プロシージャのコピー

ユーザ・ストアド・プロシージャの実行をスタンバイ・データベースにコピーするには、**sp_setrepproc** を使用して、複写するようストアド・プロシージャをマーク付けします。

sp_setrepproc を使用してマーク付けしたプロシージャは、そのプロシージャに対するサブスクリプションが作成されていれば、レプリケート・データベースでも再生成されます。

ウォーム・スタンバイ・アプリケーションでのストアド・プロシージャを実行する場合、次の 2 つのケースが考えられます。

- **sp_setrepproc** を使用して複写するようストアド・プロシージャをマーク付けした場合、Replication Server はそのプロシージャの実行をスタンバイ・データベースにコピーします。ストアド・プロシージャの結果はスタンバイ・データベースにコピーされません。
- ストアド・プロシージャを複写するようマーク付けしなかった場合は、影響を受けるテーブルが複写するようマーク付けされていれば、Replication Server はそのプロシージャによって実行された DML の変更をスタンバイ・データベースにコピーします。

sp_setrepproc システム・プロシージャの詳細については、『Replication Server 管理ガイド 第 1 巻』の「複写ファンクションの管理」を参照してください。

データベースが master データベースである場合、ユーザ・プロシージャは複写されません。

名前が同じで所有者が異なるテーブルの複写

Adaptive Server と Replication Server では、名前が同じで所有者が異なるテーブルを複写できます。

sp_reptostandby を使用して、データベースを複写するようマーク付けすると、スタンバイ・データベース内の名前と所有者が同じテーブルに、更新内容が自動的にコピーされます。

sp_setreptable を使用して、テーブルを複写するようマーク付けすると、テーブル所有者名を使用するかどうかを選択してスタンバイ・データベースで正しいテーブルを選択できます。

- **owner_on** を設定すると、Replication Server はテーブル名とテーブル所有者名をスタンバイ・データベースに送信します。
- **owner_off** を設定すると、Replication Server はテーブル名と、所有者名として “dbo” をスタンバイ・データベースに送信します。

注意： レプリケート・データベースに情報をコピーするときに、**sp_setreptable** を使用して **owner_off** を設定した場合は、Replication Server はテーブル名をレプリケート・データベースに送信します。所有者情報は送信しません。

『Replication Server 管理ガイド 第 1 巻』の「複写テーブルの管理」の「複写対象テーブルへのマーク付け」の「**owner_on** ステータスでの複写の有効化」を参照してください。

注意： ユニークでない名前のテーブルに複写するようマーク付けしてから、そのテーブルに対する複写定義を作成する場合は、複写定義に所有者情報を含める必要があります。所有者情報を指定しないと、Replication Server はレプリケート・データベースまたはスタンバイ・データベースで正しいテーブルを検出できません。

ウォーム・スタンバイ・アプリケーションでの text、unitext、image、rawobject データの複写

sp_reptostandby を使用してデータベースにマーク付けすると、複写ステータスが自動的に **replicate_if_changed** になり、Adaptive Server は変更された `text`、`unitext`、`image`、`rawobject` カラムのログだけを記録します。

これによって、スタンバイ・データベースが確実にアクティブ・データベースと同期した状態になります。このようなテーブルの複写ステータスは、**sp_setrepcol** では変更できません。

sp_setreptable を使用してテーブルを複写するようマーク付けすると、デフォルトの複写ステータスが **always_replicate** になり、Adaptive Server はすべての `text`、`unitext`、`image`、`rawobject` カラム・データのログを記録します。**sp_setreptable** を使用してマーク付けしたテーブルの `text`、`unitext`、`image`、`rawobject` カラムの複写ステータスは変更できます。**sp_setrepcol** を使用して、複写ステータスを **replicate_if_changed** または **do_not_replicate** に変更します。カラム、またはカラムの組み合わせは、各ローをユニークに識別する必要があります。

複写定義を使用している場合、プライマリ・キーは、テーブル内の各ローをユニークに識別するカラム・セットである必要があります。Adaptive Server と Replication Server の複写ステータスが一致していることを確認します。

レプリケート・データベースでの use_index オプションの使用

use_index オプションを使用して、`text`、`unitext`、`image`、または `rawobject` カラムに対する複写の設定処理を高速化します。

1 つ以上の `text`、`unitext`、`image`、または `rawobject` カラムを含む大きなテーブルの場合に特に便利です。**use_index** オプションは、データベース・レベル、テーブル・レベル、またはカラム・レベルで設定できます。たとえば、イン

デックスを使用しないようにテーブルをマーク付けしても、複写にインデックスを使用するように 1 つのカラムのみを明示的にマーク付けできます。

sp_reptostandby で **use_index** オプションを使用する場合、データベースが `text`、`unitext`、`image`、または `rawobject` カラムのインデックスを使用するようにマーク付けされ、明示的に複写対象としてマーク付けされていないテーブルに、内部インデックスが作成されます。

インデックスを使用して複写するようにマーク付けされているデータベースでは、ローでない状態のカラムを持つ新しいテーブルが作成されると、複写のインデックスも作成されます。同様に、インデックスを使用するようにマーク付けされているデータベースで、**alter table...add column** コマンドが実行されると、新しいローでない状態のカラムで内部インデックスが作成されます。**alter table...drop column** コマンドを使用すると、削除されているカラムがインデックスを使用するようにマーク付けされている場合、複写の内部インデックスも削除されます。

異なるオブジェクト・レベルでの複写のインデックスのステータスは、カラム、テーブル、データベースの順番になります。データベースがインデックスを使用して複写するようにマーク付けされていても、ユーザがインデックスを使用しないようにテーブルをマーク付けした場合、テーブルのステータスがデータベースのステータスよりも優先されます。

注意： ローでない状態のカラム (`text`、`unitext`、`image`、または `rawobject`) の複写のパフォーマンスは変わりません。データベース、テーブルまたはカラムを複写対象としてマーク付けする処理のみが影響されます。

テーブルに多数のローがある場合、またはデータベースに、非常に多くのローといくつかのローでない状態のカラムがあるテーブルが 1 つ以上ある場合は、**use_index** オプションを使用できます。

SQL 文の複写でのウォーム・スタンバイ・データベースの設定

デフォルトでは、ウォーム・スタンバイ・アプリケーションは、SQL 文の複写をサポートする DML コマンドを複写しません。ただし、SQL 文の複写を使用する方法はいくつかあります。

- **replicate SQLDML** 句と **send standby** 句を使用して、テーブル複写定義を作成する。
- **ws_sqldml_replication** パラメータを on にする。デフォルト値は **UDIS** です。ただし、**ws_sqldml_replication** の優先度は SQL の複写のテーブル複写定義よりも低い。テーブル複写定義にテーブルの **send standby** 句が含まれている場合、その句によって、DML 文をレプリケートするかどうかが決定される。**ws_sqldml_replication** パラメータの設定とは無関係。

暗号化カラムの複写

ウォーム・スタンバイ・アプリケーションで暗号化カラムを処理するときの考慮事項は、ウォーム・スタンバイ以外の環境の場合と同様です。

『Replication Server 管理ガイド 第 1 巻』の「複写テーブルの管理」の「暗号化カラムの複写」を参照してください。

引用符付き識別子の複写

ウォーム・スタンバイ・データベースおよび複写定義のサブスクライバへの複写時に、プライマリ・テーブル名は引用符付きとしてマーク付けされているが、レプリケート・テーブル名はマーク付けされていない場合 (またはその逆)、Replication Server は、プライマリ・テーブル名とレプリケート・テーブル名の両方を引用符付きとして送信します。

ウォーム・スタンバイにレプリケート・データベースが含まれる場合

アクティブ・データベースの情報は、スタンバイ・データベースにもレプリケート・データベースにもコピーできます。

Replication Server は、テーブルの `text`、`unitext`、`image`、`rawobject` カラムを、スタンバイ・データベースとレプリケート・データベースに同じ複写ステータスでコピーする必要があります。

すべての `text`、`unitext`、`image`、`rawobject` カラムをスタンバイ・データベースとレプリケート・データベースにコピーする場合は、テーブルの複写ステータスを変更しないでください。デフォルトでは、すべての `text`、`unitext`、`image`、`rawobject` カラムが、スタンバイ・データベースとレプリケート・データベースにコピーされます。

変更された `text`、`unitext`、`image`、`rawobject` カラムだけをコピーするには、**sp_setrepcol** を使用して複写ステータスを **replicate_if_changed** に設定します。

現在の isql セッションでの複写の変更

isql セッションに対して DML コマンドか DDL コマンドとプロシージャ、またはその両方の複写を制御するには、**set replication** を使用します。

set replication は、アクティブ・データベースを管理する Adaptive Server で実行します。構文は次のとおりです。

```
set replication [on | force_ddl | default | off]
```

デフォルト設定は “on” です。デフォルトの動作は、**sp_reptostandby** を使用して、データベースを複写するようマーク付けしているかどうかによって異なります。

表 8 : set replication のデフォルトの動作

sp_reptostandby を使用して複写するようデータベースをマーク付けした場合	sp_reptostandby を使用して複写するようデータベースをマーク付けしなかった場合
Replication Server は、すべてのユーザ・テーブルに対して、DML コマンドとサポートされている DDL コマンドをスタンバイ・データベースにコピーする。	Replication Server は、**sp_setreptable** を使用してマーク付けされているテーブルに対して、DML コマンドをスタンバイ・データベースとレプリケート・データベースにコピーする。

『Replication Server リファレンス・マニュアル』の「Adaptive Server コマンドとシステム・プロシージャ」の「**set replication**」を参照してください。

スタンバイ・データベースへの DDL コマンドの強制的な複写

サポートされている DDL コマンドとシステム・プロシージャを強制的に複写するには、**set replication force** を使用します。

たとえば、**isql** セッションに対して、サポートされているすべての DDL コマンドとシステム・プロシージャを強制的に複写するには、次のように入力します。

```
set replication force_ddl
```

このコマンドを実行すると、**sp_setreptable** を使用してマーク付けしたテーブルに対して、DDL コマンドとシステム・プロシージャの複写が有効になります。

force_ddl を無効にして **set replication** をデフォルトのステータスに戻すには、次のように入力します。

```
set replication default
```

スタンバイ・データベースへのすべての複写の無効化

スタンバイ・データベースへの複写をすべて無効にするには、**set replication force off** を使用します。

isql セッションに対して、スタンバイ・データベースへの複写をすべて無効にするには、次のように入力します。

```
set replication off
```

ASE ウォーム・スタンバイ・データベースの設定

ウォーム・スタンバイ・アプリケーションのデータベースを設定するには、いくつかの高度な作業が必要です。

1. 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使用する 1 つの論理コネクションを作成します。
2. Sybase Central または **rs_init** を使用して、アクティブ・データベースを複写システムに追加します。

   複写システムにすでに追加しているデータベースをアクティブ・データベースとして指定した場合は、アクティブ・データベースを追加する必要はありません。
3. **sp_reptostandby** または **sp_setreptable** を使用して、アクティブ・データベースのテーブルに対して複写を有効にします。
4. Sybase Central または **rs_init** を使用して、複写システムにスタンバイ・データベースを追加し、そのスタンバイ・データベースを初期化します。

作業を始める前に

ASE ウォーム・スタンバイ・データベースを設定するには、いくつかの前提条件があります。

- アクティブ・データベースとスタンバイ・データベースを管理する Replication Server は、インストールが完了し、実行中である必要があります。1 つの Replication Server で、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を管理します。
- アクティブ・データベースとスタンバイ・データベースがある Adaptive Server は、インストールが完了し、実行中である必要があります。これらのデータベースは、異なるマシンで実行されているデータ・サーバによって管理するのが理想的です。
- データベースをアクティブ・データベースまたはスタンバイ・データベースとして複写システムに追加するには、そのデータベースが Adaptive Server に存在していなければなりません。

参照：

- ウォーム・スタンバイの条件と制限 (70 ページ)

クライアント・アプリケーションの注意事項

ウォーム・スタンバイ・データベースを設定する前に考慮する必要のあるクライアント・アプリケーションの注意事項がいくつかあります。

使用しているクライアント・アプリケーションと、スタンバイ・データベースの初期化方法によっては、スタンバイ・データベースの初期化が完了するまでアクティブ・データベース内のトランザクション処理をサスペンドする場合があります。

トランザクション処理をサスペンドしない場合は、スタンバイ・データベースにデータをロードする間に実行されるトランザクションを保持できるように、Replication Server のステーブル・キューに十分な領域があることを確認してください。

ウォーム・スタンバイ・データベースを設定する前に、クライアント・アプリケーションを新しいアクティブ・データベースに切り替えるためのメカニズムを実装します。

参照：

- アクティブ・データ・サーバを使用するようにクライアントを設定する (116 ページ)

作業 1：論理コネクションの作成

アクティブ・データベースがすでに複写システムの一部である場合は、論理コネクションを作成し、アクティブ・データベースに対して RepAgent を再設定します。

参照：

- ウォーム・スタンバイ・アプリケーションでのデータベース・コネクション (67 ページ)

論理コネクションの名前付け

論理コネクションに割り当てる名前は、アクティブ・データベースが複写システムに追加されているかどうかによって異なります。

論理コネクションを作成する場合は、論理データ・サーバ名と論理データベース名の組み合わせを、*data_server.database* の形式で使用します。

- アクティブ・データベースが複写システムに追加されていない場合 — 論理コネクションには、アクティブ・データベースとは異なる名前を使用します。論理コネクションと物理コネクションにユニークな名前を使用すると、アクティブ・データベースを簡単に切り替えられます。
- アクティブ・データベースがすでに複写システムに追加されている場合 — 論理コネクション名には、アクティブ・データベースの *data_server* 名と *database*

名を使用します。論理コネクションは、この物理データベースを参照する既存の複写定義とサブスクリプションをすべて継承します。

ウォーム・スタンバイ・アプリケーションに対して複写定義またはサブスクリプションを作成する場合は、物理コネクションではなく論理コネクションを指定します。論理コネクションを指定すると、Replication Server は現在のアクティブ・データベースを参照できます。

参照：

- 複写を使用するウォーム・スタンバイ・アプリケーション (124 ページ)

論理コネクションの作成手順

Replication Server からコネクションを作成するには、**create logical connection** コマンドを使用します。

1. **sa** パーミッションを持つログイン名を使用して、ウォーム・スタンバイ・データベースを管理する Replication Server にログインします。
2. **create logical connection** コマンドを実行します。

   ```
   create logical connection to data_server.database
   ```

   データ・サーバ名には有効な Adaptive Server 名を、データベース名には有効なデータベース名を任意に指定できます。

RepAgent の再設定と再起動

論理コネクションを作成した後、RepAgent を再設定し再起動します。

複写システムにすでに追加しているデータベースをアクティブ・データベースとして指定すると、論理コネクションを作成するときにアクティブ・データベースの RepAgent スレッドが停止します。

1. **send_warm_standby_xacts** 設定パラメータを設定し、**sp_config_rep_agent** を使用して RepAgent を再設定します。

   『Replication Server 管理ガイド 第 1 巻』の「RepAgent の管理と Adaptive Server のサポート」の「RepAgent の準備」および『Replication Server リファレンス・マニュアル』の「Adaptive Server コマンドとシステム・プロシージャ」の「**sp_config_rep_agent**」を参照してください。
2. RepAgent を再起動します。

作業 2：アクティブ・データベースの追加

ウォーム・スタンバイ・アプリケーションに対してデータベースをアクティブ・データベースとして複写システムに追加するには、**rs_init** を使用します。

使用しているプラットフォームの『Replication Server インストール・ガイド』と『Replication Server 設定ガイド』の説明に従って、複写システムにデータベースを追加するための手順を実行します。

作業 3：アクティブ・データベースのオブジェクトに対する複写の有効化

ストアド・プロシージャの複写を有効にするには **sp_reptostandby** を使用し、アクティブ・データベースでのテーブルの複写を有効にするには **sp_reptostandby** または **sp_setreptable** を使用します。

アクティブ・データベースのテーブルに対して複写を有効にするには、次のいずれかの方法を使用します。

- **sp_reptostandby** を使用して、データベースを複写するようマーク付けして、データとサポートされているスキーマの変更に対する複写を有効にします。
- **sp_setreptable** を使用して、データの変更を複写するよう個々のテーブルをマーク付けします。

1. システム管理者またはデータベース所有者として Adaptive Server にログインし、次のコマンドを実行します。
   ```
   use active_database
   ```
2. 次の 3 つの方法のいずれかを使用して、データベース・テーブルを複写するようマーク付けします。
   - **sp_reptostandby** システム・プロシージャを実行して、すべてのユーザ・テーブルをマーク付けします。
     ```
     sp_reptostandby dbname, [ 'L1' | 'all' ]
     ```
     dbname はアクティブ・データベース名であり、**L1** は複写レベルを Adaptive Server バージョン 11.5 のレベルに設定し、**all** は複写レベルを現在の Adaptive Server のバージョンのレベルに設定します。この方法によって、DML コマンドおよび DDL コマンドとプロシージャが複写されます。
   - **sp_reptostandby** を **use_index** オプションとともに実行して、すべてのユーザ・テーブルをマーク付けします。
     ```
     sp_reptostandby dbname, [[, 'L1' | 'ALL'][, use_index]]
     ```
     dbname には、アクティブ・データベース名を指定します。**use_index** オプションを使用すると、データベースが `text`、`unitext`、`image`、または `rawobject` カラムのインデックスを使用するようにマーク付けされ、明示

的に複写対象としてマーク付けされていないそれらのテーブルに、内部インデックスが作成されます。

- スタンバイ・データベースに複写する個々のユーザ・テーブルに対して **sp_setreptable** システム・プロシージャを実行し、それぞれのテーブルをマーク付けします。

```
sp_setreptable table_name, 'true'
```

table_name はテーブル名です。この方法によって、DML コマンドが複写されます。

3. スタンバイ・データベースへの複写を実行する各ストアド・プロシージャに対して、関連するパラメータを使用して **sp_setrepproc** を実行します。

- 『Replication Server 管理ガイド 第 1 巻』の「複写ファンクションの管理」で説明した複写ファンクション機能を使用する場合は、**'function'** パラメータを使用して **sp_setrepproc** を実行します。

```
sp_setrepproc proc_name, 'function'
```

- テーブル複写定義に関連する複写ストアド・プロシージャなどの非同期プロシージャを使用する場合は、**'table'** パラメータを使用して **sp_setrepproc** を実行します。

```
sp_setrepproc proc_name, 'function'
```

参照：

- ウォーム・スタンバイの複写情報 (72 ページ)
- 非同期プロシージャ (445 ページ)

あとで追加されたオブジェクトの複写の有効化

スタンバイ・データベースに複写する新しいテーブルとユーザ・ストアド・プロシージャを、マーク付けして追加します。

- **sp_reptostandby** を使用して、データベースを複写するようマーク付けした場合、新しいテーブルが自動的に複写するようマーク付けされます。
- **sp_setreplicate** を使用して、スタンバイ・データベースに複写するようデータベース・テーブルをマーク付けした場合、複写する新しいテーブルをそれぞれ **sp_setreplicate** を使用してマーク付けする必要があります。
- 複写する新しいユーザ・ストアド・プロシージャには、それぞれ **sp_setrepproc** を使用してマーク付けする必要があります。

作業 4：スタンバイ・データベースの追加

rs_init を使用して、スタンバイ・データベースとその RepAgent を複写システムに追加し、アクティブ・データベースのデータでそのスタンバイ・データベースを初期化します。

スタンバイ・データベースを複写システムに追加したら、オペレーション可能な状態にする必要があります。

その後、スタンバイ・データベースのオブジェクトの複写を有効にしたり、スタンバイ・データベースのメンテナンス・ユーザにパーミッションを付与したりすることができます。これらの手順を実行する必要があるかどうかは、スタンバイ・データベースを初期化する方法によって異なります。

1. スタンバイ・データベースがない場合は作成します。
2. スタンバイ・データベースを初期化する方法を決定します。
3. **dump** と **load** を使用してスタンバイ・データベースを初期化する場合は、スタンバイ・データベース・メンテナンス・ユーザを追加します。
4. 複写する前に、**online database** 句を使用して、新しいデータベースをオンラインにします。

スタンバイ・データベースの作成

スタンバイ・データベースがない場合は、必要に応じて適切な Adaptive Server にスタンバイ・データベースを作成する必要があります。

データベース作成の詳細については、『Adaptive Server Enterprise システム管理ガイド』を参照してください。

スタンバイ・データベースを初期化する方法の決定

スタンバイ・データベースは、アクティブ・データベースのデータを使用して初期化します。

次の Adaptive Server のコマンドとユーティリティを使用して、スタンバイ・データベースを初期化します。

- **dump** と **load**
- **bcp**
- **quiesce database ... to** *manifest_file* を使用してマニフェスト・ファイルを生成し、**mount** を使用してスタンバイ・データベースにデータをコピーします。

『Adaptive Server Enterprise リファレンス・マニュアル：コマンド』を参照してください。

Sybase Central または **rs_init** を使用してスタンバイ・データベースを追加すると、Replication Server が「複写有効化」マーカをアクティブ・データベースのトランザ

クション・ログに書き込みます。dump database または dump transaction のいずれかを実行すると、Adaptive Server がダンプ・マーカをアクティブ・データベースのトランザクション・ログに書き込みます。

初期化中にトランザクション処理をサスペンドしない場合は、次の方法を選択します。

- Sybase Central の「ダンプ・マーカ」オプションまたは **rs_init** を選択して、**dump** コマンドと **load** コマンドを使用する。

初期化中にトランザクション処理をサスペンドする場合は、次のいずれかの方法を選択します。

- Sybase Central の「ダンプ・マーカ」オプションまたは **rs_init** を選択しないで、**dump** コマンドと **load** コマンドを使用する。
- **bcp** を使用する。
- **quiesce database ... to** *manifest_file* と **mount** を使用する。

使用されているデータベースが master データベースの場合、**dump** または **load** を使用して、ターゲット・データベースをマテリアライズすることはできません。矛盾を解決するようにデータを処理できる **bcp** などの他の方法を使用できます。

データベース初期化方法の概要

各初期化方法とそれらのマーカの役割に関する注意事項を考慮します。

表 9 : スタンバイ・データベースの初期化に関する注意事項

注意事項	「ダンプ・マーカ」を付けて dump と load を使用する場合	「ダンプ・マーカ」を付けずに dump と load を使用する場合	bcp を使用する場合	mount を使用する場合
クライアント・アプリケーションでの使用	クライアント・アプリケーションに対するトランザクション処理をサスペンドできない場合に使用する。	クライアント・アプリケーションに対するトランザクション処理をサスペンドできる場合に使用する。		クライアント・アプリケーションに対するトランザクション処理をサスペンドできる場合に使用する。

注意事項	「ダンプ・マーカ」を付けて dump と load を使用する場合	「ダンプ・マーカ」を付けずに dump と load を使用する場合	bcp を使用する場合	mount を使用する場合
Replication Server がスタンバイ・データベースへの複写を開始するタイミング	Replication Server は、複写有効化マーカの後ろにある最初のダンプ・マーカから、スタンバイ・データベースへの複写を開始する。	Replication Server は、複写有効化マーカからスタンバイ・データベースへの複写を開始する。		Replication Server は、複写有効化マーカからスタンバイ・データベースへの複写を開始する。
メンテナンス・ユーザのログイン名の作成と、すべてのユーザ ID を一致させる処理	アクティブ Adaptive Server とスタンバイ Adaptive Server の両方にスタンバイ・データベースのメンテナンス・ユーザのログイン名を追加して、サーバ・ユーザの ID を一致させる。 (**dump** と **load** を使用してアクティブ・データベースのデータでスタンバイ・データベースを初期化すると、スタンバイ・データベースの以前の内容がすべてアクティブ・データベースの内容で上書きされるため、アクティブ Adaptive Server でログイン名を作成する。)		スタンバイ・データベースを追加すると、Sybase Central または **rs_init** が、スタンバイ Adaptive Server とスタンバイ・データベースにメンテナンス・ユーザのログイン名とユーザを追加する。	アクティブ Adaptive Server とスタンバイ Adaptive Server の両方にスタンバイ・データベースのメンテナンス・ユーザのログイン名を追加する。サーバ・ユーザの ID を一致させる。 (**mount** を使用してアクティブ・データベースのデータでスタンバイ・データベースを初期化すると、スタンバイ・データベースの以前の内容がすべてアクティブ・データベースの内容で上書きされるため、アクティブ Adaptive Server でログイン名を作成する。)

注意事項	「ダンプ・マーカ」を付けて dump と load を使用する場合	「ダンプ・マーカ」を付けずに dump と load を使用する場合	bcp を使用する場合	mount を使用する場合
スタンバイ・データベースの初期化	**dump** と **load** を使用して、アクティブ・データベースのデータをスタンバイ・データベースに転送する。 データベース・ダンプかトランザクション・ダンプ、またはその両方を使用できる。		**bcp** を使用して、アクティブ・データベースの各複写テーブルをスタンバイ・データベースにコピーする。	**quiesce database ... to** *manifest_file* と **mount database** を使用して、アクティブ・データベースのデータをスタンバイ・データベースに転送する。
アクティブ・データベースのコネクション・ステータス	アクティブ・データベースへのコネクションは変更されない。	Replication Server がアクティブ・データベースへのコネクションをサスペンドする。		Replication Server がアクティブ・データベースへのコネクションをサスペンドする。
コネクションのレジューム	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する。	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て、アクティブ・データベースのトランザクション処理をレジュームする。		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て、アクティブ・データベースのトランザクション処理をレジュームする。

プラットフォーム間でのダンプとロード

プラットフォーム間のダンプとロードは、RepAgent によってスタンバイ・データベースを初期化する場合に使用します。

1. アクティブ・データベース上：
   a) **sp_stop_rep_agent** *database* を使用して RepAgent を停止します。
   b) **dbcc settrunc('ltm', 'ignore')** を使用してセカンダリ・トランケーション・ポイントを削除します。
   c) Adaptive Server でデータベースをシングル・ユーザ・モードで設定します。次のように入力します。

```
sp_dboption database_name, 'single user', true
```

d) データベースのチェックポイントを実行します。
次のように入力します。

```
checkpoint
```

e) Adaptive Server で実行することにより、データベースのトランザクション・ログをダンプします。

```
dump tran database_name with truncate_only
go
```

f) データベースのダンプを取得します。

2. スタンバイ・データベース上：

a) スタンバイ・データベースから取得したダンプをロードします。

プラットフォームのエンディアン・タイプが同じでも、**sp_post_xpload** を実行してインデックスをチェックし、再構築することをおすすめします。

b) トランザクション・ログをダンプして **sp_post_xpload** が作成するログ・レコードを削除します。

```
dump tran database_name with truncate_only
go
```

c) Adaptive Server **sp_indsuspect** システム・プロシージャを実行し、ユーザ・テーブルにサスペクトとマーク付けされたインデックスがないかチェックします。

d) 必要に応じてサスペクト・インデックスを再構築します。文字セットまたはソート順に変更があった場合は、**sp_indsuspect** を実行して、sp_indsuspect でサスペクト・インデックスのあるテーブルが表示されなくなるまで、インデックスの再構築を繰り返す必要があります。

e) **dbcc settrunc ('ltm', 'valid')** を実行して、データベース内のセカンダリ・トランケーション・ポイントを復元し、その後 **rs_zeroltm** を実行して、データベース・ロケータ値を 0 にリセットします。

これらのコマンドを実行すると、RepAgent はセカンダリ・トランケーション・ポイントで開始することができます。

f) **sp_start_rep_agent** *database* を使用して RepAgent を起動します。

『Adaptive Server Enterprise システム管理ガイド 第 2 巻』の「第 11 章 バックアップおよびリカバリ・プランの作成」の「プラットフォーム間でのデータベースのダンプとロード」を参照してください。

トランザクション処理をサスペンドしない場合

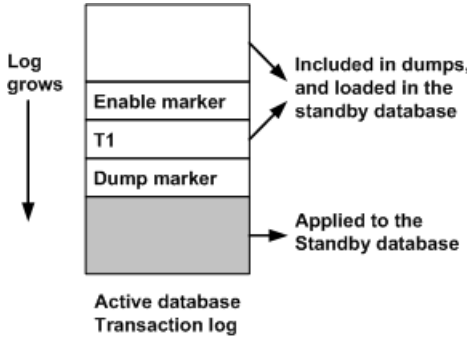
スタンバイ・データベースの初期化中にアクティブ・データベースに対してトランザクション処理をサスペンドしない場合は、スタンバイ・データベースを追加するときに「ダンプ・マーカ」オプションを選択します。

次に **dump** コマンドと **load** コマンドを使用してスタンバイ・データベースを初期化します。

Replication Server は、アクティブ・データベースのトランザクション・ログ内の「複写有効化」マーカの後ろにある最初のダンプ・マーカから、スタンバイ・データベースへの複写を開始します。

この図では、スタンバイ・データベースを追加した後に実行されたトランザクション T1 が、ログの中の複写有効化マーカの後ろに示されています。T1 はダンプに含まれるため、ダンプをロードした後のスタンバイ・データベースに存在します。Replication Server が T1 をスタンバイ・データベースに複写する必要はありません。

図 4 : ダンプ・マーカを使用した dump と load の実行



複写有効化マーカが書き込まれてからアクティブ・データベースのデータがダンプされるまでの間は、アクティブ・データベースでトランザクションを実行できます。

両方のマーカが受信されて、スタンバイ・データベースがオペレーション可能な状態になるまで、最後の完全なデータベース・ダンプと後続のすべてのトランザクション・ダンプをスタンバイ・データベースにロードできます。このあと、オプションで、アクティブ・データベースの最後のトランザクション・ダンプを使用して、スタンバイ・データベースを最新にすることもできます。ダンプに含まれていないトランザクションはすべて複写されます。

Replication Server は、複写有効化マーカと後続の最初のダンプ・マーカの両方を受信するまで、アクティブ・データベースのトランザクションをスタンバイ・データベースに複写しません。両方のマーカを受信してから、Replication Server はスタンバイ・データベースでトランザクションの実行を開始します。

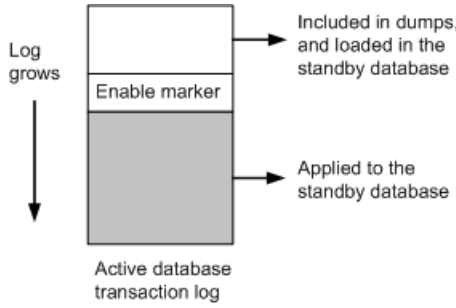
トランザクション処理をサスペンドする場合

スタンバイ・データベースの初期化中にアクティブ・データベースに対してトランザクション処理をサスペンドする場合は、スタンバイ・データベースを追加するときに「ダンプ・マーカ」オプションを選択しません。

スタンバイ・データベースは、**dump** コマンドと **load** コマンド、**bcp**、または **mount** を使用して初期化できます。

Replication Server は、アクティブ・データベースのトランザクション・ログ内の複写有効化マーカから、スタンバイ・データベースへの複写を開始します。クライアント・アプリケーションがサスペンドされるため、複写有効化マーカの後にはトランザクションは発生しません。

図 5 : ダンプ・マーカを使用しない dump と load の実行、または bcp の実行



図に示すように、複写有効化マーカが書き込まれてから、アクティブ・データベースのデータが **dump** コマンドを使用してダンプされるか、**bcp** または **mount** を使用してコピーされるまで、アクティブ・データベースではトランザクションは実行されません。

スタンバイ・データベースが複写有効化マーカを受信するまで、最後の完全なデータベース・ダンプまたは、**bcp** を使用してコピーした複写テーブルの最後のセットをスタンバイ・データベースにロードできます。

このマーカを受信してから、Replication Server はスタンバイ・データベースでトランザクションの実行を開始します。

スタンバイ・データベースのメンテナンス・ユーザの追加

「ダンプ・マーカ」オプションを指定するどうかにかかわらず、**dump** コマンドと **load** コマンドを使用してスタンバイ・データベースを初期化する場合、スタンバイ・データ・サーバとアクティブ・データ・サーバの両方で、スタンバイ・データベースのメンテナンス・ユーザのログイン名を作成する必要があります。

アクティブ・データベースを追加すると、Sybase Central と **rs_init** によって、アクティブ・データ・サーバにアクティブ・データベースのメンテナンス・ユーザが自動的に追加されます。

サーバ・ユーザの ID を一致させる

各データ・サーバ内で、各ログイン名に対するサーバ・ユーザの ID (suid) は、master データベースの syslogins テーブルと各ユーザ・データベースの sysusers テーブルで同じである必要があります。

このことは、ウォーム・スタンバイ・アプリケーションのアクティブ・データベースとスタンバイ・データベースでも当てはまる必要があります。また、サーバ・ユーザの ID とロール設定も、master データベースの syslogins テーブルと sysloginroles テーブルで同じでなければなりません。

サーバ・ユーザの ID を一致させるには、次の 3 つの方法のいずれかを使用します。

- メンテナンス・ユーザ名を含むすべてのログイン名を、両方の Adaptive Server に同じ順序で追加する。Adaptive Server はサーバ・ユーザの ID を順番に割り当てるため、すべてのログイン名に対するサーバ・ユーザの ID が一致する。
- スタンバイ・データベースにダンプをロードした後、スタンバイ・データベースの sysusers テーブルを、スタンバイ Adaptive Server の master データベースの syslogins テーブルと一致させる。
- マスタ Adaptive Server を、使用するすべてのログイン名で管理し、マスタ Adaptive Server の master データベースの syslogins テーブルを、新しく作成したすべての Adaptive Server にコピーする。

メンテナンス・ユーザの追加

スタンバイ・データベースのメンテナンス・ユーザのログイン名を、スタンバイ・データ・サーバとアクティブ・データ・サーバの両方に追加します。

1. スタンバイ・データ・サーバで、**sp_addlogin** システム・プロシージャを実行して、メンテナンス・ユーザのログイン名を作成します。

   sp_addlogin の使用方法の詳細については、『Adaptive Server Enterprise システム管理ガイド』を参照してください。
2. アクティブ・データ・サーバで、**sp_addlogin** を実行して、スタンバイ・データ・サーバで作成したのと同じメンテナンス・ユーザのログイン名を作成します。

   dump コマンドと **load** コマンドを使用してスタンバイ・データベースを設定すると、sysusers テーブルがアクティブ・データベースのその他のデータとともに、スタンバイ・データベースにロードされます。

複写システムへのスタンバイ・データベースの追加

スタンバイ・データベースを初期化してオンラインにし、コネクションをレジュームし、複写システムに追加します。

1. クライアント・アプリケーションに適しており、スタンバイ・データベースを初期化する方法に適している場合は、アクティブ・データベースのトランザクション処理をサスペンドします。

   トランザクション処理をサスペンドしない場合は、「ダンプ・マーカ」を使用して **dump** と **load** を実行してください。
2. Sybase Central または **rs_init** を使用して、スタンバイ・データベースを複写システムに追加します。複写システムにデータベースを追加するための手順を実行します。
3. 論理コネクションのステータスをいつでもモニタできるようにする場合

   次のように入力します。

   ```
   admin logical_status, logical_ds, logical_db
   ```

   `Operation in Progress` 出力カラムと `State of Operation in Progress` 出力カラムに、スタンバイ作成ステータスが表示されます。
4. **dump** と **load** を使用してスタンバイ・データベースを初期化する場合、**dump** コマンドを使用してアクティブ・データベースの内容をダンプし、スタンバイ・データベースをロードします。

   次に例を示します。

   ```
   dump database active_database to dump_device
   ```

   ```
   load database standby_database from dump_device
   ```
5. すでに前回のデータベース・ダンプと後続のトランザクション・ダンプをロードしている場合は、トランザクション・ログをダンプしてそれをスタンバイ・データベースにロードするだけで済みます。

   次に例を示します。

   ```
   dump transaction active_database to dump_device
   ```

   ```
   load transaction standby_database from dump_device
   ```
6. ロード・オペレーションが完了したら、スタンバイ・データベースをオンラインにします。

   ```
   online database standby_database
   ```

   dump と **load**、および **online database** コマンドの使用方法については、『Adaptive Server Enterprise リファレンス・マニュアル』を参照してください。
7. スタンバイ・データベースを初期化します。**bcp** または **quiesce ... to** *manifest_file* と **mount** を使用します。

- **bcp** を使用してスタンバイ・データベースを初期化するには、アクティブ・データベースの各複写テーブルをスタンバイ・データベースにコピーします。
  アクティブ・データベースを複写システムに追加するときに作成した `rs_lastcommit` テーブルをコピーする必要があります。
  bcp プログラムの使用方法については、『Adaptive Server Enterprise ユーティリティ・ガイド』を参照してください。
- **quiesce ... to** *manifest_file* と **mount** を使用してスタンバイ・データベースを初期化するには、そのデータベースをクワイス状態にし、マニフェスト・ファイルを作成します。データベースとログ・デバイスの両方のコピーを作成します。デバイスをスタンバイ・データベースにマウントします。

8. 「ダンプ・マーカ」を選択しないで **dump** と **load** を使用するか、**bcp** を使用するか、または **quiesce database ... to***manifest_file* と **mount** を使用してスタンバイ・データベースを初期化すると、Replication Server はアクティブ・データベースへのコネクションをサスペンドします。アクティブ・データベースへのコネクションをレジュームする必要があります。
   Replication Server で次のように入力します。
   ```
   resume connection to active_ds.active_db
   ```
9. スタンバイ・データベースを初期化する方法に関係なく、次のコマンドを実行して、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する必要があります。
   Replication Server で次のように入力します。
   ```
   resume connection to standby_ds.standby_db
   ```
10. アクティブ・データベースのトランザクション処理がサスペンドされている場合は、その処理をレジュームします。

スタンバイ作成でのブロック化コマンドの使用

wait for create standby Replication Server ブロック化コマンドを使用して、スタンバイ・データベースの準備ができるまで Replication Server がコマンドを受け入れないようにします。

このコマンドは、スタンバイ・データベースを作成するスクリプトで使用できます。構文は次のとおりです。

```
wait for create standby for logical_ds.logical_db
```

スタンバイ・データベースのオブジェクトに対する複写の有効化

スタンバイ・データベースへの切り替えが可能な状態にするには、切り替え後に新しいスタンバイ・データベースに複写する、スタンバイ・データベース内のテーブルとストアド・プロシージャに対して、複写を有効にする必要があります。

- **dump** コマンドと **load** コマンド、または **mount** コマンドを使用してスタンバイ・データベースを初期化した場合、スタンバイ・データベースのテーブルと

ストアド・プロシージャの複写設定は、アクティブ・データベースの複写設定と同じになります。

- **bcp** を使用してスタンバイ・データベースを初期化した場合は、**sp_setreptable** または **sp_reptostandby** と、**sp_setrepproc** を使用して、これらのオブジェクトに対して複写を有効にします。スタンバイ・データベース内のオブジェクトの複写を有効にするには、アクティブ・データベース内のオブジェクトの複写の有効化の手順を使用します。

参照：

• 作業 3：アクティブ・データベースのオブジェクトに対する複写の有効化 (90 ページ)

あとで追加されたオブジェクトの複写の有効化

新しいスタンバイ・データベースに複写する新しいテーブルとユーザ・ストアド・プロシージャを、あとで追加することができます。

- **sp_reptostandby** を使用して、スタンバイ・データベースを複写するようマーク付けした場合、新しいテーブルが自動的に複写するようマーク付けされます。
- **sp_setreplicate** を使用して、新しいスタンバイ・データベースに複写するよう個々のデータベース・テーブルをマーク付けした場合、複写する新しいテーブルをそれぞれ **sp_setreplicate** を使用してマーク付けする必要があります。
- 複写する新しいユーザ・ストアド・プロシージャには、それぞれ **sp_setrepproc** を使用してマーク付けする必要があります。

メンテナンス・ユーザへのパーミッションの付与

スタンバイ・データベースを追加したら、必要なパーミッションをメンテナンス・ユーザに付与する必要があります。

1. システム管理者またはデータベース所有者として Adaptive Server にログインし、作業対象のデータベースを指定します。

   次のように入力します。

   ```
   use standby_database
   ```

2. **replication_role** をメンテナンス・ユーザに付与します。

   次のように入力します。

   ```
   sp_role "grant", replication_role, maintenance_user
   ```

   replication_role では、スタンバイ・データベースでメンテナンス・ユーザが **truncate table** を実行できます。

3. 各テーブルに対して、**grant all** コマンドを実行します。

   次のように入力します。

   ```
   grant all on table_name to maintenance_user
   ```

ASE のウォーム・スタンバイ環境での master データベースの複写

Adaptive Server ウォーム・スタンバイ環境で master データベースを複写するには、いくつかの要件と制限事項があります。

Adaptive Server ログインは master データベース間で複写できます。master データベースの複写は、ログインとロールの管理に使用する DDL コマンドとシステム・コマンドに制限されています。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。

他の master データベースから、**load** を使用してアクティブ・データベースおよびスタンバイ・データベースを初期化しないでください。各 master で **syslogins**、**suids** およびロールを同期させるには、**bcp** を使用して適切なテーブルをリフレッシュするか、または ID とロールを手動で同期してから複写を設定します。

ウォーム・スタンバイ・アプリケーションを設定し、**sp_reptostandby** を使用して複写を有効にするにはいくつかの制限事項と要件があります。また master データベースに適用できるサポートされている複数の DLL とシステム・プロシージャがあります。

スタンバイ master データベースにはアクティブ master データベースより長いパスワード有効期間を設定することをおすすめします。これにより、アクティブ master データベースはパスワードの変更を管理し、パスワード変更の複写を続行できます。

master データベースの複写は、Replication Server のバージョン 12.0 以降ではウォーム・スタンバイ環境でサポートされ、Replication Server 12.6 以降では MSA 環境でもサポートされています。プライマリ Adaptive Server またはアクティブ Adaptive Server は、バージョン 15.0 ESD #2 以降である必要があります。

MSA 環境での master データベースの複写の詳細については、『Replication Server 管理ガイド 第 1 巻』の「MSA を使用した複写オブジェクトの管理」の「MSA 環境での master データベースの複写」を参照してください。

参照：

- sp_reptostandby 使用時の制限および条件 (75 ページ)

- サポートされている DDL コマンドとシステム・プロシージャ (76 ページ)

ウォーム・スタンバイ環境での master データベースの複写の設定

ウォーム・スタンバイ環境での master データベースの複写の設定

1. Replication Server で、ウォーム・スタンバイのペアとして、アクティブ master データベースおよびスタンバイ master データベースを設定します。

   「dump および load を使用してスタンバイを初期化」したり、「ダンプ・マーカを使用してスタンバイへの複写を開始」したりしないでください。各 master で **syslogins** と **suids** を同期するには、**bcp** を使用するか、または ID を手動で同期します。
2. 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の master データベースを、システム・プロシージャを送信するようにマーク付けします。

   次のように入力します。
   ```
   sp_reptostandby master, 'all'
   ```
3. アクティブ master データベース上で RepAgent を停止します。

   次のように入力します。
   ```
   sp_stop_rep_agent master
   ```
4. 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の Replication Agent を、ウォーム・スタンバイ・トランザクションを送信するように設定します。

   次のように入力します。
   ```
   sp_config_rep_agent master, 'send warm standby
   xacts', 'true'
   ```
5. アクティブ master データベース上で RepAgent を再起動します。

   次のように入力します。
   ```
   sp_start_rep_agent master
   ```
6. Replication Server で、アクティブ master データベースとスタンバイ master データベースへの DSI コネクションをレジュームします。

   次のように入力します。
   ```
   resume connection to active_ds.master
   go
   resume connection to standby_ds.master
   go
   ```
7. ウォーム・スタンバイのステータスを確認します。

   次のように入力します。
   ```
   admin logical_status
   ```

参照：

- ASE ウォーム・スタンバイ・データベースの設定 (87 ページ)

アクティブ ASE データベースとスタンバイ ASE データベースの切り替え

アクティブ・データベースで障害が発生した場合またはアクティブ・データベースでメンテナンスを実行する場合、スタンバイ・データベースに切り替えることができます。

切り替えが必要かどうかの判断

アクティブ・データベースからスタンバイ・データベースへの切り替えがいつ必要なのかは、アプリケーションの要件によって判断します。

通常、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・データベースが長時間使用できない状態になった場合は、切り替えが必要になります。

いつ切り替えるかは、アクティブ・データベースで必要なリカバリの規模、アクティブ・データベースとスタンバイ・データベースの同期の程度、ユーザまたはアプリケーションが許容できるダウン時間などによって判断します。

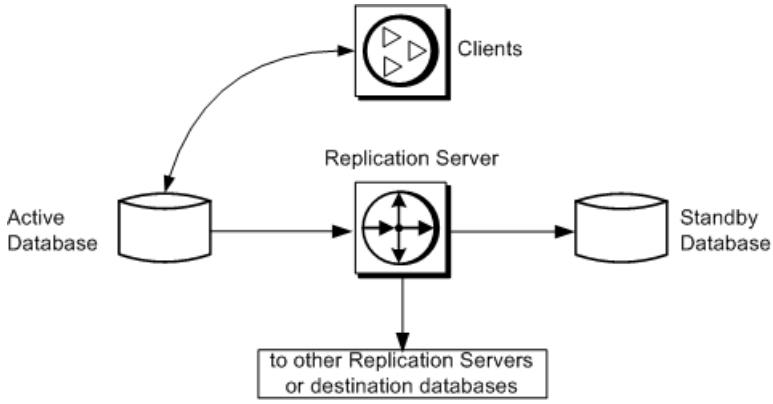
また、アクティブ・データベースまたはそのデータ・サーバで、定期メンテナンスを行うために、アクティブ・データベースとスタンバイ・データベースのロールを切り替える場合もあります。

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前

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前に、関連するプロセス、およびウォーム・スタンバイ環境でのコンポーネントのステータスについて説明します。

この図は、ウォーム・スタンバイ・アプリケーションを使用した通常の処理の例を示しています。

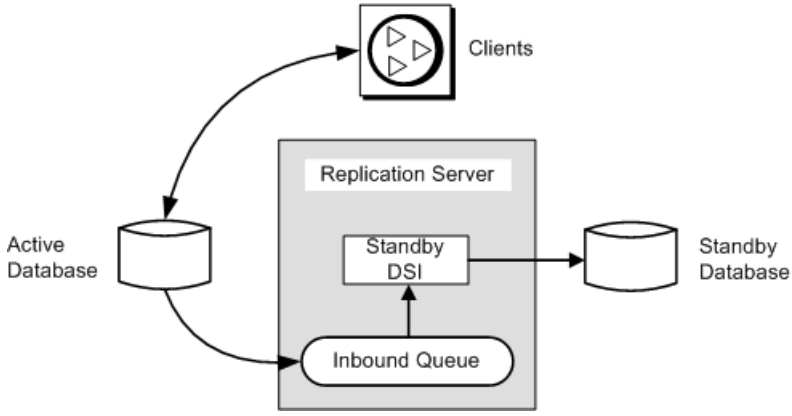
図 6 : ウォーム・スタンバイ・アプリケーション



「ウォーム・スタンバイ・アプリケーションの例 ― 切り替え前」の図

- ウォーム・スタンバイ・アプリケーション自体のアクティビティを介する場合以外は、複写システムに関与しないデータベースに対するウォーム・スタンバイ・アプリケーションを示します。
- 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前の、ウォーム・スタンバイ・アプリケーションの通常のオペレーションを示しています。
- 内部の詳細を加えたもので、次のことを示しています。
  - Replication Server は、アクティブ・データベースから受信したトランザクションを、インバウンド・メッセージ・キューに書き込みます。
    インバウンド・キューとアウトバウンド・キューの詳細については、『Replication Server 管理ガイド 第 1 巻』の「Replication Server の技術的概要」の「Replication Server でのトランザクション処理」の「分散同時制御」を参照してください。
  - このインバウンド・キューはスタンバイ・データベースの DSI スレッドによって読み取られ、このスレッドによってスタンバイ・データベースでトランザクションが実行されます。
    アクティブ・データベースから受信したメッセージは、スタンバイ DSI スレッドがそのメッセージを読み取ってスタンバイ・データベースに適用するまで、インバウンド・キューからトランケートできません。

図 7 : ウォーム・スタンバイ・アプリケーションの例 ― 切り替え前



この例では、トランザクションは、アクティブ・データベースからスタンバイ・データベースに単純に複写されます。論理データベース自体では、次のことは行われません。

- レプリケート・データベースまたはリモート Replication Server に複写されるプライマリ・データの保持
- 別の Replication Server からの複写トランザクションの受信

参照：

- 複写を使用するウォーム・スタンバイ・アプリケーション (124 ページ)

内部での切り替え手順

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場合、Replication Server はいくつかの作業を実行します。

Replication Server は次の作業を行います。

1. アクティブおよびスタンバイ RepAgent コネクションに対し、log suspend を発行します。
2. インバウンド・キューに残されているすべてのメッセージを読み取り、スタンバイ・データベースに適用します。サブスクリプション・データまたは複写ストアド・プロシージャについては、アウトバウンド・キューに適用します。インバウンド・キュー内のコミットされたトランザクションは、すべて切り替えが完了する前に処理されます。
3. スタンバイ DSI をサスペンドします。
4. 新しいアクティブ・データベースのセカンダリ・トランケーション・ポイントを有効にします。
5. 新しいアクティブ・データベースのトランザクション・ログにマーカを設定します。Replication Server はこのマーカを使用して、新しいスタンバイ・データ

ベースや任意のレプリケート・データベースに適用するトランザクションを決定します。

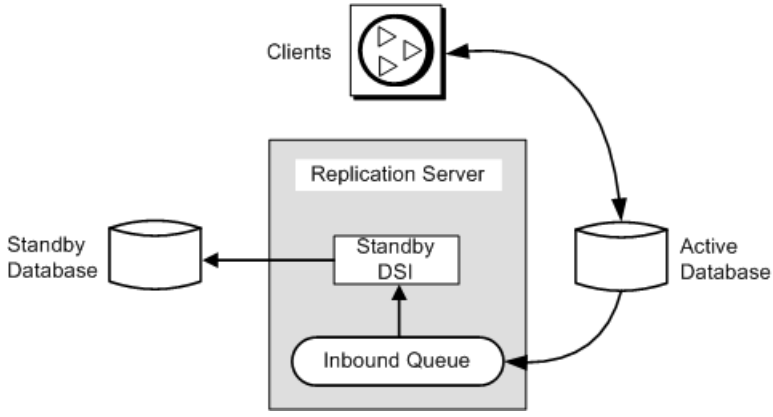
6. ウォーム・スタンバイ・データベースに属する RSSD 内のデータを更新します。
7. 新しいアクティブ・データベースのコネクションをレジュームし、そのデータベースに対するログ転送をレジュームして新しいメッセージを受信できるようにします。

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たあと

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たあとの、関連するプロセス、およびウォーム・スタンバイ環境でのコンポーネントのステータスについて説明します。

アクティブ・データベースとスタンバイ・データベースのロールを切り替えると、複写システムは次の図に示すように変更されます。

図 8 : ウォーム・スタンバイ・アプリケーションの例 ― 切り替え後



- 以前のスタンバイ・データベースが新しいアクティブ・データベースになります。クライアント・アプリケーションは、新しいアクティブ・データベースに切り替えられています。
- この例では、以前のアクティブ・データベースが新しいスタンバイ・データベースになります。以前のアクティブ・データベースに対するメッセージは、新しいアクティブ・データベースに適用するためにキューイングされます。

注意： 切り替え後、前のアクティブ・データベースの Replication Agent が停止し、新しいアクティブ・データベースの Replication Agent が起動されます。

切り替えの実行

アクティブ・データベースからスタンバイ・データベースの切り替え処理は、複数の作業で構成されます。

1. クライアント・アプリケーションがまだアクティブ・データベースを使用している場合は、アクティブ・データベースから切断する。
2. Replication Server で、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。
3. 新しいアクティブ・データベースでクライアント・アプリケーションを再起動する。
4. 新しいアクティブ・データベースに対して RepAgent を起動する。
5. 古いアクティブ・データベースを削除するか、新しいスタンバイ・データベースとして使用するかを決定する。

アクティブ・データベースからのクライアント・アプリケーションの切断

スタンバイ・データベースに切り替える前に、クライアントがアクティブ・データベースでトランザクションを実行しないようにする必要があります。

データベースに障害が発生した場合、当然クライアントはトランザクションを実行できません。ただし、データベースがオンラインに戻った後でクライアントがデータベースを更新しないようにする必要があります。

参照：

- アクティブ・データ・サーバを使用するようにクライアントを設定する (116 ページ)

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

論理コネクションに対してアクティブ・データベースとスタンバイ・データベースを切り替える手順について説明します。

前提条件

切り替える前に、アクティブ・データ・サーバを使用するようにクライアントを設定する必要があります。

手順

1. アクティブ・データベースの Adaptive Server で、RepAgent が停止していることを確認します。停止していない場合は、**sp_stop_rep_agent** を使用して RepAgent を停止します。
2. Replication Server で、switch active コマンドを実行します。

次のように入力します。

```
switch active for logical_ds.logical_db
to data_server.database
```

data_server.database は、新しいアクティブ・データベースです。

3. 切り替えの進行状況をモニタするには、**admin logical_status** を使用します。

   次のように入力します。

   ```
   admin logical_status, logical_ds, logical_db
   ```

   `Operation in Progress` 出力カラムと `State of Operation in Progress` 出力カラムに、切り替えステータスが表示されます。

4. アクティブ・データベースの切り替えが完了したら、新しいアクティブ・データベースに対して RepAgent を再起動する必要があります。

   次のように入力します。

   ```
   sp_start_rep_agent dbname
   ```

次のステップ

注意： Replication Server が切り替えの途中で停止した場合は、Replication Server を再起動すると、切り替えがレジュームします。

参照：

- アクティブ・データ・サーバを使用するようにクライアントを設定する (116 ページ)
- 内部での切り替え手順 (107 ページ)

Switch Active でのブロック化コマンドの使用

wait for switch Replication Server ブロック化コマンドを使用して、スタンバイ・データベースの準備ができるまで Replication Server が待機するようにします。

このコマンドは、アクティブ・データベースを切り替えるスクリプトで使用できます。構文は次のとおりです。

```
wait for switch for logical_ds.logical_db
```

切り替えのモニタリング

admin logical_status を使用すると、切り替えの進行を妨げる複写システムの問題をチェックできます。

このような問題には、スタンバイ・データベースまたはサスペンドされたスタンバイ DSI に対してトランザクション・ログが満杯であることなどがあります。問題を解決できない場合は、**abort switch** コマンドを使用して切り替えをアボートできます。

`Operation in Progress` 出力カラムと `State of Operation in Progress` 出力カラムに、切り替えステータスが表示されます。

たとえば、**admin logical_status** コマンドによって、`State of Operation in Progress` 出力カラムに次のいずれかのメッセージが繰り返し返されると仮定します。

```
Standby has some transactions that have not been applied
```

または

```
Inbound Queue has not been completely read by Distributor
```

これらのメッセージは解決できない問題を示している場合があり、この場合は切り替えをアボートできます。**admin who** コマンドを使用すると、切り替えオペレーションのステータスの詳細を取得できます。

参照：

- ウォーム・スタンバイ・アプリケーションをモニタするためのコマンド (114 ページ)

切り替えのアボート

Replication Server でアクティブ・データベースとスタンバイ・データベースの切り替えがそれほど進行していなければ、**abort switch** コマンドを使用してプロセスをアボートできます。

構文は次のとおりです。

```
abort switch for logical_ds.logical_db
```

abort switch コマンドを実行して **switch active** コマンドを正常にキャンセルした場合、アクティブ・データベースの RepAgenet の再起動が必要になることがあります。

ある特定の時点を経過すると、**switch active** コマンドをキャンセルできなくなります。この場合、**switch active** コマンドが完了するのを待ち、元のアクティブ・データベースに戻すために再度コマンドを使用します。

クライアント・アプリケーションの再起動

admin logical_status コマンドを実行して進行中のオペレーションがないことが表示された場合、または **wait for switch** コマンドを実行して **isql** プロンプトが返された場合は、新しいアクティブ・データベースでクライアント・アプリケーションを再起動できます。

クライアント・アプリケーションは、Replication Server での新しいアクティブ・データベースへの切り替えが完了するまで待機してから、新しいアクティブ・データベースでトランザクションの実行を開始する必要があります。クライアン

トを古いアクティブ・データベースから新しいアクティブ・データベースに切り替えるには、正しい順序で指定する必要があります。

参照：
- アクティブ・データ・サーバを使用するようにクライアントを設定する (116 ページ)

ペーパトレール・トランザクションの解決
古いアクティブ・データベースで障害が発生した場合、新しいアクティブ・データベースに送信されていないトランザクションがあるかどうか確認します。このトランザクションの実行が外部に記録されている場合、このトランザクションを「*ペーパトレール・トランザクション*」と呼びます。

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、切り替えが完了する前に、インバウンド・キュー内のコミットされたトランザクションがすべて新しいアクティブ・データベースに適用されます。ただし、障害が発生する前にアクティブ・データベースでコミットされたいくつかのトランザクションが Replication Server で受信されなかったため、スタンバイ・データベースに適用されていない可能性があります。

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場合、新しいアクティブ・データベースでペーパトレール・トランザクションを再実行できます。依存性がある場合は、ペーパトレール・トランザクションを再実行してからでないと、新しいトランザクションを実行できないことがあります。ペーパトレール・トランザクションを実行するときは、メンテナンス・ユーザのログイン名ではなく必ず元のクライアントのログイン名を使用してください。

古いアクティブ・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する場合、最初にペーパトレール・トランザクションをリバースし、それらがスタンバイ・データベースで複製されないようにします。

古いアクティブ・データベースの管理
新しいアクティブ・データベースに切り替えたら、古いアクティブ・データベースをどうするかを決定する必要があります。

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** を使用してデータベース・コネクションを削除し、あとで新しいスタンバイ・データベースとして再度追加します。データベースを削除すると、そのデータベースにキューイングされていたメッセージはすべて削除されます。『Replication Server リファレンス・マニュアル』の「Replication Server コマンド」の「**drop connection**」を参照してください。

古いアクティブ・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する
古いアクティブ・データベースが損傷を受けていない場合は、新しいスタンバイ・データベースとしてオンラインに戻すことができます。

次のように入力します。

```
resume connection to data_server.database
```

data_server.database は、古いアクティブ・データベースの物理データベース名です。

トランザクションの重複を防ぐには、データベースでペーパトレール・トランザクションを解決する必要があります。アプリケーションによっては、古いアクティブ・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戻す前に、この作業が必要です。

ペーパトレール・トランザクションは、新しいアクティブ・データベースで再実行する必要があるため、トランザクションが複写システムを介して配信されたときにトランザクションを再度受信できるように、新しいスタンバイ・データベースを準備する必要があります。

重複は、次のいずれかの処理で解決できます。

- 新しいスタンバイ・データベース内の重複したトランザクションを取り消すかリバースする。
- 重複したトランザクションを無視し、あとで処理する。

ウォーム・スタンバイ・アプリケーションのモニタリング

Replication Server ログ・ファイルまたはいくつかのコマンドを使用して、2 つの Adaptive Server データベース間または Oracle データベース間でウォーム・スタンバイ・アプリケーションをモニタすることができます。

Replication Server ログ・ファイル

Replication Server ログ・ファイルでは、スタンバイ・データベースの追加時に表示されるメッセージなど、ウォーム・スタンバイ・オペレーションに関するメッセージを参照できます。

作成されたスタンバイ・コネクション
次の例は、スタンバイ・データベースに対する物理コネクションを作成しているときに、Replication Server が書き込むメッセージです。

```
I. 95/11/01 17:47:50. Create starting : SYDNEY_DS.pubs2
I. 95/11/01 17:47:58. Placing marker in TOKYO_DS.pubs2 log
I. 95/11/01 17:47:59. Create completed : SYDNEY_DS.pubs2
```

この例では、SYDNEY_DS がスタンバイ・データ・サーバで、TOKYO_DS がアクティブ・データ・サーバです。

スタンバイ・データベースに対して物理コネクションを作成すると、Replication Server はアクティブ・データベースのトランザクション・ログに「複写有効化」マーカを書き込みます。スタンバイ DSI は、このマーカを受け取るまですべてのトランザクションを無視します。ただし、「ダンプ・マーカ」オプションを選択した場合、スタンバイ DSI はログ内で次のダンプ・マーカを検出するまで引き続きメッセージを無視します。

アクティブ・データベースの Replication Agent からの適切なマーカをスタンバイ・データベースが受信すると、スタンバイ DSI は Replication Server ログ・ファイルにメッセージを書き込み、スタンバイ・データベースで後続のトランザクションの実行を開始します。

上記のメッセージ例では、Replication Server はスタンバイ・データベース SYDNEY_DS.pubs2 のコネクションを作成して、その DSI スレッドをサスペンドしています。この時点で、データベース管理者はアクティブ・データベース TOKYO_DS.pubs2 の内容をダンプし、スタンバイ・データベースにロードします。

初期化後にレジュームされるスタンバイ・コネクション
データベース管理者がスタンバイ・データベースにダンプをロードし、スタンバイ・データベースへの接続をレジュームすると、スタンバイ DSI はアクティブ・データベースからのメッセージの処理を開始します。Replication Server は、次のようなログ・メッセージを書き込みます。

```
I. 95/11/01 18:50:34. The DSI thread for database 'SYDNEY_DS.pubs2'
is started.
I. 95/11/01 18:50:41. Setting LTM truncation to 'ignore' for
SYDNEY_DS.pubs2 log
I. 95/11/01 18:50:43. DSI for SYDNEY_DS.pubs2 received and processed
Enable
    Replication Marker. Waiting for Dump Marker
I. 95/11/01 18:50:43. DSI for SYDNEY_DS.pubs2 received and processed
Dump
    Marker. DSI is now applying commands to the Standby
```

ログ・ファイルにこの最後のメッセージが表示された時点で、ウォーム・スタンバイ・データベースの作成プロセスが完了します。

ウォーム・スタンバイ・アプリケーションをモニタするためのコマンド

ウォーム・スタンバイ・アプリケーションのステータスをモニタするには、**admin** コマンドを使用します。

これらのコマンドの詳細については、『Replication Server リファレンス・マニュアル』の「Replication Server コマンド」を参照してください。

admin logical_status

admin logical_status コマンドを実行すると、次のことを確認できます。

- スタンバイ・データベースの追加、またはアクティブ・データベースとスタンバイ・データベースの切り替えの進行状況。
- アクティブ・データベース・コネクションまたはスタンバイ・データベース・コネクションがサスペンドされているかどうか。
- スタンバイ DSI がメッセージを無視しているかどうか。スタンバイ DSI は、マーカがアクティブ・データベースからトランザクション・ストリームに到達するのを待機している間、メッセージを無視する。

admin who, dsi

スタンバイ DSI のステータスをチェックする別の方法として、**admin who, dsi** コマンドを使用できます。`IgnoringStatus` 出力カラムには、次のいずれかが表示されます。

- “Applying” — DSI がメッセージをスタンバイ・データベースに適用している場合
- “Ignoring” — DSI がマーカを待機している場合

admin who, sqm

admin who, sqm コマンドを実行すると、ステーブル・キューのステータスに関する情報が表示されます。ウォーム・スタンバイ・アプリケーションでは、インバウンド・キューは (ディストリビュータ・スレッドを無効にしていない場合) ディストリビュータ・スレッドとスタンバイ DSI スレッドによって読み取られます。Replication Server は、両方のスレッドがインバウンド・キューのメッセージを読み取って配信するまで、そのメッセージを削除できません。

Replication Server がインバウンド・キューのメッセージを削除していない場合、**admin who, sqm** コマンドを使用して問題を調査できます。コマンドを実行すると、キューを読み取っているスレッド数とキュー内の最小削除ポイントを確認できます。

admin sqm_readers

admin sqm_readers コマンドを実行すると、インバウンド・キューを読み取っている個々のスレッドの読み取りポイントと削除ポイントがモニタされます。インバウンド・キューが削除されていない場合、**admin sqm_readers** を実行して、キューを読み取っていないスレッドを検出できます。

admin sqm_readers コマンドは、論理コネクションのキュー番号とキュー・タイプの 2 つのパラメータを取ります。

キュー番号とキュー・タイプは、**admin who, sqm** コマンド出力の `Info` カラムに表示されます。コロンの左側の 3 桁の数字がキュー番号で、コロンの右側の数字がキュー・タイプです。

キュー・タイプ 1 は、インバウンド・キューです。キュー・タイプ 0 は、アウトバウンド・キューです。論理コネクションのインバウンド・キューは、複数のスレッドによって読み取ることができます。たとえば、キュー番号 102 のインバウンド・キューを読み取っているスレッドについて調べるには、次のように **admin sqm_readers** を実行します。

```
admin sqm_readers, 102, 1
```

アクティブ・データ・サーバを使用するようにクライアントを設定する

Replication Server で **switch active** 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・データベースを切り替える場合、Replication Server ではクライアント・アプリケーションを新しいアクティブ・データ・サーバとアクティブ・データベースに自動的に切り替えないため、クライアント・アプリケーションを切り替える方法を工夫する必要があります。

現在のアクティブ・データ・サーバに接続するようクライアント・アプリケーションを設定するための方法の例を 3 つ挙げます。作成できるのは、次のものです。

- 2 つの interfaces ファイル
- クライアント・アプリケーションに対するデータ・サーバの記号名が指定された 1 つの interfaces ファイル・エントリ
- 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メカニズム

この方法は、ウォーム・スタンバイ・データベースを設定する前に実行する必要があります。

アプリケーションを切り替える方法に関係なく、Replication Server によって使用される interfaces ファイル・エントリは修正しないでください。

2 つの interfaces ファイル

この方法では、クライアント・アプリケーション用と Replication Server 用の 2 つの interfaces ファイルを設定します。

クライアントを切り替えた場合、interfaces ファイル・エントリを修正して、新しいアクティブ・データベースのデータ・サーバのホスト名とポート番号を使用できます。

クライアント・アプリケーション用のデータ・サーバの記号名

この方法では、クライアント・アプリケーションに対するデータ・サーバの記号名を指定した interfaces ファイル・エントリを作成します。

interfaces ファイルには、データ・サーバ名、ホスト名、ポート番号のエントリが含まれる可能性があります。

表 10 : interfaces ファイル内のデータ・サーバの記号名

	データ・サーバ名	ホスト名	ポート番号
クライアント・アプリケーション	CLIENT_DS	*machine_1*	2800
アクティブ・データベース	TOKYO_DS_X	*machine_1*	2800
スタンバイ・データベース	TOKYO_DS_Y	*machine_2*	2802

サーバ名が CLIENT_DS のデータ・サーバに対して interfaces エントリを作成できます。クライアント・アプリケーションは常に CLIENT_DS に接続します。CLIENT_DS エントリは、アクティブ・データベースのデータ・サーバと同じホスト名とポート番号を使用します。

Replication Server が接続するホスト名とポート番号はクライアント・アプリケーションと同じですが、使用するデータ・サーバ名は異なります。この例では、Replication Server は TOKYO_DS_X データ・サーバと TOKYO_DS_Y データ・サーバ間で切り替えを行います。

アクティブ・データベースを切り替えた後に、CLIENT_DS の interfaces エントリを、新しいアクティブ・データベースのデータ・サーバのホスト名とポート番号に変更します。この例では、*machine_2* とポート番号 2802 に変更します。

現在のアクティブ・データ・サーバへのクライアント・データ・サーバのマップ

この方法では、クライアント・アプリケーションのデータ・サーバ・コネクションが現在のアクティブ・データ・サーバに自動的にマップされる、中間の Open Server アプリケーションなどのメカニズムを作成します。

Open Server アプリケーションなどの作成方法の詳細については、『Open Server Server-Library/C リファレンス・マニュアル』などの Open Server のマニュアルを参照してください。

ウォーム・スタンバイ・データベース・コネクションの変更

論理データベース・コネクションと物理データベース・コネクションを再設定または修正するオプションについて説明します。通常の環境では、ウォーム・スタ

ンバイ・アプリケーションを一般的な手順で設定すると、デフォルトの設定が正しく機能します。

論理コネクションの変更

ウォーム・スタンバイ論理データベース・コネクションのパラメータを修正するには、**alter logical connection** コマンドを使用します。

次のようなパラメータを修正するには、**alter logical connection** コマンドを使用します。

- 論理コネクションに影響を与えるパラメータ
- ディストリビュータ・スレッドを有効または無効にするパラメータ
- スタンバイ・データベースへの **truncate table** の複写を有効または無効にするパラメータ

論理コネクションに影響を与えるパラメータの変更

論理コネクションに影響するパラメータを更新するには、**alter logical connection** コマンドを使用します。

Replication Server にログインし、**isql** プロンプトで次のコマンドを入力します。

```
alter logical connection
 to logical_ds.logical_db
set logical_database_param to 'value'
```

logical_ds は論理コネクションのデータ・サーバ名、*logical_db* は論理コネクションのデータベース名、*logical_database_param* は論理データベース・パラメータ、*value* はパラメータの文字列設定です。

新しい設定は、すぐに有効になります。

論理コネクションに影響を与える設定パラメータ

論理コネクションを設定するために使用できるいくつかのパラメータがあります。

警告！ ステーブル・キューの領域がかなり不足している場合にのみ、論理コネクション・パラメータ **materialization_save_interval** と **save_interval** を再設定してください。これらの値を (**strict** から分単位の指定値に) 再設定すると、スタンバイ・データベースでメッセージのロスが発生する可能性があります。

表 11 : 論理コネクションに影響を与える設定パラメータ

logical_database_param	*value*
deferred_name_resolution	Replication Server で遅延名前解決を有効にし、Adaptive Server での遅延名前解決をサポートする。 Replication Server で遅延名前解決のサポートを有効にする前に、レプリケート Adaptive Server で遅延名前解決がサポートされていることを確認する。 デフォルト値は off **注意：** このパラメータは Adaptive Server でのみ使用可能です。
materialization_save_interval	マテリアライゼーション・キューのセーブ・インターバル。このパラメータは、ウォーム・スタンバイ・アプリケーション内のスタンバイ・データベースに対してのみ使用する。 デフォルト値はスタンバイ・データベースでは “strict”
replicate_minimal_columns	Replication Server がすべてのトランザクションのすべての複写定義カラムを送信するか、スタンバイ・データベースで更新オペレーションまたは削除オペレーションを実行する必要があるカラムだけを送信するかを指定する。値は “on” または “off”。 Replication Server は、複写定義のパラメータに “send standby” パラメータが含まれていないか、複写定義が存在しない場合にのみ、スタンバイ時にこの値を使用する。 それ以外の場合は、複写定義内の "replicate minimal columns" パラメータまたは "replicate all columns" パラメータの値を使用する。 デフォルト値は on **dsi_compile_enable** を ‘on’ に設定すると、**replicate_minimal_columns** に設定した値は無視される。
save_interval	メッセージが送信先データ・サーバに正常に渡された後に、Replication Server がそのメッセージを保存しておく時間 (分単位)、すなわちセーブ・インターバルを指定する。 デフォルト値は 0 分

logical_database_param	*value*
send_standby_repdef_cols	論理コネクション用にスタンバイ・データベースに送信されるカラムを指定する。これは、Replication Server に対して、スタンバイ・データベースに送信するテーブル・カラムを指示する複写定義内の "send standby" オプションよりも優先される。指定できる値は、次のとおり。 • **on** – 照合する複写定義で指定されているテーブル・カラムだけを送信する。複写定義内の "send standby" オプションは無視される。 • **off** – すべてのテーブル・カラムをスタンバイ・データベースに送信する。複写定義内の "send standby" オプションは無視される。 • **check_repdef** – "send standby" オプションの設定に基づいて、すべてのテーブル・カラムをスタンバイ・データベースに送信する。 デフォルト値は **check_repdef**

参照：
- リカバリのためのセーブ・インターバル (386 ページ)

ディストリビュータ・スレッドの無効化

ディストリビュータ・スレッドを無効にするには、**alter logical connection** コマンドを使用します。

アクティブ・データベースのデータをスタンバイ・データベース以外のデータベースに複写しない場合、Replication Server では論理コネクション用のディストリビュータ・スレッドは必要ありません。ディストリビュータ・スレッドを無効にすると、Replication Server リソースを節約できます。

ディストリビュータ・スレッドを無効にするには、最初に論理データベースのデータに対するサブスクリプションをすべて削除する必要があります。次に、Replication Server で **alter logical connection** を実行します。

```
alter logical connection
 to logical_ds.logical_db
set distribution off
```

あとでアクティブ・データベースからデータを複写する場合、このコマンドを使用してディストリビュータ・スレッドを再度有効にできます。

警告！ ディストリビュータ・スレッドを無効にしたあとで複写システムからスタンバイ・データベースを削除すると、アクティブ・データベースからインバウンド・キューを読み取るための Replication Server のスレッドが存在しなくなります。別のスタンバイ・データベースを追加して論理コネクションに対する分配を "on"

に設定するか、複写システムからアクティブ・データベースを削除するまで、インバウンド・キューへの書き込みが続きます。

スタンバイ・データベースへのトランケート・テーブルの複写

alter logical connection コマンドを使用すると、**truncate table** コマンドの複写を有効または無効にできます。

Replication Server は、**truncate table** の実行をウォーム・スタンバイ・データベースにコピーします。アクティブ・データベースとスタンバイ・データベースは、この機能をサポートするよう Adaptive Server バージョン 11.5 以降である必要があります。

truncate table の複写を有効または無効にするには、送信元 Replication Server にログインし、次のコマンドを入力します。

```
alter logical connection
 to logical_ds.logical_db
set send_truncate_table to {on | off}
```

Replication Server バージョン 11.5 以降にアップグレードするか、それをインストールする*前に*ウォーム・スタンバイ・アプリケーションを作成した場合、**alter logical connection** を使用してこの機能を有効にするまで、Replication Server は **truncate table** をスタンバイ・データベースにコピーしません。既存のウォーム・スタンバイ・アプリケーションとの互換性を維持するため、デフォルト設定は "off" です。

Replication Server バージョン 11.5 以降にアップグレードしたか、それをインストールした*あとに*ウォーム・スタンバイ・アプリケーションを作成した場合、**alter logical connection** を使用してこの機能を無効にするまで、Replication Server は **truncate table** をスタンバイ・データベースに自動的にコピーします。デフォルト設定は "on" です。

物理コネクションの変更

ウォーム・スタンバイ・アプリケーションに対する物理コネクションに影響するパラメータを変更するには、送信元 Replication Server で **alter connection** コマンドを使用します。

構文は次のとおりです。

```
alter connection to data_server.database
set database_param to 'value'
```

data_server は送信先データ・サーバ、*database* はデータ・サーバが管理するデータベース、*database_param* はコネクションに影響するパラメータ、*value* は *database_param* に対する設定です。

変更前に必ずコネクションをサスペンドしてください。その後、**alter connection** を実行してから、新しいパラメータ設定のコネクションをレジュームして有効にします。『Replication Server 管理ガイド 第 1 巻』の「データベース・コネクションの管理」の「データベース・コネクションの変更」を参照してください。

スタンバイ・データベースでのトリガの設定

alter connection コマンドを使用して、トリガの起動を有効または無効にするようにコネクションを設定することができます。

デフォルトでは、スタンバイ DSI スレッドは、スタンバイ・データベースにログインするときに Adaptive Server の **set triggers off** コマンドを実行します。これによって Adaptive Server は複写トランザクションに対してトリガを起動できないため、スタンバイ・データベースでの更新の重複を回避できます。

alter connection コマンドを使用して、トリガの起動を有効または無効にするようにコネクションを設定することで、デフォルトの動作を変更できます。そのためには、**dsi_keep_triggers** 設定パラメータを "on" または “off” に設定します。スタンバイ・データベースを除くすべてのコネクションに対するデフォルトの **dsi_keep_triggers** 設定は "on" です。

スタンバイ・データベースでの複写の設定

dsi_replication 設定パラメータを設定して、DSI によって適用されたトランザクションを、複写されたものとしてトランザクション・ログ内でマーク付けするかどうかを指定します。

アクティブ・レプリケート・データベースに対しては、"on" に設定する必要があります。デフォルトでは、スタンバイ・データベースに対しては "off"、他のすべてのデータベースに対しては "on" に設定されます。

dsi_replication を “off” に設定すると、DSI はデータベースで **set replication off** を実行し、DSI が実行するトランザクションのログ・レコードに、Adaptive Server が複写情報を追加しないようにします。これらのトランザクションは、メンテナンス・ユーザによって実行されるため、(スタンバイ・データベースがある場合を除いて) これ以上複写されることはありません。そのため、必要に応じてこのパラメータを "off" に設定すると、トランザクション・ログに書き込まれる情報は少なくなります。

コネクションに対してこのパラメータがどのように設定されているかを確認するには、**admin who, dsi** を使用します。

スタンバイ・データベースの設定パラメータの変更

スタンバイ・データベースを作成するときに、複数の設定パラメータがアクティブ・データベースに対して設定されている場合は、そのパラメータがアクティ

ブ・データベースからスタンバイ・データベースにコピーされます。これらの設定パラメータの設定は変更できます。

表 12 : スタンバイ・データベースにコピーされる設定パラメータ

batch	batch_begin	command_retry
db_packet_size	dsi_cmd_separator	dsi_charset_convert
dsi_cmd_batch_size	dsi_keep_triggers	dsi_fadeout_time
dsi_isolation_level	dsi_max_text_to_log	dsi_large_xact_size
dsi_max_cmds_to_log	dsi_replication	dsi_num_large_xact_threads
dsi_num_threads	dsi_xact_group_size	dsi_serialization_method
dsi_sqt_max_cache_size	dsi_xact_in_group	dump_load
parallel_dsi	use_batch_markers	

『Replication Server 管理ガイド 第 1 巻』の「データベース・コネクションの管理」を参照してください。

論理データベース・コネクションの削除

ウォーム・スタンバイ・アプリケーションを取り除く場合、複写システムから論理データベースを削除する必要があります。

これには、スタンバイ・データベースを削除してから、**drop logical connection** コマンドを実行します。必ずスタンバイ・データベースを削除してから、このコマンドを実行してください。物理データベース・コネクションの削除の詳細については、『Replication Server 管理ガイド 第 1 巻』の「データベース・コネクションの管理」の「データベース・コネクションの削除」を参照してください。

drop logical connection の構文は、次のとおりです。

```
drop logical connection to data_server.database
```

data_server は論理データ・サーバ、*database* は論理データベースです。

たとえば、LDS 論理データ・サーバにある `pubs2` 論理データベースへのコネクションを削除するには、次のように入力します。

```
drop logical connection to LDS.pubs2
```

ID サーバからの論理データベースの削除

ウォーム・スタンバイ・アプリケーションが複写システムに存在する場合、ID サーバの RSSD 内の `rs_idnames` システム・テーブルに、論理データベースが物理データベース、データ・サーバ、Replication Server とともにリストされます。場

合によっては、このシステム・テーブルから論理データベースのエントリを削除する必要があります。

たとえば、**drop logical connection** コマンドが失敗した場合、ID サーバで、論理データベースに対応するローを `rs_idnames` システム・テーブルから強制的に削除する必要があります。論理データベース・コネクションは、`ltype` カラムに "L" を示します。

sysadmin dropldb コマンドを実行すると、ID サーバにログインして、指定の論理データベースのエントリを削除します。構文は次のとおりです。

```
sysadmin dropldb, data_server, database
```

data_server は論理データ・サーバ名、*database* は論理データベース名です。

sysadmin コマンドを実行するには、**sa** パーミッションが必要です。

複写を使用するウォーム・スタンバイ・アプリケーション

複写に関与するウォーム・スタンバイ・アプリケーションについて説明します。ここで、論理データベースは、複写システムのプライマリ・データベースまたはレプリケート・データベースとして機能します。

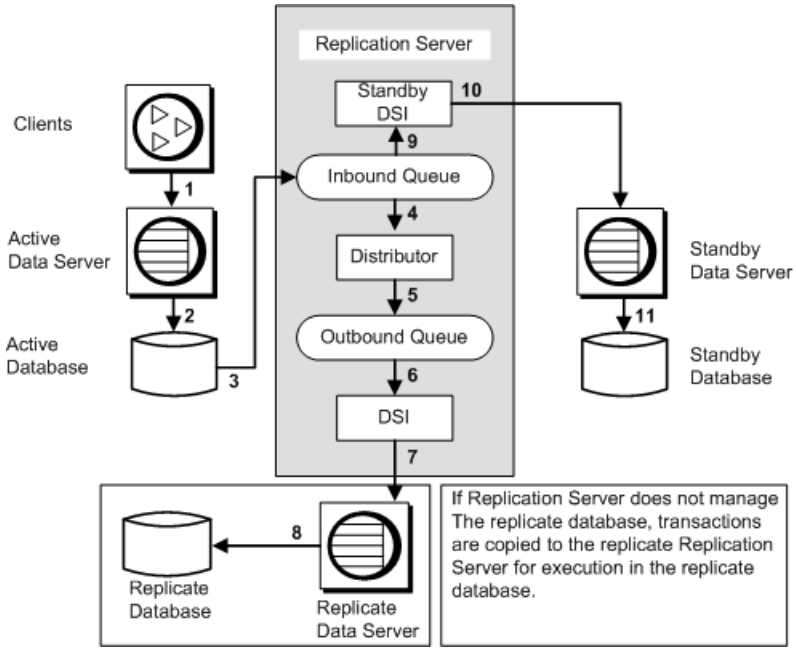
プライマリ・データベース用のウォーム・スタンバイ・アプリケーション

プライマリ・データベース用のウォーム・スタンバイ・アプリケーションについて説明します。

この図は、プライマリ・データベース用のウォーム・スタンバイ・アプリケーションの例を示しています。この例では、1 つの Replication Server が次の 3 つのデータベースを管理しています。

- 論理プライマリ・データベース用のアクティブ・データベース
- 論理プライマリ・データベース用のスタンバイ・データベース
- 論理プライマリ・データベースのデータに対するサブスクリプションを持つレプリケート・データベース

図 9 : プライマリ・データベース用のウォーム・スタンバイ・アプリケーション



この例では、1 つの Replication Server がプライマリ・データベースとレプリケート・データベースの両方を管理しています。他のインスタンスでは、プライマリ・データベースとレプリケート・データベースを管理する Replication Server がそれぞれ異なることがあります。

図中の番号は、プライマリ・データベース用のウォーム・スタンバイ・アプリケーション内の複写システムを介した、クライアント・アプリケーションからのトランザクションの流れを示しています。

クライアント・アプリケーションからインバウンド・キューまで
この図では、番号 1 ～ 3 は、クライアントから Replication Server 内のインバウンド・キューへのトランザクションを示します。

- クライアントは、アクティブ・プライマリ・データ・サーバでトランザクションを実行します。
- アクティブ・プライマリ・データ・サーバは、アクティブ・プライマリ・データベースを更新します。
- アクティブ・プライマリ・データベースの Replication Agent スレッドが、データベース・ログの複写データに対するトランザクションを読み込み、それらを Replication Server に転送します。Replication Server は、それらのトランザクションをインバウンド・キューに書き込みます。

複写データに対するトランザクションは、メンテナンス・ユーザによって実行されるものも含めて、すべてスタンバイ・データベース内のアプリケーション用の Replication Server に送信されます。

インバウンド・キューからレプリケート・データベースまで
この図では、番号 4 ～ 8 は、インバウンド・キューからレプリケート・データベースへのトランザクションの流れを示します。

- ディストリビュータ・スレッドは、インバウンド・キューからトランザクションを読み取ります。
- ディストリビュータ・スレッドは、サブスクリプションに対してトランザクションを処理し、複写トランザクションをアウトバウンド・キューに書き込みます。
  メンテナンス・ユーザが実行するトランザクションは、スタンバイ・データベースには常に複写されますが (**sp_config_rep_agent** を使用して RepAgent を設定するときに、**send_warm_standby_xacts** パラメータを設定したため)、RepAgent に対して **send_maint_xacts_to_replicate** パラメータも設定しないとレプリケート・データベースには複写されません。

  注意： Oracle の場合、**filter_maint_userid** 設定パラメータは、パラメータが "true" または "false" のどちらに設定されているかにかかわらず、Replication Agent for Oracle では無効なため、メンテナンス・ユーザが実行したトランザクションは、常にレプリケート・データベースに複写されます。
- DSI スレッドは、アウトバウンド・キューからトランザクションを読み取ります。
- DSI スレッドは、レプリケート・データ・サーバでトランザクションを実行します。
- レプリケート・データ・サーバは、レプリケート・データベースを更新します。
  別の Replication Server が管理するデータベースに複写されるトランザクションは、DSI スレッドではなく RSI スレッドが管理する RSI アウトバウンド・キューに書き込まれます。RSI スレッドは、その別の Replication Server にトランザクションを配信します。

インバウンド・キューからスタンバイ・データベースまで
図では、番号 9 ～ 11 は、インバウンド・キューから論理プライマリ・データベース用のスタンバイ・データベースへのトランザクションの流れを示します。

- スタンバイ DSI スレッドは、インバウンド・キューからトランザクションを読み取ります。
- スタンバイ DSI スレッドは、スタンバイ・データ・サーバでトランザクションを実行します。

- スタンバイ・データ・サーバは、スタンバイ・データベースを更新します。

インバウンド・キューは、スタンバイ DSI スレッドとディストリビュータ・スレッドによって読み取られます。この 2 つのスレッドは同時に動作します。両方のスレッドがメッセージを読み取って送信先に配信するまで、それらのメッセージはインバウンド・キューからトランケートできません。DSI がメッセージをスタンバイ・データベースに適用するまで、それらのメッセージはキューに残ります。また、サブスクリプションまたは複写ストアド・プロシージャの実行がある場合は、ディストリビュータ・スレッドがそれらをアウトバウンド・キューに書き込みます。

使用している複写システムによっては、トランザクションがレプリケート・データベースより先にスタンバイ・データベースに複写されることがあります。ただし、Replication Server では、スタンバイ・プライマリ・データベースとレプリケート・データベースは確実にアクティブ・プライマリ・データベースと同期がとれた状態になります。

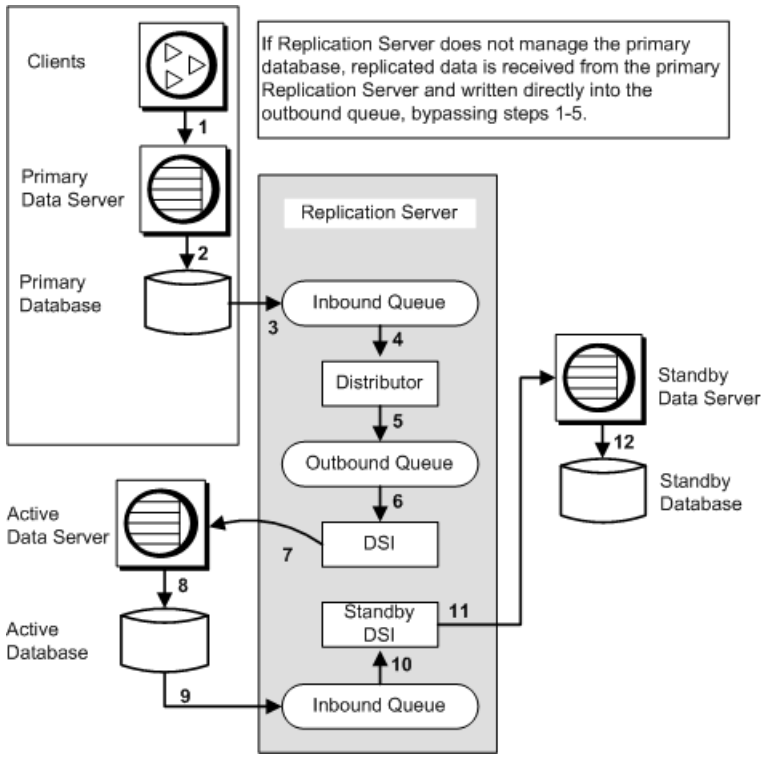
レプリケート・データベース用のウォーム・スタンバイ・アプリケーション

レプリケート・データベース用のウォーム・スタンバイ・アプリケーションについて説明します。

この図は、レプリケート・データベース用のウォーム・スタンバイ・アプリケーションの例を示しています。この例では、1 つの Replication Server が次の 3 つのデータベースを管理しています。

- プライマリ・データベース
- 論理レプリケート・データベース用のアクティブ・データベース
- 論理レプリケート・データベース用のスタンバイ・データベース

図 10 : レプリケート・データベース用のウォーム・スタンバイ・アプリケーション



論理レプリケート・データベースには、プライマリ・データベースのデータに対するサブスクリプションがあります。このため、プライマリ・データベースからの更新は、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に複写されます。

この例では、1 つの Replication Server がプライマリ・データベースとレプリケート・データベースの両方を管理しています。他のインスタンスでは、プライマリ・データベースとレプリケート・データベースを管理する Replication Server がそれぞれ異なることがあります。

図中の番号は、レプリケート・データベース用のウォーム・スタンバイ・アプリケーション内の複写システムを介した、クライアント・アプリケーションからのトランザクションの流れを示しています。

クライアント・アプリケーションからプライマリ・データベースおよびアクティブ・データベースまで
図では、番号 1 ～ 8 は、クライアントからプライマリ・データベースと通常の複写を使用したアクティブ・レプリケート・データベースへのトランザクションの流れを示します。

- クライアントは、プライマリ・データ・サーバでトランザクションを実行します。
- プライマリ・データ・サーバは、プライマリ・データベースを更新します。
- プライマリ・データベースの Replication Agent は、トランザクション・ログの複写データに対するトランザクションを読み込み、それらを Replication Server に転送します。Replication Server は、それらのトランザクションをインバウンド・キューに書き込みます。
- ディストリビュータ・スレッドは、インバウンド・キューからトランザクションを読み取ります。
- ディストリビュータ・スレッドは、サブスクリプションに対してトランザクションを処理し、複写トランザクションをアウトバウンド・キューに書き込みます。
  レプリケート・データベース用のウォーム・スタンバイ・アプリケーションを管理する Replication Server がプライマリ・データベースを管理しない場合は、複写データがプライマリ Replication Server から受信され、アウトバウンド・キューに直接書き込まれます。手順 1 ～ 5 は省略されます。
- DSI スレッドは、アウトバウンド・キューからトランザクションを読み取ります。
- DSI スレッドは、ウォーム・スタンバイ・アプリケーションのアクティブ・データ・サーバであるレプリケート・データ・サーバでトランザクションを実行します。
- アクティブ・データ・サーバは、アクティブ・データベースを更新します。
  別の Replication Server が管理するプライマリ・データベースで開始するトランザクションは、プライマリ Replication Server のディストリビュータ・スレッドによって RSI アウトバウンド・キューに書き込まれます。その後、これらのトランザクションは、論理レプリケート・データベースのアクティブ・データベースに適用されるように、レプリケート Replication Server の DSI アウトバウンド・キューに複写されます。

アクティブ・データベースからスタンバイ・データベースまで
図では、番号 9 ～ 12 は、論理レプリケート・データベース用のアクティブ・データベースからスタンバイ・データベースへのトランザクションの流れを示します。

- アクティブ・データベースの Replication Agent は、アクティブ・データベース・ログのトランザクションを読み込み、それらを Replication Server に転送し

ます。Replication Server は、それらのトランザクションをインバウンド・キューに書き込みます。
複写データに対するトランザクションは、メンテナンス・ユーザによって実行されるものも含めて、すべてスタンバイ・データベース内のアプリケーション用の Replication Server に送信されます。

- スタンバイ DSI スレッドは、インバウンド・キューからトランザクションを読み取ります。
- スタンバイ DSI スレッドは、スタンバイ・データ・サーバでトランザクションを実行します。
- スタンバイ・データ・サーバは、スタンバイ・データベースを更新します。

論理コネクションのセーブ・インターバルの設定

DSI キューのセーブ・インターバルまたはマテリアライゼーション・キューのセーブ・インターバルを使用すると、論理レプリケート・データベースのセーブ・インターバルを再設定できます。

コネクションのセーブ・インターバルは、メッセージがステーブル・キューに保持されてから削除されるまでの時間を指定します。ウォーム・スタンバイ・アプリケーションを一般的な手順で設定すると、デフォルトの設定が正しく機能します。

configure logical connection を使用すると、論理コネクションの DSI キューとマテリアライゼーション・キューのセーブ・インターバルを設定できます。

『Replication Server リファレンス・マニュアル』の「Replication Server コマンド」の「**configure replication server**」を参照してください。

警告！ 論理コネクションの DSI キューとマテリアライゼーション・キューのセーブ・インターバル設定は、ステーブル・キューの領域不足によって重大な問題が発生している場合にのみ再設定してください。これらのセーブ・インターバルを (**strict** から分単位の指定値に) 再設定すると、スタンバイ・データベースでメッセージのロスが発生する可能性があります。Replication Server では、このようなロスが検出できないため、ユーザ自身がスタンバイ・データベースの整合性を確認しなければなりません。

DSI キューのセーブ・インターバル

デフォルトでは、論理コネクションの DSI キューのセーブ・インターバルは、スタンバイ・データベースの作成時に **'strict'** に設定されます。

これで、Replication Server は DSI キューのメッセージを、メッセージがスタンバイ・データベースに配信されるまで保持します。論理コネクションの DSI キューのセーブ・インターバルを変更する必要がある場合は、**configure logical connection** コマンドを使用します。

たとえば、レプリケート Replication Server に対して、その論理レプリケート・データ・サーバ LDS に送信先が指定されているメッセージを、1 時間 (60 分) 保存させるようにするには、次のコマンドを入力します。

```
configure logical connection to LDS.logical_pubs2
set save_interval to '60'
```

このセーブ・インターバルを **'strict'** にリセットするには、次のように入力します。

```
configure logical connection to LDS.logical_pubs2
set save_interval to 'strict'
```

マテリアライゼーション・キューのセーブ・インターバル
デフォルトでは、論理コネクションのマテリアライゼーション・キューのセーブ・インターバルは、サブスクリプションの作成時に **'strict'** に設定されます。

これで、Replication Server はマテリアライゼーション・キューのメッセージを、メッセージがスタンバイ・データベースに配信されるまで保持します。論理コネクションのマテリアライゼーション・キューのセーブ・インターバルを変更する必要がある場合は、**configure logical connection** コマンドを使用します。

たとえば、レプリケート Replication Server に対して、その論理レプリケート・データ・サーバ LDS のマテリアライゼーション・キューにあるメッセージを、1 時間 (60 分) 保存させるようにするには、次のコマンドを入力します。

```
configure logical connection to LDS.logical_pubs2
set materialization_save_interval to '60'
```

このセーブ・インターバルを **'strict'** にリセットするには、次のように入力します。

```
configure logical connection to LDS.logical_pubs2
set materialization_save_interval to 'strict'
```

ウォーム・スタンバイ・データベースの複写定義とサブスクリプション

ウォーム・スタンバイ・アプリケーションの複写定義とサブスクリプションを作成できます。

Adaptive Server データベースしか含まれていない複写システムでは、複写定義の目的がプライマリ・キー、引用符付き識別子情報、またはカスタム・ファンクション文字列を指定することだけである場合、ウォーム・スタンバイ環境または Multi-Site Availability (MSA) 環境におけるテーブルの複写定義の必要性を軽減できます。『Replication Server 管理ガイド 第 1 巻』の「MSA を使用した複写オブジェクトの管理」で「複写定義およびサブスクリプションの使用の削減」を参照してください。

ただし、複写定義は論理データベースの各テーブルに対して作成できます。スタンバイ・データベースへの複写時には、ファンクション複写定義も使用できます。複写定義によって Replication Server がスタンバイ・データベースにデータを複写する方法を変更できるため、ウォーム・スタンバイ・アプリケーションを最適化したり、アプリケーションで必要なデフォルト以外の動作を有効にしたりできます。

ウォーム・スタンバイ・アプリケーションの複写定義は、論理データベースへの通常の複写または論理データベースからの通常の複写に使用できます。

参照：
- 複写を使用するウォーム・スタンバイ・アプリケーション (124 ページ)

ウォーム・スタンバイにおける alter table のサポート

Adaptive Server Enterprise バージョン 12.0 以降では、既存テーブルの変更 (null 入力不可能なカラムの追加、カラムの削除、カラム・データ型の変更) が可能です。

Oracle ウォーム・スタンバイ・アプリケーションの場合は、ユーザ定義のデータ型を有効にするために複写定義が必要となります。Replication Agent for Oracle は、初期化されるときに複写定義を自動的に作成します。このような状況では、新しい複写定義の作成または既存の複写定義の変更を手動で行って、どのユーザ定義のデータ型がスタンバイ・データベースに複写されているかを複写定義で明示的に指定する必要があります。

Replication Server は、テーブルにサブスクリプションがない場合の **alter table** コマンドによるテーブルの変更をサポートします。

注意： テーブルにサブスクリプションが存在する場合は、**alter table** によるテーブルの変更をサポートするために、テーブルの複写定義を変更する必要があります。その方法については、『Replication Server 管理ガイド 第 1 巻』の「複写テーブルの管理」で「複写定義の修正」を参照してください。

以前のリリースでは、テーブルに複写定義を定義した場合、Replication Server は常にウォーム・スタンバイの複写定義で定義したカラム・データ型を使用していました。Replication Server バージョン 12.0 以降では、Replication Server は、テーブルの複写定義を、状況に応じて使用する場合と使用しない場合があります。

複写定義がない場合

複写定義がないテーブルに対して **alter table** コマンドを使用すると、Replication Server はプライマリ・サーバから受信したのと同じ情報をウォーム・スタンバイ・データベースに送信します。

alter table のオプションはすべてサポートされています。プライマリで **alter table** を実行すると、このコマンドがウォーム・スタンバイに複写され、スタンバイへの複写が続行されます。Replication Server で必要な処理はありません。

構文と詳細については、『Adaptive Server Enterprise リファレンス・マニュアル：コマンド』の「コマンド」で「**alter table**」を参照してください。

alter table によるデフォルト値のカラムの追加

ここでは、アクティブ・データベースで **alter table** コマンドを発行して、デフォルト値を指定したカラムを追加するときの、DSI エラーを防ぐ方法について説明します。

アクティブ・データベースで alter table コマンドを発行し、デフォルト値を指定したカラムを追加すると、Adaptive Server が自動生成された名前の制約を作成します。このコマンドがスタンバイ・データベースに複写されると、スタンバイ・データベースも自動生成された別の名前の同じ制約を作成します。アクティブ・データベースで制約を削除すると、スタンバイ・データベースでは制約名が認識されず、DSI エラーが生成されます。

これを回避するため、制約は最初にアクティブ・データベースで削除します。DSI は自動的に停止します。次に、スタンバイ・データベースで作成された制約を削除し、**resume dsi skip transaction** コマンドを発行します。

別の回避策としては、次のコマンドを実行します。

```
alter table table name
replace column name
default null
```

これにより、アクティブ・サイトとスタンバイ・サイトの両方に作成された制約が自動的に削除されます。

send standby 句を使用しないウォーム・スタンバイ

どの複写定義にも **send standby** 句が関連付けられていない場合、Replication Server は複写定義を参照しないで、プライマリ・テーブルから受信したデータをすべて送信します。

Replication Server は元のカラム名とデータ型を使用し、Replication Agent から受信したデータを送信します。複写定義はプライマリ・キーを検索するためだけに使用されます。プライマリ・キーは、テーブルに対するすべての複写定義内のプライマリ・キーの結合です。

スキーマを変更しても、テーブルのすべての複写定義にあるすべてのプライマリ・キー・カラムが削除されることがない場合は、複写定義がない状況と同じ内容が当てはまります。**alter table** のオプションはすべてサポートされており、Replication Server で必要な処理はありません。

プライマリ・テーブルを変更する前であればいつでも、複写定義を変更して複写定義内のプライマリ・キーをすべて削除し、新しいプライマリ・キー・カラムをその中に追加できます。

古いプライマリ・キーは、古いデータ・ローをすべて複写システムから削除してから削除します。そうしないと、DSI が停止します。停止した場合は、リカバリの手順を参照してください。

参照：
- 複写定義がない場合 (132 ページ)

send standby all columns 句を使用したウォーム・スタンバイ

複写定義に **send standby all columns** が関連付けられている場合、Replication Server は元のカラム名とデータ型を使用して、Replication Agent から受信するデータをすべて送信します。複写定義はプライマリ・キーを検索するためだけに使用されます。

スキーマを変更しても、send standby all columns 句が指定された複写定義にあるすべてのプライマリ・キー・カラムが削除されることがない場合は、複写定義がない状況と同じ内容が当てはまります。**alter table** のオプションはすべてサポートされており、Replication Server で必要な処理はありません。

プライマリ・テーブルを変更する前であればいつでも、複写定義を変更して **send standby all columns** 句が指定された複写定義内のすべてのプライマリ・キーを削除し、新しいプライマリ・キー・カラムをその中に追加できます。

参照：
- 複写定義がない場合 (132 ページ)

send standby replication definition columns 句を使用したウォーム・スタンバイ

複写定義に **send standby replication definition columns** 句がある場合、スタンバイは、レプリケート・テーブル名とレプリケート・カラム名、およびテーブルの対応する複写定義に指定されているデータ型を継続して使用します。

複写定義のデータ型をスタンバイで使用する場合、常に **send standby replication definition columns** 句を使用して複写定義を作成してください。

複写定義を使用してパフォーマンスを最適化する

ウォーム・スタンバイ環境で複写システムのパフォーマンスを向上させるには、複写定義を使用します。

スタンバイ・データベースへの複写に複写定義を使用するよう指定した場合、次のようになります。

- ユーザは、スタンバイ・データベースへの複写に対して、Replication Server が複写定義の **replicate minimal columns** 設定を使用するかどうかを指定できます。

この設定は、更新によってすべてのカラムの値が置き換えられるか、変更された値を持つカラムの値のみが置き換えられるかを示します。

- ユーザは、Replication Server がテーブルのすべてのカラムをスタンバイ・データベースに複写するか、ストアド・プロシージャのすべてのパラメータをスタンバイ・データベースに複写するか、またはテーブルやファンクション複写定義にリストされているカラムまたはパラメータのみを複写するかを指定できます。

スタンバイ・データベースへの複写に対する複写定義の作成
スタンバイ・データベースへの複写のためにのみ複写定義を作成するには、**create replication definition** コマンドで **send standby** 句を使用します。

複写定義のプライマリ・キーと **replicate minimal columns** の設定が、スタンバイ・データベースへの複写に使用されます。

『Replication Server リファレンス・マニュアル』の「Replication Server コマンド」の「**create replication definition**」を参照してください。

プライマリ・キーの指定
テーブル複写定義の有無は、プライマリ・キー・カラムがデータベースの **where** 句でどのようにパックされるかを決定します。『Replication Server 管理ガイド 第 1 巻』の「MSA を使用した複写オブジェクトの管理」の「複写定義およびサブスクリプションの使用の削減」で、「プライマリ・キー・カラムと引用符付きのテーブル名またはカラム名」を参照してください。

最小カラムの更新
スタンバイ・データベースへの複写に対して複写定義を作成すると、複写システムのパフォーマンス最適化を実現するもう 1 つの手段である最小カラム設定を利用できます。

replicate minimal columns 句を使用すると、複写された **update** トランザクションと **delete** トランザクションに必要なカラムのみが含まれます。変更されていないカラムの値は、**update** コマンドから除外できます。不必要なカラムを除外することにより、複写システムを介して配信されるメッセージのサイズが小さくなり、Adaptive Server での処理を軽減できます。

スタンバイへの複写に複写定義を使用しない場合でも、これによってパフォーマンスの向上を実現できます。

テーブルに複写定義がない場合、またはテーブルに複写定義はあるがスタンバイ・データベースへの複写には使用されない場合、最少カラムの複写は自動的に実行されます。

スタンバイ・データベースに複写するカラムの指定
スタンバイ・データベースへの複写に対して複写定義を作成する場合、複写するカラムのセットを指定できます。

- テーブル内のすべてのカラムをスタンバイ・データベースに複写するには、**send standby** または **send standby all columns** を指定します。
- 複写定義のカラムだけをスタンバイ・データベースに複写するには、**send standby replication definition columns** を指定します。

このコマンドにおける **send standby** 句の使用の詳細については、『Replication Server リファレンス・マニュアル』の「Replication Server コマンド」の「**create replication definition**」を参照してください。

スタンバイ・データベースに複写するパラメータの指定
ファンクション複写定義を作成する場合、複写するパラメータのセットを指定できます。

- ストアド・プロシージャのすべてのパラメータをスタンバイ・データベースに複写するには、**send standby all parameters** を指定します (または **all parameters** 句を省略します)。
- 複写定義のパラメータのみをスタンバイ・データベースに複写するには、**send standby replication definition parameters** を指定します。

複写ストアド・プロシージャにファンクション複写定義がない場合、そのストアド・プロシージャを実行すると、Replication Server はそのパラメータをすべてアクティブ・データベースからスタンバイ・データベースに複写します。複写ストアド・プロシージャごとに、1 つのファンクション複写定義を作成できます。

create applied function replication definition および **create request function replication definition** コマンドでの **send standby** 句の使用の詳細については、『Replication Server リファレンス・マニュアル』の「Replication Server コマンド」を参照してください。

複写定義を使用して重複した更新をコピーする

重複した更新を複写する場合は、カラムに対して **send standby replication definition** パラメータ・オプションを指定した複写定義を作成します。

複写定義がない場合、Replication Server は重複した更新をウォーム・スタンバイに複写しません。つまり、更新が現在の値を同じ値に変更するだけで更新前イメージと更新後イメージが同一である場合、Replication Server は更新を複写しません。

重複した更新を複写する場合は、**send standby replication definition** パラメータ・オプションを使用します。

テーブルに対して複写定義を作成すると、**replicate minimal columns** オプションを指定して複写定義を作成した場合でも、Replication Server は常に重複した更新を送信します。

ウォーム・スタンバイ・アプリケーションでのサブスクリプションの使用

ウォーム・スタンバイ・アプリケーションではサブスクリプションを使用できます。

アクティブ・データベースからスタンバイ・データベースへの複写にサブスクリプションは使用されませんが、次のことは可能です。

- 論理プライマリ・データベースのデータに対してサブスクリプションを作成する。
- 他のデータベースのデータを論理レプリケート・データベースに複写するためにサブスクリプションを作成する。

create subscription コマンドと **define subscription** コマンドは、物理名ではなく、論理データベース名と論理データ・サーバ名を使用します。

サブスクリプションとサブスクリプション・マテリアライゼーションの詳細については、『Replication Server 管理ガイド 第 1 巻』の「サブスクリプションの管理」を参照してください。

参照：

- 複写を使用するウォーム・スタンバイ・アプリケーション (124 ページ)

サブスクリプションの使用に関する制限

ウォーム・スタンバイ・データベースから、またはウォーム・スタンバイ・データベースにデータを複写するサブスクリプションを作成する場合は、いくつかの制限が適用されます。

Replication Server は、ウォーム・スタンバイ・アプリケーションにおいて、すべての形式のサブスクリプション・マテリアライゼーションおよびマテリアライゼーション解除をサポートします。サブスクリプションの作成に適用される制限には、次のものがあります。

- データベースの論理コネクションがある場合、物理アクティブ・データベースまたは物理スタンバイ・データベースに対するサブスクリプションを作成できない。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に、または両方からサブスクリプション・データを複写するには、論理データベースに対してサブスクリプションを作成する必要がある。
- スタンバイ・データベースを複写システムに追加している間は、サブスクリプションを作成できない。スタンバイ・データベースが正常に初期化されるまで待機する必要がある。

- サブスクリプションの作成中は、スタンバイ・データベースを複写システムに追加できない。
- **switch active** コマンドの実行中は、新しいサブスクリプションを作成できない。

論理プライマリ・データベースのサブスクリプション・マテリアライゼーション

ここでは、論理プライマリ・データベースに関連したサブスクリプション・マテリアライゼーションの問題、および、サブスクリプション・マテリアライゼーション中に論理プライマリ・データベースに対して **switch active** コマンドを実行した場合に何が起こるかについて説明します。

サブスクリプション・マテリアライゼーション中に、データは、アクティブ・プライマリ・データベースから選択され、マテリアライゼーション・キューに移動します。

switch active コマンドを実行すると、アクティブ・データベースが変更されたことをレプリケート・サイトに通知するために、プライマリ Replication Server が RSSD 情報を複写します。マテリアライズするサブスクリプションを持つレプリケート Replication Server がこの情報を受信すると、マテリアライゼーション・キューが削除されます。新しいアクティブ・プライマリ・データベースでサブスクリプション・データを再度選択すると、新しいキューが作成されます。

注意： アクティブ・データベースが変更されたことをレプリケート Replication Server で検出できるように、プライマリ Replication Server の RSSD の Replication Agent が実行されている必要があります。

論理レプリケート・データベースに対するサブスクリプション・マテリアライゼーション

ここでは、論理レプリケート・データベースに関連したサブスクリプション・マテリアライゼーションの問題、および、サブスクリプション・マテリアライゼーション中に論理レプリケート・データベースに対して **switch active** コマンドを実行した場合に何が起こるかについて説明します。

アトミック・マテリアライゼーション

アトミック・マテリアライゼーションを使用すると、Replication Server はマテリアライゼーション・キューのセーブ・インターバルを **'strict'** に設定します。

トランザクションは、データがアクティブ・データベースに適用されてスタンバイ・データベースに複写されるまで、マテリアライゼーション・キューから削除されません。

マテリアライゼーション・キューが適用されると、Replication Server がアクティブ・レプリケート・データベースでマーカを実行します。このマーカによって、マテリアライゼーション・キューの適用後に実行するトランザクションの開始がマーク付けされます。

アクティブ・レプリケート・データベースでマーカが実行されると、Replication Server は次のような情報メッセージをログに書き込みます。

```
I. 95/10/03 18:00:15. REPLICATE RS: Created atomic subscription
publishers_sub for replication definition publishers_rep at active
replicate
for <LDS.pubs2>
```

マーカがスタンバイ・レプリケート・データベースで受信されると、Replication Server は次のような情報メッセージをログに書き込みます。

```
I. 95/10/03 18:00:15. REPLICATE RS: Created atomic subscription
publishers_sub for replication definition publishers_rep at standby
replicate for <LDS.pubs2>
```

これでマテリアライゼーションが完了し、Replication Server はマテリアライゼーション・キューを削除します。サブスクリプションは、アクティブ・レプリケート・データベースとスタンバイ・レプリケート・データベースの両方で VALID とみなされます。

マテリアライゼーション・キューの処理中に **switch active** コマンドを実行すると、Replication Server はマテリアライゼーション・キューを新しいアクティブ・データベースに再度適用します。**incrementally** オプションを使用してサブスクリプションを作成した場合、新しいアクティブ・データベースにまだ複写されていないマテリアライゼーション・ローのバッチのみが再実行されます。

ノンアトミック・マテリアライゼーション
ノンアトミック・マテリアライゼーションを使用すると、セーブ・インターバルは 0 に設定されるため、マテリアライゼーション・キューのローがアクティブ・データベースに適用されると、Replication Server がそのローを削除できます。

switch active コマンドの実行時にサブスクリプションがマテリアライズ中である場合、Replication Server はマテリアライゼーション・キューの処理を終了しますが、サブスクリプションに “suspect” とマーク付けします。アクティブ・データベースとレプリケート・データベースでサブスクリプション・ステータスを確認するには、**check subscription** コマンドを使用します。suspect とマーク付けされたサブスクリプションは、削除して再作成する必要があります。

注意： ノンアトミック・マテリアライゼーションは、異機種間複写ウォーム・スタンバイ・アプリケーションではサポートされていません。『Replication Server 異機種間複写ガイド』の「マテリアライゼーション」を参照してください。

バルク・マテリアライゼーション
バルク・マテリアライゼーションを使用し、データをウォーム・スタンバイ・アプリケーションに複写するサブスクリプションを作成する場合、サブスクリプション・データがアクティブ・レプリケート・データベースとスタンバイ・レプリケート・データベースに確実にロードされるようにする必要があります。

ログ付きの **bcp** など、挿入されたローのログを記録するメソッドでデータをロードすると、Replication Server はローをスタンバイ・データベースに複写します。ログを記録しないメソッドでデータをロードする場合は、アクティブ・データベースのログにはスタンバイ・データベースに複写するための挿入レコードがないため、そのデータをスタンバイ・データベースにもロードする必要があります。

バルク・マテリアライゼーション中は、サブスクリプション・データをレプリケート・データベースにロードする前に、**activate subscription with suspension** コマンドを実行します。デフォルトでは、**activate subscription with suspension** を実行すると、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に対して DSI スレッドがサスペンドされます。DSI スレッドがサスペンドされると、データを両方のデータベースにロードできるようになります。

ログ付きの **bcp** またはローのログを記録するその他のメソッドを使用してデータをロードする場合、**activate subscription with suspension at active replicate only** を実行し、Replication Server がアクティブ・データベースの DSI のみをサスペンドするようにします。これによって、挿入されたローをアクティブ・データベースからスタンバイ・データベースに複写できるようになります。

サブスクリプションの確認

論理レプリケート・データベース用のウォーム・スタンバイ・アプリケーションでは、**check subscription** コマンドを使用してサブスクリプション・ステータスをチェックできます。

ウォーム・スタンバイ・アプリケーションを管理する Replication Server は、アクティブ・データベースとスタンバイ・データベースのステータスが異なるかどうかによって、1 つまたは 2 つのステータス・メッセージを返します。

たとえば、サブスクリプションの作成中にマテリアライゼーション・ステータスが、アクティブ・データベースでは VALID、スタンバイ・データベースでは ACTIVATING の場合があります。

サブスクリプションの削除

論理レプリケート・データベースでは、**with purge** オプションを指定して **drop subscription** コマンドを使用すると、サブスクリプションを削除できます。

drop subscription マーカは、DSI キューからアクティブ・データベースへとマテリアライゼーション解除データを追跡し、スタンバイ・データベースに移動します。マーカが両方のデータベースで受信されると、サブスクリプション・データが両方のデータベースから削除されます。

switch active を実行している間

drop subscription コマンドに **with purge** オプションを指定してサブスクリプションを削除する間に、レプリケート Replication Server で **switch active** コマンドを実行できます。

Replication Server は DSI スレッドをサスペンドし、一時的にマテリアライゼーション解除をサスペンドします。**switch active** の完了後、DSI スレッドがレジュームされ、マテリアライゼーション解除が再開します。

疑わしい drop subscription

論理レプリケート・データベースに対して **with purge** オプションを使用してサブスクリプションを削除すると、次の状況が重なったときに、疑わしい (suspect) **drop subscription** が発生する可能性があります。

- サブスクリプションがアクティブ・データベースでマテリアライズされている。
- 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た。
- その後、新しいアクティブ・データベースでマテリアライズ中のサブスクリプションを削除する。

マテリアライゼーション解除は、新しいアクティブ・データベースで再開して正常に処理されますが、新しいスタンバイ (古いアクティブ) データベースに、パージされていないサブスクリプション・データが一部保持されている場合があります。この不整合を解決するには、**rs_subcmp** プログラムを使用してアクティブ・データベースとスタンバイ・データベースを一致させるか、スタンバイ・データベースを削除して再生成します。

たとえば、**drop subscription** を実行しようとすると、次のような警告メッセージが表示される場合があります。

```
W. 95/10/02 20:59:15. WARNING #28171 DSI(111 SYDNEY_DS.pubs2) - /
sub_dsi.c(1231)
 REPLICATE RS: Dropped subscription publishers_sub for replication
 definition publishers_rep at standby replicate for
<SYDNEY_DS.pubs2> before
 it completed materialization at the Active Replicate. Standby
replicate may
 have some subscription data rows left in the database
```

スタンバイ・データベースの作成時に消失するカラム

複写定義を持つ既存のデータベースに対してスタンバイ・データベースを作成する場合、特定の状況が重なると、カラムが消失する可能性があります。

次の状況が重なると、カラムが消失する可能性があります。

- 既存のデータベースに、テーブルの一部のカラムが含まれていない複写定義がある。
- インバウンド・キューに、コミットされていない挿入または更新トランザクションがある。
- 既存のデータベース (現在のアクティブ・データベース) に対してスタンバイ・データベースを作成する。
- その後、トランザクションがコミットする。

デフォルトでは、スタンバイ・データベースがすべてのカラムを受け取るものと想定されますが、トランザクションの開始時にスタンバイ・データベースが存在していません。この場合、Replication Server は複写定義にないカラムの値を廃棄しています。カラムが複写定義になく、スタンバイ・データベースでそのカラムに対して null 値が許可される場合は、値を失うことなく、ローをスタンバイ・データベースで挿入または更新できます。そうでない場合は、ユーザがデータベースを調整する必要があります。

ロス検出とリカバリ

ウォーム・スタンバイ・アプリケーションを作成すると、複写システムに新しいタイプのロス検出メッセージが追加されます。

ウォーム・スタンバイ・アプリケーションに関与する Replication Server でキューを再構築する場合、Replication Server によって次のいずれかのデータベース間でロスが検出されることがあります。

表 13 : ウォーム・スタンバイ・アプリケーションのロス検出

ロスの検出場所	目的
論理レプリケート・データベース	論理プライマリ・データベース
論理プライマリ・データベース	物理レプリケート・データベース
物理プライマリ・データベース	論理レプリケート・データベース
物理アクティブ・データベース	物理スタンバイ・データベース
論理プライマリ・データベース	Replication Server

ウォーム・スタンバイ・アプリケーションが関与するデータベースのリカバリ・オペレーションで **ignore loss** コマンドを使用するには、受信したロス検出メッセージに表示されているのと同じ論理または物理データ・サーバとデータベース名を使用します。

参照：
- 複写システム・リカバリ (381 ページ)

パフォーマンス・チューニング

使用する Replication Server システムの要件を満たすには、リソースを効率的に管理し、個々の Replication Server のパフォーマンスを最適化する必要があります。

Replication Server のパフォーマンスは、設定パラメータの値の変更、並列 DSI スレッドの使用、ディスク割り付けの選択によって影響を受けます。これらのリソースを正しく管理するには、Replication Server の内部処理についてある程度理解しておく必要があります。

Replication Server の内部処理

複写中は、Replication Server のいくつかの「*スレッド*」によってデータ・オペレーションが実行されます。

UNIX プラットフォームでは、これらは POSIX スレッドです。Windows プラットフォームでは、WIN32 スレッドです。また、Replication Server は、データをキューに保存し、重要なシステム情報を RSSD (Replication Server システム・データベース) から取得します。これらの内部オペレーションは、プライマリ Replication Server やレプリケート Replication Server のさまざまな処理をサポートしています。

スレッド、モジュール、およびデーモン

ここでは、Replication Server におけるスレッド、モジュール、およびデーモンの動作について説明します。

Replication Server は、複数のスレッドを並列実行します。スレッドの合計数は、Replication Server が管理するデータベースの数と、その Replication Server が直接ルートを持つ Replication Server の数によって異なります。各スレッドは、ユーザ・セッションの管理、RepAgent からのメッセージの受信、別の Replication Server からのメッセージの受信、データベースへのトランザクションの適用など、特定の機能を実行します。

スレッドの中には、Replication Server の特定の部分 (*モジュール*) を呼び出して、メッセージとトランザクションの送信先を判断し、複写する操作とその複写方法を決めるものがあります。

*デーモン・スレッド*は、バックグラウンドで実行され、事前に定義された時刻に、または特定のイベントに応じて、指定されたオペレーションを実行します。そし

て、サブスクリプション・マテリアライゼーションなどの Replication Server のアクティビティ中で実行されます。

複写システムをトラブルシューティングする場合、Replication Server のスレッド、モジュール、デーモンのステータスを確認します。

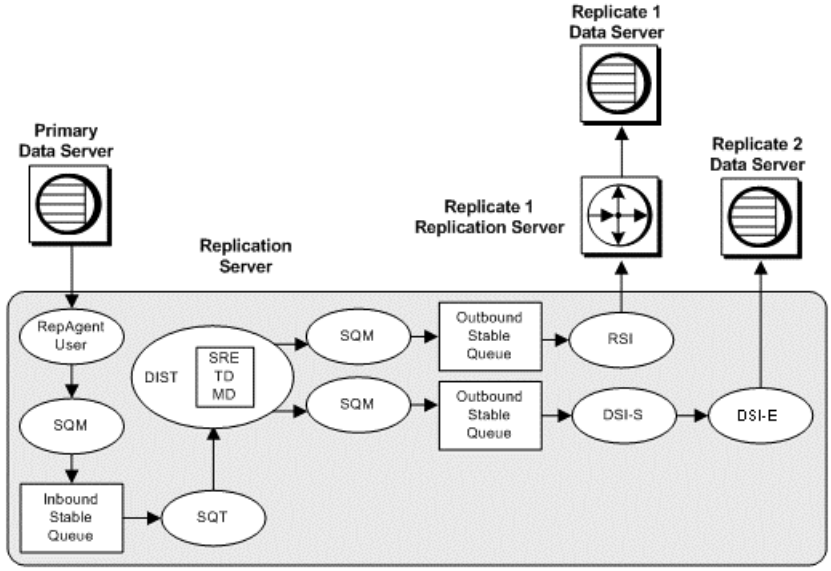
参照：

- プライマリ Replication Server での処理 (146 ページ)
- Replication Server の検証とモニタリング (5 ページ)

プライマリ Replication Server での処理

ここでは、プライマリ・データ・サーバで開始されるトランザクションが、どのようにプライマリ Replication Server に送信され、続いてレプリケート Replication Server に分配されるかについて説明します。

図 11 : プライマリ Replication Server での処理に使用されるスレッド



Replication Agent ユーザ・スレッド

ここでは、RepAgent をはじめとした複写エージェントが Replication Server と連動して、サブスクリプションを作成するレプリケート・データベースにトランザクション情報を分配する仕組みについて説明します。

RepAgent は、Open Client™ インタフェースを介して Replication Server にログインします。RepAgent はトランザクション・ログをスキャンし、ログ・レコードを

LTL (ログ変換言語) コマンドに直接変換し、このコマンドのログが取られるとすぐにこのコマンドを Replication Server にバッチでまたは一度に 1 つずつ送信します。次に、Replication Server がトランザクション情報を、サブスクリプションを作成するレプリケート・データベースに分配します。

Replication Server には、Replication Server が管理するプライマリ・データベースごとに 1 つの RepAgent ユーザ・スレッドがあります。そのため、Replication Server には、RepAgent ごとに 1 つの RepAgent ユーザ・スレッドがあります。RepAgent ユーザ・スレッドは、RepAgent からの送信が有効であることを確認して、送信されてきたデータをデータベースの受信ステーブル・キューに書き込みます。

ステーブル・キュー・マネージャ・スレッド

インバウンド・キュー、アウトバウンド・キューのいずれの場合でも、プライマリ Replication Server がアクセスするステーブル・キューに対して、それぞれ 1 つのステーブル・キュー・マネージャ (SQM) スレッドがあります。各 RepAgent ユーザ・スレッドは、専用の SQM スレッドとともに動作します。この SQM スレッドは、トランザクションがデータ・サーバまたは別の Replication Server に転送された後のステーブル・キュー領域を再利用します。

ステーブル・キュー・トランザクション・スレッド

ステーブル・キュー・トランザクション (SQT) スレッドは、トランザクションを再編成して、トランザクションをコミット順に配置します。

トランザクション・ログ・レコードやインバウンド・キューに格納されるコマンドは、必ずしもトランザクションごとにグループ化されるわけではありませんが、コミットされた順に並べられます。送信先のデータ・サーバへの最終的な適用、およびマテリアライゼーション処理では、トランザクションをコミット順に並べる必要があります。

SQT スレッドは、そのステーブル・インバウンド・キューからコマンドを読み取る順序でトランザクションを再編成し、トランザクションのリンク・リストを保持します。アウトバウンド・キューに対して、DSI/S スレッドは、トランザクションをスケジュールし、トランザクションの再編成および並び替えの SQT ファンクションを実行します。また、コミット・レコードを読み取ると、SQT によるトランザクションの並べ替えが必要な処理に応じて、ディストリビュータ (DIST) スレッドまたは DSI スレッドに対してトランザクションを使用可能にします。

SQT スレッドは、ロールバック・レコードを読み取ると、影響を受けたレコードをすべてのステーブル・キューから削除するよう SQM スレッドに指示します。また、DSI/S スレッドのオペレーションにより、SQT ライブラリは、トランザクションがラージ・トランザクション・スレッショルドを超えると、DSI に通知します。

参照：
- 並列 DSI スレッド (202 ページ)

ディストリビュータ・スレッドと関連モジュール

Replication Server が管理する各プライマリ・データベースには、ディストリビュータ (DIST) スレッドが 1 つずつあります。DIST スレッドは、SQT を使用してインバウンド・キューからトランザクションを読み取り、SQM スレッドを使用してトランザクションをアウトバウンド・キューに書き込みます。

このため、たとえば 3 つのプライマリ・データベースがある場合、3 つのインバウンド・キュー、3 つの DIST スレッド、3 つの SQT スレッドがあります。

注意： トランザクションの送信先がスタンバイ・データベースのみの場合は、DIST スレッドを無効にして SQT スレッドも無効にすることをおすすめします。SQM スレッドは存在し、キューへの書き込みを行います。

各トランザクション・ローの送信先を決定する場合、DIST スレッドは、サブスクリプション・レゾルーション・エンジン (SRE)、トランザクション・デリバリ (TD: Transaction Delivery)、メッセージ・デリバリ (MD: Message Delivery) の各モジュールを呼び出します。すべての DIST スレッドが、これらのモジュールを共有します。

Subscription Resolution Engine (サブスクリプション・レゾルーション・エンジン)

サブスクリプション・レゾリューション・エンジン (SRE) は、トランザクション・ローをサブスクリプションと照合します。

一致するものが見つかると、SRE は送信先データベース ID を各ローに付加します。SRE はサブスクリプションで必要なローのみをマーク付けするため、ネットワーク・トラフィックを最小限に抑えます。一致するサブスクリプションがない場合、DIST スレッドがロー・データを破棄します。

SRE は、各ローについて、サブスクリプション・マイグレーションが発生するかどうかを判断します。

- ローは、サブスクリプションと一致するようにローのカラム値が変更されると、サブスクリプションにマイグレート・インします。また、そのローはレプリケート・テーブルに追加される必要があります。
- ローは、サブスクリプションと一致しなくなるようにローのカラム値が変更されると、サブスクリプションからマイグレート・アウトします。また、そのローはレプリケート・テーブルから削除される必要があります。

SRE は、サブスクリプション・マイグレーションを検出すると、レプリケート・テーブルとプライマリ・テーブルとの間の一貫性を維持するために複写すべきオペレーション (挿入、削除、または更新) を決定します。

トランザクション・デリバリ・モジュール
トランザクション・デリバリ (TD) モジュールは、DIST スレッドによって呼び出され、トランザクション・ローをデータ・サーバやその他の Replication Server への分配用にまとめます。

メッセージ・デリバリ・モジュール
メッセージ・デリバリ (MD) モジュールは、DIST スレッドによって呼び出され、データ・サーバやその他の Replication Server へのトランザクションのルート指定を最適化します。

DIST スレッドは、トランザクション・ローと送信先 ID を MD モジュールに渡します。MD モジュールは、この情報や RSSD のルート指定情報を使用して、次のようにトランザクションの送信先を決定します。

- DSI スレッドを介してデータ・サーバに送信する
- RSI スレッドを介して Replication Server に送信する

MD モジュールは、トランザクションの送信方法を決定した後、トランザクションを適切なアウトバウンド・キューに配置します。

ディストリビュータ・ステータスの記録
Replication Server は、RSSD の Replication Server `rs_databases` システム・テーブルにディストリビュータ・スレッドの DIST ステータスを記録します。

ディストリビュータ (DIST) スレッドは、インバウンド・キューからトランザクションを読み取り、アウトバウンド・キューに複写トランザクションを書き込みます。DIST スレッドは、Replication Server がプライマリ・データベースに接続するときに作成され、手動で、または Replication Server 設定を使用して、サスペンドまたはレジュームできます。DIST スレッドをサスペンドまたはレジュームすると、スレッドの DIST ステータスが変更されます。

`rs_databases` の記録によって、DIST スレッドは Replication Server が停止した後でもそのステータスを保持できます。

『Replication Server リファレンス・マニュアル』の「Replication Server システム・テーブル」で「`rs_databases`」を参照してください。

データ・サーバ・インタフェース・スレッド

Replication Server は、データ・サーバ・インタフェース (DSI) スレッドを起動して、コネクションを維持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送信します。

各 DSI スレッドは、1 つのスケジューラ・スレッド (DSI-S) と 1 つ以上のエグゼキュータ・スレッド (DSI-E) で構成されます。各 DSI エグゼキュータ・スレッドは、データベースへの Open Client コネクションをオープンします。

Replication Server から、Replication Server が管理するレプリケート・データベースにトランザクションを送信するときのパフォーマンスを向上させるには、複数の DSI エグゼキュータ・スレッド (つまり、並列 DSI スレッド) を使用してトランザクションが適用されるように、データベース・コネクションを設定します。

DSI スケジューラ・スレッドは、SQT インタフェースを呼び出して次のことを実行します。

- 小さなトランザクションをコミット順にグループ化します。
- トランザクション・グループを次に使用可能な DSI エグゼキュータ・スレッドにディスパッチします。

DSI エグゼキュータ・スレッドは、次のことを実行します。

- データベース・コネクションに割り当てられたファンクション文字列クラスに従って、ファンクションに定義されたファンクション文字列を使用してファンクションをマップします。
- レプリケート・データベースのトランザクションを実行します。
- 割り当てられたエラー・アクションに従って、データ・サーバから返されるエラーに対してアクションを実行します。失敗したトランザクションがある場合は、例外ログに記録します。

DSI スレッドは、Replication Server によってサポートされるすべてのプライマリ・データベースのトランザクションの混合を適用する場合があります。これらのトランザクションは、レプリケート・データ・サーバの単一のアウトバウンド・ステーブル・キューから読み取られます。

参照：

- 並列 DSI スレッド (202 ページ)

Replication Server インタフェース・スレッド

RSI スレッドは、1 つの Replication Server から別の Replication Server にメッセージを送信する非同期インタフェースです。ソース・データベースが直接ルートを持つ送信先 Replication Server ごとに 1 つの RSI スレッドがあります。

プライマリ Replication Server の DIST スレッドがトランザクションを処理すると、他の Replication Server 宛てのトランザクションが RSI アウトバウンド・キューに書

き込まれます。RSI スレッドは、各レプリケート Replication Server にログインし、ステーブル・キューからレプリケート Replication Server にメッセージを転送します。

ある Replication Server から別の Replication Server に直接ルートが作成されると、送信元 Replication Server の RSI スレッドは、レプリケート Replication Server にログインします。間接ルートが作成される場合、Replication Server は、新しいステーブル・キューと RSI スレッドを作成しません。代わりに、間接ルートに対するメッセージは、直接ルートに対する RSI スレッドによって処理されます。『Replication Server 管理ガイド 第 1 巻』の「ルートの管理」を参照してください。

その他のデーモン・スレッド

Replication Server のデーモン・スレッドの中には、複写システムでその他のタスクを実行するものがいくつかあります。

表 14 : その他の Replication Server デーモン・スレッド

スレッド名またはデーモン名	説明
アラーム・デーモン (dALARM)	アラーム・デーモンは、コネクションのフェードアウト時間やサブスクリプション・リトライ・デーモンのインターバルなど、他のスレッドによって設定されたアラームを追跡する。
非同期 I/O デーモン (dAIO)	非同期 I/O デーモンは、Replication Server ステーブル・キューへの非同期 I/O を管理する。
コネクション・マネージャ・デーモン (dCM)	コネクション・マネージャ・デーモンは、データ・サーバや他の Replication Server へのコネクションを管理する。
リカバリ・デーモン (dREC)	リカバリ・デーモンは、ウォーム・スタンバイ・アプリケーション、ルート指定、リカバリ・プロシージャと関連する各種オペレーションを管理する。
サブスクリプション・リトライ・デーモン (dSUB)	サブスクリプション・リトライ・デーモンは、設定可能なタイムアウト時間 (`rs_config` システム・テーブルの **sub_daemon_sleep_time** 設定パラメータ) 経過後にウェイクアップし、失敗した可能性のあるサブスクリプションの処理のレジュームを試みる。
バージョン・デーモン (dVERSION)	バージョン・デーモンは、アップデード後 Replication Server が最初に起動されたときに一時的にアクティブになる。Replication Server の新しいバージョン番号を ID サーバに知らせる。

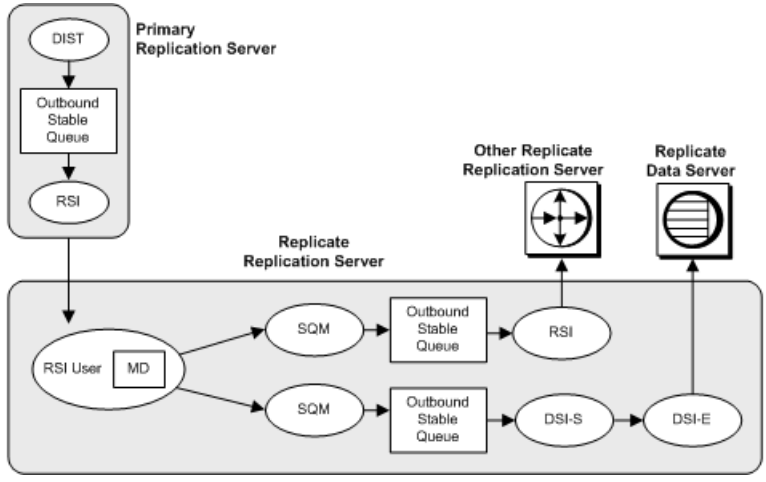
スレッド名またはデーモン名	説明
RS ユーザ・スレッド	RS ユーザ・スレッドは、サブスクリプションの作成または削除処理中のレプリケート Replication Server からのコネクションを管理する。 サブスクリプションの作成と削除に関するデータ・フローについては、『Replication Server 管理ガイド 第 1 巻』の「サブスクリプションの管理」で「サブスクリプション・マテリアライゼーション・メソッド」を参照してください。
USER スレッド	USER スレッドは、ユーザが Replication Server にログインしたときに、RCL コマンドを実行するために作成される。

レプリケート Replication Server での処理

ここでは、レプリケート Replication Server がプライマリ Replication Server から受信メッセージを受信するときに関連する処理について説明します。

この図は、プライマリ Replication Server での処理にも関与しているスレッドの例 (SQM、RSI、DSI) を示しています。

図 12 : レプリケート Replication Server でのトランザクション処理



参照：

- プライマリ Replication Server での処理 (146 ページ)

RSI ユーザ・スレッド

RSI ユーザ・スレッドは、他の Replication Server からの受信メッセージ用のクライアント・コネクション・スレッドです。

RSI ユーザ・スレッドは、メッセージ・デリバリ (MD) モジュールを呼び出して、メッセージを次のものに送信するかどうかを判別します。

- データ・サーバ (DSI スレッドを使用)。DSI スレッドは、1 つのスケジューラ・スレッド (DSI-S) と 1 つ以上のエグゼキュータ・スレッド (DSI-E) で構成される。
- 別の Replication Server (RSI スレッドを使用)。

RSI ユーザ・スレッドは、他の Replication Server やデータベースに送信されるコマンドをアウトバウンド・キューに書き込みます。プライマリ Replication Server での処理に関与しているスレッドは、メッセージがアウトバウンド・キューに格納された後でメッセージの処理を行います。

参照：

- データ・サーバ・インタフェース・スレッド (150 ページ)
- Replication Server インタフェース・スレッド (150 ページ)
- プライマリ Replication Server での処理 (146 ページ)

パフォーマンスに影響する設定パラメータ

Replication Server には、パフォーマンスを向上させるための設定パラメータが用意されています。パラメータには、サーバ全体に影響を与えるものと、個々のコネクションやルートを対象とするものがあります。

パフォーマンスに影響する Replication Server パラメータ

設定パラメータの値を変更すると、Replication Server のパフォーマンスを向上させることができます。

Replication Server のインストール後、**rs_init** によってデフォルトの設定パラメータが設定されます。

configure replication server を使用してこれらのパラメータを変更する方法については、『Replication Server 管理ガイド 第 1 巻』の「複写システムの管理」の「Replication Server 設定パラメータの設定」で「Replication Server パラメータの変更」を参照してください。

表 15 : パフォーマンスに影響する Replication Server パラメータ

設定パラメータ	説明
block_size to '*value*' **with shutdown**	ステーブル・キュー構造で使用される連続メモリ・ブロックのバイト数であるキュー・ブロック・サイズを指定する。 有効な値 16KB、32KB、64KB、128KB、または 256KB デフォルト値は 16KB **注意：** コマンドを実行してブロック・サイズを変更すると、Replication Server が停止します。Replication Server 15.6 より前のバージョンでブロック・サイズを指定した後は、**with shutdown** 句を含める必要があります。バージョン 15.6 以降では、**with shutdown** 句はオプションです。キュー・ブロック・サイズの変更を有効にするために Replication Server を再起動する必要はありません。このパラメータは、**configure replication server** コマンドのみを使用して変更してください。そうしないとキューが破損します。 ライセンス：Advanced Services オプションで個別にライセンス供与される。
db_packet_size	ネットワーク・パケットの最大サイズ。データベースと通信する場合、ネットワーク・パケットの値はそのデータベースで受け入れられる範囲内にする必要がある。Adaptive Server を再設定して使用する場合は、この値を変更できる。 最大値：16,384 バイト デフォルト値はすべての Adaptive Server データベースに対して、512 バイトのネットワーク・パケット
deferred_queue_size	Open Server の遅延キューの最大サイズ。Open Server の制限を超える場合は、最大サイズを増やす。この値は 0 よりも大きくすること。 **注意：** 変更した場合は、Replication Server を再起動して変更を有効にする必要がある。 デフォルト値は 2,048 (Linux および HPIA32 の場合)、1,024 (その他のプラットフォームの場合)
disk_affinity	次のパーティションを割り当てるための割り付けヒントを指定する。現在のパーティションが満杯になった場合に、次のセグメントの割り付け先となるパーティションの論理名を入力する。指定できる値は “*partition_name*” と “off”。 デフォルト値は off

設定パラメータ	説明
dist_direct_cache_read	ディストリビュータ (DIST) スレッドを有効にしてステーブル・キュー・スレッド (SQT) キャッシュから SQL 文を直接読み取る。これにより、SQT からの負荷、および SQT と DIST の間の依存性が減少し、SQT と DIST の両方の効率が向上する。 デフォルト値は off
dsi_bulk_copy	コネクションのバルク・コピー・イン機能を on または off にする。**dynamic_sql** と **dsi_bulk_copy** の両方を on にすると、DSI によってバルク・コピー・インが適用される。バルク・コピー・インが使用されない場合は、動的 SQL が使用される。Sybase では、大量のデータを挿入する必要がある場合にパフォーマンスを向上させるため、**dsi_bulk_copy** を on にすることをおすすめしている。 デフォルト値は off
dsi_bulk_threshold	トランザクション内の連続する **insert** コマンドの数。この数に到達すると、バルク・コピー・インを使用するように Replication Server をトリガする。ステーブル・キュー・トランザクション (SQT) は、大量の **insert** コマンドのバッチを検出すると、バルク・コピー・インを適用するかどうかを決定するために、指定された数の **insert** コマンドをメモリに保持する。これらのコマンドはメモリに保持されるため、この値を **dsi_large_xact_size** の設定値よりも大きい値に設定しないことをおすすめする。 Replication Server は、Sybase IQ への Real-Time Loading (RTL) Replication と Adaptive Server への High Volume Adaptive Replication (HVAR) に、**dsi_bulk_threshold** を使用する。1 つのテーブルに対する insert、delete、または update の各オペレーションに対するコマンドの数が、コンパイル後に指定する数よりも小さい場合、RTL と HVAR はバルク・インタフェースを使用せず、代わりに言語を使用する。 最小値：1 デフォルト値は 20
dsi_cmd_batch_size	Replication Server が、コマンド・バッチ内に配置する最大バイト数。 デフォルト値は 8,192 バイト

設定パラメータ	説明
dsi_cmd_prefetch	データ・サーバからの応答の待機中に、DSI がコマンドの次のバッチを構築することを許す。このため、DSI の効率が向上する。データ・サーバのパフォーマンスを高めるように調整する場合、この機能を使用するとパフォーマンスがさらに向上する可能性がある。 デフォルト値は off **dsi_compile_enable** を 'on' に設定すると、**dsi_cmd_prefetch** に設定した値は無視される。 ライセンス：Advanced Services オプションで個別にライセンス供与される。
dsi_max_xacts_in_group	グループ化できるトランザクションの最大数を指定する。大きい値を指定するほど、レプリケート・データベースでのデータ遅延時間が短縮される。値の範囲：1 – 1000 デフォルト値は 20 **dsi_compile_enable** が on の場合、このパラメータは無視される。
dsi_non_blocking_commit	コミット後に Replication Server がメッセージを保存している期間を延長する長さ (分単位) を指定する。値の範囲：0 ～ 60 分 デフォルト値は 0 – 非ブロッキング・コミットは無効。 Adaptive Server 15.0 以降の **delayed_commit** オプションを使用する場合、または Oracle 10g v2 以降で同等の機能を使用する場合には、このパラメータを有効にすることで複写パフォーマンスが向上する。
dsi_xact_group_size	1 つにグループ化されたトランザクションに配置する最大バイト数 (ステーブル・キュー・オーバヘッドを含む)。グループ化されたトランザクションとは、DSI が単一のトランザクションとして適用するトランザクション・セットのことである。–1 はグループ化が行われないことを意味する。 Sybase では、**dsi_xact_group_size** を最大値に設定し、**dsi_max_xacts_in_group** でグループ内のトランザクション数を制御することをおすすめしている。 **注意：** このパラメータは、Replication Server バージョン 15.0 以降では使用されなくなっていますが、旧バージョンの Replication Server との互換性を保つために保持されています。 最大値：2,147,483,647 デフォルト値は 65,536 バイト **dsi_compile_enable** が on の場合、このパラメータは無視される。

設定パラメータ	説明
dynamic_sql	動的 SQL 機能を有効または無効にする。このパラメータが on の場合にのみ、動的 SQL 関連の他の設定パラメータが有効になる。 デフォルト値は off
dynamic_sql_cache_size	コネクションの動的 SQL 文を使用できるデータベース・オブジェクトの数を Replication Server に指定する。 デフォルト値は 100 最小値：1 最大値：65,536
dynamic_sql_cache_management	DSI エグゼキュータ・スレッドの動的 SQL キャッシュを管理する。値：**mru** - **dynamic_sql_cache_size** に達すると、最後に使用された文を保持し、それ以外の文の割り付けを解除して新しい動的文を割り付ける。**fixed** (デフォルト) - **dynamic_sql_cache_size** に達すると、Replication Server は新しい動的文の割り付けを停止する。
exec_cmds_per_timeslice	**exec_cmds_per_timeslice** を使用して、LTI または RepAgent エグゼキュータ・スレッドが CPU を解放する前に実行できる LTL コマンドの数を指定することによって、RepAgent エグゼキュータが処理できるコマンドの数を制御する。この値を大きくすると、RepAgent エグゼキュータ・スレッドが CPU リソースをより長時間制御でき、RepAgent から Replication Server へのスループットが向上する。 このパラメータは、**alter connection** を使用してコネクション・レベルで設定する。 デフォルト値は 2,147,483,647 最小値：1 最大値：2,147,483,647
exec_nrm_request_limit	正規化待機中のプライマリ・データベースからのメッセージ用に使用可能なメモリ量を指定する。 **configure replication server** で **nrm_thread** を 'on' に設定してから、**exec_nrm_request_limit** を使用する。 デフォルト値は 1,048,576 バイト (1MB) 最小値：16,384 バイト (16KB) 最大値：2,147,483,647 バイト (2GB) ライセンス：Advanced Services オプションで個別にライセンス供与される。

設定パラメータ	説明
exec_sqm_write_request_limit	インバウンド・キューへの書き込み待ちメッセージ用に使用可能なメモリ量を指定する。 デフォルト値は 1MB 最小値：16KB 最大値：2GB
init_sqm_write_delay	満杯になっていないメッセージ・ブロックをキューに書き込む前に、SQM ライタがさらに多くのメッセージを待機する時間の初期値。SQM ライタは、常に満杯のブロックをキューに書き込もうとする。満杯になっていないブロックを満杯にできない場合、SQM ライタは **init_sqm_write_delay** に指定された時間だけ待機した後、メッセージがブロックに追加されるのを待っているかどうかを再度チェックする。メッセージがない場合、SQM ライタは **init_sqm_write_delay** の時間を倍にする。SQM ライタは、**init_sqm_write_max_delay** の値に達するまで遅延時間を倍にしていく。この値に達すると、SQM ライタは満杯になっていないブロックを書き込む。 デフォルト値は 100 ミリ秒
init_sqm_write_max_delay	満杯になっていないメッセージ・ブロックをキューに書き込む前に、SQM ライタ・スレッドがさらに多くのメッセージを待機する時間の最大値。詳細については、**init_sqm_write_delay** の説明を参照。 デフォルト値は 1,000 ミリ秒
mem_reduce_malloc	大きな単位でメモリを割り付けできるようにすることで、メモリ割り付け回数を削減し、Replication Server のパフォーマンスを向上させる。 デフォルト値は off ライセンス：Advanced Services オプションで個別にライセンス供与される。
mem_thr_dsi	DSI スレッドによる SQT キャッシュの入力を停止する合計メモリのパーセンテージを指定する。 デフォルト値は **memory_limit** 値の 80%。 範囲：1 – 100
mem_thr_exec	EXEC スレッドによる RepAgent からのコマンドの受信を停止する合計メモリのパーセンテージを指定する。 デフォルト値は **memory_limit** 値の 90%。 範囲：1 – 100

設定パラメータ	説明
mem_thr_sqt	SQT スレッドでキャッシュからの最大トランザクションをフラッシュする合計メモリのパーセンテージを指定する。 デフォルト値は **memory_limit** 値の 85%。 範囲：1 – 100
mem_warning_thr1	この値を超えると最初の警告メッセージが生成される、合計メモリのスレッショルド・パーセンテージを指定する。「**memory_limit**」を参照。 デフォルト値は **memory_limit** 値の 80%。 範囲：1 – 100
mem_warning_thr2	合計メモリのスレッショルド・パーセンテージを指定して、この値を超えると 2 番目の警告メッセージが生成されるようにする。「**memory_limit**」を参照。 デフォルト値は **memory_limit** 値の 90%。 範囲：1 – 100
memory_control	メモリを大量に必要とするスレッドのメモリ制御動作を管理する。「**memory_limit**」を参照。 指定できる値は、次のとおり。 • on – メモリ制御を有効化する • off – メモリ制御を無効化する デフォルト値は on

設定パラメータ	説明
memory_limit	Replication Server が使用できる合計メモリの最大値 (メガバイト単位)。 その他のいくつかの設定パラメータの値は、**memory_limit** によって示された、メモリ・プールから使用可能なメモリ量に直接関連する。これらの設定パラメータには、**exec_nrm_request_limit**、**exec_sqm_write_request_limit**、**md_sqm_write_request_limit**、**queue_dump_buffer_size**、**sqt_max_cache_size**、**sre_reserve**、**sts_cachesize** がある。 デフォルト値は 2,047 32 ビット版 Replication Server の場合： • 最小値 – 0 • 最大値 – 2,047 64 ビット版 Replication Server の場合： • 最小値 – 0 • 最大値 – 2,147,483,647 **memory_control** の状態： • on – メモリ消費が **memory_limit** を超過しても Replication Server は停止しない • off – メモリ消費が **memory_limit** を超過すると Replication Server は自動的に停止する メモリ使用量を監視し、必要に応じて **memory_limit** を増加させる。 Replication Server でメモリを大量に必要とするスレッドは、次のとおり。 • EXEC • SQT • DST これらのスレッドは、メモリ使用量チェックを実行してから新しいデータを受信または処理することで、メモリ制御を実行する。メモリ制御時にメモリ使用量が多いことが判明すると、次の動作によりスレッド機能が調整される。 • スレッドによる新しいデータのグループ化を停止し、既存データのクリーニングと処理を行う。または、 • 空きメモリが確保されるまで新しいデータを受信しないよう、スレッドをスリープ・モードにする。

設定パラメータ	説明
	2,047 より大きい値に設定されていた場合は、ダウングレードすると、オーバフローから保護するために 2,047 にリセットされる。
md_sqm_write_request_limit	アウトバウンド・キューへの書き込み待ちメッセージ用にディストリビュータが使用可能なメモリ量を指定する。 **注意：** Replication Server 12.1 の場合、**md_source_memory_pool** は **md_sqm_write_request_limit** で置換されます。**md_source_memory_pool** は、旧バージョンの Replication Server との互換性を保つために保持されています。 デフォルト値は 1MB 最小値：16K 最大値：2GB
nrm_thread	Replication Server が、ログ転送言語 (LTL: Log Transfer Language) コマンドを正規化してパックし、それと並行して RepAgent エグゼキュータ・スレッドによる解析を行うことができる、NRM スレッドを有効化する。NRM スレッドとの並列処理によって RepAgent エグゼキュータ・スレッドは応答時間を短縮する。NRM スレッドは RepAgent エグゼキュータ・スレッドから分離したスレッドである。 **configure replication server** コマンドを使用して **nrm_thread** を on に設定してから、**exec_nrm_request_limit** を使用する。 デフォルト値は off ライセンス：Advanced Services オプションで個別にライセンス供与される。
rec_daemon_sleep_time	リカバリ・デーモンのスリープ時間を指定することによって、ウェイクアップ・インターバルを設定する。このデーモンは、ウォーム・スタンバイ・アプリケーションや他のいくつかのオペレーションで “strict” セーブ・インターバル・メッセージを処理する。 デフォルト値は 2 分

設定パラメータ	説明
smp_enable	対称型マルチプロセッシング (SMP) を有効にする。Replication Server スレッドが、Replication Server によって内部的にスケジュールされるか、オペレーティング・システムによって外部的にスケジュールされるかを指定する。Replication Server スレッドが内部的にスケジュールされる場合、使用可能なマシン・プロセッサの数にかかわらず、Replication Server で使用できるプロセッサは 1 つに制限される。値は "on" または "off"。 デフォルト値は on アップグレードまたはダウングレードでは、設定されていた値は変更されない。
sqm_async_seg_delete	セグメントを削除する専用デーモンを有効にし、インバウンドとアウトバウンドのキュー処理のパフォーマンスを向上させるには、**sqm_async_seg_delete** を on に設定する。 デフォルト値は on このパラメータ設定の変更を有効にするには、Replication Server を再起動する必要がある。 **sqm_async_seg_delete** が on の場合、Replication Server に大規模なパーティションが必要になる可能性がある。パーティションを拡大するには、**alter partition** を使用する。 • 『Replication Server 設定ガイド』の「Replication Server のインストールと設定の準備」の「複写システムのプラン作成」の「各 Replication Server の最初のディスク・パーティション」 • 『Replication Server 管理ガイド 第 1 巻』の「Replication Server の技術的概要」の「Replication Server でのトランザクション処理」の「ステーブル・キュー」の「ステーブル・キューのパーティション」 を参照。
sqm_cache_enable	SQM キャッシュと大容量 I/O がステーブル・デバイスで有効になっているかどうかを示す。 デフォルト値は on
sqm_cache_size	キャッシュにあるページの数を示す。ページのサイズは **sqm_page_size** によって指定される。範囲は 1 〜 4096。 デフォルト値は 16

設定パラメータ	説明
sqm_page_size	ページ内のブロックの数を示す。 サーバワイドなステーブル・キューのページ・サイズをページあたりのブロック単位で設定する。ページ・サイズを一重引用符または二重引用符で囲む。たとえば、ページ・サイズを4に設定すると、64K チャンクでステーブル・キューに書き込むように Replication Server に指示される。 ページ・サイズを設定すると、Replication Server の I/O サイズも設定される。値の範囲は、1 〜 64。 デフォルト値は 4
sqm_recover_segs	Replication Server が RSSD をリカバリ QID 情報で更新する前に割り付けるステーブル・キュー・セグメント数を指定する。 Sybase では、**sqm_recover_segs** の値を増加して、パフォーマンスを向上させることをおすすめしている。 デフォルト値は 1 最小値：1 最大値：2,147,483,648
sqm_write_flush	ステーブル・デバイスを考慮する場合、**sqm_write_flush** は、書き込みオペレーションが完了する前にメモリ・バッファへの書き込みをディスクにフラッシュするかどうかを指定する。値は "on"、"off"、または "dio"。 デフォルト値は on
sqt_init_read_delay	SQT スレッドが SQM 読み込みを待機し、コマンド・キュー内の新しい命令のチェックを開始するまでスリープする時間の長さ。それぞれのスリープ時間の終了時にコマンド・キューが空の場合、SQT はスリープ時間を **sqt_max_read_delay** に設定されている値を超える直前まで、繰り返し倍の値を設定していく。 デフォルト値は 1 ミリ秒 最小値：0 ミリ秒 最大値：86,400,000 ミリ秒 (24 時間)

設定パラメータ	説明
sqt_max_cache_size	SQT キャッシュのサイズを SQT の最大キャッシュ・メモリのバイト数に設定するには、**sqt_max_cache_size** を使用する。 32 ビット版 Replication Server の場合： • デフォルト – 1,048,576 • 最小値 – 0 • 最大値 – 2,147,483,647 64 ビット版 Replication Server の場合： • デフォルト – 20,971,520 • 最小値 – 0 • 最大値 – 2,251,799,813,685,247 2,147,483,647 バイトより大きい値に設定されていた場合は、ダウングレードすると、オーバフローから保護するために 2,147,483,647 バイトにリセットされる。
sqt_max_prs_size	HVAR および RTL のトランザクション。プロファイリング処理によってアンパックされたコマンドが使用する最大メモリ (バイト)。 32 ビット版 Replication Server の場合： • デフォルト – 2,147,483,647 (2GB) • 最小値 – 0 • 最大値 – 2,147,483,647 64 ビット版 Replication Server の場合： • デフォルト – 2,147,483,647 (2GB) • 最小値 – 0 • 最大値 – 2,251,799,813,685,247 **configure replication server** を使用して、すべてのコネクションのサーバ・レベルのパラメータを設定するか、**alter connection** を使用して、個々のコネクションのデータベース・レベルを設定する。データベース・レベルのデフォルト値は 0 である。データベースレベルのデフォルトを保持するか、デフォルトにリセットすると、Replication Server はサーバ・レベルで設定された値を使用する。 **注意：** Replication Server 15.7.1 以降にアップグレードした場合、32 ビットと 64 ビットの Replication Server の両方でデフォルトを 2GB に設定する必要があります。

設定パラメータ	説明
sqt_max_read_delay	SQT スレッドがコマンド・キューに新しい命令があるかどうかをチェックするまで SQM 読み込みを待機している間、SQT スレッドがスリープする最長時間。 デフォルト値は 1 ミリ秒 最小値：0 ミリ秒 最大値：86,400,000 ミリ秒 (24 時間)
sts_cachesize	システム・テーブルをキャッシュする場合は、**sts_cachesize** を使用して、キャッシュされた RSSD システム・テーブルごとにキャッシュされるローの合計数を指定する。この値をアクティブな複写定義の数まで増やすと、Replication Server では負荷の高いテーブル検索が実行されなくなる。 カウンタ 11008 - STSCacheExceed を確認するか、Replication Server ログでローが STS キャッシュから削除されたことを示す警告を確認し、STS キャッシュが小さすぎるかどうかをモニタする。 デフォルト値は 1000
sts_full_cache_*system_table_name*	システム・テーブルをキャッシュする場合は、**sts_full_cache**_*system_table_name* を使用して、完全にキャッシュする RSSD システム・テーブルを指定する。完全にキャッシュされたテーブルでは、簡単な **select** 文に対して RSSD へのアクセスは不要になる。一部の RSSD テーブルのみを完全にキャッシュできる。 デフォルト値は `rs_asyncfuncs`、`rs_clsfunctionsrs_columns`、`rs_objects`、`rs_objfunctions`、`rs_repobjs`、および `rs_users` は完全にキャッシュされる。Sybase では、これらのテーブルをキャッシュしてパフォーマンスを向上させることをおすすめしている。
sub_daemon_sleep_time	ウェイクアップ・インターバルを設定する場合は、**sub_daemon_sleep_time** を使用して、サブスクリプション・デーモンが、ウェイクアップしてサブスクリプションをリカバリするまでにスリープする時間 (秒単位) を設定する。値の範囲は、1 ～ 31,536,000。 デフォルト値は 120 秒

設定パラメータ	説明
sub_sqm_write_request_limit	アウトバウンド・キューへの書き込み待ちメッセージ用に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解除スレッドが使用可能なメモリ量を指定する。 デフォルト値は 1MB 最小値：16K 最大値：2GB

参照：
- キュー・ブロック・サイズの増加 (288 ページ)
- Advanced Services Option (266 ページ)
- SQM ライタの待機時間の設定 (180 ページ)
- ウェイクアップ・インターバルの設定 (193 ページ)
- ステーブル・デバイス：考慮事項 (166 ページ)
- SQT キャッシュのサイズ設定 (194 ページ)
- システム・テーブルのキャッシュ (180 ページ)
- RepAgent エグゼキュータが処理できるコマンド数の制御 (196 ページ)
- SMP の効率的な使用 (197 ページ)
- 割り付けられるステーブル・キュー・セグメント数の指定 (197 ページ)

ステーブル・デバイス：考慮事項

アプリケーションと同様、Replication Server にも標準 I/O と I/O デバイスの推奨事項が当てはまります。ステーブル・デバイスを使用してステーブル・キューをサポートする方法について計画を立てる場合、ディスクの読み込み／書き込みヘッドや I/O チャネルの競合の影響を考慮してください。

各キューに専用のデバイスを 1 つ以上用意できれば、I/O がパフォーマンスの問題を招くことはほとんどありません。キュー専用のデバイスを用意することは、プライマリ・データベースやレプリケート・データベース、または RSSD などのその他のプロセスによってデバイスが使用されることを防ぐことにもなります。データベース・コネクション・パラメータ **disk_affinity** を使用すると、専用デバイスがサポートする特定のパーティションとキューとの間の関係を確立できます。

UNIX オペレーティング・システム・ファイル上で初期化されたステーブル・キューに関して、書き込みオペレーションの完了前にメモリ・バッファへの書き込みがディスクにフラッシュされるかどうかは、**sqm_write_flush** 設定パラメータによって制御されます。

sqm_write_flush が on の場合、Replication Server は O_DSYNC フラグを使用してステーブル・キューをオープンします。このフラグによって、書き込みオペレー

ションの完了前に、書き込みがメモリ・バッファからディスクにフラッシュされます。データは物理的なメディアに格納されるため、システム障害が発生してもデータを常に復元できます。デフォルトの設定です。

sqm_write_flush が off の場合、書き込みは UNIX ファイル・システムにバッファされます。その後の書き込みが失敗した場合、自動リカバリは保証されません。テストした結果では、パーティション・タイプと I/O フラッシュのさまざまなオプションの書き込み速度を比較した場合、バッファされたファイル・システムに **sqm_write_flush** を on にして書き込むと、ロー・パーティションに書き込む場合よりも処理速度が最大 5 倍遅くなりました。

さらに、ロー・パーティションに書き込むと、バッファされたファイル・システムに **sqm_write_flush** を off にして書き込む場合よりも処理速度が最大 7 倍遅くなりました。ステーブル・デバイスに対して UNIX のバッファされたファイル・システムを使用するときに **sqm_write_flush** を off にすると、I/O のパフォーマンスが最高になりますが、データ・ロスが発生する可能性が高くなります。プライマリ・データベースのトランザクション・ログのバックアップを保存しておくと、そのリスクを減らしたり、取り除いたりすることができます。

ファイル・システムのパーティションでは、同期 I/O である DSYNC と比較した場合、ダイレクト I/O によって I/O の遅延時間が減少します。ダイレクト I/O を設定するには、次のコマンドを実行します。

```
configure replication server set sqm_write_flush to
"dio"
```

このコマンドを実行すると、ダイレクト I/O が有効になります。ただし、このコマンドは、ステーブル・キューがファイル・システム上に存在する場合にのみ有効です。ダイレクト I/O メソッドにより、Replication Server は、ファイル・システムをバッファリングせずに直接ディスクに対して読み書きできます。ステーブル・キュー・キャッシュを適切に調整する必要があります。適切なキャッシュ・サイズに調整すると、ほとんどの読み込みトランザクションがキャッシュ内で完了します。

注意： ダイレクト I/O は、Solaris プラットフォームと Linux プラットフォームの Replication Server 15.1 以降でのみサポートされます。

このコマンドは静的であるため、コマンドを有効にするにはサーバを再起動する必要があります。

注意： ロー・パーティションまたは Windows ファイル上で初期化されたステーブル・キューでは、**sqm_write_flush** の設定は無視されます。このような場合、常にメディアに対して直接書き込みオペレーションが行われます。

I/O のパフォーマンスを向上させるため、Replication Server 15.1 以降ではステーブル・デバイスのキャッシュがサポートされています。

参照：
- ステーブル・キューのキャッシュ (184 ページ)

パフォーマンスに影響するコネクション・パラメータ

Replication Server には、パフォーマンスに影響する、いくつかのデータベース・コネクション・パラメータが用意されています。

コネクション・パラメータの完全なリストについては、『Replication Server 管理ガイド 第 1 巻』の「データベース・コネクションの管理」を参照してください。

表 16 : パフォーマンスに影響するコネクション・パラメータ

設定パラメータ	説明
async_parser	Replication Server が RepAgent から非同期にコマンドを解析できるようにする。 **async_parser** を on に設定すると、次のように設定される。 • 非同期パーサが on – **exec_prs_num_threads** が 2 • ASCII パッキングが on – **ascii_pack_ibq** が on • インバウンド・コマンドの直接レプリケーションが on – **cmd_direct_replicate** が on • アウトバウンド・コマンドの直接レプリケーションが on – **dist_cmd_direct_replicate** が on デフォルト値は off **注意：** 非同期パーサを設定する前に、**smp_enable** が on であり、Replication Server のホスト・マシンが解析用の追加スレッドをサポートできることを確認してください。**ascii_pack_ibq** を on に設定する前に、Replication Server のサイト・バージョンを 1571 以降に設定する必要があります。サイトのバージョンが 1571 だと、**async_parser** を on に設定しても、**exec_prs_num_threads**、**cmd_direct_replicate**、および **dist_cmd_direct_replicate** だけが設定されます。
ascii_pack_ibq	ASCII パッキングを使用して、インバウンド・キューにパックされたコマンドが消費するステーブル・キューの記憶領域を低減する。 デフォルト値は off **注意：** インバウンド・キューで ASCII パッキングの恩恵を受けるには、Replication Server で非同期パーサを有効にする必要があります。**ascii_pack_ibq** を on に設定する前に、Replication Server のサイト・バージョンを 1571 以降に設定する必要があります。

設定パラメータ	説明
batch	デフォルトの “on” の場合は、レプリケート・データベースに対してコマンド・バッチを使用できる。 デフォルト値は on
cmd_direct_replicate	エクゼキュータ・スレッドの **cmd_direct_replicate** を on に設定して、解析データをバイナリまたは ascii データと一緒にディストリビュータ・スレッドに直接送信する。ディストリビュータ・スレッドは、必要に応じて解析済みデータからデータを直接取得して処理できるので、バイナリ・データの解析に費やされる時間を節約してレプリケーション・パフォーマンスを向上させることができる。 デフォルト値は off
dist_cmd_direct_replicate	**dist_cmd_direct_replicate** を on に設定すると、DIST モジュールで内部解析データをメモリ内キャッシュから DSI に送信することができる。 デフォルト値は on **dist_cmd_direct_replicate** を off に設定すると、DIST モジュールはアウトバウンド・キューからデータを DSI に送信する。
db_packet_size	ネットワーク・パケットの最大サイズ。データベースと通信する場合、ネットワーク・パケットの値はそのデータベースで受け入れられる範囲内にする必要がある。 最大値：16,384 バイト デフォルト値はすべての Adaptive Server データベースに対して、512 バイトのネットワーク・パケット
disk_affinity	次のパーティションを割り当てるための割り付けヒントを指定する。現在のパーティションが満杯になった場合に、次のセグメントの割り付け先となるパーティションの論理名を入力する。指定できる値は “*partition_name*” と “off”。 デフォルト値は off

設定パラメータ	説明
dist_sqt_max_cache_size	インバウンド・キューの最大ステーブル・キュー・トランザクション (SQT) キャッシュ・サイズ (バイト単位)。デフォルトの 0 では、**sqt_max_cache_size** パラメータの現在の設定値が、コネクションの最大キャッシュ・サイズとして使用される。 デフォルト値は 0 32 ビット版 Replication Server の場合： • 最小値 – 0 • 最大値 – 2,147,483,647 64 ビット版 Replication Server の場合： • 最小値 – 0 • 最大値 – 2,251,799,813,685,247
dsi_cdb_max_size	Replication Server が HVAR または RTL 用に生成できる最終的な変更を保管するデータベースの最大サイズ (メガバイト単位)。 • デフォルト – 1024 • 最小値 – 0 • 最大値 – 2,147,483,647 HVAR では、Replication Server は **dsi_cdb_max_size** をスレッショルドに使用して次のことを行う。 • 連続レプリケーション・モードを使用して複写された大規模トランザクションを検出する。 • **dsi_cdb_max_size** よりも大きく、データベースの最終的な変更を必要とするグループに、小規模なコンパイル可能なトランザクションを分けることを停止する。 RTL では、Replication Server は **dsi_cdb_max_size** を使用して、フル・インクリメンタル・コンパイルによって大規模なトランザクション・グループをフラッシュする。
dsi_cmd_batch_size	Replication Server が、コマンド・バッチ内に配置する最大バイト数。 デフォルト値は 8,192 バイト

設定パラメータ	説明
dsi_cmd_prefetch	データ・サーバからの応答の待機中に、DSI がコマンドの次のバッチを事前構築することを許す。このため、DSI の効率が向上する。データ・サーバのパフォーマンスを高めるように調整する場合、この機能を使用するとパフォーマンスがさらに向上する可能性がある。 デフォルト値は off **dsi_compile_enable** を 'on' に設定すると、**dsi_cmd_prefetch** に設定した値は無視される。 ライセンス：Advanced Services オプションで個別にライセンス供与される。
dsi_commit_check_locks_intrvl	DSI エグゼキュータ・スレッドが `rs_dsi_check_thread_lock` ファンクション文字列の実行と実行の間に待機するミリ秒 (ms) 数。並列 DSI とともに使用される。 デフォルト値は 1,000 ミリ秒 (1 秒) 最小値：0 最大値：86,400,000 ミリ秒 (24 時間)
dsi_commit_check_locks_max	トランザクションのロールバックとリトライが実行されるまでに、DSI エグゼキュータ・スレッドが `rs_dsi_check_thread_lock` ファンクション文字列を実行する最大回数。並列 DSI とともに使用される。 デフォルト値は 400 最小値：1 最大値：1,000,000
dsi_commit_control	コミット制御について、Replication Server が内部テーブルを使用して内部的に処理するか (on)、`rs_threads` システム・テーブルを使用して外部的に処理するか (off) を指定する。並列 DSI とともに使用される。 デフォルト値は on

設定パラメータ	説明
dsi_isolation_level	トランザクションの独立性レベルを指定する。ANSI 標準および Adaptive Server でサポートされている値は、次のとおり。 • 0 ― 個々のトランザクションで書き込まれたデータが実際のデータであることを保証する。 • 1 ― ダーティ・リードを防止し、個々のトランザクションで書き込まれたデータが実際のデータであることを保証する。 • 2 ― 繰り返し不可能読み出しとダーティ・リードを防止し、個々のトランザクションで書き込まれたデータが実際のデータであることを保証する。 • 3 ― 幻ロー、繰り返し不可能読み出し、ダーティ・リードを防止し、個々のトランザクションで書き込まれたデータが実際のデータであることを保証する。 カスタム・ファンクション文字列を使用することで、Replication Server は、レプリケート・データ・サーバが使用できる任意の独立性レベルをサポートできる。サポートは ANSI 標準のみに制限されない。 デフォルト値はターゲット・データ・サーバの現在のトランザクションの独立性レベル
dsi_large_xact_size	ラージ・トランザクションと見なされるまでに 1 つのトランザクション内で許可されるコマンドの数。 デフォルト値は 100 最小値：4 最大値：2,147,483,647 **dsi_compile_enable** が on の場合、このパラメータは無視される。
dsi_max_cmds_in_batch	出力コマンドのバッチ処理の対象にできるソース・コマンドの最大数を定義する。 パラメータの変更を反映させるには、コネクションをサスペンドしてレジュームする必要がある。 範囲：1 – 1000 デフォルト値は 100
dsi_max_xacts_in_group	グループ化できるトランザクションの最大数を指定する。大きい値を指定するほど、レプリケート・データベースでのデータ遅延時間が短縮される。値の範囲：1 – 1000 デフォルト値は 20 **dsi_compile_enable** が on の場合、このパラメータは無視される。

設定パラメータ	説明
dsi_num_large_ xact_threads	ラージ・トランザクションで使用するために予約されている並列 DSI スレッドの数。最大値は、**dsi_num_threads** の値から 1 を引いた値。 デフォルト値は 0
dsi_num_threads	使用する並列 DSI スレッドの数。最大値は 255。 デフォルト値は 1
dsi_partitioning_rule	利用可能な並列 DSI スレッド間でトランザクションを分割するために DSI が使用する、1 つ以上のパーティショニング・ルールを指定する。指定できる値は **origin**、**origin_sessid**、**time**、**user**、**name**、**none** のいずれか。 デフォルト値はなし **dsi_compile_enable** が on の場合、このパラメータは無視される。

設定パラメータ	説明
dsi_serialization_method	一貫性を保ちながら、トランザクションを開始できるタイミングを決定するために使用するメソッドを指定する。どの場合でもコミット順は保持される。 これらのメソッドは並列処理量の多い順になる。並列処理が多いほど、レプリケート・データベースに適用されるときに並列トランザクション間の競合が増える可能性がある。競合を減らすには、**dsi_partitioning_rule** オプションを使用する。 • **no_wait** ― 他のトランザクションの状態に関係なく、トランザクションが準備でき次第すぐに開始できることを指定する。 **注意：** **dsi_commit_control** を “on” に設定している場合は、**dsi_serialization_method** は **no_wait** にのみ設定できる。 • **wait_for_start** ― 開始直前にコミットするようにスケジュールされているトランザクションが開始した直後に、トランザクションを開始できるよう指定する。 • **wait_for_start** (デフォルト) ― 直前にコミットするようスケジュールされているトランザクションの準備が終了するまでは、トランザクションを開始できないよう指定する。 • **wait_after_commit** ― 直前にコミットするようにスケジュールされているトランザクションのコミットが完了するまで、トランザクションを開始できないよう指定する。 以下のオプションは、Replication Server の旧バージョンとの下位互換性のためにのみ維持されている。 • **none** ― **wait_for_start** と同じ。 • **single_transaction_per_origin** ― **dsi_partitioning_rule** が **origin** に設定された **wait_for_start** と同じ。 **注意：** **isolation_level_3** 値は逐次化メソッドとしてサポートされなくなりましたが、**dsi_serialization_method** を **wait_for_start** に設定し、**dsi_isolation_level** を 3 に設定した場合と同じです。 デフォルト値は **wait_for_commit**

設定パラメータ	説明
dsi_sqt_max_cache_size	アウトバウンド・キューの SQT (ステーブル・キュー・トランザクション) インタフェース・キャッシュ・サイズの最大量 (バイト単位)。デフォルトの 0 では、**sqt_max_cache_size** パラメータの現在の設定値が、コネクションの最大キャッシュ・サイズとして使用される。 デフォルト値は 0 32 ビット版 Replication Server の場合： • 最小値 – 0 • 最大– 2GB (2,147,483,648 バイト) 64 ビット版 Replication Server の場合： • 最小値 – 0 • 最大 – 2 ペタバイト (2,251,799,813,685,247 バイト)
dsi_xact_group_size	1 つにグループ化されたトランザクションに配置する最大バイト数 (ステーブル・キュー・オーバヘッドを含む)。グループ化されたトランザクションとは、DSI が単一のトランザクションとして適用するトランザクション・セットのことである。–1 はグループ化が行われないことを意味する。 Sybase では、**dsi_xact_group_size** を最大値に設定し、**dsi_max_xacts_in_group** でグループ内のトランザクション数を制御することをおすすめしている。 **注意：** このパラメータは、Replication Server バージョン 15.0 以降では使用されなくなっていますが、旧バージョンの Replication Server との互換性を保つために保持されています。 最大値：2,147,483,647 デフォルト値は 65,536 バイト **dsi_compile_enable** が on の場合、このパラメータは無視される。

設定パラメータ	説明
exec_cmds_per_timeslice	CPU を解放する前に、LTL または RepAgent エグゼキュータ・スレッドが処理できる LTL コマンドの数を指定する。この値を大きくすると、RepAgent エグゼキュータ・スレッドが CPU リソースをより長時間制御でき、RepAgent から Replication Server へのスループットが向上する。 このパラメータは、**alter connection** を使用してコネクション・レベルで設定する。 デフォルト値は 2,147,483,647 最小値：1 最大値：2,147,483,647
exec_max_cache_size	エグゼキュータ・コマンド・キャッシュに割り当てるメモリ量を指定する。 デフォルト値は 1,048,576 バイト 32 ビット版 Replication Server の場合： • 最小値 – 0 • 最大値 – 2,147,483,647 64 ビット版 Replication Server の場合： • 最小値 – 0 • 最大値 – 2,251,799,813,685,247
exec_nrm_request_limit	正規化待機中のプライマリ・データベースからのメッセージ用に使用可能なメモリ量を指定する。 **configure replication server** で **nrm_thread** を 'on' に設定してから、**exec_nrm_request_limit** を使用する。 最小値：16,384 バイト 最大値：2,147,483,647 バイト デフォルト値： • 32 ビット版 – 1,048,576 バイト (1MB) • 64 ビット版 – 8,388,608 バイト (8MB) **exec_nrm_request_limit** の設定を変更した後、Replication Agent をサスペンドしてレジュームする。 ライセンス：Advanced Services オプションで個別にライセンス供与される。

設定パラメータ	説明
exec_prs_num_threads	プライマリ・データベースから特定のコネクションの複数パーサ・スレッドを開始して非同期パーサ機能を有効にし、コネクションの非同期パーサ・スレッドの数を指定する。 デフォルト値は 0 最小値：0 (非同期パーサを無効にする) 最大値：20 **注意：** 非同期パーサを設定する前に、**smp_enable** が on であり、Replication Server のホスト・マシンが解析用の追加スレッドをサポートできることを確認してください。
exec_sqm_write_request_limit	インバウンド・キューへの書き込み待ちメッセージ用に使用可能なメモリ量を指定する。 デフォルト値は 1MB 最小値：16KB 最大値：2GB
md_sqm_write_request_limit	アウトバウンド・キューへの書き込み待ちメッセージ用にディストリビュータが使用可能なメモリ量を指定する。 **注意：** Replication Server 12.1 の場合、**md_source_memory_pool** は **md_sqm_write_request_limit** で置換されます。**md_source_memory_pool** は、旧バージョンの Replication Server との互換性を保つために保持されています。 デフォルト値は 1MB 最小値：16K 最大値：2GB
parallel_dsi	並列 DSI をデフォルト値に設定するための簡易な設定法。値を “on” にすると、**dsi_num_threads** が 5、**dsi_num_large_xact_threads** が 2、**dsi_serialization_method** が **wait_for_commit**、**dsi_sqt_max_cache_size** が 100 万バイトに設定される。値を “off” にすると、並列 DSI 値はデフォルトに設定される。このパラメータを “on” に設定した後、並列 DSI の各設定パラメータを適切な値に微調整できる。 デフォルト値は off
sqm_async_seg_delete	セグメントを削除する専用デーモンを有効にするには、**sqm_async_seg_delete** を on に設定する。 デフォルト値は on

設定パラメータ	説明
sqm_cmd_cache_size	Replication Server が SQM コマンド・キャッシュに保存できる解析データの最大サイズ (バイト単位)。 32 ビット版 Replication Server の場合： • デフォルト – 1,048,576 • 最小値 – 0 (SQM コマンド・キャッシュは無効) • 最大値 – 2,147,483,647 64 ビット版 Replication Server の場合： • デフォルト – 20,971,520 • 最小値 – 0 • 最大値 – 2,251,799,813,685,247 Replication Server では、**cmd_direct_replicate** または **sqm_cache_enable** が off に設定されている場合、**sqm_cmd_cache_size** が無視される。
sqm_max_cmd_in_block	各 SQM ブロックで、解析データが関連付けられるエントリの最大数を指定する。 デフォルト値は 320 最小値：0 最大値：4096 **sqm_max_cmd_in_block** の値を SQM ブロックのエントリ数に設定する。データのプロファイルによっては、ブロック・サイズが固定されており、メッセージ・サイズは予測できないため、ブロックごとにエントリが異なる場合がある。設定した値が大きすぎると、メモリを無駄にすることになる。値が小さすぎると、レプリケーションのパフォーマンスが低下する。 Replication Server では、**cmd_direct_replicate** または **sqm_cache_enable** が off に設定されている場合、**sqm_max_cmd_in_block** が無視される。
use_batch_markers	**use_batch_markers** が **on** に設定されている場合、ファンクション文字列 **rs_batch_start** と **rs_batch_end** が実行される。 **注意：** **rs_begin** および **rs_commit** ファンクション文字列に含まれていないコマンドのバッチの開始時および終了時に、追加の SQL 変換が送信される必要があるレプリケート・データ・サーバの場合にのみ、このパラメータを **on** に設定する必要がある。 デフォルト値は off

参照：

- Advanced Services Option (266 ページ)
- 並列 DSI スレッド (202 ページ)
- パーティショニング・ルール：競合を減らして並列処理を増やす (217 ページ)
- グループ内のトランザクション数の指定 (198 ページ)
- RepAgent エグゼキュータが処理できるコマンド数の制御 (196 ページ)

パフォーマンスに影響するルート・パラメータ

Replication Server には、パフォーマンスに影響する、いくつかのルート設定パラメータが用意されています。

ルート・パラメータの完全なリストについては、『Replication Server 管理ガイド 第 1 巻』の「ルートの管理」を参照してください。

表 17 : パフォーマンスに影響するルート・パラメータ

設定パラメータ	説明
rsi_batch_size	トランケーション・ポイントが要求される前に、別の Replication Server に送信されるバイト数。 デフォルト値は 256KB 最小値：1K 最大値：128MB
rsi_packet_size	他の Replication Server との通信に使用するパケット・サイズ (バイト単位)。値の範囲は、1,024 ～ 16,384。 デフォルト値は 4096 バイト
rsi_sync_interval	RSI 同期確認メッセージ間の秒数。Replication Server では、これらのメッセージを使用して、RSI アウトバウンド・キューと送信先 Replication Server が同期される。この値は 0 よりも大きくすること。デフォルト値：60 秒

チューニング・パラメータの使用についての注意事項

Replication Server のパフォーマンスを向上させるための基本的な推奨事項がいくつかあります。これらの設定値を変更してシステムのパフォーマンスが向上するかどうかは、システム設定やサイトでの Replication Server の使用方法によって異なります。

SQM ライタの待機時間の設定

SQM ライタの待機時間を設定するには、Replication Server の設定パラメータである **init_sqm_write_delay** と **init_sqm_write_max_delay** を使用します。

低容量システムでは、**init_sqm_write_delay** と **init_sqm_write_max_delay** を小さい値に設定すると、満杯になっていないブロックを書き込むまでの SQM ライタの待機時間を短くできます。高容量システムでは、SQM ライタはほとんど待機することなくブロックを書き込むため、これらのパラメータを大きい値に設定します。

SQM ライタの待機頻度をモニタするには、カウンタ 6038 - WritesTimerPop を参照します。

書き込まれた満杯のブロックや満杯になっていないブロックの数を確認するには、次のカウンタを参照します。

- 6002 - BlocksWriten
- 6041 - BlocksFullWrite

カウンタ 62006 - SleepsWriteQ がカウンタ 62002 - BlocksRead よりもかなり大きい値を示す場合、次のメッセージ・ブロックがダウンストリームに配信されるのを SQM リーダが待機する頻度が多くなり、遅延時間が生じます。満杯になっていないブロックを書き込むまでの SQM ライタの待機時間を短くするように、**init_sqm_write_delay** と **init_sqm_write_max_delay** の値を小さくしてください。

理想的なのは、カウンタ 62002 - BlocksRead に対するカウンタ 62004 - BlocksReadCached の比率を高くし、カウンタ 62006 - SleepsWriteQ を比較的小さく設定することです。この設定により、SQM ライタが SQM リーダと同程度の速さで適切に動作し、ディスクからの読み込みをせずに、前者から後者にブロックを渡すことができます。ただし、これらは Replication Server 全体に対するパラメータであり、これらを調整すると、あるキューの効率が上がり、別のキューの効率が下がることになります。

システム・テーブルのキャッシュ

システム・テーブルをキャッシュするには、Replication Server の設定パラメータである **sts_cache_size** と **sts_full_cache**_*table_name* を使用します。

簡単な **select** 文で特定のシステム・テーブルが RSSD にアクセスする必要がなくなるように、それらのシステム・テーブルを完全にキャッシュすることができます。デフォルトでは、`rs_repobjs` と `rs_users` が完全にキャッシュされます。使用する複写定義とサブスクリプションの数によっては、これらのテーブルを完全にキャッシュすると、RSSD へのアクセス要求を大幅に抑えることができます。ただし、`rs_objects` のユニークなローの数が **sts_cachesize** の値とほぼ同じ場合は、これらのテーブルがすでに完全にキャッシュされていることがあります。

複写システムに多数の複写定義があり、複写定義の変更要求も数多くある場合は、rs_objects、rs_columns、および rs_objfunctions の **sts_full_cache** が off になっていることを確認してください。これは、完全にキャッシュされるテーブルに変更が加えられると、そのテーブルのキャッシュ全体が更新されるためです。

キャッシュできるシステム・テーブル

キャッシュできるのは一部のシステム・テーブルだけです。

表 18 : キャッシュできるシステム・テーブル

テーブル			
rs_classes	rs_dbsubsets	rs_version	rs_datatype
rs_databases	rs_columns	rs_config	rs_routes
rs_objects	rs_diskaffinity	rs_asyncfuncs	rs_users
rs_sites	rs_queues	rs_repdbs	rs_dbreps
rs_repobjs	rs_systext	rs_publications	rs_objfunctions
rs_clsfunctions	rs_translation	rs_targetobjs	

複写定義の変更プロセス

複写定義の作成、変更、削除やファンクション文字列のカスタマイズといった多数の変更を RSSD に加える場合は、複写定義の変更プロセスを開始する前に rs_objects、rs_columns、および rs_objfunctions の **sts_full_cache** を無効にしておいて、複写定義の変更プロセスの後でこれらのテーブルの **sts_full_cache** を元の値に設定することをおすすめします。

ヒント： RSSD の変更が多い場合は、定期的に RSSD テーブルで Adaptive Server の **update statistics** コマンドを実行します。複写定義の作成、変更、または削除などの複写定義の変更要求で影響を受けるテーブルは rs_objects、rs_columns、および rs_objfunctions です。ファンクション文字列の作成、変更、または削除などのファンクション文字列の変更要求で変更を受けるテーブルは rs_funcstrings および rs_systext です。

sts_full_cache を無効にするには、次のように入力します。ここで、*system_table_name* はテーブルの名前です。

```
configure replication server
set sts_full_cache_system_table_name to 'off
```

『Replication Server 管理ガイド 第 1 巻』の「複写テーブルの管理」の「複写定義の修正」で、「複写定義の変更」の「複写定義の変更要求プロセス」を参照してください。

エグゼキュータ・コマンド・キャッシュ

Sybase RepAgent が最初にプライマリ Adaptive Server データベース・テーブルの **insert**、**delete**、または **update** LTL コマンドを送信するとき、エグゼキュータ・コマンド・キャッシュを使用して、そのテーブルのカラム名とデータ型をキャッシュします。

カラム名やデータ型などのメタデータは、RepAgent が **insert**、**delete**、**update** コマンドに関連付けられたデータとともに送信するテーブル・スキーマに含まれています。ただし、キャッシュでは、以下のようになります。

- RepAgent が **insert**、**update**、**delete** コマンドに関連付けられたメタデータおよびデータを送信するのは、RepAgent が起動してから、または Replication Server とのコネクションが再開してから、RepAgent がその特定のテーブルに対するオペレーションを初めて処理するときだけです。Replication Agent は、それ以降にそのテーブルのトランザクションを処理するとき、テーブル・メタデータを送信しません。
- スキーマ定義をすべて維持するのに十分なメモリが RepAgent にない場合、RepAgent はメタデータとデータを再送信できます。
- RepAgent は、Adaptive Server の **alter table** オペレーションの後など、テーブル・スキーマの変更後に特定のテーブルでの変更を処理するとき、テーブルのメタデータとデータを送信します。

同じテーブルに対するそれ以降のオペレーションをレプリケートするため、RepAgent はカラム・データのみを送信します。これは、Replication Server エグゼキュータ・コマンド・キャッシュがメタデータを格納しているからです。RepAgent のメタデータの低減と Replication Server エグゼキュータ・コマンド・キャッシュでのキャッシュを組み合わせることで、レプリケーション・パフォーマンスが向上します。その要因は、キャッシュに以下の性質があるためです。

- RepAgent がメタデータをログ転送言語 (LTL) パケットにパックするために費やす時間を短縮します。
- 各パケットで送信されるデータの量を増やすことで、ネットワーク・トラフィックを低減します。
- RepAgent は節約した時間をメタデータのパックではなくプライマリ・データベース・ログのスキャンに充てることができます。
- Replication Server エグゼキュータは多くのカラムがあるテーブルを効率よく処理できます。

注意： キャッシュには、**insert**、**update**、または **delete** オペレーションによって変更されたテーブルのメタデータのみが格納されます。

システムの稼働条件
テーブル・メタデータを低減するには、LTL バージョン 740 以降および Adaptive Server 15.7 以降が必要です。

テーブル・メタデータの低減の有効化

Sybase RepAgent のテーブル・メタデータの低減を有効にします。RepAgent のテーブル・メタデータの低減を有効にした場合、Replication Server ではエグゼキュータ・コマンド・キャッシュが自動的に有効になります。

1. Adaptive Server で、次のコマンドを実行します。

   ```
   sp_config_rep_agent database_name, 'ltl metadata reduction',
   'true'
   ```

   ここで *database_name* はプライマリ Adaptive Server データベースになります。

   注意： デフォルトでは、**ltl metadata reduction** は fase に設定されているため、Adaptive Server の RepAgent ではテーブル・メタデータの低減が有効になりません。

2. 変更を有効にするには、RepAgent を再起動します。

   ```
   sp_start_rep_agent database_name
   ```

エグゼキュータ・コマンド・キャッシュのサイズの設定

exec_max_cache_size を使用して、エグゼキュータ・コマンド・キャッシュに割り当てるメモリ量を指定します。

キャッシュに割り当てたメモリが不足していると、レプリケーションのパフォーマンスが低下し、次のメッセージが頻繁に表示されるようになります。

```
Executor Command Cache exceeds its maximum limit defined by
exec_max_cache_size (current value is current_exec_max_cache_size).
```

レプリケーションのパフォーマンスのさらなる低下を防ぐためには、**exec_max_cache_size** を使用してキャッシュのメモリを増やすか、Replication Agent でテーブル・メタデータの低減を無効にします。

32 ビット版 Replication Server の場合は 0 ～ 2,147,483,647 バイトの値を設定でき、64 ビット版 Replication Server の場合は 0 ～ 2,251,799,813,685,247 バイトの値を設定することができます。デフォルトは 32 ビット版と 64 ビット版の両方の Replication Server で 1,048,576 バイトです。

たとえば、エグゼキュータ・コマンドのキャッシュ・サイズを 2,097,152 バイトに設定するには、次のようにします。

- サーバ・レベル ― Replication Server へのすべてのプライマリ・データベース・コネクションで、次のように入力します。

```
configure replication server
set exec_max_cache_size to '2097152'
```

- コネクション・レベル – 特定のプライマリ・データベース・コネクションについては、次のように入力します。

```
alter connection to dataserver_name.database_name
set exec_max_cache_size to '2097152'
```

両方のレベルに設定がある場合、Replication Server は常にコネクション・レベルの設定を使用します。変更内容を有効にするために、Replication Server を再起動する必要はありません。

ステーブル・キューのキャッシュ

Replication Server は、単純なキャッシュ・メカニズムを使用して I/O を最適化します。このメカニズムにより、通常はキャッシュからデータを高速に読み取ることができるため、書き込みに対する遅延時間が短縮され、読み取り速度が向上します。

キャッシュは複数のページで構成され、各ページは隣接する複数のブロックで構成されています。キャッシュは、起動時に各キューに割り付けられます。ページ・サイズを変更すると、ステーブル・キュー・デバイス内の I/O のサイズが変化します。ページがいっぱいになると、単一の書き込み操作でページ全体が書き込まれます。

ステーブル・キュー・キャッシュでは、ページ・ポインタが前進し、キャッシュの終端で先頭に戻ります。ライタがメッセージ・キューを満杯にし、メッセージを待機しているときにブロックされると、SQM によって現在のページがフラッシュされます。いっぱいになっていないページがフラッシュされると、データを含むブロックだけがディスクに書き込まれます。

ステーブル・キュー・キャッシュのパラメータの設定

ステーブル・キュー・キャッシュについて設定できるパラメータがいくつかあります。

サーバ全体のキャッシュのデフォルト値を設定するには、次のコマンドを実行します。

```
configure replication server set sqm_cache_enable to
"on|off"
```

キューのキャッシュを有効または無効にしてサーバ・レベルの設定を無効にするには、次のコマンドを実行します。

```
alter queue q_number, q_type, set sqm_cache_enable to
"on|off"
```

sqm_cache_enable パラメータが無効になっている場合、SQM モジュールは、16K に固定された 1 ブロックのバッファを保持する以前のメカニズムに戻ります。

サーバ全体のページ・サイズのデフォルト値を設定するには、次のコマンドを実行します。

```
configure replication server set sqm_page_size to
"num_of_blocks"
```

指定したキューのページ・サイズを設定するには、次のコマンドを実行します。

```
alter queue q_number, q_type, set sqm_page_size to
"num_of_blocks"
```

num_of_blocks では、ページ内の 16K ブロックの数を指定します。ページ・サイズを設定すると、Replication Server の I/O サイズも設定されます。たとえば、ページ・サイズを 4 に設定すると、Replication Server は 64K のまとまりでステーブル・キューに書き込みを行います。

サーバ全体のキャッシュ・サイズのデフォルト値を設定するには、次のコマンドを実行します。

```
configure replication server set sqm_cache_size to
"num_pages"
```

指定したキューのキャッシュ・サイズを設定するには、次のコマンドを実行します。

```
alter queue q_number, q_type, set sqm_cache_size to
"num_pages"
```

num_pages では、キャッシュ内のページの数を指定します。

すべての SQM 設定コマンドは静的であるため、コマンドを有効にするにはサーバを再起動する必要があります。

これらの設定パラメータの詳細については、『Replication Server リファレンス・マニュアル』を参照してください。

非同期パーサ、ASCII パッキング、および直接コマンド・レプリケーション

非同期パーサ、ASCII パッキング、およびインバウンドとアウトバウンドの直接コマンド・レプリケーションの各機能を同時に活用することで、データの変換と転送時にレプリケーション・プロセス全体が改善されます。

Replication Server は、エグゼキュータ (EXEC) スレッド、ディストリビュータ (DIST) スレッド、およびデータ・サーバ・インタフェース (DSI) スレッドを順序どおりに処理します。

1. EXEC スレッドは、LTL コマンドを Replication Agent から Replication Server に転送します。
2. また、LTL コマンドを解析しそれらを内部解析形式で格納し、解析済みデータをバイナリ形式でパックします。

3. EXEC スレッドは、インバウンド SQM スレッドを介してバイナリ・データをインバウンド・キューに書き込みます。
4. DIST モジュールは、バイナリ・コマンドを取得し、コマンドを元の形式にリストアし、コマンドの送信先を決定します。
5. DIST モジュールがコマンドを DSI モジュールに送信する前に、DIST はコマンドを内部 ASCII 形式でパックし、アウトバウンド SQM スレッドを介してコマンドをアウトバウンド・ステーブル・キューに書き込みます。
6. DSI モジュールは、ASCII コマンドを読み取って解析し、コマンドを元の形式にリストアします。

非同期パーサ、およびインバウンドとアウトバウンドの直接コマンド・レプリケーションの各機能を使用すると、シーケンス内の特定の手順のレプリケーション・パフォーマンスが向上し、その一方で、ASCII パッキングにより、キューのストレージ消費量が削減されます。ただし、これらの機能をすべて同時に使用することで、パフォーマンスが最大になりキューのストレージ消費量を低減できます。これらの機能を個別に設定するのではなく、**async_parser** と **alter connection** を使用して同時に設定します。

非同期パーサ機能を設定する前に、**smp_enable** が on であり、Replication Server のホスト・マシンが解析用の追加スレッドをサポートできることを確認してください。

async_parser を on に設定すると、次のように設定されます。

- 非同期パーサを on に設定 – **exec_prs_num_threads** を 2 に設定する
- ASCII パッキングを on に設定 – **ascii_pack_ibq** を on に設定する
- インバウンド・コマンドの直接レプリケーションを on に設定 – **cmd_direct_replicate** を on に設定する
- アウトバウンド・コマンドの直接レプリケーションを on に設定 – **dist_cmd_direct_replicate** を on に設定する

async_parser を on に設定してから個々のパラメータを個別に設定し、パフォーマンスとリソース消費量を調整してバランスを取ることもできます。**async_parser** のデフォルト値は off です。個々のパラメータをデフォルト設定にリセットするには、**async_parser** を off に設定します。**ascii_pack_ibq** を on に設定する前に、Replication Server のサイト・バージョンを 1571 以降に設定する必要があります。サイトのバージョンが 1571 だと、**async_parser** を on に設定しても、**exec_prs_num_threads**、**cmd_direct_replicate**、および **dist_cmd_direct_replicate** だけが設定されます。

suspend distributor と **resume distributor** を使ってディストリビュータを再開するには、**async_parser** を使用します。**async_parser** を on に設定すると、関連する Replication Agent が再起動します。

非同期パーサ
追加のエグゼキュータ・スレッドを設定して Replication Agent からのコマンドをパースし、Replication Agent がエグゼキュータを待機する時間を短縮します。

1 つの関連付けられたエグゼキュータ・スレッドを使用する場合、Replication Agent はスレッドが LTL コマンドの解析を完了するのを待機してから、次のバッチ・コマンドを Replication Server に送信できます。複数のスレッドを LTL コマンドの解析専用に設定する場合、Replication Server は個々の解析タスクを専用の非同期パーサ・スレッドに割り当てるため、LTL コマンドのいくつかのパケットが並列に解析されます。さらに、データの転送と解析は複数のスレッドと非同期であるため、Replication Agent はエグゼキュータ・スレッドを待機しなければならない時間が短縮されることで、レプリケーションのスループットが向上します。

非同期パーサ機能を設定する前に、**smp_enable** が on であり、Replication Server のホスト・マシンが解析用の追加スレッドをサポートできることを確認してください。非同期パーサ機能を使用するには、**exec_prs_num_threads** を **alter connection** で設定し、プライマリ・データベースから特定のコネクションの複数パーサ・スレッドを開始し、コネクションの非同期パーサ・スレッドの数を指定します。Replication Server は、**exec_prs_num_threads** を設定すると、Replication Agent を再起動します。起動できる最大スレッド数は 20 です。非同期パーサを無効にするには、0 に設定します。最小値 (デフォルト値) は 0 です。非同期パーサ・スレッドのほかに、コマンド・バッチの同期スレッドも起動します。

exec_prs_num_threads の設定に失敗し、Replication Server がシャットダウンする場合は、合計数が次のような場合です。

- 起動するスレッドが **num_threads** で指定したプールで使用可能なスレッドの数よりも大きい場合
- 作成されるメッセージ・キューが **num_msg_queues** で指定したプールで使用可能なメッセージ・キューの数を超えている場合

例

TOKYO_DS データ・サーバ内の `pdb1` プライマリ・データベースへのコネクションの 4 つのパーサ・スレッドを起動するには、次のように入力します。

```
alter connection to TOKYO_DS.pdb1
set exec_prs_num_threads to 4
go
```

非同期パーサ・スレッドの確認
存在する非同期パーサ・スレッドの数を表示するには、**admin who** を使用します (**exec_prs_num_threads** が 0 より大きい値に設定されている場合)。さらに、**admin who** は、コマンド・バッチの同期スレッドが存在することも示します。

ASCII パッキング

ASCII パッキングを非同期パーサと併用することで、インバウンド・キューにパックされたコマンドが消費するステーブル・キューの記憶領域を低減します。

インバウンド・キューでコマンドのデフォルトのバイナリ・パッキング・モードを使用する代わりに、**ascii_pack_ibq** を on に設定し、Replication Server がバイナリ・パッキングよりも小さいステーブル・キューの記憶領域を使用する ASCII パッキングを使用してコマンドをパックします。デフォルトは off です。バイナリ形式でパックされたコマンドは多くのステーブル・キューの記憶領域を使用しますが、Replication Server はバイナリ形式でパックされたコマンドを ASCII でパックされたコマンドより高速に解釈します。

ascii_pack_ibq を on に設定すると、インバウンド・キューのコマンドのみがパックされます。Replication Server では、ASCII パッキングを使用してコマンドが既にパッキングされているため、インバウンド・キューでコマンドのパッキング・モードを変更することはできません。Replication Server でインバウンド・キューの ASCII パッキングを使用するメリットを得るには、非同期パーサ機能を有効にし、**ascii_pack_ibq** を on に設定する前に Replication Server のサイト・バージョンを 1571 以降に設定する必要があります。

例

TOKYO_DS データ・サーバ内の `pdb1` プライマリ・データベースからコネクションに使用するパッキング・モードを ASCII パッキングに設定するには、次のように入力します。

```
alter connection to TOKYO_DS.pdb1
set ascii_pack_ibq to on
go
```

インバウンド・コマンドの直接レプリケーション

Replication Server EXEC モジュールと DIST モジュールとの間のインバウンド・レプリケーション・パスにおけるコマンド変換を低減し、レプリケーション・パフォーマンスを向上させることができます。

インバウンド・データの場合、**cmd_direct_replicate** を on に設定すると、エグゼキュータ・スレッドはバイナリ・データまたは ASCII フォーマット・データとともに内部解析データを送信します。Replication Server は、解析済みデータを独立した SQM コマンド・キャッシュに格納します。SQM コマンド・キャッシュ内の解析済みデータは、SQM キャッシュに格納されたバイナリ・データまたは ASCII フォーマット・データにマッピングされます。ディストリビュータ・モジュールは、必要に応じて内部フォーマット解析済みデータからデータを直接取得して処理できるので、バイナリ・データまたは ASCII フォーマット・データの解析に費やされる時間を節約できます。ASCII フォーマットの解析にはより多くのリソー

スが必要になるため、コマンド変換の低減と ASCII フォーマットのパック・データのレプリケーション・パフォーマンスの向上は、バイナリ・パックのデータよりも多くなります。デフォルトは off です。

ディストリビュータ・スレッドを SQM コマンド・キャッシュから読み取ることができるのは、最初にスレッドが物理ディスクでなく SQM キャッシュから読み込むことができ、**sqm_cache_size** を使用して適切な SQM キャッシュ・サイズを設定する場合のみです。ディストリビュータ・スレッドが SQM キャッシュからコマンドを読み取ると、スレッドが SQM コマンド・キャッシュ内のこのコマンドの解析済みバージョンを見つけることができるかどうかは、**sqm_cmd_cache_size** を使用して設定する SQL コマンド・キャッシュ・サイズと **sqm_max_cmd_in_block** を使用して設定可能な解析済みコマンドに関連付けることができる SQM ブロックの最大エントリ数によって決まります。

configure replication server を使用して、すべてのデータベースへの Replication Agent コネクションに対してサーバ・レベルで **cmd_direct_replicate** を設定します。それ以外の場合は、**alter connection** を設定して、個々のコネクションのパラメータを設定します。**alter connection** を使用した場合に設定を有効にするには、Replication Agent を再起動します。**configure replication server** を使用する場合は、Replication Server を再起動します。

sqm_cmd_cache_size パラメータと **sqm_max_cmd_in_block** パラメータを使用して、SQM コマンド・キャッシュ・メモリ設定を設定します。**cmd_direct_replicate**、**sqm_cmd_cache_size** および **sqm_max_cmd_in_block** を同じコマンド内または個別に設定できます。Replication Server は **sqm_cmd_cache_size** および **sqm_max_cmd_in_block** の設定を無視します (**sqm_cache_enable** が off の場合)。

例 1

64 ビット版 Replication Server のすべてのコネクションとキューの設定を行うには、次のようにします。

```
configure replication server
set cmd_direct_replicate to 'on'
set sqm_cmd_cache_size to '40971520'
set sqm_max_cmd_in_block to '640'
go
```

例 2

TOKYO_DS データ・サーバの `pdb1` プライマリ・データベースへのコネクションの設定と 32 ビット版 Replication Server のインバウンド・キュー番号 2 の設定を行うには、次のようにします。

```
alter connection to TOKYO_DS.pdb1
set cmd_direct_replicate to 'on'
go
```

```
alter queue 2, 1,
set sqm_cmd_cache_size to '2048576'
set sqm_max_cmd_in_block to '640'
go
```

参照：

- キュー・ブロック・サイズの増加 (288 ページ)
- カウンタを使ったパフォーマンスのモニタリング (339 ページ)
- ステーブル・キュー・キャッシュのパラメータの設定 (184 ページ)

SQM コマンド・キャッシュ・カウンタを使ったパフォーマンスのモニタリング

sqm_cache_enable と **cmd_direct_replicate** が on で、**sqm_cmd_cache_size** と **sqm_max_cmd_in_block** がゼロ以外の値に設定されている場合は、エグゼキュータとディストリビュータのスレッドが解析済みデータと対話するときに、いくつかのカウンタを使用してレプリケーション・パフォーマンスをモニタできます。

表 19 : SQM コマンド・キャッシュ・カウンタ

カウンタ	説明
RACmdsDirectRepSend	解析済みデータに関連付けられたエグゼキュータ・スレッドから送信されるコマンドの数。
DISTCmdsDirectRepRecv	エグゼキュータから直接文に関連付けられた解析データを持つため、解析処理をスキップできるディストリビュータが受け取るコマンドの数。
SQMNoDirectReplicateInCache	エグゼキュータ・スレッドから送信された解析済みデータを持つコマンドの数。ただし、コマンド・キャッシュが **sqm_cmd_cache_size** を超えるため、ディストリビュータへのレプリケーション経路に沿って解析済みデータをそれ以上送信できない。
SQMNoDirectReplicateInSQMCache	エグゼキュータ・スレッドから送信された解析済みデータを持つコマンドの数。ただし、これらのコマンドが読み取られる前に SQM キャッシュで上書きされたため、ディストリビュータへのレプリケーション経路に沿って解析済みデータをそれ以上送信できない。
SQMNoDirectReplicateInBlock	エグゼキュータ・スレッドから送信された解析済みデータを持つコマンドの数。ただし、現在の SQM ブロックの解析済みデータ・エントリの数が **sqm_max_cmd_in_block** を超えるため、ディストリビュータへのレプリケーション経路に沿って解析済みデータをそれ以上送信できない。

アウトバウンド・コマンドの直接レプリケーション

Replication Server DIST モジュールと DSI モジュールとの間のアウトバウンド・レプリケーション・パスにおけるコマンド変換を低減し、レプリケーション・パフォーマンスを向上させることができます。

アウトバウンド・データでは、**dist_cmd_direct_replicate** を on に設定すると、DIST モジュールはパックされた ASCII 形式データとともに内部解析データを送信できます。DSI モジュールは、必要に応じて解析済みデータからデータを直接取得して処理できるので、ASCII 形式データの解析に費やされる時間を節約できます。**dist_cmd_direct_replicate** を off に設定すると、DIST モジュールはパックされた ASCII データを DSI に送信します。デフォルトは on です。

DSI モジュールを SQM コマンド・キャッシュから読み取ることができるのは、最初にスレッドが物理ディスクでなく SQM キャッシュから読み込むことができ、**sqm_cache_size** を使用して適切な SQM キャッシュ・サイズを設定する場合のみです。DSI モジュールが SQM キャッシュからコマンドを読み取ると、モジュールが SQM コマンド・キャッシュ内のこのコマンドの解析済みバージョンを見つけることができるかどうかは、**sqm_cmd_cache_size** を使用して設定する SQM コマンド・キャッシュ・サイズと **sqm_max_cmd_in_block** を使用して設定可能な解析済みコマンドに関連付けることができる SQM ブロックの最大エントリ数によって決まります。

configure replication server を使用して、すべてのディストリビュータに対してサーバ・レベルで **dist_cmd_direct_replicate** を設定します。それ以外の場合は、**alter connection** を使用して、個々のディストリビュータのパラメータを設定します。**alter connecion** を使用する場合、設定を有効にするには、**suspend distributor** および **resume distributor** を使用して特定のディストリビュータを再開します。**configure replication server** を使用する場合は、Replication Server を再起動します。

sqm_cmd_cache_size パラメータと **sqm_max_cmd_in_block** パラメータを使用して、SQM コマンド・キャッシュ・メモリ設定を設定します。**dist_cmd_direct_replicate**、**sqm_cmd_cache_size**、および **sqm_max_cmd_in_block** を同じコマンド内または個別に設定できます。Replication Server は **sqm_cmd_cache_size** および **sqm_max_cmd_in_block** の設定を無視します (**sqm_cache_enable** が off の場合)。

例 1

64 ビット版 Replication Server のすべてのコネクションとキューの設定を行うには、次のようにします。

```
configure replication server
set dist_cmd_direct_replicate to 'on'
set sqm_cmd_cache_size to '40971520'
set sqm_max_cmd_in_block to '640'
go
```

例 2

TOKYO_DS データ・サーバの pdb1 プライマリ・データベースへのコネクションの設定と 32 ビット版 Replication Server のインバウンド・キュー番号 3 の設定を行うには、次のようにします。

```
alter connection to TOKYO_DS.pdb1
set dist_cmd_direct_replicate to 'on'
go
alter queue 2, 1,
set sqm_cmd_cache_size to '2048576'
set sqm_max_cmd_in_block to '640'
go
```

SQM コマンド・キャッシュ・カウンタを使ったパフォーマンスのモニタリング

sqm_cache_enable と **dist_cmd_direct_replicate** が on で、**sqm_cmd_cache_size** と **sqm_max_cmd_in_block** がゼロ以外の値に設定されている場合は、EXEC、DIST、および DSI スレッドとモジュールが解析済みデータと対話するときに、いくつかのカウンタを使用してレプリケーション・パフォーマンスをモニタできます。

表 20 : SQM コマンド・キャッシュ・カウンタ

カウンタ	説明
DISTCmdsDirectRepSend	DIST モジュールから直接送信される解析済みデータ形式のコマンドの数。
DSIECmdsDirectRepRecv	アウトバウンド・コマンドを伝達する DIST モジュールまたはスタンバイ・インバウンド・コマンドを伝達する EXEC モジュールから直接 DSI-E スレッドが受信したコマンドの数。DSI モジュールがウォーム・スタンバイ・コネクション用である場合は、インバウンド・コマンドがカウントされ、そうでない場合は、アウトバウンド・コマンドがカウントされる。
SQMNoDirectReplicateInCache	ディストリビュータ・スレッドから送信された解析済みデータを持つコマンドの数。ただし、コマンド・キャッシュが **sqm_cmd_cache_size** を超えるため、DSI スレッドへのレプリケーション経路に沿って解析済みデータをそれ以上送信できない。
SQMNoDirectReplicateInSQMCache	ディストリビュータ・スレッドから送信された解析済みデータを持つコマンドの数。ただし、これらのコマンドが読み取られる前に SQM キャッシュで上書きされたため、DSI スレッドへのレプリケーション経路に沿って解析済みデータをそれ以上送信できない。

カウンタ	説明
SQMNoDirectReplicateInBlock	ディストリビュータ・スレッドから送信された解析済みデータを持つコマンドの数。ただし、現在の SQM ブロックの解析済みデータ・エントリの数が **sqm_max_cmd_in_block** を超えるため、DSI スレッドへのレプリケーション経路に沿って解析済みデータをそれ以上送信できない。

SQM コマンド・キャッシュ・メモリ設定

SQM コマンド・キャッシュ・メモリの設定内容は、Replication Server で使用できる合計メモリ量、インバウンド・キューとアウトバウンド・キューの数、およびトランザクション・プロファイルに応じて異なり、これらはコマンド・サイズによって変化します。

SQM コマンド・キャッシュ・メモリ設定を設定する場合は、以下を実行します。

- **sqm_cmd_cache_size** を増やします (Replication Server で SQM キャッシュの合計数が多い場合)。SQM キャッシュの合計 = **sqm_cache_size** (ページ) * **sqm_page_size** (ブロック) * **block_size** (キロバイト)
- **sqm_max_cmd_in_block** を小さくします (コマンド・サイズまたはテーブルのロー・サイズが大きい場合)
- **sqm_max_cmd_in_block** を増やします (**block_size** が大きい場合)。

初期値を設定したら、レプリケーション・パフォーマンスとモニタ・カウンタのデータに基づいて値を調整します。

- **sqm_cmd_cache_size** を増やします (SQMNoDirectReplicateInCache に大きな値が表示される場合)。
- **sqm_max_cmd_in_block** を増やします (SQMNoDirectReplicateInBlock に大きな値が表示される場合)。

configure replication server を使用して、Replication Server へのすべてのデータベース・コネクションに対して **sqm_cache_size**、**sqm_page_size**、および **block_size** を変更します。それ以外の場合は、**alter queue** を使用して、特定のデータベース・コネクションの設定を行います。

パラメータのデフォルト値と有効な値の範囲については、『Replication Server リファレンス・マニュアル』の「Replication Server コマンド」を参照してください。

ウェイクアップ・インターバルの設定

ウェイクアップ・インターバルを設定するには、Replication Server の設定パラメータである **rec_daemon_sleep_time** と **sub_daemon_sleep_time** を使用します。

デフォルトでは、リカバリ・デーモンとサブスクリプション・デーモンが 2 分ごとにウェイクアップし、RSSD のメッセージをチェックします。一般的な運用環

境では、サブスクリプション・デーモンはほとんど使用されません。そのため、サブスクリプション・デーモンのウェイクアップ・インターバルを最大値 (31,536,000 秒) に設定できます。また、リカバリ・デーモンのウェイクアップ・インターバルを長くすべきかどうかを評価できます。

SQT キャッシュのサイズ設定

SQT キャッシュのサイズを設定するには、Replication Server の設定パラメータである **sqt_max_cache_size** とデータベース・コネクション設定パラメータである **dsi_sqt_max_cache_size** を使用します。

SQT キャッシュの使用状況をモニタするには、カウンタ 24005 - CacheMemUsed を参照します。代わりに、カウンタ 24009 – TransRemoved をモニタしてください。TransRemoved が 0 のままの場合は、トランザクションがキャッシュからフラッシュされず、他のトランザクションが使用する領域が作成されないことを示しますが、**sqt_max_cache_size** を調整する必要はありません。

警告！ **sqt_max_cache_size** に大きすぎる値を設定すると、サーバがシャットダウンすることがあり、サーバの **memory_limit** が SQT キャッシュのサイズ変更に対応できる十分高い値に設定されていない場合に Replication Server のリソース全体に影響を及ぼすことがあります。

sqt_max_cache_size は、DIST クライアントをサポートするすべての SQT キャッシュに適用され、DSI クライアントをサポートする SQT キャッシュに対するデフォルト値を用意します。DIST はトランザクションをすぐに処理できるため、その SQT キャッシュを DSI の SQT キャッシュと同じサイズにする必要はありません。そのため、コネクション設定パラメータ **dsi_sqt_max_cache_size** や DIST SQT キャッシュ専用の **sqt_max_cache_size** を使用して、DSI の SQT キャッシュ・サイズを個々に設定することをおすすめします。

注意： 15.5 より前のバージョンの Replication Server では、**sqt_max_cache_size** の設定が高すぎるとレプリケーションが遅くなります。Replication Server バージョン 15.5 以降の場合、このアドバイスは該当しません。

未処理のバイト数の制御

exec_nrm_request_limit、**exec_sqm_write_request_limit**、および **md_sqm_write_request_limit** データベース・コネクション設定パラメータを使用して、メモリの未処理のバイト数を制御します。

exec_nrm_request_limit は RepAgent エグゼキュータ・スレッドの効率の向上のために別途にライセンス供与されるオプションです。

参照：

- RepAgent エグゼキュータ・スレッドの効率の向上 (286 ページ)

exec_sqm_write_request_limit データベース設定パラメータ

exec_sqm_write_request_limit は、インバウンド・キューへの書き込み待ちメッセージ用に使用可能なメモリ量を制御します。

md_sqm_write_request_limit データベース設定パラメータ

md_sqm_write_request_limit は、DIST スレッドが保持できる未処理のバイト数を制御します。最大数に達すると、DIST スレッドはこれらのバイトの一部がアウトバウンド・キューに書き込まれるのを待機しなければならなくなります。

パフォーマンスをモニタリングするためのカウンタの使用

カウンタを使用して、RepAgent エグゼキュータと NRM スレッドのパフォーマンスをモニタできます。

ノーマライゼーションが完了するのを待機する間、RepAgent エグゼキュータがスリープする回数や時間をモニタするには、次のカウンタを参照します。

- 58038 – RAWaitNRMTime

RepAgent エグゼキュータまたは NRM がインバウンド・キューに書き込みメッセージが入るのを待つ間にスレッドがスリープする回数や時間をモニタするには、次のカウンタを参照します。

- 58019 – RAWriteWaitsTime

RAWriteWaitsTime が常に大きい場合は、StableDevice I/O を確認します。

参照：

- カウンタを使ったパフォーマンスのモニタリング (339 ページ)

ネットワーク・オペレーション数の制御

DSI コマンド・バッチのサイズを制御するには、**dsi_cmd_batch_size** データベース・コネクション設定パラメータを使用します。

dsi_cmd_batch_size は、DSI がレプリケート・データ・サーバにコマンドを送信するときに使用するバッファのサイズを制御します。DSI 設定バッチを on に設定すると、DSI は 1 つのコマンド・バッチに含められるだけのコマンドを配置してから、そのバッチをレプリケート・データ・サーバに送信します。
dsi_cmd_batch_size の値を大きくすると、レプリケート・データベースにコマンド・バッチあたりの作業をより多く提供することで、スループットを向上できる場合があります。

バッチとバッチ・サイズをモニタするためのカウンタ

Replication Server では、バッチとバッチ・サイズをモニタするためにカウンタを用意しています。

バッチの平均サイズをモニタするには、カウンタ 57076 – DSIEBatchSize を参照します。バッチの平均処理時間 (バッチが作成されてから、フラッシュされて結果が処理されるまでの時間) をモニタするには、カウンタ 57070 – DSIEBatchTime を参照します。

バッチの処理効率やバッチ・サイズをモニタする場合、次のカウンタも役立つことがあります。

57037 – SendTime	57079 – DSIEOCmdCount	57063 – DSIEResultTime
57070 – DSIEBatchTime	57092 - DSIEBFMaxBytes	57076 – DSIEBatchSize

RepAgent エグゼキュータが処理できるコマンド数の制御

exec_cmds_per_timeslice データベース・コネクション設定パラメータを使用して、RepAgent エグゼキュータ・スレッドが処理できるコマンド数を制御します。

デフォルトで、**exec_cmds_per_timeslice** パラメータの値は 2,147,483,647 です。これは、他のスレッドに CPU を解放しなければならくなるまでに RepAgent エグゼキュータ・スレッドが処理できるコマンド数が 2,147,483,647 個以下であることを示します。環境によっては、この値を変更するとパフォーマンスが向上する場合があります。

インバウンド・キューの処理が遅い場合、これらの値を大きくして、RepAgent エグゼキュータ・スレッドと DIST スレッドの処理に費やせる時間を増やしてください。ただし、アウトバウンド・キューの処理が遅い場合は、これらのパラメータ値を小さくして、DSI スレッドの処理に費やせる時間を増やしてください。

Replication Server がサポートするコネクション数について CPU リソースが制限されている場合、**exec_cmds_per_timeslice** の値を大きくすると全体のパフォーマンスが低下することがあります。この場合、RepAgent エグゼキュータによる CPU リソースの制御を厳しくすると、他の Replication Server スレッドに対するリソースが減少することがあります。

RepAgent エグゼキュータ・スレッドが CPU を解放する回数や時間をモニタするには、次のカウンタを参照します。

- 58016 – RAYieldTime

割り付けられるステーブル・キュー・セグメント数の指定

sqm_recover_segs Replication Server 設定パラメータを使用して、Replication Server が RSSD をリカバリ QID 情報で更新する前に割り付けるステーブル・キュー・セグメント数を指定します。

sqm_recover_segs を小さく設定すると、RSSD の更新が増え、パフォーマンスが低下する場合があります。**sqm_recover_segs** を大きく設定すると、RSSD の更新が減り、リカバリ時間が長くなる代わりにパフォーマンスが向上する場合があります。

SQM ライタが `rs_oqids` テーブルを更新する頻度をモニタするには、カウンタ 6036 - UpdsRsoqid を参照します。通常は、**sqm_recover_segs** の値を大きくすると、セグメントの割り付けに必要な時間とシステム・リソースが減り、パフォーマンスが向上します。ただし、各オリジンに対して正常に書き込まれた最後のメッセージを決定するために SQM ライタがスキャンしなければならないキューが増えるため、キューの起動と再起動に時間がかかります。各セグメントには 1MB のキュー領域が必要です。SQM ライタが起動時や再起動時にスキャンできるメガバイト数を計算して、**sqm_recover_segs** の値を決定します。たとえば、SQM ライタが Replication Server の起動や再起動を遅らせることなく 50MB のキューをスキャンできる場合、**sqm_recover_segs** を 50 に設定します。

ステーブル・キューのディスク・パーティションの選択

disk_affinity データベース・コネクション設定パラメータを使用して、現在のパーティションが満杯になった場合に、次のセグメントの割り付け先となるパーティションの論理名を指定します。

Replication Server のパーティション関係の機能を使用すると、Replication Server によるステーブル・キューのセグメントの割り付け先のディスク・パーティションを選択できます。全体のスループットを向上させるため、処理の遅いステーブル・キューには処理速度の速いデバイスを関連付けることをおすすめします。

参照：

- キュー・セグメントの割り付け (334 ページ)

SMP の効率的な使用

対称型マルチプロセッシング (SMP) を有効にするには、**smp_enable** Replication Server 設定パラメータを使用します。

SMP を効率的に使用するために必要なプロセッサ数を決定するには、2 つのプロセッサをベースとし、4 つのキューごとにプロセッサを 1 つずつ増やしていきます。プロセッサの処理速度によって、これらの数がパフォーマンス要件を満たすかどうかが決まります。並列 DSI をサポートするアウトバウンド・キューがあり、

13 個以上の DSI エグゼキュータ・スレッドがある場合は、2 つまたは 3 つのアウトバウンド・キューごとにプロセッサを 1 つ増やすなど、アウトバウンド・キューに対するプロセッサ／スレッドの比率を大きくすることがあります。

Replication Server は、サポートされるコネクションやルートの数に基づいて、常に限られた数のスレッドを使用します。すべてのスレッドが常にビジーである場合でも、Replication Server で使用できるプロセッサ数をさらに増やしてゆくと、最終的に「CPU 飽和」が発生し、それ以上プロセッサを増やしてもパフォーマンスが向上しなくなります。この場合、処理速度のより速い CPU を使用すると、CPU リソースが原因で発生するパフォーマンスの問題を解決できることがあります。

場合によっては、Replication Server が使用可能なプロセッサの数が多すぎるとパフォーマンスが低下することがわかっています。この原因は、使用可能なプロセッサ間で、強制的にスレッド・コンテキストを切り替えるのにかかる時間が問題であると考えることができます。オペレーティング・システム (OS) による Replication Server のプロセスやスレッドの管理をモニタするには、OS のモニタ・ユーティリティを使用してください。これらのユーティリティは、Replication Server が使用可能な CPU を減らすとこのようなコンテキストの切り替え数が減るかどうかを判断するのに役立ちます。

グループ内のトランザクション数の指定

さまざまな設定パラメータを使用すると、グループ内のトランザクション数を制御できます。

データベース設定パラメータ：dsi_max_xacts_in_group

dsi_max_xacts_in_group を使用してグループ内のトランザクションの最大数を指定します。

数が大きいほど、レプリケート・データベースでのコミット処理が減少し、スループットが向上する可能性がある。

グループ・サイズを制御するには、**dsi_max_xacts_in_group** を使用します。**dsi_xact_group_size** は最大値の 2,147,483,647 に設定し、その値を変更しないでください。**dsi_max_cacts_in_group** の値を 1 に減らしてグループ化なしにすると、並列トランザクション間の競合が減る場合があります。

DSI-E スレッドごとにグループ内に配置される平均トランザクション数をモニタするには、カウンタ 57001 – UnGroupedTransSched を確認します。

DSI コネクション全体に対するグループごとの平均トランザクション数をモニタするには、次のカウンタを確認します。

- 5000 – DSIReadTranGroups
- 5002 – DSIReadTransUngrouped

グループがクローズされる理由をモニタするには、次のカウンタを確認します。

- 5042 - GroupsClosedBytes
- 5043 - GroupsClosedNoneOrig
- 5044 - GroupsClosedMixedUser
- 5045 - GroupsClosedMixedMode
- 5049 - GroupsClosedTranPartRule
- 5051 - UserRuleMatchGroup
- 5053 - TimeRuleMatchGroup
- 5055 - NameRuleMatchGroup
- 5063 - GroupsClosedTrans
- 5068 - GroupsClosedLarge
- 5069 - GroupsClosedWSBSpec
- 5070 - GroupsClosedResume
- 5071 - GroupsClosedSpecial
- 5072 – OriginRuleMatchGroup
- 5074 – OSessIDRuleMatchGroup
- 5076 – IgOrigRuleMarchGroup

データベース設定パラメータ：dsi_xact_group_size、dsi_max_xacts_in_group

レプリケート・データベースに適用するために 1 つのトランザクションとしてグループ化できるトランザクション数を増やすには、これらの設定パラメータをともに使用します。

トランザクションごとの平均コマンド数が少ない (5 個以下) 場合、**dsi_xact_group_size** と **dsi_max_xact_in_group** を使用してトランザクションの処理時間を向上させることができます。

dsi_xact_group_size を最大値に設定し、**dsi_max_xact_in_group** でトランザクション・グループのサイズを制御することをおすすめします。

トランザクション・サイズの設定

単一の DSI コネクションに対しては、**dsi_large_xact_size** の値を最大値の 2,147,483,647 に設定します。並列 DSI が設定されていない場合でも、DSI/S は、**dsi_large_xact_size** によって設定された文の制限を読み取り、並列 DSI に関連する複数のタスクを実行します。

非ブロッキング・コミットの有効化

dsi_non_blocking_commit Replication Server 設定パラメータを使用して、Replication Server がコミット後にメッセージを保存している期間を延長する長さ (分単位) を指定して、非ブロッキング・コミットを有効にします。

Adaptive Server 15.0 以降で遅延コミット機能を使用できる場合、または Oracle 10g v2 で同等の遅延コミット機能を使用できる場合には、非ブロッキング・コミット機能により複写パフォーマンスが向上します。

値の範囲：0 ～ 60 分

デフォルト値は 0 – 非ブロッキング・コミットを無効にします。

メモリ消費の制御

メモリ消費量が指定スレッショルドを超えると Replication Server は警告メッセージを表示し、EXEC、DSI、および SQT スレッショルドで使用するメモリを制御できます。

メモリのスレッショルドの警告メッセージ

メモリ消費量が使用できる合計メモリの指定スレッショルド・パーセンテージを超えると、警告メッセージを表示するように Replication Server を設定します。

警告メッセージを設定するには、次のコマンドを使用します。

- **mem_warning_thr1** – この値を超えると最初の警告メッセージが生成される、合計メモリのスレッショルド・パーセンテージを指定します。
  デフォルト値は **memory_limit** 値の 80%。
  範囲：1 – 100
- **mem_warning_thr2** – この値を超えると 2 番目の警告メッセージが生成される前に使用される合計メモリのスレッショルド・パーセンテージを指定します。
  デフォルト値は **memory_limit** 値の 90%。
  範囲：1 – 100

Replication Server スレッドのメモリ制御

Replication Server スレッショルドのメモリ消費が **memory_limit** で指定された使用できるメモリを超過すると、Replication Server の自動停止を回避できます。

Replication Server でメモリを大量に必要とするスレッドは、次のとおりです。

- DSI
- EXEC
- SQT

これらのスレッドは、メモリ使用量チェックを実行してから新しいデータを受信または処理することで、メモリ制御を実行します。メモリ制御時にメモリ使用量が多いことが判明すると、次の動作によりスレッド機能が調整されます。

- スレッドによる新しいデータのグループ化を停止し、既存データのクリーニングと処理を行います。または、
- 空きメモリが確保されるまで新しいデータを受信しないよう、スレッドをスリープ・モードにします。

EXEC、DST、および SQT スレッショルドでフロー制御を管理するには、次のコマンドを使用します。

- **mem_thr_dsi** － DSI スレッドによる SQT キャッシュの入力を停止する合計メモリのパーセンテージを指定します。
  デフォルト値は **memory_limit** 値の 80%。
- **mem_thr_exec** － EXEC スレッドによる RepAgent からのコマンドの受信を停止する合計メモリのパーセンテージを指定します。
  デフォルト値は **memory_limit** 値の 90%。
- **mem_thr_sqt** － SQT スレッドでキャッシュからの最大トランザクションをフラッシュする合計メモリのパーセンテージを指定します。
  デフォルト値は **memory_limit** 値の 85%。

memory_control を使用して、スレッショルドのメモリ制御動作を管理します。**memory_control** の有効な値は、enable (デフォルト値) または disable です。これにより、Replication Server はメモリ消費を制御し、メモリの問題で停止することはなくなりました。

これらの設定パラメータのデフォルト値を変更するには、**configure replication server** を使用します。デフォルト値または既存の値を表示するには、**admin config** を使用します。

『Replication Server リファレンス・マニュアル』の「Replication Server コマンド」の「**configure replication server**」を参照してください。

スレッド情報をモニタする

admin who を使用して、スレッショルドのメモリ制御動作に関する情報を提供します。

状態	説明
Controlling Mem	スレッドはメモリ制御を実行している。
Sleeping For Mem	空きメモリが確保されるまで、スレッドはスリープする。

『Replication Server リファレンス・マニュアル』の「Replication Server コマンド」の「**admin who**」を参照してください。

メモリ管理統計

admin stats を使用して、メモリ管理統計を表示します。

メモリ・カウンタは、rsh モジュールで有効です。メモリ・カウンタをレポートするには、次のコマンドを使用します。

```
admin stats,rsh display_name instance_id
```

構文の説明は次のとおりです。

- *display_name* – カウンタ名。有効な表示名を確認するには、**rs_helpcounter** を使用します。*display_name* は必ず *module_name* と組み合わせて使用します。
- *instance_id* – SQT や SQM などのモジュールの特定のインスタンスを識別します。インスタンス ID を確認するには、**admin who** を実行し、*Info* カラムを表示します。rsh モジュールの場合、*SPID* を使用してください。*SPID* を確認するには、**admin who** を実行し、*Spid* カラムを表示します。

『Replication Server リファレンス・マニュアル』の「Replication Server コマンド」の「**admin stats**」を参照してください。

並列 DSI スレッド

単一の DSI スレッドではなく並列 DSI スレッドを使用してトランザクションがレプリケート・データ・サーバに適用されるように、データベース・コネクションを設定できます。

トランザクションを並列で適用すると、複写速度を上げて、なおかつプライマリ・サイトで発生したトランザクションのコミット順序を維持できます。

並列 DSI スレッドがアクティブな場合、Replication Server は通常、先行するトランザクションがコミットされる前に、DSI が次のトランザクションのコミット・レコードを確認*してから*、トランザクションの処理を開始します。コミットは、先行するトランザクションがすべてコミットされたことが確認されるまで延期されます。Replication Server は、トランザクションのコミット順序を維持し、次のいずれかの方法で、同時に並列して実行されているトランザクションでの更新の競合を検出できます。

- 内部的に、Replication Server の内部テーブルとファンクション文字列を使用する。
- 外部的に、レプリケート・データベースの rs_threads システム・テーブルを使用する。

Replication Server は、並列 DSI スレッドで多数のオペレーションを含むトランザクションを処理する方法によって、さらに並列処理を実現できます。ラージ・トランザクションは、DSI スレッドがコミット・レコードを確認する前に処理が開始

されます。これは、ラージ・トランザクションをより早い段階で処理できるだけでなく、ウォーム・スタンバイ・モードのときに、Replication Server が最終的にロールバックされるトランザクションの処理を開始する可能性があることも意味します。ただし、サブスクリプションの複写の場合、ロールバック・トランザクションは、DIST スレッドによって検出されます。

Replication Server で、並列処理が最大限に実行され、トランザクション間の競合が最小限に抑えられるようにする方法は、他にもあります。次に例を示します。

- トランザクションの逐次化メソッドを使用すると、システムが競合を起こすことなく処理できる並列度を選択できます。
- トランザクション・パーティショニング・ルールによって、レプリケート・データベースでの競合を避けるため、トランザクションのグループ化方法や分配方法に影響する細かなチューニングが可能です。

並列 DSI スレッドの使用の利点とリスク

ほとんどのプライマリ・データベースでは、多数のユーザやアプリケーションがトランザクションを同時に生成できます。これらすべてのトランザクションを 1 つのコネクション経由でレプリケート・データベースに送信すると、深刻なボトルネックが生じることがあります。このボトルネックによって、プライマリ・データベースとレプリケート・データベースとの間に、不要な遅延が発生する場合があります。

Replication Server で並列 DSI を有効にした場合のメリットは、複数のレプリケート・データベースで複数のトランザクションを同時に処理することによって、ボトルネックが発生する可能性を抑えられることです。

並列 DSI を有効にした場合のリスクは、複数のレプリケート・コネクションとそれらのトランザクションの間に競合が発生することです。レプリケートに対して複数のトランザクションを同時に適用すると、トランザクション間でレプリケート・リソースの競合が発生し、別のボトルネックを招く場合があります。

このため、並列 DSI スレッドを使用するには、複写環境についての十分な知識を持っていることと、何度もテストをして並列 DSI のチューニング・パラメータの中でどれが最も効果的かを決定することが必要です。目的は、レプリケートで発生する競合の量を制御しながら高いスループットを実現することです。

たとえば、複写する必要がある 1,000 個のトランザクションを含む一連の処理について考えてみます。1 つのレプリケート・コネクションで 1,000 個のトランザクションをすべて送信すると時間がかかります。しかし、各トランザクションに 1 つずつ、合計 1,000 個のコネクションを設定して使用すると、競合が発生したり、サーバ・リソースの負荷が大きくなったりします。適切に設定するには、この 2 つの方法のバランスを取る必要があります。そのためは、トランザクションのプロファイルと、並列 DSI を使用してレプリケートにそれらのトランザクションを発行する場合の影響の両方を考慮する必要があります。

2つ目の例として、プライマリで発行された2つの逐次トランザクションがそれぞれ同じテーブル・ローに対して1つの更新オペレーションを実行する場合を考えます。これらの2つのトランザクションが、レプリケートで2つのコネクションによって並列に試行される場合、テーブル・ローにアクセスするための最初のトランザクションに排他アクセスが付与されます。2番目のトランザクションは、最初のトランザクションがコミットするかロールバックしてローを解放するまで待機しなければなりません。最終的には両方のトランザクションが適用されますが、並列 DSI の設定によって得られるメリットはありません。トランザクションは、並列 DSI を使用しない場合と同じように逐次処理されます。競合が発生して並列 DSI を使用するメリットがなくなったためです。

並列 DSI のパラメータ

並行 DSI スレッド環境をカスタマイズすることができます。

並列 DSI スレッドを個々のコネクション用にチューニングするには、これらの設定パラメータを **alter connection** で使用します。

並列 DSI のコネクションを設定するには、**parallel_dsi** パラメータを **on** に設定し、個々の並列 DSI 設定パラメータ設定して環境を微調整します。

たとえば、SYDNEY_DS データ・サーバにある `pubs2` データベースへのコネクションに対して並列 DSI を有効にするには、次のコマンドを入力します。

```
alter connection to SYDNEY_DS.pubs2
 set parallel_dsi to 'on'
```

注意： 個々の並列 DSI 設定パラメータを設定するには、**configure replication server** コマンドを使用します。

並列 DSI 設定パラメータ

Replication Server は複数の並列 DSI 設定パラメータを提供します。

表 21 : 並列 DSI 設定パラメータ

パラメータ	説明
dsi_commit_check_locks_intrvl	DSI エグゼキュータ・スレッドが `rs_dsi_check_thread_lock` ファンクション文字列の実行と実行の間に待機するミリ秒 (ms) 数。 デフォルト値は 1,000 ミリ秒 (1 秒) 最小値：0 最大値：86,400,000 ミリ秒 (24 時間)

パラメータ	説明
dsi_commit_check_locks_log	警告メッセージがログに記録されるまでに、DSI エグゼキュータ・スレッドが `rs_dsi_check_thread_lock` ファンクション文字列を実行する回数。 デフォルト値は 200 最小値：1 最大値：1,000,000
dsi_commit_check_locks_max	トランザクションのロールバックとリトライが実行されるまでに、DSI エグゼキュータ・スレッドが `rs_dsi_check_thread_lock` ファンクション文字列を実行する最大回数。 デフォルト値は 400 最小値：1 最大値：1,000,000
dsi_commit_control	コミット制御について、Replication Server が内部テーブルを使用して内部的に処理するか (on)、`rs_threads` システム・テーブルを使用して外部的に処理するか (off) を指定する。 デフォルト値は on
dsi_ignore_underscore_names	**dsi_partitioning_rule** が “name” に設定されている場合に、Replication Server がアンダースコアで始まるトランザクション名を無視するかどうかを指定する。値は “on” または “off”。 デフォルト値は on

パラメータ	説明
dsi_isolation_level	トランザクションの独立性レベルを指定する。ANSI 標準および Adaptive Server でサポートされている値は、次のとおり。 • 0 — 個々のトランザクションで書き込まれたデータが実際のデータであることを保証する。 • 1 — ダーティ・リードを防止し、個々のトランザクションで書き込まれたデータが実際のデータであることを保証する。 • 2 — 繰り返し不可能読み出しとダーティ・リードを防止し、個々のトランザクションで書き込まれたデータが実際のデータであることを保証する。 • 3 — 幻ロー、繰り返し不可能読み出し、ダーティ・リードを防止し、個々のトランザクションで書き込まれたデータが実際のデータであることを保証する。 カスタム・ファンクション文字列を使用することで、Replication Server は、レプリケート・データ・サーバが使用できる任意の独立性レベルをサポートできる。サポートは ANSI 標準のみに制限されません。 デフォルト値はターゲット・データ・サーバの現在のトランザクションの独立性レベル
dsi_large_xact_size	ラージ・トランザクションと見なされるまでに 1 つのトランザクション内で許可される文の数。 デフォルト値は 100 最小値：4 最大値：2,147,483,647 (バイト)
dsi_max_xacts_in_group	グループ化できるトランザクションの最大数を指定する。大きい値を指定するほど、レプリケート・データベースでのデータ遅延時間が短縮される。 値の範囲：1 – 1000.Default:20
dsi_max_cmds_in_batch	出力コマンドのバッチ処理の対象にできるソース・コマンドの最大数を定義する。 範囲：1 – 1000 デフォルト値は 100
dsi_num_large_xact_threads	ラージ・トランザクションで使用するために予約されている並列 DSI スレッドの数。最大値は、**dsi_num_threads** の値から 1 を引いた値。 デフォルト値は 0

パラメータ	説明
dsi_num_threads	コネクションに使用する並列 DSI スレッドの数。値が 1 の場合、並列 DSI 機能が無効になる。 デフォルト値は 1 最小値：1 最大値：255
dsi_partitioning_rule	利用可能な並列 DSI スレッド間でトランザクションを分割するために DSI が使用する、1 つ以上のパーティショニング・ルールを指定する。指定できる値は **origin**、**origin_sessid**、**time**、**user**、**name**、**none**、**ignore_origin** のいずれか。 デフォルト値はなし

パラメータ	説明
dsi_serialization_method	一貫性を保ちながら、トランザクションを開始できるタイミングを決定するために使用するメソッドを指定する。どの場合でもコミット順は保持される。 これらのオプション・メソッドは並列処理量の多い順になる。並列処理が多いほど、レプリケート・データベースに適用されるときに並列トランザクション間の競合が増える可能性がある。競合を減らすには、**dsi_partition_rule** オプションを使用する。 • **no_wait** ― 他のトランザクションの状態に関係なく、トランザクションが準備でき次第すぐに開始できることを指定する。 **注意：** **dsi_commit_control** を “on” に設定している場合は、**dsi_serialization_method** は **no_wait** にのみ設定できる。 • **wait_for_start** ― 開始直前にコミットするようにスケジュールされているトランザクションが開始した直後に、トランザクションを開始できるよう指定する。 • **wait_for_start** (デフォルト) ― 直前にコミットするようスケジュールされているトランザクションの準備が終了するまでは、トランザクションを開始できないよう指定する。 • **wait_after_commit** ― 直前にコミットするようにスケジュールされているトランザクションのコミットが完了するまで、トランザクションを開始できないよう指定する。 以下のオプションは、Replication Server の以前のバージョンとの下位互換性のためにのみ維持されている。 • **none** ― **wait_for_start** と同じ。 • **single_transaction_per_origin** ― **dsi_partitioning_rule** が **origin** に設定された **wait_for_start** と同じ。 • **isolation_level_3** – **wait_for_start** と **dsi_isolation_level** が **3** に設定された場合と同じ。

パラメータ	説明
dsi_sqt_max_cache_size	アウトバウンド・キュー用の最大 SQT キャッシュ・サイズ (バイト単位)。デフォルトの 0 では、**sqt_max_cache_size** パラメータの現在の設定値が、コネクションの最大キャッシュ・サイズとして使用される。 デフォルト値は 0 32 ビット版 Replication Server の場合： • 最小値 – 0 • 最大値 – 2,147,483,647 (バイト単位) 64 ビット版 Replication Server の場合： • 最小値 – 0 • 最大値 – 2,251,799,813,685,247 (バイト単位)
parallel_dsi	並列 DSI スレッドの簡易設定。値を “on” に設定すると、**dsi_num_threads** が 5、**dsi_num_large_xact_threads** が 2、**dsi_serialization_method** が **wait_for_commit**、そして **dsi_sqt_max_cache_size** が 100 万バイト (32 ビット・プラットフォーム) および 2000 万バイト (64 ビット・プラットフォーム) になる。値を “off” にすると、並列 DSI 値はデフォルトに設定される。このパラメータを “on” に設定した後、並列 DSI の各設定パラメータを適切な値に微調整できる。 デフォルト値は off

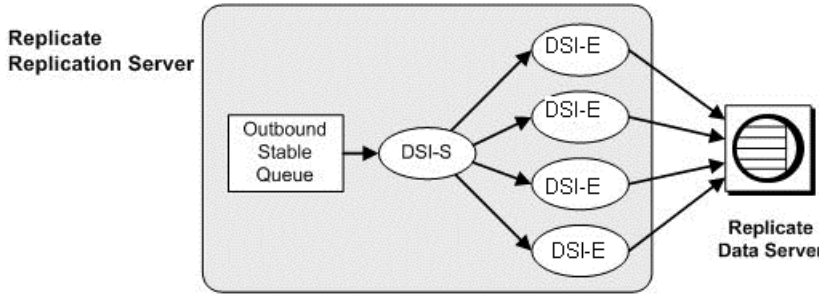
参照：

- パーティショニング・ルール：競合を減らして並列処理を増やす (217 ページ)
- SQT キャッシュのサイズ設定 (194 ページ)
- パフォーマンスを最適化するための並列 DSI の設定 (228 ページ)

並列 DSI のコンポーネント

並列 DSI のコンポーネントについて説明します。

図 13 : 並列 DSI のコンポーネント



DSI スケジューラ・スレッド

DSI スケジューラ・スレッド (DSI-S) は、スモール・トランザクションをコミット順にグループ化します。

トランザクションがグループ化されると、DSI スケジューラは、それらのグループを次に使用可能な DSI エグゼキュータ・スレッドにディスパッチします。DSI スケジューラは、異なるオリジンに対してグループを並列にディスパッチしようとしますが、それはこれらのグループを並列にコミットできるからです。異なるオリジンからのトランザクション間の競合が多すぎる場合は、**dsi_partitioning_rule** パラメータの **ignore_origin** オプションを設定します。

トランザクション・パーティショニング・ルールを使用すると、DSI スケジューラがトランザクションのグループ化に使用できる追加の条件を指定できます。

参照：

- パーティショニング・ルール：競合を減らして並列処理を増やす (217 ページ)

DSI エグゼキュータ・スレッド

DSI エグゼキュータ・スレッド (DSI-E) は、ファンクションをファンクション文字列にマッピングし、レプリケート・データベースのトランザクションを実行します。

また、DSI エグゼキュータ・スレッドは、レプリケート・データ・サーバが返すあらゆるエラーに対してアクションを実行します。

並列 DSI スレッドによるトランザクションの処理

dsi_large_xact_size データベース・コネクション設定パラメータを使用して、ラージ・トランザクションとスモール・トランザクションを定義できます。

dsi_large_xact_size は、ラージ・トランザクションと見なされるまでに 1 つのトランザクション内で許可されるコマンドの数を指定します。通常、Replication Server でのスモール・トランザクションとラージ・トランザクションの処理は、異なります。

スモール・トランザクション

Replication Server は、同じようなトランザクションをグループ化して 1 つのラージ・トランザクションとして処理しようと試みます。

この方法により、Replication Server は、個々のトランザクションをコミットせずに、グループに対して 1 つのコミットを発行できます。いくつかの条件のいずれか 1 つが満たされると、トランザクションのグループはそこで完了し、次に使用可能な DSI エグゼキュータ・スレッドに送信されます。次に例を示します。

- 次のトランザクションが、別のオリジンから発行された。
- グループ内のトランザクション数が、**dsi_max_xacts_in_group** で指定されている値を超える。
- グループ内のトランザクションの合計サイズ (バイト数) が、**dsi_xact_group_size** で指定されている値を超える。
- 次のトランザクションが、常に単独でグループ化されるラージ・トランザクションである。
- トランザクション・パーティショニング・ルールにより、次のトランザクションを既存のグループでグループ化できないと判断された。

完了したグループは、次に使用可能な DSI エグゼキュータ・スレッドに送信できます。グループに追加できるのは、コミットされたトランザクションのみです。つまり、トランザクションは、そのコミット・レコードが読み込まれるまで、トランザクション・グループに追加されません。

ラージ・トランザクション

ラージ・トランザクションは、ラージ・トランザクション用に予約されている次に使用可能な DSI エグゼキュータ・スレッドに送信されます。

DSI エグゼキュータ・スレッドは、コミット・レコードを確認するまで待機せずに、トランザクションをレプリケート・データ・サーバに送信します。トランザクションがプライマリ・データ・サーバでロールバックされた場合、DSI エグゼキュータ・スレッドはレプリケート・データ・サーバでそのトランザクションをロールバックします。

Replication Server でラージ・トランザクションが発生したときにラージ・トランザクション専用のスレッドが使用不可能であると、トランザクションはスモール・トランザクションと同じ方法で処理されます。

独立性レベルの選択

トランザクションの独立性レベルを選択することで、トランザクション中に他のユーザがデータにアクセスできる度合いを制御できます。

ANSI SQL 規格では、トランザクションの 4 つの独立性レベルを定義しています。各独立性レベルでは、同時実行トランザクションの処理中に許可されないアクションの種類が指定されます。上位レベルには、下位レベルで課された制限が含まれます。独立性レベルの詳細については、『Adaptive Server Enterprise Transact-SQL ユーザーズ・ガイド』を参照してください。

注意： Replication Server は、ANSI 標準値だけではなく、サポートされている任意のデータ・サーバを複写するために必要なすべての値をサポートしています。

- レベル 0 – コミットされていないトランザクションが修正したデータを、他のトランザクションが変更できないようにする。ただし、他のトランザクションはコミットされていないデータを読み込むことができる。その結果、ダーティ・リードとなる。
- レベル 1 – ダーティ・リードを防止する。あるトランザクションがローを修正し、その変更をコミットする前に別のトランザクションがそのローを読み込むと、ダーティ・リードが発生する。
- レベル 2 – 繰り返し不可能読み出しを防止する。あるトランザクションがローを読み込み、2 番目のトランザクションがこのローを修正する場合に発生する。2 番目のトランザクションがこの修正内容をコミットすると、最初のトランザクションによる後続の読み込みは元の読み込みと異なる結果になる。
- レベル 3 – あるトランザクションが読み込んだデータは、そのトランザクションが終了するまで有効であることを保証する。トランザクション終了までインデックス・ページやテーブル・ロックを適用することで、「繰り返し不可能読み出し」と「幻ロー」を防止する。
  トリガを使用してデータベース全体のデータの参照整合性を保つには、独立性レベル 3 を選択します。独立性レベル 3 は、トリガの実行中に、テーブルで幻ローが発生するのを防ぎます。

独立性レベルは、**create connection** または **configure connection** と **dsi_isolation_level** オプションを使用して設定できます。たとえば、SYDNEY_DS データ・サーバにある `pubs2` データベースへのコネクションの独立性レベルを 3 に変更するには、次のように入力します。

```
alter connection to SYDNEY_DS.pubs2
    set dsi_isolation_level to '3'
```

Replication Server は、*rs_isolation_level* システム変数を使用して、独立性レベルの値を **rs_set_isolation_level** ファンクション文字列に設定します。

rs_set_isolation_level は、Replication Server が、レプリケート・データ・サーバとのコネクションを確立するときに実行されます。値が設定されていない場合、Replication Server は **rs_dsi_isolation_level** を実行せず、代わりにデータ・サーバの独立性レベルを使用します。Adaptive Server の独立性レベルのデフォルトは 1 です。

独立性レベルを Sybase 以外のレプリケート・データ・サーバに合わせて設定します。

独立性レベルは、レプリケート・データ・サーバによって異なる場合があります。これは、Replication Server の設定パラメータ DSI に影響を与えます。

Sybase 以外のレプリケート・データ・サーバーの独立性レベル：

- Oracle – READ COMMITTED と SERIALIZABLE
- Microsoft SQL Server – READ UNCOMMITTED、READ COMMITTED、REPEATABLE READ、SNAPSHOT、および SERIALIZABLE
- IBM DB2 UDB – REPEATABLE READ、READ STABILITY、CURSOR STABILITY、および UNCOMMITED READ

Sybase 以外のレプリケート・データ・サーバの場合は、**rs_set_isolation_level** ファンクション文字列を編集し、*rs_isolation_level* システム定義変数を含める必要があります。**rs_set_isolation_level** 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。

Adaptive Server 以外のデータ・サーバを使用している場合は、使用しているデータ・サーバ用に *rs_isolation_level* ファンクション文字列を変更するときに、**rs_set_isolation_level** 変数を含めるようにします。

独立性ラベルを設定するには、該当するファンクション文字列クラスにファンクション文字列を作成します。次に例を示します。

- Oracle – SERIALIZABLE 独立性ラベルを設定するには：

```
create function string rs_set_isolation_level
for rs_oracle_function_class
output language
'set transaction isolation level serializable'
```

- Microsoft SQL Server – SERIALIZABLE 独立性レベルを設定するには：

```
create function string rs_set_isolation_level
for rs_msss_function_class
output language
'set transaction isolation level serializable
```

- IBM DB2 UDB – REPEATABLE READ 独立性レベルを設定するには：

```
create function string rs_set_isolation_level
for rs_udb_function_class
```

```
output language
'set current isolation = RR'
```

トランザクションの逐次化メソッド

Replication Server には、並列化のレベルを指定するための 4 つの逐次化メソッドが用意されています。

並列スレッドと複写環境間で予想される競合の度合いに応じてメソッドを選択してください。各逐次化メソッドでは、トランザクションが直前のトランザクションのコミットを待機しなければならなくなる前に、そのトランザクションをどの程度の量だけ開始できるようにするかを定義します。

逐次化メソッドによって割り当てられた並列度を下げずに競合の確率を下げるには、**dsi_partitioning_rule** パラメータを使用します。

逐次化メソッドは、次のとおりです。

- **no_wait**
- **wait_for_start**
- **wait_for_commit**
- **wait_after_commit**

alter connection コマンドと **dsi_serialization_method** パラメータを使用して、データベース・コネクションに逐次化メソッドを選択します。たとえば、次のコマンドを入力して、**wait_for_commit** 逐次化メソッドをコネクションに選択します。このコネクションは pubs2 データベースへのもので、これは SYDNEY_DS データ・サーバにあります。

```
alter connection to SYDNEY_DS.pubs2
 set dsi_serialization_method to 'wait_for_commit'
```

トランザクションは次の 3 つの部分で構成されています。

- 先頭部分。
- トランザクション本体。**insert**、**update**、**delete** などのオペレーションで構成されます。
- トランザクションの末尾部分。コミットまたはロールバックで構成されます。

逐次化メソッドは、コミットの一貫性を保ちながら、トランザクションの先頭部分が直前のトランザクションのコミット準備の完了を待機するかどうか、またはトランザクションの先頭部分がそれよりも前に処理されるようにするかどうかを定義します。

参照：

- パーティショニング・ルール：競合を減らして並列処理を増やす (217 ページ)

no_wait

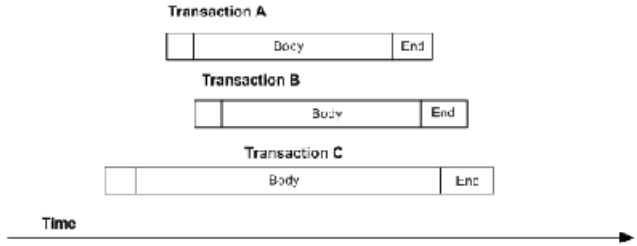
no_wait メソッドは、直前のトランザクションのコミットを待機せずに次のトランザクションを開始するよう DSI に指示します。

このメソッドは、使用するプライマリ・アプリケーションが更新の競合を回避するよう設計されているか、**dsi_partitioning_rule** を効果的に使用して競合を減少させたり、取り除いたりしていることが前提となります。Adaptive Server は、**dsi_isolation_level** が **3** に設定されなければ、更新ロックを保持しません。このメソッドは、並列トランザクション間の競合がほとんどないことを前提とし、結果的には図に示すように並列に近い形で実行されます。

no_wait を指定すると、パフォーマンスが向上する可能性が高くなりますが、競合が発生する危険性も高くなります。

注意： **dsi_commit_control** を “on” に設定している場合は、**dsi_serialization_method** は **no_wait** にのみ設定できます。

図 14 : no_wait 逐次化メソッドによるスレッドのタイミング

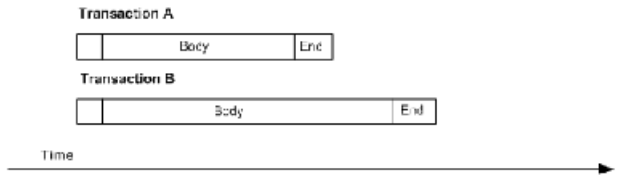


wait_for_start

wait_for_start は、開始直前にコミットするようにスケジュールされているトランザクションが開始した直後に、トランザクションを開始できるよう指定します。

dsi_serialization_method を **wait_for_start** に設定し、同時に **dsi_commit_control** を **off** に設定することがないようにしてください。

図 15 : wait_for_start 逐次化メソッドによるスレッドのタイミング



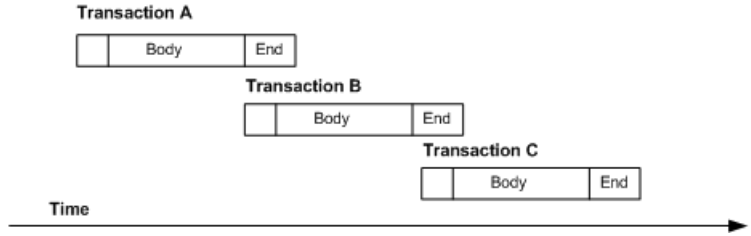
wait_for_commit

wait_for_commit メソッドでは、直前のトランザクションの処理が正常に完了してコミットの送信が開始されるまで、次のスレッドのトランザクション・グループが処理のために送信されることはありません。

デフォルトの設定です。このメソッドは、並列トランザクション間にかなりの競合があることを前提とし、図に示すように実行にずれが生じます。

このメソッドでは、1 つのトランザクションのコミットの準備ができるまで待機してから次のトランザクションを開始するように DSI に指示し、トランザクションの逐次化を維持します。最初のトランザクションは必要なロックをすでに保持しているため、最初のトランザクションのコミット中に次のトランザクションをレプリケート・データ・サーバに送信できます。

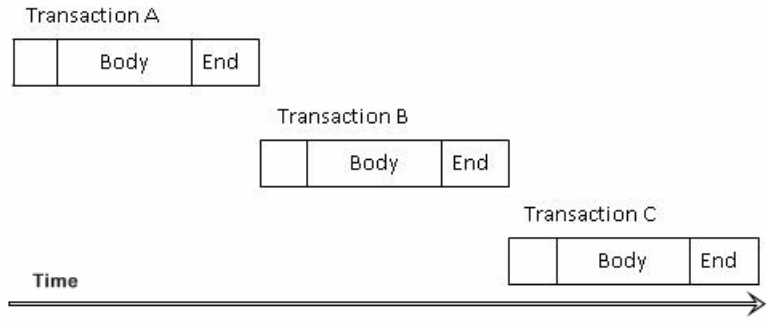
図 16 : wait_for_commit 逐次化メソッドによるスレッドのタイミング



wait_after_commit

wait_after_commit は、直前にコミットするようにスケジュールされているトランザクションのコミットが完了するまで、トランザクションを開始できないよう指定します。

図 17 : wait_after_commit 逐次化メソッドによるスレッドのタイミング



パーティショニング・ルール：競合を減らして並列処理を増やす

dsi_partitioning_rule を使用して設定されるパーティショニング・ルールによって、並列 DSI 機能で、共通名、共通ユーザ、重複する begin/commit 時刻のいずれかまたはこの組み合わせを持つトランザクションがトランザクション・グループおよび並列処理の対象であると判断することが可能になります。

パーティショニング・ルールによって、並列 DSI 機能での処理順序をプライマリとほぼ同じにできます。また、レプリケートでの競合を減らすために使用できます。

各並列 DSI パラメータを使用すると、インストール環境の条件に基づいて機能を微調整できます。**dsi_num_threads** は、コネクションで使用できる DSI スレッド数を制御します。**dsi_serialization_method** はコネクションの並列処理の量を制御しますが、並列処理の向上とレプリケートでの競合の可能性のバランスを取る必要があります。**dsi_partitioning_rule** を使用すると、並列 DSI の全体的な機能を低下させずに競合を減少させることができます。

トランザクション・パーティショニング・ルール

Replication Server では、1 つ以上の属性に従い、各コネクションに対してトランザクションを分割することができます。

各属性は次のとおりです。

- オリジン
- オリジンとセッション ID
- なし (パーティショニング・ルールが適用されない)
- ユーザ名
- オリジンの begin/commit 時刻
- トランザクション名
- オリジンの無視

注意： パフォーマンスを向上させるためにパーティショニング・ルールを使用する場合、**dsi_serialization_method** は **wait_for_commit** に指定しないでください。**wait_for_commit** は並列処理を少なくすることで競合を減らすためです。

パーティショニング・ルールを選択するには、**alter connection** コマンドを **dsi_partitioning_rule** オプションと使用します。構文は次のとおりです。

```
alter connection to data_server.database
    set dsi_partitioning_rule to '{ none|rule[, rule ] }'
```

rule の値は **user**、**time**、**origin**、**origin_sessid**、**name**、および **ignore_origin** です。ユーザ名とオリジンの begin/commit 時刻に従ってトランザクションを分割するには、次のように入力します。

```
alter connection to TOKYO_DS.pubs2
    set dsi_partitioning_rule to 'user,time'
```

パーティショニング・ルール：オリジン

origin では、同じオリジンのトランザクションが、レプリケート・データベースに適用されるときに逐次化されます。

パーティショニング・ルール：オリジンとプロセス ID

origin_sessid では、オリジンとプロセス ID が同じトランザクションが、レプリケート・データベースに適用されるときに逐次化されます。

最初は `origin_sessid,time` の設定でパーティショニング・ルールを開始することをおすすめします。

注意： Application Server のプロセス ID は、セッション・プロセス ID (SPID: Session Process ID) です。

パーティショニング・ルール：なし

none はデフォルトの動作であり、DSI スケジューラが各トランザクション・グループまたはラージ・トランザクションを次に使用可能な並列 DSI スレッドに割り当てます。

パーティショニング・ルール：ユーザ

ユーザ名によるトランザクションの分割を選択する場合、同じプライマリ・データベースのユーザ ID で入力されたトランザクションが逐次処理されます。異なるユーザ ID で入力されたトランザクションのみが並列処理されます。

このパーティショニング・ルールを使用すると競合を回避できますが、並列処理で不要なロスが発生する場合があります。たとえば、複数のバッチ・ジョブを実行している DBA を考えてみます。DBA が同じユーザ ID を使用して各バッチ・ジョブを送信する場合、Replication Server は各バッチを逐次処理します。

ユーザ名によるパーティショニング・ルールの使用が最も適しているのは、プライマリにおける各ユーザ・コネクションがユニークな ID を持つ場合です。“sa” など、同じ ID を使用して複数のユーザがログオンする場合にはあまり適していません。この場合には、**orig_sessid** がより適しています。

パーティショニング・ルール：オリジンの begin/commit 時刻

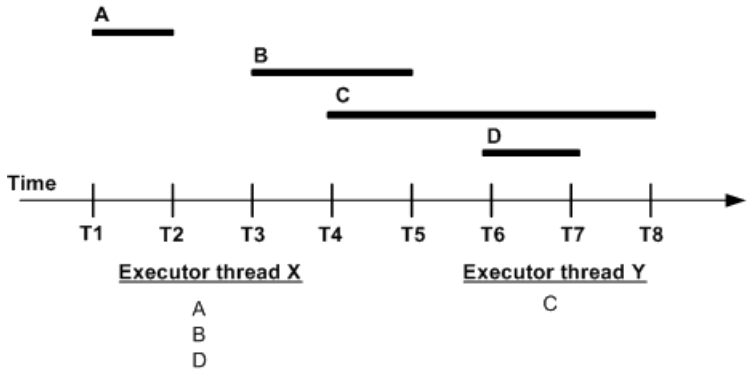
time パーティショニング・ルールが使用されている場合、DSI スケジューラはトランザクションにおけるオリジンの begin/commit 時刻を参照し、プライマリ・

データベースで同じプロセスによって実行できないトランザクションを判別します。

オリジンの begin 時刻が直前のトランザクションの commit 時刻より前であるトランザクションは、異なる DSI エグゼキュータ・スレッドで処理できます。

オリジンの begin/commit 時刻のパーティショニング・ルールが選択されており、図に示されるトランザクションと処理時刻がすべて同じプライマリ・データベースからのものと仮定します。

図 18 : トランザクションのオリジンの begin/commit 時刻



この例では、DSI スケジューラは、DSI エグゼキュータ・スレッド X にトランザクション A を割り当てます。次に DSI スケジューラは、トランザクション B の begin 時刻とトランザクション A の commit 時刻を比較します。トランザクション A がトランザクション B の開始前にコミットした場合、スケジューラはエグゼキュータ・スレッド X にトランザクション B を割り当てます。つまり、トランザクション A と B がグループ化され、同じ DSI エグゼキュータ・スレッドで処理されます。ただし、トランザクション C はトランザクション B のコミット前に開始します。このため、DSI スケジューラはトランザクション B と C がプライマリの異なるプロセスで適用されたと判断し、エグゼキュータ・スレッド Y にトランザクション C を割り当てます。トランザクション B と C は、同じグループに入れることができないので、異なる DSI エグゼキュータ・スレッドで処理されます。トランザクション D はトランザクション C のコミット前に開始するため、スケジューラは問題なくエグゼキュータ・スレッド X にトランザクション D を割り当てることができます。

注意： オリジンの begin/commit 時刻のパーティショニング・ルールを使用すると、ラージ・トランザクションがコミットの発生前にスケジュールされるため、そのラージ・トランザクションの処理時に競合が発生する場合があります。

パーティショニング・ルール：名前
DSI スケジューラは、トランザクション名を使用して、トランザクションを逐次処理するためにグループ化できます。

Adaptive Server でトランザクションを作成するときに、**begin transaction** コマンドを使用してトランザクション名を割り当てることができます。

トランザクション名のパーティショニング・ルールが適用される場合、DSI スケジューラは同じ名前のトランザクションを同じエグゼキュータ・スレッドに割り当てます。異なる名前のトランザクションは、並列で処理されます。名前が null またはブランクであるトランザクションは、**name** パラメータでは無視されます。トランザクションの処理は、他の DSI 並列処理パラメータや、他のエグゼキュータ・スレッドが使用可能かどうかによって決まります。

注意： このパーティショニング・ルールは、トランザクション名をサポートしているのであれば、Sybase 以外のデータ・サーバでも使用できます。

デフォルトのトランザクション名
デフォルトでは、Adaptive Server は常に各トランザクションに名前を割り当てます。名前が **begin transaction** を使用して明示的に割り当てられていない場合、Adaptive Server は、アンダースコア (_) 文字で始まりトランザクションを説明する文字が続く名前を割り当てます。たとえば、Adaptive Server は、1 つの **insert** コマンドにデフォルトで “_ins” という名前を割り当てます。

トランザクション名に基づいてトランザクションを分割するときに Replication Server がこれらの名前を無視するかどうかを指定するには、**alter connection** に **dsi_ignore_underscore_name** オプションを指定して使用します。デフォルトでは、**dsi_ignore_underscore_name** は **on** であり、Replication Server はアンダースコアで始まる名前のトランザクションを、名前が null であるトランザクションと同様に処理します。

パーティショニング・ルール：オリジンの無視
ignore_origin は異なるオリジンからのトランザクションに対するデフォルトの処理方法を無効にするため、すべて同じオリジンからのトランザクションであるかのようにパーティション化することができます。

ignore_origin 以外のすべてのパーティション・ルールでは、他に指定されているパーティション・ルールに関係なく、異なるオリジンからのトランザクションを並列に適用することが許可されます。

次に例を示します。

```
alter connection dataserver.db
   set dsi_partitioning_rule to "name"
```

上の例では、名前が同じであるかどうかに関係なく、オリジンの異なるトランザクションは並列に適用されます。

name パーティショニング・ルールは、同じオリジンからのトランザクションにのみ適用されます。したがって、同じオリジンからの同じ名前のトランザクションは直列に適用され、同じオリジンからの名前の違うトランザクションは並列に適用されます。

ignore_origin が **alter connection** 文の最初に記述されている場合は、2 番目またはその後のルールに基づいて同じオリジンまたは異なるオリジンからのトランザクションがパーティション化されます。次に例を示します。

```
alter connection dataserver.db
   set dsi_partitioning_rule to "ignore_origin, name"
```

上の例では、同じ名前のトランザクションはすべて直列に適用され、名前の異なるトランザクションは並列に適用されます。トランザクションのオリジンは関係ありません。

ignore_origin が **alter connection** 文の 2 番目以降に記述されている場合、Replication Server はそれを無視します。

複数のトランザクション・ルールの使用

1 つのコネクションには複数のトランザクション・ルールを設定できます。

たとえば、オリジンのセッション ID と begin/commit 時刻を適用すると、プライマリ・データベースの処理環境に最も近いものになります。

複数のトランザクション・ルールを指定すると、Replication Server は **alter connection set dsi_partitioning_rule** 構文に入力した順序でルールを適用します。

たとえば、**dsi_partitioning_rule** が “time, user” に設定されていると、Replication Server はユーザ ID をチェックする前にオリジンの begin/commit 時刻をチェックします。オリジンの begin/commit 時刻競合がない場合は、Replication Server はユーザ ID をチェックします。オリジンの begin/commit 時刻に競合がある場合は、Replication Server はユーザ ID をチェックせずに **time** ルールを適用します。したがって、後のトランザクションのオリジンの begin 時刻が commit 時刻より早い場合は、両方のトランザクションのユーザ ID が同じでも、2 つのトランザクションは異なる並列 DSI スレッドに割り当てられます。

グループ化のロジックとトランザクション・パーティショニング・ルール

パーティショニング・ルールは、グループ化やスケジューリングに関する判断に影響を与えることがあります。

パーティショニング・ルールにより、2 つのトランザクションの時刻が重複しているか (**time** ルール)、トランザクション名が異なっているか (**name** ルール)、ユー

ザが異なっている (**user** ルール) と判断された場合、その 2 つのトランザクションを同じグループに入れることはできません。それ以外の場合、トランザクションのサイズやオリジンなどに基づき、通常どおりグループ・サイズに従ってグループ化されます。

参照：

- スモール・トランザクション (211 ページ)

競合する更新の解決

並列 DSI 処理では、トランザクションのコミット順をプライマリ・データベースと同じにする必要がありますが、トランザクションの更新は同時に処理できます。その結果として発生するトランザクションの競合は、すべて解決されなければなりません。

前のトランザクションのコミットを待機しなければならないためトランザクションがコミットされず、必要なリソースが後のトランザクションによってロックされているため前のトランザクションがコミットできない場合、コミット順のデッドロックによるトランザクションの競合 (競合デッドロック) が発生することがあります。

たとえば、DSI スレッド A と B がトランザクションを並列に処理するとします。スレッド A のトランザクションは、スレッド B のトランザクションより先にコミットされなければなりません。スレッド B のトランザクションが、スレッド A に必要なリソースをロックします。スレッド B のトランザクションは、スレッド A のトランザクションがコミットされるまでコミットされず、スレッド A のトランザクションは、必要なリソースがスレッド B によってロックされているのでコミットされません。

Replication Server には、コミット順のデッドロックを解決する 2 つの方法が用意されています。

- 内部的に、Replication Server の内部テーブルとファンクション文字列を使用する。
- 外部的に、レプリケート・データベースの `rs_threads` システム・テーブルおよび複数のファンクション文字列を使用する。

内部的な解決方法では、主に Replication Server 内で処理し、`rs_dsi_check_thread_lock` ファンクション文字列を使用してコミット順のデッドロックを検出します。外部的な解決方法では、Replication Server とレプリケート・データベースを必要とし、`rs_threads` システム・テーブルを使用してコミット順の検証や、コミット順のデッドロックの検出を行います。

Sybase のデータ・サーバでも Sybase 以外のデータ・サーバでも、デフォルトの内部的な解決方法を使用することをおすすめします。この方法では、外部的な解決

方法よりも必要なネットワーク I/O が少なく、コミット順のデッドロックが発生した場合も、1 つのトランザクションをロールバックするだけで済みます。外部的な解決方法では、必要なネットワーク I/O が増え、複数のトランザクションのロールバックが必要になります。外部的な解決方法は、以前のバージョンの Replication Server との互換性を維持する場合に使用します。

Replication Server にコミット順のデッドロックが発生し、**dsi_commit_control** が on である場合、Replication Server は 1 つのトランザクションをロールバックして再試行します。ただし、Replication Server にコミット順のデッドロックが発生し、**dsi_commit_control** が off である場合は、Replication Server はすべてのトランザクションを逐次的にロールバックして再試行します。

解決方法を選択するには、**alter connection** コマンドに **dsi_commit_control** オプションを指定して入力します。たとえば、TOKYO_DS データ・サーバの `pubs2` に対して内部的な解決方法を選択するには、次のコマンドを入力します。

```
alter connection to TOKYO_DS.pubs2
set dsi_commit_control to 'on'
```

dsi_commit_control を “on” に設定すると内部的な解決方法が指定され、**dsi_commit_control** を “off” に設定すると外部的な解決方法が指定されます。

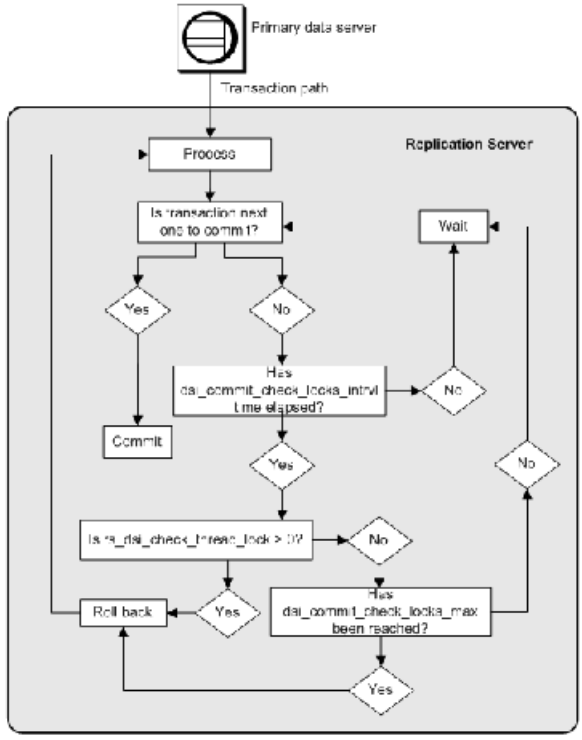
競合する更新の解決

Replication Server が **rs_dsi_check_thread_lock** ファンクション文字列を使用してどのように Replication Server のコミット順のデッドロックを解決するかを説明します。

トランザクションの整合性を維持するために、Replication Server はトランザクションのコミット順序を管理し、コミット順序の一貫性に関するデッドロックを解決する必要があります。

この図は、コミット順序のデッドロックを解決するために Replication Server が使用するロジックを示しています。

図 19 : rs_dsi_check_thread_lock ファンクション文字列を使用した競合の解決ロジック



注意： 内部的な解決方法は、Replication Server が検出するコミット順のデッドロックを解決し、Replication Server 内でのみ更新の競合を解決します。レプリケート・データベースは、デッドロックを検出すると、ロールバックするトランザクションを選択します。コミット順を維持するため、Replication Server は現在レプリケート・データベースに対して実行中のすべてのトランザクションをロールバックする必要があります。次に、Replication Server は、トランザクションを逐次的に再適用します。

コミット順の管理

Replication Server は、プライマリ・データベースから送信されたコミット情報を読み込み、この情報を使用してレプリケート・データベースでのトランザクションのコミット順を定義および管理します。

DSI エグゼキュータ・スレッドのトランザクション処理が完了して、「次に」コミットするトランザクションであることが予期されている場合、このコミットを実行できます。スレッドのトランザクション処理が完了して、そのトランザク

ションが「次に」コミットが予期されるトランザクションでない場合、スレッドはコミットの順番になるまで待機する必要があります。

コミットの一貫性のデッドロックの解決
スレッドのトランザクション処理が完了して、そのトランザクションが次にコミットが予期されるトランザクションでない場合、そのトランザクションが、先にコミットするようにスケジュールされているトランザクションに必要なリソースを保持している場合もあります。DSI エグゼキュータ・スレッドは、**dsi_commit_check_locks_intrvl** パラメータに指定された時間だけ待機すると、**rs_dsi_commit_check_thread_lock** ファンクション文字列を実行し、前のトランザクションで必要であったリソースに対するロックをスレッドが保持しているかどうかを判断します。

- スレッドが別のトランザクションをブロックしている場合 (**rs_dsi_check_thread_lock** > 0)、現在のトランザクションがロールバックし、コミット順のデッドロックが解決されて前のトランザクションをコミットできます。ブロックしているトランザクションのみがロールバックし、その他のトランザクションは通常どおりに処理を実行します。
- スレッドが別のトランザクションをブロックしていない場合、スレッドは **rs_dsi_check_thread_lock** の実行回数が **dsi_commit_check_locks_max** パラメータで定義された回数を上回っているかどうかをチェックします。
  - スレッドによる **rs_dsi_check_thread_lock** の実行回数が **dsi_commit_check_locks_max** で定義された回数を上回らなければ、次のトランザクションがある場合はそのトランザクションがコミットされるか、スレッドが **dsi_commit_check_locks_intrvl** で指定された時間だけ再度待機します。
  - スレッドによる `rs_dsi_check_thread_lock` の実行回数が **dsi_commit_check_locks_max** で定義された回数を上回る場合、現在のトランザクションがロールバックします。

内部コミット制御のファンクション文字列
Replication Server は、**rs_dsi_check_thread_lock** ファンクションを使用して、現在の DSI エグゼキュータ・スレッドが別のレプリケート・データベースのプロセスをブロックしているかどうかをチェックします。

rs_dsi_check_thread_lock は、DSI エグゼキュータ・スレッドがレプリケート・データベース・プロセスをブロックするロックを保持しているかどうかを判別する。戻り値が 0 より大きい場合、別のデータベース・プロセスに必要なリソースをスレッドが保持しており、スレッドがトランザクションをロールバックして再試行する必要があることを示す。

このファンクションは、ファンクション文字列クラス・スコープを持ちます。DSI エグゼキュータ・スレッドのコミット準備が完了しても、次にコミットされるべきスレッドではないのでコミットできず、**dsi_commit_check_locks_intrvl** に定

義された時間が経過した場合にのみ、このファンクションが呼び出されます。コミット順の競合が頻繁に発生する場合は、**dsi_commit_check_locks_intrvl** で指定されている待機時間を減らすことを検討してください。

注意： Replication Server は、Replication Server がデフォルトのファンクション文字列を生成するファンクション文字列クラスの上記のファンクションに対して、ファンクション文字列を自動的に作成します。その他のファンクション文字列クラスでは、これらのファンクション文字列を作成してから、**dsi_commit_control** を on に設定して並列 DSI 機能を使用する必要があります。

外部的に競合する更新を解決

Replication Server が `rs_threads` テーブルを外部的に使用してどのようにコミット順のデッドロックを解決するかを説明します。

`rs_threads` テーブルは、レプリケート・データベースにあります。このテーブルには、各 DSI エグゼキュータ・スレッドのローがあります。ロー・レベル・ロックをシミュレートするため、テーブルには `id` および `seq` の 2 つのカラムと、1 つのローだけがページに収まるようにするためのダミー・カラムがあります。`id` カラムは、ユニークなクラスタード・インデックスとして使用されます。

トランザクションの開始時に、DSI エグゼキュータ・スレッドは、`rs_threads` テーブル内のローを次に使用可能なシーケンス番号で更新します。トランザクションのコミット準備が完了すると、スレッドは、そのトランザクションよりも先にコミットされるべきトランザクションのシーケンス番号を `rs_threads` テーブルから選択するため、**select** 文をレプリケート・データ・サーバに送信します。

先行するトランザクションが `rs_threads` のこのローに対するロックを保持するため、先行するトランザクションがコミットされるまで、このスレッドはブロックされます。

返されたシーケンス番号が予期された値より小さい場合、スレッドはトランザクションをロールバックするかどうか、および **select** オペレーションを再試行するかどうかを決定します。DSI は多数のコマンドを 1 つのバッチにフォーマットしてから Adaptive Server に送信するため、先行するトランザクションが何らかのコマンドを Adaptive Server に送信する前に、スレッドのコミット準備が完了する場合があります。この場合、`rs_threads` テーブルの **select** が複数回送信される場合があります。

返されたシーケンス番号が予期された値と一致する場合、トランザクションをコミットできます。

デッドロックの解決策

Replication Server がどのようにデッドロックを解決するか説明します。

トランザクションのコミット準備が完了しても、そのトランザクションが正しいコミット順において次にコミットされるトランザクションでないためにコミットできず、直前にコミットする必要があるトランザクションに必要なリソースのロックを保持している場合、レプリケート・データベースでデータベース・リソースのデッドロックが発生します。

このデータベース・リソースのデッドロックは、次にコミットされるトランザクションが保持している `rs_threads` のロックと、そのトランザクションが必要とするリソースに対して保持されているロックによって起きるものです。レプリケート・データベースは、データベース・リソースのデッドロックを検出し、ロールバックするトランザクションを選択します。

Replication Server は、コミット順を維持する必要があります。そのため、レプリケート・データベースによってこのロールバックが強制実行されると、Replication Server は、レプリケート・データベースに対して実行しているすべてのトランザクションをロールバックし、コミット順に従って再び逐次適用します。

rs_threads を使用したコミット制御用のファンクション文字列

Replication Server は、次に示すシステム・ファンクションによって `rs_threads` システム・テーブルを操作します。

これらのファンクションは、ファンクション文字列クラス・スコープを持ちます。コネクションに複数の DSI スレッドが定義されている場合のみ、これらのファンクションが実行されます。

注意： これらのファンクション文字列は、外部 `rs_threads` メソッドがコミット制御に使用される場合のみ必要です。

表 22 : rs_threads システム・テーブルを変更するシステム・ファンクション

機能	説明
rs_initialize_threads	`rs_threads` システム・テーブルの各エントリのシーケンスを 0 に設定する。このファンクションは、コネクションの初期化中に実行される。
rs_update_threads	`rs_threads` システム・テーブル内の指定したエントリのシーケンス番号を更新する。
rs_get_thread_seq	`rs_threads` システム・テーブルの指定されたエントリの現在のシーケンス番号を返す。

機能	説明
rs_get_thread_seq_nohold-lock	**noholdlock** オプションを使用して、rs_threads システム・テーブルの指定されたエントリの現在のシーケンス番号を返す。このスレッドは、**dsi_isolation_level** が 3 の場合に使用される。

パフォーマンスを最適化するための並列 DSI の設定

並列処理と許容できる競合レベルとのバランスを取りながら並列 DSI の処理をチューニングして、最高の複写パフォーマンスを提供します。

競合は常に発生するものです。競合を回避する唯一の方法は、並列 DSI 処理をオフにすることです。

また、すべての並列 DSI パラメータを最大限の並列処理が実行されるように設定すると、Replication Server で、実際にトランザクションをレプリケートに適用する時間よりも競合からのリカバリ時間が長くなる場合があります。運用環境を十分に理解し、並列処理と許容できる競合レベルとのバランスを適正に保つことでパフォーマンスを最適化することができます。

パフォーマンスを最適化するための並列 DSI の設定準備

パフォーマンスをチューニングする前に考慮することがいくつかあります。

1. トランザクション・プロファイルについて理解します。

   どのような種類のトランザクションが複写されるのか。これらのトランザクションは同じローやテーブルに影響を与えるか。これらのトランザクションを並列に適用すると競合が発生しやすくなるか。日付や月などの時間によってトランザクション・プロファイルが変化するか一定であるか。トランザクション・プロファイルについて十分に理解することによって、最も有効なパラメータと設定を選択できるようになります。
2. 競合を処理するようレプリケート・データベースをチューニングします。

   大半のプライマリ・データベースは、クラスタード・インデックス、パーティショニング、ロー・レベルのロックなどを使用することにより、競合を最小限に抑えるようチューニングされています。レプリケート・データベースも同様にチューニングされていることを確認してください。
3. 使用している複写環境を正確に反映した繰り返し可能なトランザクションのセットを定義します。

   並列 DSI 環境をチューニングするには、繰り返して処理を行う必要があります。パラメータの設定、テスト、パフォーマンスの測定、前回の測定との比較を、最良の結果が得られるまで繰り返します。
4. 最初に **dsi_serialization_method** パラメータをリセットします。

注意： **dsi_commit_control** を “on” に設定している場合は、**dsi_serialization_method** は **no_wait** にのみ設定できます。

dsi_serialization_method パラメータを **no_wait** に設定し、最大限の並列処理が可能になるようにします。次に、その他のパラメータで競合が減るかどうか試します。**wait_for_commit** (デフォルト) の設定では並列処理が最小限にとどまり、メリットも最も少なくなります。そのため、**no_wait** に設定して競合を減少させるためのあらゆる手段を使ってもパフォーマンスが向上しない場合にかぎり、**dsi_serialization_method** を **wait_for_commit** に再設定してください。

5. **dsi_num_threads** パラメータを正しく設定します。

   dsi_num_threads パラメータは DSI エグゼキュータ・スレッドの総数を定義し、**dsi_num_large_xact_threads** パラメータはラージ・トランザクション用に予約された DSI エグゼキュータ・スレッドの総数を定義します。このため、DSI エグゼキュータ・スレッドの総数 (**dsi_num_threads**) は、ラージ・トランザクション用に予約された DSI スレッド数とスモール・トランザクションで使用可能なスレッド数の合計になります。

   まず **dsi_num_threads** を 5 に設定し、**dsi_num_large_xact** スレッドを 2 に設定します。その後、**dsi_serialization_method** と **dsi_partitioning_rule** を選択します。
   - 競合が増加しない場合は、**dsi_num_threads** を増やす。
   - 競合が減少しない場合は、**dsi_num_threads** を減らす。

   dsi_num_threads がデフォルトより大きくなるようにし、**dsi_num_threads** の値が **dsi_num_large_xact_threads** の値よりも大きくなるようにします。

競合を減らす

パフォーマンスを最適にするために並列 DSI の設定を準備するための作業をすべて行い、パフォーマンス・テストの結果、競合がパフォーマンスに影響を与えていることが明らかになった場合には、競合を減少させるために並列 DSI パラメータのチューニングを開始してください。

次に例を示します。

- レプリケートがアクティビティをブロックしている。
- デッドロック状態であるために Replication Server が大部分のトランザクションをロールバックして再適用している。カウンタ 5060 – TrueCheckThrdLock を参照してください。

最初に **dsi_max_xacts_in_group** パラメータをチューニングします。このパラメータは、1 つの begin/commit ブロック内でグループ化されるトランザクション数を決定します。**dsi_max_xacts_in_group** の値を小さくすると、DSI エグゼキュータ・スレッドのコミットの頻度が増えます。こうすることで、DSI エグゼキュータ・スレッドがレプリケート・リソースを保持する量も時間も少なくなるため、競合は減少します。

dsi_num_threads パラメータの調整も競合に影響を与えます。使用できる DSI エグゼキュータ・スレッド数が増えると、スレッド間で競合が発生しやすくなります。

dsi_num_threads の値は小さくし、ラージ・トランザクション用に予約されたスレッドの場合でも 3 を指定します。これを繰り返しながら、パフォーマンスを最適化する値を見つけます。全体のパフォーマンスが向上するのであれば、一部の競合は許容できます。

参照：

- パフォーマンスを最適化するための並列 DSI の設定準備 (228 ページ)

パーティショニング・ルールの使用

パーティショニング・ルールによっても競合を減少させることができますが、使用しているトランザクション・プロファイルについて十分に理解する必要があります。

トランザクション名ルール

トランザクションにトランザクション名があるか、また競合が同じ名前のトランザクションによって引き起こされているかどうかを調べます。同じ名前のトランザクションをひとつずつ個別にレプリケートに送られるようにするトランザクション名ルールを設定します。

トランザクションに名前が付けられていない場合は、名前を付けられるようにアプリケーションを変更できます。次に、**name** ルールを使用して、指定したトランザクションのみを逐次化します。DSI エグゼキュータ・スレッドが複数のトランザクションの並列処理を試みる場合に、特定のタイプのラージ・トランザクションで常に問題が発生するとします。そのような場合は、問題のトランザクションに同じ名前を付けて、**name** ルールを適用すると、問題のトランザクションを逐次処理できます。ただし、**name** ルールはすべてのトランザクションに適用されるので、同じ名前のトランザクションがすべて逐次処理されることに注意してください。

ユーザ名ルール

ユーザ名を設定すると、同じユーザ ID のトランザクションが並列処理されることで引き起こされる競合を削減できることがあります。

ユーザ名ルールはトランザクション名ルールと同様に、すべてのトランザクションに適用されるので、同じユーザ ID のトランザクションはすべて逐次処理されます。

オリジンの begin/commit 時刻ルール

時刻ルールでは、commit/begin 時刻が重複しないトランザクションが逐次実行されます。

つまり、最初のトランザクションの commit 時刻が次のトランザクションの begin 時刻より早い場合、これら 2 つのトランザクションが逐次的に実行されます。

パーティショニング・ルールの組み合わせ
ルールは組み合わせることができます。その場合、先に条件を満たすルールが優先されます。

つまり、**origin_sessid, time** ルールを指定した場合、オリジンのセッション ID が同じ 2 つのトランザクションは逐次実行され、**time** ルールは適用されません。

更新の競合が頻繁に発生する場合
トランザクション間の競合が頻繁に発生する場合、並列 DSI 設定パラメータを設定します。

並列 DSI 設定パラメータを設定します。

- **dsi_serialization_method** – このパラメータを **wait_for_commit** に設定します。
- **dsi_num_large_xact_threads** ― このパラメータは、2 に設定します。ウォーム・スタンバイ・アプリケーションで並列 DSI を設定する場合は、スタンバイ・データベースの **dsi_num_larg_xact_threads** パラメータを、アクティブ・データベースで実行される同時ラージ・トランザクションの数よりも 1 つ多い数に設定します。
- **dsi_num_threads** ― このパラメータは、**dsi_num_large_xact_threads** パラメータの値に 3 を加えた値に設定します。通常のトランザクションが 1 つまたは 2 つの文のように小さい場合、**dsi_num_threads** は **dsi_num_large_xact_threads** の値に 1 を加えた値に設定します。

parallel_dsi 設定パラメータを on に設定すると、上記の並列 DSI を設定する簡易設定になります。この場合、**dsi_sqt_max_cache_size** パラメータは 100 万バイトに設定されます。

更新の競合が頻繁には発生しない場合
トランザクション間の競合がたまにしか発生しない場合、並列 DSI 設定パラメータを次のように設定します。

並列 DSI 設定パラメータを設定します。

- **dsi_isolation_level** – 使用しているレプリケート・データ・サーバが Adaptive Server の場合は、このパラメータを独立性レベル 3 に設定します。Sybase 以外のデータ・サーバでは、**rs_set_isolation_level** カスタム・ファンクション文字列を使用して、ANSI 標準レベル 3 に相当するレベルに設定します。
  - Oracle と Microsoft SQL Server – SERIALIZABLE レベルは ANSI SQL Isolation Level 3 に相当。
  - DB2 – REPEATABLE READ レベルは ANSI SQL Isolation 3 に相当。
- **dsi_num_large_xact_threads** ― このパラメータは、2 に設定します。ウォーム・スタンバイ・アプリケーションで並列 DSI を設定する場合は、スタンバイ・

データベースの **dsi_num_larg_xact_threads** パラメータを、アクティブ・データベースで実行される同時ラージ・トランザクションの数よりも 1 つ多い数に設定します。

- **dsi_num_threads** ― このパラメータは、**dsi_num_large_xact_threads** パラメータの値に 3 を加えた値に設定します。

参照：

• 独立性レベルを Sybase 以外のレプリケート・データ・サーバに合わせて設定します。 (213 ページ)

独立レベルの設定

並列 DSI が有効であり、レプリケート・テーブルがロー・レベル・ロックに設定されている場合に、DSI 独立性レベルを使用して、トランザクションの一部の消失を防ぎます。

このような場合、トランザクション自体が適切な順序でコミットされても、トランザクション内の個々のオペレーションの順序がプライマリでの順序と一致しないことがあります。

たとえば、2 番目にコミットするトランザクションが、最初にコミットするトランザクションによって挿入されるローを更新する場合、この更新が、コミットの前に実行される可能性があります。その場合、トランザクションは正常にコミットされ、挿入は残りますが、更新は失われます。

DML オペレーションの順序不整合を回避するには、**dsi_isolation_level** を 3 に設定します。たとえば、**dsi_isolation_level** が **3** の場合、2 番目にコミットするトランザクションは、更新の対象となる、まだ存在しないローに対する範囲ロックを取得します。これにより、最初にコミットするトランザクションで、デッドロックが発生します。データ・サーバは、データベース・リソースのデッドロックを宣言します。Replication Server はすべてのオープン・トランザクションをロールバックして、それらを逐次的に再適用するため、更新は失われません。

ラージ・トランザクションのサイズの設定

dsi_large_xact_size を最大値 (2,147,483,647) などの大きな数に設定した場合、コミット・ポイントが読み取られる前にラージ・トランザクションを開始できるようにするよりも、ラージ・トランザクションを処理するオーバヘッドを削除したほうがパフォーマンスが向上する可能性があります。

並列 DSI と rs_origin_commit_time システム変数

rs_origin_commit_time システム変数の値は、並列 DSI 機能を使用しているかどうかによって異なります。

- 並列 DSI を使用せずにラージ・トランザクションを処理する場合、*rs_origin_commit_time* の値には、トランザクション・グループの最後のトランザクションがプライマリ・サイトでコミットされた時刻が指定されます。
- ラージ・トランザクションのコミットが DSI キューから読み取られる前に、並列 DSI を使用してこれらのトランザクションを処理する場合、DSI スレッドがこれらのトランザクションのいずれかの処理を開始するときに、*rs_origin_commit_time* の値は *rs_origin_begin_time* の値に設定されます。
  トランザクションのコミット文が読み込まれると、*rs_origin_commit_time* の値は実際のコミット時間に設定されます。そのため **dsi_num_large_xact_threads** がゼロより大きい値に設定されている場合、*rs_origin_commit_time* の値は **rs_commit** 以外のシステム・ファンクションにおいて、信頼できる値にはなりません。

DSI バルク・コピー・イン

Replication Server は、大量の **insert** 文をレプリケート・データベース内の同じテーブルで複写するときのパフォーマンスを向上するバルク・コピー・インをサポートします。

通常の複写では、レプリケート・データベースにデータを複写するときに、Replication Server は、SQL の **insert** コマンドを生成し、コマンドをレプリケート・データベースに送信して、レプリケート・データベースがローを処理し、オペレーションの結果を送り返してくるのを待機します。このプロセスは、1 日の終わりのバッチ処理や取引の統合などの大量のデータが複写される場合に、Replication Server のパフォーマンスに影響します。

データベースのサポート

バルク・コピー・インは、Adaptive Server データベースと ExpressConnect for Oracle によって更新される Oracle レプリケート・データベースでサポートされます。DSI のバルク・コピー・インを on にしていて、レプリケート・データベースがサポートされていない場合、DSI がシャット・ダウンされ、エラーが返されます。「Replication Server Options」の『ExpressConnect for Oracle インストールおよび設定ガイド』の「システムの稼動条件」を参照してください。

DSI バルク・コピー・イン設定パラメータ

DSI のバルク・オペレーションを制御するデータベース・コネクション・パラメータ

パラメータ	説明
dsi_bulk_copy	コネクションのバルク・コピー・イン機能を on または off にする。**dynamic_sql** と **dsi_bulk_copy** の両方が on の場合、Replication Server は必要に応じてバルク・コピー・インを適用し、Replication Server がバルク・コピー・インを使用できない場合は、動的 SQL を使用します。 デフォルト値は off
dsi_bulk_thresh-old	トランザクション内の連続する **insert** コマンドの数。この数に到達すると、バルク・コピー・インを使用するように Replication Server をトリガする。ステーブル・キュー・トランザクション (SQT) は、大量の **insert** コマンドのバッチを検出すると、バルク・コピー・インを適用するかどうかを決定するために、指定された数の **insert** コマンドをメモリに保持する。これらのコマンドはメモリに保持されるため、この値を **dsi_large_xact_size** の設定値よりも大きい値に設定しないことをおすすめする。 最小値：1 デフォルト値は 20

バルク・コピー・インの設定

バルク・コピー・イン・パラメータの値を設定するには、**alter connection** または **configure replication server** を使用します。

次のいずれかの方法を使用してください。

- **alter connection** を使用して、コネクション・レベルでバルク・コピー・イン・コネクション・パラメータを変更する。

```
alter connection to dataserver.database
set {dsi_bulk_copy | dsi_bulk_threshold} to value
```

- **configure replication server** を使用してサーバのデフォルトを変更する。

```
configure replication server
set {dsi_bulk_copy | dsi_bulk_threshold} to value
```

dsi_bulk_copy と **dsi_bulk_threshold** の値をチェックするには、**admin config** を使用します。

dsi_bulk_copy を on にすると、SQT によって、トランザクションに含まれる同じテーブルでの連続する **insert** 文の数がカウントされます。この数が **dsi_bulk_threshold** に達すると、DSI によって、以下が実行されます。

1. DSI が、**insert** でないコマンドまたは異なるレプリケート・テーブルに属するコマンドに到達するまで、データを Adaptive Server にバルク・コピーします。
2. トランザクションの残りのコマンドの実行を続行します。

Adaptive Server が、バルク・オペレーションが成功した場合はその終了時点、またはオペレーションが失敗した時点でバルク・コピー・インの結果を送信します。

注意： DSI でのバルク・コピー・インの実装により、複数文のトランザクションがサポートされるため、バルク・コピーに含まれないコマンドがトランザクションに含まれている場合でも、DSI でバルク・コピー・インを実行できます。

サブスクリプション・マテリアライゼーションに変更

バルク・コピー・インにより、サブスクリプション・マテリアライゼーションのパフォーマンスが向上します。

dsi_bulk_copy を on にすると、各トランザクションの **insert** コマンドの数が **dsi_bulk_threshold** を超えた場合に、Replication Server は、バルク・コピー・インを使用してサブスクリプションをマテリアライズします。

注意： 通常の複写で、**autocorrection** が on の場合、テーブルではバルク・オペレーションが無効になります。ただし、マテリアライゼーションでは、**dsi_bulk_threshold** に達していて、マテリアライゼーションが障害からリカバリするノンアトミック・サブスクリプションでない場合は、**autocorrection** が有効になっていても、バルク・オペレーションが適用されます。

サブスクリプション・マテリアライゼーション・メソッドの詳細については、『Replication Server 管理ガイド 第 1 巻』の「サブスクリプションの管理」を参照してください。

バルク・コピー・イン用のカウンタ

バルク・コピー・インをサポートするカウンタ。

カウンタ	説明
DSINoBulkDatatype	データに含まれているデータ型にバルク・コピー・インとの互換性がないため、スキップされたバルク・オペレーションの数。
DSINoBulkFstr	テーブルに **rs_insert** または **rs_writetext** のカスタマイズされたファンクション文字列が含まれているため、スキップされたバルク・オペレーションの数。

カウンタ	説明
DSINoBulkAutoc	テーブルで **autocorrection** が有効になっているため、スキップされたバルク・オペレーションの数。
DSIEBFBulkNext	次のコマンドがバルク・コピーであるために実行されたバッチ・フラッシュの数。
DSIEBulkSucceed	ターゲット・データベースでデータ・サーバ・インタフェース・エグゼキュータ (DSI/E: Data Server Interface executor) が **blk_done(CS_BLK_ALL)** を呼び出した回数。
DSIEBulkCancel	ターゲット・データベースで DSI/E が **blk_done(CS_BLK_CANCEL)** を呼び出した回数。
DSIEBulkRows	DSI/E がバルク・コピー・インを使用してレプリケート・データ・サーバに送信したローの数。
BulkTime	DSI/E がバルク・コピー・インを使用してレプリケート・データ・サーバにデータを送信するのにかかった時間 (ミリ秒)。

バルク・コピー・インの制限

バルク・コピー・インを使用する際には制限事項がいくつかあります。

Replication Server DSI は以下の場合バルク・コピー・インを使用しません。

- オートコレクションがオンになっており、データがサブスクリプション・マテリアライゼーションの一部ではない場合。
- **rs_insert** にユーザ定義のファンクション文字列が含まれている場合。
- `text` カラムに、出力が `none` または `rpc` の、**rs_writetext** のユーザ定義のファンクション文字列が含まれている場合。
- データ・ローに `opaque` データ型または `rs_datatype.canonic_type` の値が 255 のユーザ定義データ型 (UDD: User-Defined Datatype) が含まれている場合。

バルク・コピー・イン機能は、以下の条件下ではサポートされません。以下の場合は、バルク・コピー・インを無効にしてください。

- レプリケート・データベースがバルク・コピー・インをサポートしない場合。この場合、DSI のバルク・コピー・インを有効にすると、DSI が終了し、エラー・メッセージが返される。『Replication Server 異機種間複写ガイド』を参照。
- Replication Server とレプリケート Adaptive Server の文字セット間でデータ・サイズが変化し、データ・ローに text カラムが含まれる場合。この場合、DSI バルク・コピー・インを有効にすると、DSI が終了し、次のメッセージが返される。

```
Bulk-Lib routine 'blk_textxfer' failed.
Open Client Client-Library error: Error: 16843015,
```

```
Severity 1 -- 'blk_textxfer(): blk layer: user
error: The given buffer of xxx bytes exceeds the
total length of  the value to be transferred.'
```

- owner、tablename の長さが 255 バイトを超え、レプリケート・データベースがバージョン 15.0.3 中間リリースよりも前のバージョンである場合。DSI のバルク・コピー・インを有効にすると、Replication Server が終了し、次のメッセージが返される。

  ```
  Bulk-Lib routine 'blk_init' failed.
  ```

  owner.tablename の長さが 255 バイトを超えている場合にバルク・コピー・インを使用しないように指定するには、次の手順に従います。

  1. トレースを on にします。

     ```
     trace "on", rsfeature, ase_cr543639
     ```

  2. 以下を、Replication Server の設定ファイルに追加します。

     ```
     trace=rsfeature,ase_cr543639
     ```

その他の制限事項：

- **insert** コマンドとは異なり、バルク・コピー・インでは、タイムスタンプは生成されない。複写に timestamp カラムが含まれていない場合、timestamp カラムには NULL 値が挿入される。バルク・コピー・インを無効にするか、または timestamp カラムを含めるように複写定義を設定する。
- **writetext** ファンクション文字列を **no log** に変更した場合でも、text カラムと image カラムのログは常に記録される。
- バルク・コピーは、Adaptive Server で **insert** トリガを呼び出さない。
- 設定パラメータ **send_timestamp_to_standby** は、バルク・コピー・インに影響しない。timestamp データは常にスタンバイに複写される。

SQL 文の複写

SQL 文の複写 – Replication Server では、ログベースの複写を補完し、バッチ・ジョブによるパフォーマンスの低下に対処するための Adaptive Server 内での SQL 文の複写がサポートされています。

SQL 文の複写は、次の場合に使用することをおすすめします。

- データ操作言語 (DML: Data Manipulation Language) 文が複写されたテーブル上の多数のローに影響を及ぼす場合。
- 基本のアプリケーションを変更してストアド・プロシージャの複写を有効にすることが困難な場合。

注意： Log Transfer Manager は SQL 文の複製をサポートしません。また、Adaptive Server 以外のデータベースへの SQL 文複製もサポートされていません。

SQL 文の複写の概要

SQL 文の複写では、Replication Server は、個々のローの変更ではなく、プライマリ・データを変更した SQL 文をトランザクション・ログから受け取ります。

Replication Server は、SQL 文をレプリケート・サイトに適用します。Adaptive Server RepAgent は、SQL データ操作言語 (DML: Data Manipulation Language) と個々のローの変更の両方を送信します。設定に応じて、Replication Server が、個々のローの変更によるログの複写または SQL 文の複写のどちらかを選択します。

SQL 文の複写には、複写後に、変更されたローの数がプライマリ・データベースとレプリケート・データベースで一致していることを確認するためのロー・カウントの検証が含まれます。ローの数が一致しない場合は、Replication Server でこのエラーを処理する方法を指定できます。

SQL 文の複写を有効化して設定するには、以下を設定してください。

- SQLDML のログを記録するようにプライマリ・データベースを設定する。
- SQLDML を複写するように Replication Server を設定する。
  1. テーブルと Multi-Site Availability (MSA) の複写のための SQLDML の複写定義を作成する。
  2. Replication Server で、ウォーム・スタンバイ複写の **WS_SQLDML_REPLICATION** パラメータを on に設定する。

ログベースのレプリケーションのパフォーマンス問題

ログベースのレプリケーションがどのようにパフォーマンスに影響するか、そしてパフォーマンスの問題にどのように対処するかを説明します。

Sybase の複写テクノロジはログベースです。複写テーブルで実行された変更はデータベースのトランザクション・ログに書き込まれます。Adaptive Server は影響されるローに追加された各変更のログを記録します。単一の DMS 文が、Adaptive Server が生成する複数のログ記録になることがあります。DML 文のタイプによって、Adaptive Server は影響を受けるすべてのローについて、「事前の」イメージと「事後の」イメージを 1 つずつ記録することがあります。Sybase Replication Agent はログを読み込み、Replication Server に転送します。Replication Server は DML オペレーション (**insert**、**delete**、**update**、**insert**、**select**、またはストアド・プロシージャの実行) を識別し、各オペレーションに対応する SQL 文を生成します。

ログベースのレプリケーションにはこれらの固有の問題があります。

- 単一の DML 文が複数のローに影響を与える場合、Replication Server は複数の DML 文を、単一のオリジナルの DML だけでなく、レプリケート・サイトに適用します。たとえば、テーブル `t` がレプリケートされた場合は、次のようにします。

```
1> delete tbl where c < 4
2> go
(3 rows affected)
```

delete 文はトランザクション・ログに、ローが削除されたそれぞれのログについての 3 つの記録を行います。これらのログ・レコードは、データベース・リカバリとレプリケーションに使用されます。Replication Agent は 3 つのログ・レコードに関する情報を Replication Server に送信し、Replication Server は 3 つの **delete** 文に戻します。

```
delete t where c = 1
delete t where c = 2
delete t where c = 3
```

- Adaptive Server は、レプリケート・データベースのリソースの非対称ロードをもたらすレプリケート・サイトの最適化は実行できません。
- 複数のローに影響を与える多数のステートメントの処理は、システムの遅延時間を増加します。
- Adaptive Server は **select into** に関する情報を一部しかログしません。したがって、レプリケーション・システムは DMI コマンドを正しくレプリケートできません。

これらの問題に対処するには、2 つの異なる方法があります。

- ストアド・プロシージャの複写
- SQL 文の複写

ストアド・プロシージャの複写

ストアド・プロシージャの複写を使用して、複雑な DML オペレーションまたは多数のローに影響を与えるオペレーションをカプセル化することができます。

そのストアド・プロシージャの呼び出しだけが複写され、個々のローの変更を無視すると、ストアド・プロシージャの複写のパフォーマンスが高まります。ネットワーク・トラフィックが減少し、Replication Server がレプリケート・サイトでストアド・プロシージャに適用する処理が少なくなります。

DDL を複写するウォーム・スタンバイ設定では、**select into** オペレーションは最小限しかログされないため、複写できません。複写プロセスに固有なトランザクション管理の制限と **select into** コマンドによって、ストアド・プロシージャの複写は使用できません。

さらに、サード・パーティ・アプリケーションによっては、ストアド・プロシージャの複写をサポートするために変更することが困難なものもあります。その結果、ストアド・プロシージャの複写が Replication Server のパフォーマンスを改善しても、すべての状況で使用できるとは限りません。

Replication Server トポロジが SQL 文の複製に与える影響

SQL 文の複製を使用する場合、基本となる Replication Server トポロジを考慮する必要があります。

Replication Server は複数の Replication Server、ウォーム・スタンバイ設定、Multi-Site Availability (MSA) 設定を含む「基本的プライマリ・コピー」モデルをはじめ、広範囲におよぶトポロジをサポートしています。

従来の複写と同様に SQL 文の複写はログ・ベースです。SQL 文を複写するために必要な情報 (プライマリ・データで実行される) はトランザクション・ログに格納されます。Log Reader、Sybase Replication Agent、その他のアプリケーションについては、トランザクション・ログを読み込み、変更されたレプリケート・テーブルについて Replication Server に通知してください。

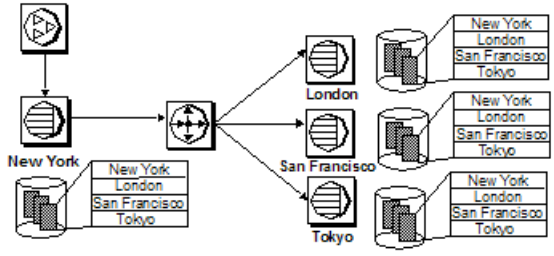
簡単な MSA やウォーム・スタンバイ設定では、送信元と送信先データは同じで、プライマリ・テーブルで実行される DML 文はレプリケート・テーブルの同じデータ・セットに影響を与えます。

注意： SQL 文の複製は DML 文にのみ適用されます。

レプリケート・サイト内の同じデータ

この図は Replication Server トポロジとニューヨーク内の単一のプライマリ・データベースを示します。テーブルは 他の 3 つのサイトに複写されます。ロンドン、東京、およびサンフランシスコ。すべてのテーブルは完全に複写されます。

図 20 : 基本プライマリ・コピー・モデル：レプリケート・サイト内の同じデータ



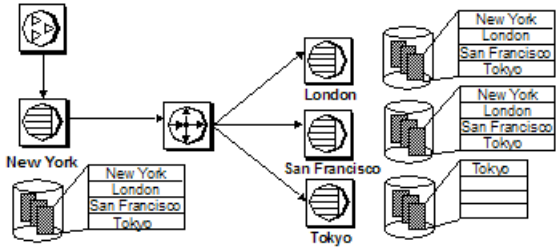
ニューヨーク・サイトに接続しているクライアントによって実行された次の文を考えてみます。

```
delete t1 where a>5
```

このコマンドが東京、ロンドン、サンフランシスコで実行される場合、データはすべてのサイトで同じなので、同じデータ・セットがすべてのレプリケート・サイトに影響を与えます。この場合、すべての複写済みサイトは SQL 文の複写を使用するように設定できます。

レプリケート・サイト内の異なるデータ
この図は複製されたサイト東京がデータのサブネットのみをサブスクライブし、`site` が「東京」と等しい場合を示しています。

図 21 : 基本プライマリ・コピー・モデル：レプリケート・サイト内の異なるデータ



ニューヨーク・サイトで実行された次の文を考えてみます。

```
delete t1 where a>5
```

Replication Servers はロンドンとサンフランシスコで同じ文を実行できますが、東京ではできません。これはこのサイトがデータのサブネットのみをサブスクライブしているためです。SQL 文の複写がこのケースで使用されると、東京サイトのように複写されたデータベースは、従来の複写に基づいてプライマリ・トランザクション・ログから個々のログ・レコード変更を受け取ります。ロンドン・サイトのような他のサイトは、SQL 文を受け取ります。

プライマリ・テーブルとレプリケート・テーブルの異なるデータ・セットもまた、プライマリ・データベースやレプリケート・データベースが異なるオブジェクト・スキーマを持つ場合、あるいはユーザが別のテーブルで **join** のコマンドを使用して DML 文を実行する場合に影響を受けます。この状況では、異なるデータはプライマリとレプリケートで影響を受けます。**join** に使用されたテーブルは、レプリケーションにマークされていなかったり、そのテーブル内の値が部分的であったり、プライマリ・データベースからのものとは異なることが考えられます。

SQL 文をプライマリ・データを格納する Adaptive Server 内と Replication Server 内でアクティベートすることが必要です。プライマリの Adaptive Server で SQL 文の複製をいったん有効にすると、Adaptive Server は、SQL 文の複製がアクティベート

された DMS 文が実行されたトランザクション・ログに追加情報を記録します。Replication Agent または他の Log Reader は、個々のログ・レコード変更と SQL 文の複製の情報を Replication Server に送ります。

注意： Sybase Replication Agent は、Replication Server 15.2 またはそれ以降の SQL 文の複製情報を送ります。

文がレプリケート・サイトに適用されると異なるデータ・セットに影響する可能性がある場合、Adaptive Server は SQL 文の複製を許可しません。

SQL 文の複写の有効化

SQL 文の複製はデータベース、テーブルまたはセッション・レベルで有効にできます。セッション設定により、テーブル設定とータベース設定の両方が上書きされます。テーブル設定により、データベース設定が上書きされます。

いくつかの Adaptive Server ストアド・プロシージャで、SQL 文の複写がサポートされます。

SQL 文の複写をデータベース・レベルで設定する

sp_setrepdbmode を使用することで、特定の DML オペレーションについて SQL 文の複写をデータベース・レベルで有効にできます。

SQL 文の複写に該当する DML オペレーションには、次のものがあります。

- **U – update**
- **D – delete**
- **I – insert select**
- **S – select into**

たとえば、SQL 文として **delete** 文を複写し、**select into** の複写も有効にするには、次のように入力します。

```
sp_setrepdbmode pdb, 'DS', 'on'
```

ユーザが **delete** をデータベース `pdb` のテーブルで実行すると、Adaptive Server は SQL 文の複製に関する追加情報をログに記録します。RepAgent は個々のログ・レコードと Replication Server が必要とする情報の両方を送信し、SQL 文を構築します。SQL 文の複製をデータベース・レベルで実行できるのは、**sp_setreptostandby** が **ALL** または **L1** に設定することで、データベースが複製用にマークされている場合だけです。

threshold パラメータは、DML 文が影響を与えるローの最小数を定義して SQL 分の複写をアクティベートします。デフォルトのスレッショルドは 50 ローです。つまり、DML 文が少なくとも 51 ローに影響を与えると、Adaptive Server は 自動的に SQL 文の複写を使用します。

たとえば、データ操作言語 (DML) 文が 100 を超えるローに影響を及ぼすときには SQL 文の複写をトリガするようスレッショルドをデータベース・レベルで設定します。

```
sp_setrepdbmode pubs2, 'threshold', '100'
go
```

『Replication Server リファレンス・マニュアル』の「Adaptive Server コマンドとシステム・プロシージャ」の「**sp_setrepdbmode**」を参照してください。

参照：

- SQL 文の複写スレッショルドの設定 (245 ページ)

SQL 文レプリケーション・ステータスの表示

sp_reptostandby を使用して、SQL 文の複写ステータスをデータベース・レベルで表示します。

次に例を示します。

```
sp_reptostandby pdb
go
The replication status for database 'pdb' is 'ALL'.
The replication mode for database 'pdb' is 'off'.
```

SQL 文の複写をテーブル・レベルで有効にする

SQL 文の複写をテーブル・レベルで設定するには、**sp_setrepdefmode** を使用します。テーブル・レベルの設定により、データベース・レベルの設定が上書きされます。

sp_setrepdefmode には、以下のためのオプションがあります。

- 特定の DML オペレーションについての SQL 文の複写の有効化または無効化
- スレッショルドの設定。SQL 文の複写をアクティブにするには、このスレッショルドに達する必要がある。

SQL 文の複写に該当する DML オペレーションには、次のものがあります。

- **U – update**
- **D – delete**
- **I – insert select**

たとえば、テーブル `t` での **update**、**delete**、および **insert select** オペレーションに対して SQL 文の複製を有効にするには、以下を使用します。

```
sp_setrepdefmode t, 'UDI', 'on'
go
```

ユーザが **deletes**、**updates**、または **insert select** DML 文をテーブル `t` で実行すると、Adaptive Server は SQL 文の複製に対する追加情報をログに記録します。

RepAgent は、個々のログ・レコードと Replication Server が SQL 文を作成するために必要な情報の両方を読み込み、ログに記録します。

threshold パラメータは、DML 文が影響を与えるローの最小数を定義して SQL 分の複写をアクティベートします。デフォルトのスレッショルドは 50 ローです。つまり、DML 文が少なくとも 51 ローに影響を与えると、Adaptive Server は 自動的に SQL 文の複写を使用します。

たとえば、スレッショルド値を 100 に設定するには、次のように入力します。

```
sp_setreptable t, true
go
sp_setrepdefmode t, 'UD', 'on'
go
sp_setrepdefmode t, 'threshold','100'
go
```

この例では、テーブル `t` によって実行される **update** および **delete** 文は、101 ロー以上の文が影響される場合は、SQL 文の複写を使用します。

『Replication Server リファレンス・マニュアル』の「Adaptive Server コマンドとシステム・プロシージャ」の「**sp_setrepdefmode**」を参照してください。

注意： **select into** オペレーションはテーブル・レベルで設定できません。これはターゲット・テーブルが存在しないためです。

参照：

- SQL 文の複写スレッショルドの設定 (245 ページ)

SQL 文の複写をセッション・レベルで有効にする

セッション中、指定された DML オペレーションについて SQL 文の複写を設定するには、**set repmode** を使用します。セッションでの設定はデータベース・レベルおよびオブジェクト・レベルの設定を上書きします。

セッション・レベルの設定は、ログイン時に login trigger を使用して設定するか、バッチの先頭で設定します。

たとえば、セッション中に **select into** と **delete** のみを SQL 文として複製するには、以下を使用します。

```
set repmode on 'DS'
```

セッション・レベルで、すべての SQL 文の複写設定を解除するには、**set repmode off** を使用します。

set オプションは、セッション中にのみ有効です。ストアド・プロシージャ内で設定したオプションは、ストアド・プロシージャが終了するとデフォルト値に戻ります。

注意： ログイン・トリガ内で設定したオプションは、トリガが実行を終えても維持されます。

set repmode on を実行すると SQL 文の複製が有効になるのは、セッション・レベルのオプション **set replication on** が設定されている場合だけです。この例では、SQL 文の複写は有効になりません。

```
set replication off
go
set repmode on 'S'
go
```

この例では、SQL 文の複写は有効になります。

```
sp_reptostandby pdb, 'ALL'
go
set repmode on 'S'
go
```

threshold パラメータは、DML 文が影響を与えるローの最小数を定義して SQL 分の複写をアクティベートします。デフォルトのスレッショルドは 50 ローです。つまり、DML 文が少なくとも 51 ローに影響を与えると、Adaptive Server は 自動的に SQL 文の複写を使用します。

次の例は、セッション・レベルで 1000 ローとしてスレッショルドを定義する方法を示します。

```
set repmode 'threshold', '1000'
go
```

『Replication Server リファレンス・マニュアル』の「Adaptive Server コマンドとシステム・プロシージャ」の「**set repmode**」を参照してください。

参照：

- SQL 文の複写スレッショルドの設定 (245 ページ)

SQL 文の複写スレッショルドの設定

個々のテーブルにスレッショルドを設定しなくても、SQL 文の複写をトリガできます。

スレッショルドを次のレベルで設定できます。

- データベース・レベル - Adaptive Server 15.0.3 ESD #1 以降
- セッション・レベル - Adaptive Server 15.0.3 ESD #2 以降

Adaptive Server 15.0.3 では、スレッショルドはテーブル・レベル以外では設定できません。

デフォルトでは、SQL 文の複写がトリガされるのは、SQL 文が 50 を超えるローに影響を及ぼす場合です。セッション、データベース、およびテーブルの各レベルで、さまざまなスレッショルド値を設定できます。

ただし、セッション・レベルで設定したスレッショルドは、テーブル・レベルとデータベース・レベルのスレッショルドよりも優先されます。テーブル・レベルで設定したスレッショルドは、データベース・レベルで設定したスレッショルドよりも優先されます。

データベース・レベルでのスレッショルドとオペレーションの設定

threshold パラメータを **sp_setrepdbmode** コマンドに対して使用して、スレッショルドをデータベース・レベルで設定します。

以降の例で、スレッショルドをデータベース・レベルとテーブル・レベルで設定すると同時に、オペレーションをさまざまなレベルで定義する方法を示します。

例 1

次の例では、pubs2 データベースと table1 テーブルについて異なるスレッショルドをデータベース・レベルとテーブル・レベルで設定する方法を示します。

1. データベース・レベルでのスレッショルドをデフォルト値の 50 ローにリセットします。
   ```
   sp_setrepdbmode pubs2, 'threshold', '0'
   go
   ```
2. **update**、**delete**、**insert**、および **select into** の各オペレーションの SQL 文の複写を pubs2 に対して有効にします。
   ```
   sp_setrepdbmode pubs2, 'udis', 'on'
   go
   ```
3. 手順 2 で定義されたオペレーションが 1,000 を超えるローに影響を及ぼすときは、これらのオペレーションに対してのみ、table1 (pubs2) の SQL 文の複写をトリガします。
   ```
   sp_setrepdefmode table1, 'threshold', '1000'
   go
   ```

例 2

次の例では、pubs2 のスレッショルドをデータベース・レベルで定義すると同時に、table1 や table2 など、pubs2 データベースにあるテーブルに対してさまざまなオペレーションを定義する方法を示します。

1. データ操作言語 (DML) 文が 100 を超えるローに影響を及ぼすときには SQL 文の複写をトリガするようスレッショルドをデータベース・レベルで設定します。
   ```
   sp_setrepdbmode pubs2, 'threshold', '100'
   go
   ```

2. SQL 文の複写を使用してオペレーションを複写させる 2 つの特定のテーブルについて、別のオペレーション・セットを定義します。table1 には更新、削除、および挿入オペレーション、table2 には削除オペレーションです。

```
sp_setrepdefmode table1, 'udi', 'on'
go
sp_setrepdefmode table2, 'd' 'on'
go
```

delete オペレーションが table2 に対して実行されるかまたは任意の DML が table1 上で実行される場合、データベース・レベルで定義されたスレッショルド 100 ローに達すると、SQL 文の複写がトリガされます。

セッション・レベルでのスレッショルドとオペレーションの設定

スレッショルドをセッション・レベルで設定するには、**set repthreshold** を使用します。

セッション・レベルで定義されたスレッショルドは、テーブル・レベルまたはデータベース・レベルで設定されたスレッショルドをオーバーライドします。テーブル・レベルで設定されたスレッショルドは、データベース・レベルで設定されたスレッショルドをオーバーライドします。

以降の例で、スレッショルドをセッション・レベル、データベース・レベル、およびテーブル・レベルで設定すると同時に、オペレーションをさまざまなレベルで定義する方法を示します。

例 1

次の例では、データベース・レベルとテーブル・レベルのスレッショルド設定がない状態で、スレッショルドをセッション・レベルで 23 に定義するか、またはデータベース・レベルとテーブル・レベルでのスレッショルド設定をオーバーライドする方法を示します。

```
set repthreshold 23
go
```

例 2

次の例は、セッション・レベルでスレッショルドをデフォルトの 50 にリセットする方法を示します。

```
set repthreshold 0
go
```

例 3

次の例では、pubs2 データベースと table1 テーブルについて、別のスレッショルドをデータベース・レベルとテーブル・レベルで設定してから、このセッション限定で定義された別のオペレーションを実行する方法を示します。

1. データベース・レベルでのスレッショルドをデフォルト値の 50 ローにリセットします。

```
sp_setrepdbmod pubs2, 'threshold', '0'
go
```

2. **update**、**delete**、**insert**、および **select into** の各オペレーションの SQL 文の複写を pubs2 に対して有効にします。

```
sp_setrepdbmode pubs2, 'udis', 'on'
go
```

3. DML オペレーションが 1,000 を超えるローに影響を及ぼすときだけ、table1 (pubs2) の SQL 文の複写をトリガします。

```
sp_setrepdefmode table1, 'threshold', '1000'
go
```

4. **update** オペレーション だけの SQL 文の複写を任意のテーブルについてこのセッション限定で有効にします。これにより、手順 2 のデータベース・レベルの設定がオーバーライドされます。

```
set repmode on 'u'
go
```

例 4

Adaptive Server ストアド・プロシージャ では **set reptheshold** を呼び出すことができます。次の例では、**set_rep_threshold_23** ストアド・プロシージャを作成し、**my_proc** ストアド・プロシージャ内で呼び出す方法を示します。

1. **set_rep_threshold_23** ストアド・プロシージャを作成します。

```
create procedure set_rep_threshold_23
as
set repthreshold 23
update my_table set my_col = 2  (statement 2)
go
```

2. **my_proc** ストアド・プロシージャを作成します。

```
create procedure my_proc
as
update my_table set my_col = 1   (statement 1)
exec set_rep_threshold_23
update my_table set my_col = 3   (statement 3)
go
```

3. **my_proc** を実行し、**set_repthreshold_23** を呼び出します。

```
exec my_proc
go
```

my_proc ストアド・プロシージャ内で、最初に文 1 がスレッショルド 50 で実行されます。次に文 2 がスレッショルド 23 で実行されます。次に文 3 がスレッショルド 50 で実行されます。これは、**set repthreshold 23** コマンドが有効であるのが **set_rep_threshold_23** プロシージャの実行中だけだからです。

例 5

セッション・レベルの複写スレッショルドはエクスポート可能です。このため、プロシージャについて **export_options** 設定を 'on' に設定し、外部スコープのプロシージャがストアド・プロシージャにより設定された SQL 文の複写スレッショルドを設定するように、SQL 文の複写スレッショルドを設定します。

1. **set_repthreshold_23** ストアド・プロシージャを作成し、**export_options** を 'on' に設定します。

```
create procedure set_repthreshold_23
as
set repthreshold 23             (statement 4)
set export_options on
update my_table set my_col = 2  (statement 2)
go
```

2. **my_proc** ストアド・プロシージャを作成します。

```
create procedure my_proc
as
update my_table set my_col = 1   (statement 1)
exec set_rep_threshold_23
update my_table set my_col = 3   (statement 3)
go
```

3. **my_proc** を実行し、**set_repthreshold_23** を呼び出します。

```
exec my_proc
go
```

最初に文 1 がスレッショルド 50 で実行されます。次に文 2 がスレッショルド 23 で実行されます。次に文 3 がスレッショルド 23 で実行されます。**set repthreshold 23** コマンドのスコープがセッションのスコープであるためです。

例 6

ログイン・トリガを作成し、特定のログイン ID の複写スレッショルドを自動的に設定できます。

1. **threshold** ストアド・プロシージャをスレッショルド設定 23 で作成し、エクスポートを有効にします。

```
create proc threshold
as
set repthreshold 23
set export_options on
go
```

2. ユーザー "Bob" がログインしたときに **threshold** ストアド・プロシージャを自動的に実行するように Adaptive Server で指定します。

```
sp_modifylogin Bob, 'login script', threshold
go
```

Bob が Adaptive Server にログインすると、セッションの SQL 文の複写スレッショルドが 23 に設定されます。

スレッショルドの設定と複写の設定

複写するよう設定されていないデータベースを用意すると同時に、SQL 文の複写のデータベース・レベルでのスレッショルドを設定することができます。

次に例を示します。

```
sp_reptostandby pubs2, 'none'
go
sp_setrepdbmode pubs2, 'threshold', '23'
go
```

ただし、データベース・レベルでのオペレーションを定義するには、複写をデータベース・レベルでも設定してください。たとえば、次は実行できません。

```
sp_reptostandby pubs2, 'none'
go
sp_setrepdbmode pubs2, 'udis', 'on'
go
```

SQL 文の複写の複写定義の設定

複写定義について、SQL 文の複写をデータベース・レベルとテーブル・レベルで変更できます。

データベース複写定義

MSA 環境で SQL 文を複写するには、**replicate SQLDML** 句を **create database replication definition** コマンドまたは **alter database replication definition** コマンドで含めます。

create database replication definition または **alter database replication definition** の構文には、以下の句を含めてください。

```
[[not] replicate setname [in (table list)] ]
```

構文の説明は次のとおりです。

setname = **DDL** | **tables** | **functions** | **transactions** | **system procedures** | **SQLDML** | **'options'**

'options パラメータは、以下の組み合わせです。

- **U** – update
- **D** – delete
- **I** – insert select
- **S** – select into

SQLDML パラメータも、**U**、**D**、**I**、**S** 文の組み合わせとして定義されます。

次の例は、**'options'** パラメータを使用して、テーブル `tb1` と `tb2` で SQLDML を複写する方法を示しています。

```
replicate 'UDIS' in (tb1,tb2)
```

次の例は、**SQLDML** パラメータを使用して、前の例の **'options'** パラメータを使用した場合と同じ結果をもたらす方法を示しています。

```
replicate SQLDML in (tb1,tb2)
```

create database replication 定義では、複数の replicate 句を使用できます。しかし、**alter database replication** 定義では、1 つの句しか使用できません。

複写定義でフィルタを指定しない場合、デフォルトは **not replicate** 句です。SQLDML フィルタを変更するには、**alter database replication definition** を適用します。**replicate** 句では、1 つまたは複数の SQLDML フィルタを指定できます。

次の例は、すべてのテーブルで **select into** 文をフィルタする方法を示しています。2 つ目の句 **not replicate 'U' in (T)** は、テーブル `T` での update をフィルタします。

```
create database replication definition dbrepdef
       with primary at ds1.pdb1
       not replicate 'S'
       not replicate 'U' in (T)
go
```

次の例では、replicate 'UD' 句を使用して、すべてのテーブルでの **update** 文と **delete** 文を有効にします。

```
create database replication definition dbrepdef_UD
       with primary at ds2.pdb1
       replicate 'UD'
go
```

複数の句を使用して、同じ定義で 1 つのテーブルを複数回指定できます。ただし、**U**、**D**、**I**、**S** はそれぞれ、定義ごとに一度しか使用できません。

```
create database replication definition dbrepdef
       with primary at ds2.pdb1
       replicate tables in (tb1,tb2)
       replicate 'U' in (tb1)
       replicate 'I' in (tb1,tb2)
go
```

次の例では、テーブル `tb1` と `tb2` での **update** 文と **delete** 文が該当します。

```
alter database replication definition dbrepdef
      with primary at ds1.pdb1
      replicate 'UD' in (tb1,tb2)
go
```

テーブル複写定義

SQL 文の複写をサポートするには、テーブル複写定義作成時に **replicate SQLDML** 句を含めます。

create replication definition の構文には、以下の句を含めてください。

```
[replicate {SQLDML [‘off’] | ‘options’}]
```

‘options’ パラメータは、以下の文の組み合わせです。

- **U** – update
- **D** – delete
- **I** – insert select

注意： 複写定義に **[replicate {minimal | all} columns]** 句が含まれる場合、**[replicate {minimal | all} columns]** 句を常に **[replicate {SQLDML [‘off’] | ‘options’}]** 句の前に置いてください。

以下は、テーブルのサンプルの **create replication definition** です。

```
create replication definition repdef1
       with primary at ds3.pdb1
       with all tables named 'tb1'

           (id_col int,
           str_col char(40))

       primary key (id_col)
       replicate all columns
       replicate ‘UD’
go
```

send standby 句を含むテーブル複写定義を使用して、**replicate 'I'** 文を指定できます。**insert select** 文を SQL 文の複写として複写できるのは、ウォーム・スタンバイまたは MSA 環境のみです。**send standby** 句が含まれないテーブル複写定義では、**insert select** 文を複写できません。

ウォーム・スタンバイと SQL 文の複写

SQL 文の複写向けにウォーム・スタンバイ・アプリケーションを設定する方法を学びます。

デフォルトでは、ウォーム・スタンバイ・アプリケーションは、SQL 文の複写をサポートする DML コマンドを複写しません。SQL の複写を使用するには、以下を行います。

- **replicate SQLDML** 句と **send standby** 句を使用して、テーブル複写定義を作成する。
- **WS_SQLDML_REPLICATION** パラメータを on に設定する。デフォルト値は **UDIS**。

ただし、**WS_SQLDML_REPLICATION** の優先度は SQL の複写のテーブル複写定義よりも低い。テーブル複写定義にテーブルの **send standby** 句が含まれている場合、その句によって、DML 文を複写するかどうかが決定される。**WS_SQLDML_REPLICATION** パラメータの設定とは無関係。

SQL 文の複写に対するロー・カウントの検証

SQL 文の複写中に発生する可能性のある SQLDML のロー・カウント・エラーに Replication Server が対応する方法を指定できます。

SQLDML のロー・カウント・エラーは、SQL 文の複写後に、変更されたローの数がプライマリ・データベースとレプリケート・データベースで一致しない場合に発生します。デフォルトのエラー・アクションは、複写の停止です。

SQLDML のロー・カウント・エラーのその他のエラー・アクションを指定するには、Replication Server のエラー・クラスのプライマリ・サイトで **assign action** コマンドを使用します。

```
assign action
   {ignore | warn | retry_log | log | retry_stop | stop_replication}
   for error_class
   to server_error1 [, server_error2]...
```

構文の説明は次のとおりです。

- *error_class* は、アクションを割り当てるエラー・クラスの名前です。デフォルトの **rs_repserver_error_class** エラー・クラスなどの Replication Server のエラー・クラスを指定できます。
- *server_error* はエラー番号です。次の Replication Server のエラー番号を指定できます。

たとえば、Replication Server でエラー番号 5186 が発生したときの **warn** エラー・アクションを割り当てるには、次のように入力します。

```
assign action warn for rs_repserver_error_class to 5186
```

次に示すのは、ロー・カウント・エラーが発生した場合に生成されるエラー・メッセージの例です。

```
DSI_SQLDML_ROW_COUNT_INVALID 5186
Row count mismatch for SQLDML command executed on
'mydataserver.mydatabase'.
The command impacted 1000 rows but it should impact 1500 rows.
```

参照：

- データ・サーバのエラー処理 (360 ページ)

SQL 文複写に対するエラー・アクション

Replication Server では、SQL 文の複写エラーに対するエラー・アクションがいくつかサポートされています。

表 23 : SQL 文複写に対するエラー・アクション

server_error	エラー・メッセージ	デフォルトのエラー・アクション	説明
5186	`Row count mismatch for the command executed on 'dataserver.database'. The command impacted x rows but it should impact y rows.`	**stop_replication**	影響を受けたローの数が想定された数と異なる場合の SQL 文の複写におけるロー・カウントの検証エラー。
5193	`You cannot enable autocorrection if SQL Statement Replication is enabled. Either enable SQL Statement Replication only or disable SQL StatementReplication before you enable autocorrection.`	**stop_replication**	SQL 文の複写が有効になっている場合、オートコレクションを有効にできない。オートコレクションを有効にする前に、SQL 文の複写を有効にするか、SQL 文の複写を無効にする。

SQL 文の複写のスコープ

SQL 文の複写が、バッチ処理での DML 文、トリガ、およびストアド・プロシージャに適用される方法について学びます。

バッチ処理

SQL 文の複写をバッチ実行される DML 文に適用するには、要件をいくつか満たさなければなりません。

- 設定で SQL 文の複写が許可されていなければならない。
- DML 文が、SQL 文の複写の使用の条件および例外に該当してはならない。

以下の例では、**delete** と **insert** を付けてバッチ文を実行する間、最初の文だけが SQL 文の複写を使用します。`table2` では従来の複写を使用しますが、これは `table2` が SQL 文の複写を使用するよう設定されていないからです。

```
create table table1 (c int, d char(5))
go
create table table2 (c int, d char(5))
go
insert table1 values (1, 'ABCDE')
go 100
sp_setreptable table1, true
go
sp_setreptable table2, true
go
sp_setrepdefmode table1, 'UDI', 'on'
go
delete table1 where c=1
insert table2 select * from table1
go
```

参照：

- SQL 文の複写使用の例外 (258 ページ)

ストアド・プロシージャ

ストアド・プロシージャ内の DML 文が文として複写されるかどうかは、ストアド・プロシージャの複写ステータスで決まります。

- ストアド・プロシージャが、複写するようマーク付けされていない場合、その内の DML 文が文として複写されるのは、以下の条件が満たされる場合のみです。
  - 設定で SQL 文の複写が許可されている。
  - DML 文が、SQL 文の複写の使用の条件および例外に該当してはならない。
- ストアド・プロシージャが、複写するようマーク付けされている場合、そのストアド・プロシージャの呼び出しだけが複写されます。そのストアド・プロシージャを構成している個別の文は複写されません。

参照：

- SQL 文の複写使用の例外 (258 ページ)

トリガ

Adaptive Server がトリガ内で DML 文に SQL 文を使用するには、要件をいくつか満たさなければなりません。

- 設定で SQL 文の複写が許可されている。
- DML 文が、SQL 文の複写の使用の条件および例外に該当してはならない。

以下の例では、**delete** 文が `table1` で実行されるとき、従来の複写を使用して複写されます。トリガを介して実行される **delete** (`table2` 上で実行) は、SQL 文の複写を使用して複写されます。これは、テーブルが SQL 文の複写向けに設定されており、かつ **delete** が文として複写されるための条件を満たしているからです。

```
create table table1 (c int, d char(5))
go
create table table2 (c int, d char(5))
go
sp_setreptable table1, true
go
sp_setreptable table2, true
go
insert table1 values (1,'one')
go
insert table2 values (2,'two')
go 100
sp_setrepdefmode table2, 'udi', 'on'
go
create trigger del_table1 on table1
for delete
as
begin
delete table2
end
go
delete table1 where c=1
go
```

参照：

- SQL 文の複写使用の例外 (258 ページ)

ストアド・プロシージャとトリガの再コンパイル

2 つの連続するストアド・プロシージャまたはトリガの間での SQL 複写設定が “off” から “on” に変更されている場合、ストアド・プロシージャとトリガは自動的に再コンパイルされます。

コンパイル時の SQL 文の複写設定	実行時の SQL 文の複写設定	ストアド・プロシージャ/トリガを自動的に再コンパイルするか?
off	On	Yes
On	off	No

データベース間トランザクション

1 件のトランザクションがさまざまなデータベースの複数のテーブルに影響を及ぼすことがあります。別のデータベースにあるテーブルを変更すると、そのテーブルがあるデータベースにその変更に関するログが作成されます。

そのデータベース用に設定された Sybase Replication Agent は、トランザクション・ログに格納された Replication Server 情報を送信します。

次の例では、`db1` と `db2` が、設定済み Sybase Replication Agent を持つ複写データベースです。データベース `db1` は、SQL 文の複写を使用するよう設定されています。

```
use db2
go
begin tran
go
delete t1 where c between 1 and 10000000
delete db1..t1 where c between 1 and 1000000
commit tran
go
```

2 番目の **delete** (db1 データベース上) では、SQL 文の複写を使用するのに対して、1 番目の **delete** (db2 データベース上) では従来の複写を使用します。db1 上で実行されている Sybase Replication Agent は、文を複写します。

Replication Server では、複数データベース間でのトランザクションの整合性は保証されていません。たとえば、1 番目の **delete** の DSI がサスペンドしている間に 2 番目の **delete** の DSI がアクティブである場合、2 番目の **delete** が 1 番目の **delete** より先に複写します。

SQL 文の複写で解決される問題

場合によっては、従来の複写方法ではデータが複写できないことがあります。このような状況でもデータを正常に複写するために用意されたのが、SQL 文の複写です。

ウォーム・スタンバイ設定での select into オペレーションの複写

select into は、**select** リストで指定されたカラムと **where** 句で選択されたローに基づいて、新しいテーブルを作成します。このオペレーションは、リカバリ目的でログが最小限に作成されるにすぎないので、従来の複写では複写できません。

SQL 文の複写を使用すると、ウォーム・スタンバイ設定で **select into** を複写できます。SQL 文の複写をデータベース・レベルで設定するには、以下を使用します。

```
sp_setrepdbmode pdb, 'S', 'on'
go
```

このオプションがデータベース・レベルでアクティブになると、すべての **select into** オペレーション (pdb データベース内) が、SQL 文の複写で複写されます。SQL 文の複写使用の例外を見直して、SQL 文の複写でクエリを複写できることを確認してください。複写対象が **select into** のサブセットだけの場合は、代わりに **set repmode** を使用してください。

参照：

- SQL 文の複写使用の例外 (258 ページ)

プライマリ・キーでの遅延更新の複写

ユニークなカラム・インデックスを持つテーブルでの更新は、従来の複写ではサポートされていないため、Replication Server はエラーを報告します。

たとえば、テーブル `t` にはカラム `c` 上にユニークなインデックスがあり、値は 1、2、3、4、および 5 です。このテーブルに単一の **update** 文が適用されます。

```
update t set c = c+1
```

従来の複写を使用すると、この文は以下のようになります。

```
update t set c = 2 where c = 1
update t set c = 3 where c = 2
update t set c = 4 where c = 3
update t set c = 5 where c = 4
update t set c = 6 where c = 5
```

最初の更新で、値 c=2 をテーブルに挿入しようとします。ところが、この値はテーブルに既に存在します。Replication Server は、エラー 2601 (重複キーの挿入の試み) を表示します。

SQL 文の複写を使用して、この問題に対処できます。テーブルにユニークなインデックスがあり、かつ SQL 文の複写が **update** 文向けに設定されている場合、Adaptive Server は SQL 文の複写を使用して update を複写します。

SQL 文の複写使用の例外

SQL 文の複写使用には、制限事項がいくつかあります。

次の場合、SQL 文の複写はサポートされません。

- レプリケート・データベースに、プライマリ・データベースとは異なるテーブル・スキーマがある。
- Replication Server が、データまたはスキーマの変換を実行する必要がある。
- サブスクリプションまたはアーティクルに **where** 句が含まれている。
- update に 1 個または複数の `text` または `image` カラムが含まれている。
- ファンクション文字列 `rs_delete`、`rs_insert`、`rs_update` がカスタマイズされている。
- DML 文が、ここに示されている条件の 1 個または複数に一致する。以下の場合、従来の複写が使用される。
  - 文が他のデータベースのビュー、テンポラリ・テーブル、またはテーブルを参照している。

    ```
    insert tbl select * from #tmp_info
    where column = 'remove'
    ```

  - 文がゼロより大きい値に設定された **set rowcount** オプションを付けて実行される。

```
set rowcount 1
update customers
set information = 'reviewed'
where information = 'pending'
```

- 文で **top n** 句 (**select** または **select into** 文内) 文、Java 関数、または SQL ユーザ定義ファンクション (UDF) が使用される。

```
delete top 5
from customers
where information = 'obsolete'
```

- ベース・テーブルに暗号化カラムが埋め込まれており、文は **set** 句または **where** 句にあるこれらのカラムのいずれかを参照する。
- 文がシステム・カタログまたは偽のテーブルを参照する ('deleted' や 'inserted' など)。この例では、トリガによって実行される **delete** は SQL 文の複写を使用しない。偽のテーブル `deleted` を使用しているからである。

```
create trigger customers_trg on customers for delete as
delete customers_hist
from customers_hist, deleted
where deleted.custID = customers_hist.custID
go
delete customers where  state = 'MA'
go
```

- 文が **insert** 文であり、`identity` または `timestamp` の新しい値を生成する。
- 文が **update** 文であり、`timestamp` または `identity` の新しい値を生成する。
- 文が **update** 文であり、値をローカル変数に割り当てる。次に例を示す。

```
update t set @a = @a + 2, c = @a where c > 1
```

- 文がマテリアライズされた計算カラムを参照する。
- 文が **select into** 文であり、暗号化カラムがある複写テーブルに影響を及ぼす。
- 文が **insert select** または **select into** であり、**union** 句を使用する。

```
select c1, c2 from tbl2
union
select cc1, cc2 from tbl3
```

- 文が **update**、**insert select**、または **select into** であり、`text/image` カラムがあるテーブル上にある。
- 文がビルトインを使用するクエリである。
  ビルトインが定数値に解決できる場合、クエリは SQL 文として複写されます。次に例を示します。

```
update tbl set value = convert(int, "15")
```

  ところが、以下のクエリは SQL 文の複写で複写されません。

```
update tbl set value = convert(int, column5)
```

ウォーム・スタンバイ・トポロジでは、以下のビルトインを含むクエリは SQL 文の複写で複写できます。これは、ビルトインを定数値に解決できない場合でも当てはまります。

abs	cot	ltrim	sqrt
acos	datalength	patindex	str
ascii	degrees	power	strtobin
asin	exp	replicate	stuff
atan	floor	reverse	substring
atn2	hextoint	right	tan
bintostr	inttohex	round	to_unichar
ceiling	len	rtrim	upper
char	log	sign	
convert	log10	soundex	
cos	lower	space	

SQL 文の複写ではオートコレクションはサポートされない

SQL 文の複写ではオートコレクションを実行できません。データ・サーバ・インタフェース (DSI) で、SQL 文の複写対象の DML コマンドが検出され、オートコレクションがデフォルトで on になっている場合、DSI がサスペンドされ、複写が停止します。

Replication Server でこのエラーを処理する方法を指定するには、エラー番号 5193 で **assign action** コマンドを使用します。

テーブル・レベルのサブスクリプションが確定化されるまで、Replication Server は SQLDML を複写しません。

RSSD システム・テーブルの変更

SQL 文の複写をサポートするために、Replication Server システム・データベース (RSSD) 内のシステム・テーブルが変更されました。

RSSD の以下のシステム・テーブルが SQL 文の複写をサポートします。

- `rs_dbreps` - `status` カラムに、それぞれが DML フィルタに対応する 2 ビット・セットの 4 つの新しいセットが含まれる。セットの最初のビットは空のフィルタかどうかを示し、2 つ目のビットは否定文のセットかどうかを示す。

- `rs_dbsubsets – type` カラムに 4 つの新しいタイプが含まれる。それは、**U**、**L**、**I**、および **S** であり、これらは DML **UDIS** フィルタに対応している。この場合、**D** ではなく **L** が delete に使用される。
- `rs_objects - attributes` カラムに、新しく 5 ビットが含まれる (**U**、**D**、**I**、または **S** オペレーションごとの 1 ビットと、テーブル複写定義に含まれるカラムが着信データ・ローの数よりも少ないかどうかを示す 1 ビット)。
  システム・ファンクション複写定義 **rs_sqldml** も、SQL 文の複写をサポートします。

SQL 文の複写用の Adaptive Server モニタリング・テーブル

Adaptive Server モニタリング・テーブルを使用して、SQL 文の複写中に Adaptive Server の状態の統計的スナップショットを撮影します。このテーブルを使用して、Adaptive Server と Adaptive Server のパフォーマンスを解析できます。

テーブル	説明
`monSQLRepActivity`	SQL 文の複写を使用して複写された DML 文においてオープンしている全オブジェクトの統計情報を提供する。
`monSQLRepMisses`	SQL 文の複写が使用されなかった複写オペレーションの統計情報を提供する。`threshold`、`querylimitation`、および `configuration` カラムは、これらの要因のいずれかによってオブジェクトに対する SQL 文の複写が防止された回数を示す。

Adaptive Server Enterprise の『パフォーマンス＆チューニング・シリーズ：モニタリング・テーブル』の「モニタリング・テーブルの概要」の「Monitoring Tables in Adaptive Server」を参照してください。

製品および混合バージョンの要件

SQL 文の複写には、Adaptive Server バージョン 15.0.3 以降、プライマリおよびレプリケート Replication Server バージョン 15.2 以降、およびルート・バージョン 15.2 以降が必要です。

ダウングレードと SQL 文の複写

SQL 文の複写を使用しており、Adaptive Server を 15.0.2 ESD #3 より前のバージョンにダウングレードするか、Replication Server を 15.2 より前のバージョンにダウングレードする場合のダウングレード手順を学びます。

Adaptive Server のダウングレード

Adaptive Server を以前のバージョンにダウングレードする一方で、ログ内に SQL 文の複写に関係するトランザクション・レコードを残すことができます。

15.0.2 ESD #3 より前のバージョンにダウングレードする場合は、複写データベースを含む Adaptive Server をダウングレードするために記載されている標準的な手順に従ってください。この手順には、トランザクション・ログの排出も含まれます。Adaptive Server Enterprise 15.0.3 の『インストール・ガイド』を参照してください。

Adaptive Server 15.0.3 には、Sybase Replication Agents バージョン 15.0.2 ESD #3 以降向けの次に示すダウングレード・サポートが用意されています。

- ログに SQL 文の複写の情報が含まれている場合でも、Sybase Replication Agent は引き続きデータを複写し続けます。
- Sybase Replication Agent は、SQL 文の複写を含むトランザクションを読み込むと、その文のアトミック変更を送信し、SQL 文の複写に関係する情報は無視します。

Replication Server のダウングレード

Replication Server を 15.2 より前のバージョンにダウングレードできます。

Sybase Replication Agent は、コネクション確立時にネゴシエートされたログ転送言語 (LTL) のバージョンに基づいて、Replication Server に送信される情報の量とタイプを制限します。

15.2 より前の Replication Server の場合、Sybase Replication Agent は SQL 文の複写の情報を送信せず、標準の複写を続行します。

動的 SQL で強化された Replication Server のパフォーマンス

Replication Server の動的 SQL により、複写パフォーマンスが強化されます。これは、Replication Server データ・サーバ・インタフェース (DSI) を使用して、ターゲット・ユーザ・データベースで動的 SQL 文を作成し、繰り返し実行できるようにすることで実現されます。

SQL 言語コマンドをターゲット・データベースに送信するのではなく、実行するたびにリテラルだけが送信されます。これにより、SQL 文の構文チェックと最適化されたクエリ・プランの構築によって生じるオーバヘッドがなくなります。さらに、DSI は動的 SQL 文を最適化します。具体的には、動的 SQL コマンドの実行に失敗したときだけ言語コマンドを生成し、準備されていた文が初めて使用されるときにその準備されていた文を 1 回だけ生成します。

動的 SQL が有効になっている場合、ユーザ・データベース・コネクションでは、次のような場合に言語コマンドの代わりに動的 SQL が使用されます。

- コマンドが **insert**、**update**、または **delete** の場合。
- `text`、`image`、`java`、または `opaque` の各カラムがコマンドに含まれていない場合。
- **update** コマンドまたは **delete** コマンドの **where** 句に NULL 値が含まれていない場合。
- コマンドに含まれるパラメータの数が 255 以下の場合。
  - **insert** コマンドには、255 カラムまで含めることができる。
  - **update** コマンドでは、**set** 句と **where** 句に合計 255 カラムまで含めることができる。
  - **delete** コマンドでは、**where** 句に 255 カラムまで含めることができる。
- コマンドでユーザ定義ファンクション文字列を使用していない場合。

動的 SQL 設定パラメータ

動的 SQL 設定パラメータを使用して、動的 SQL を有効にし調整します。

- **dynamic_sql** - 複写コネクションの動的 SQL を on または off にする。このパラメータを on に設定した場合にのみ、他の動的 SQL 設定パラメータが有効になる。
- **dynamic_sql_cache_size** - コネクションの動的 SQL を使用できるデータベース・オブジェクトの数を Replication Server に通知する。このパラメータは、データ・サーバのリソースの消費を制限する。
- **dynamic_sql_cache_management** - コネクションの動的 SQL キャッシュを管理する。動的 SQL 文がコネクションの **dynamic_sql_cache_size** に達すると、新しい動的 SQL 文の割り付けを停止するか (値が **fixed** の場合)、最後に使用された文を保持し、新しい文を割り付けるためにそれ以外の文の割り付けを解除する (値が **mru** の場合)。

動的 SQL を使用するための設定パラメータの設定

動的 SQL をサーバ・レベルまたはデータベース・コネクション・レベルで有効にしたり設定したりできます。

サーバ・レベルおよびコネクション・レベルでは動的 SQL はデフォルトで無効になっています。

動的 SQL 設定パラメータをサーバ・レベルで設定して、今後作成または開始されるコネクションのデフォルト値とします。動的 SQL をサーバ・レベルで設定するには、以下のように入力します。

```
configure replication server
set { dynamic_sql |
```

```
      dynamic_sql_cache_size |
      dynamic_sql_cache_management }
to value
```

動的 SQL をコネクション・レベルで設定するには、以下のように入力します。

```
alter connection to server.db
set { dynamic_sql |
      dynamic_sql_cache_size |
      dynamic_sql_cache_management }
to value
```

テーブル・レベルでの動的 SQL の制御

create replication definition と **alter replication definition** では、複写定義を介して各テーブルで動的 SQL のアプリケーションの制御が可能です。

「**create replication definition**」と「**alter replication definition**」(『Replication Server リファレンス・マニュアル』の「Replication Server コマンド」) を参照してください。

次のコマンドを使用することにより、特定のレプリケート・データベースに対する動的 SQL の実行をテーブル・レベルで変更できます。

```
set dynamic_sql {on | off}
for replication definition with replicate at
data_server.database
```

複写定義レベルでは、デフォルトで動的 SQL が使用されます。デフォルトを変更するのは、動的 SQL からテーブルを除外するときだけにしてください。動的 SQL の使用状況を確認するには、**stats_sampling** を有効にして **admin stats, dsi** コマンドを実行してから、DSIEDsqlPrepared、DSIEDsqlExecuted、その他の動的 SQL カウンタを探します。

各複写定義に対する動的 SQL の設定を表示するには、**rs_helprep**、**rs_helpsub**、**rs_helppubsub** の各ストアド・プロシージャを使用します。

『Replication Server リファレンス・マニュアル』の「RSSD ストアド・プロシージャ」を参照してください。

replicate minimal columns 句と動的 SQL

複写処理で動的 SQL が使用されるのは、複写定義に replicate minimal columns が含まれているか、またはあるコネクションについて **replicate_minimal_columns** が on に設定されているときです。

replicate_minimal_columns を物理コネクション環境とウォーム・スタンバイ環境で使用できます。DSI はこのパラメータを使用して、複写定義がまったくない場合、または複写定義に **replicate minimal columns** 句が含まれていない場合に、最少数のカラムを使用するかどうかを判断します。

デフォルトでは、**replicate_minimal_columns** はすべてのコネクションに対してオンです。コネクションの **replicate_minimal_columns** 設定は、**replicate all columns** 句で設定された複写定義よりも優先されます。

カスタム・ファンクション文字列を使用した場合、あるコネクションに対して **replicate_minimal_columns** を on に設定すると、現在の複写環境の動作が変わる可能性があります。アプリケーションのトリガ処理が、レプリケート・データベースに送信されるコマンドに依存する場合、**replicate_minimal_columns** 設定がデフォルト値の on であっても、ロー内のどのカラムも変更されないかぎりこのコマンドは送信されません。元の動作をリストアするには、そのコネクションの **replicate_minimal_columns** を off に設定します。

たとえば、**replicate_minimal_columns** を SYDNEY_DS データ サーバの `pubs2` データベースへのコネクションに対して有効にするには、次のように入力します。

```
alter connection to SYDNEY_DS.pubs2
set replicate_minimal_columns to 'on'
```

ロー内のどのカラムの値も変化しない場合でもトリガが起動されることを望む場合、**replicate_minimal_columns** がトリガ処理に影響を及ぼす可能性があります。

admin config を使用して、**replicate_minimal_columns** 設定を表示できます。

注意： **dsi_compile_enable** を 'on' に設定すると、**replicate_minimal_columns** 設定は無視されます。

動的 SQL の制限事項

動的 SQL を使用すべき状況を見極めることには、制限事項がいくつかあります。

- 内部複写定義を使用して、スタンバイ・コネクションまたは MSA コネクションにテーブルを複写し、そのコネクションの動的 SQL が有効になっている場合は、テーブルの新しい複写定義で、プライマリ・データベースのカラム順序と一致するカラム順序を定義する。このように定義しないと、既存の準備文が無効になることがあり、スタンバイ・コネクションまたは MSA コネクションの再起動が必要になる場合がある。
- Replication Server は、動的 SQL コマンドにあるユーザ定義データ型を Open Client/Server™ (OCS) データ型に変換する。

  Sybase の範囲外のデータによって動的 SQL が失敗すると、DSI は、エラー・メッセージをログに記録し、言語コマンドを使用して動的 SQL を再送信します。言語コマンドも失敗した場合のみ、DSI が停止します。

  この状態が頻繁に発生する場合は、テーブル複写定義によって動的 SQL を無効にするか、**set dynamic_sql off** コマンドを使用します。

動的 SQL の無効化

動的 SQL を無効化するために使用できるコマンドはいくつかあります。

- **alter connection... set dynamic_sql off** - このコネクションのすべてのコマンドに対して動的 SQL を無効にする。
- **create/alter replication definition...without dynamic_sql** - この複写定義を使用するすべてのコマンドに対して動的 SQL を無効にする。
- **set dynamic_sql off for replication definition with replicate at...**― このレプリケート・コネクションでこの複写定義を使用するすべてのコマンドに対して動的 SQL を無効にする。

Advanced Services Option

Replication Server には、複写パフォーマンスが強化された Advanced Services Option が搭載されています。

Advanced Services Option のいずれかの強化機能をアクティブ化するには、REP_HVAR_ASE ライセンス・ファイルが必要です。『Replication Server インストール・ガイド』の「インストールの計画」の「ライセンスの取得」を参照してください。

参照：

- Adaptive Server への High-Volume Adaptive Replication (266 ページ)
- リトライ・メカニズムの強化 (276 ページ)
- DSI 効率の向上 (285 ページ)
- RepAgent エグゼキュータ・スレッドの効率の向上 (286 ページ)
- ディストリビュータ・スレッドの読み込み効率の向上 (287 ページ)
- メモリ割り付けの強化 (288 ページ)
- キュー・ブロック・サイズの増加 (288 ページ)

Adaptive Server への High-Volume Adaptive Replication

Replication Server には、High-Volume Adaptive Replication (HVAR) が組み込まれています。これは、同一のデータベース・スキーマを持つデータベースに複写するとき、現行の連続複写モードより高いパフォーマンスを発揮します。

連続複写モードでは、Replication Server はログされた変更をプライマリ・データベースのログ順に従ってレプリケート・データベースに送信します。HVAR は、以下を使用してパフォーマンスの向上を達成します。

- コンパイル ― 複写データを各テーブル別、各 **insert**、**update**、**delete** オペレーション別にし、それらのオペレーションを最終的なローのオペレーションにコンパイルして整理する。
- バルク適用 ― コンパイルされた最終的な結果に対して最も効率の良いバルク・インタフェースを使用して、最終的な結果をバルク適用します。Replication Server は、メモリ内の最終的な変更が保管されるデータベースを使って変更を保管し、それをレプリケート・データベースに適用する。

コンパイル処理では、ログに記録されている個々のオペレーションを送信する代わりに、グループ化された一連のオペレーションから **insert**、**update**、または **delete** の中間オペレーションを削除し、複写されたトランザクションの最終コンパイル状態のみを送信します。トランザクション・プロファイルによって異なりますが、これは通常 Replication Server がレプリケート・データベースに送信して処理させるコマンド数を少なくします。

HVAR はできるだけ多くのコンパイル可能なトランザクションをグループ化して、グループ内のトランザクションをまとめて最終的な変更としてコンパイルしてから、レプリケート・データベースでバルク・インタフェースを使用してその変更をレプリケート・データベースに適用します。

Replication Server が大量のトランザクションを組み合わせてコンパイルし 1 つのグループにまとめるので、バルク・オペレーション処理が向上し、複写スループットとパフォーマンスも向上します。グループ・サイズを調整して、バルク適用のためにグループ化されるデータ量を制御できます。

HVAR は、プライマリ・データベースと同じスキーマを持つレプリケート・データベースのあるシステムのアーカイブとレポートを行うオンライン・トランザクション処理 (OLTP: creating online transaction processing) の作成に特に役立ちます。

データベースとプラットフォームのサポート

HVAR では Adaptive Server 12.5 以降への複写がサポートされています。64 ビットのハードウェア・プラットフォームを使用すると、最適なパフォーマンスを得ることができます。

『Replication Server 15.5 新機能ガイド』の「Replication Server バージョン 15.5 の新機能」の「新しくサポートされたオペレーティング・システム」の「64 ビット・サポート」を参照してください。

HVAR のコンパイルとバルク適用

HVAR ではコンパイル時に、複写するデータがテーブルごとおよび、**insert**、**update**、**delete** オペレーション別に整理してまとめられ、それらのオペレーションが最終的なローのオペレーションにコンパイルされます。

HVAR は複写定義内のプライマリ・キーによって異なったデータ・ローを区別します。複写定義がない場合は、`text` と `image` のカラム以外はすべて、プライマリ・キーとみなされます。

通常の複写環境で見られるオペレーションの組み合わせでは、同一のプライマリ・キーを持つテーブルとローがあると、HVAR は次のオペレーションの組み合わせルールに従います。

- **insert** の後に **delete** があると、結果はオペレーションなしになる。
- **delete** の後に **insert** があると、結果は削減なしになる。
- **update** の後に **delete** があると、結果は **delete** になる。
- **insert** の後に **update** があると、2 つのオペレーションは 1 つのオペレーションに集約され、結果のオペレーションは **insert** になる。結果のオペレーションの内容は、最初のオペレーション結果に次のオペレーションの相違点を上書きした結果となる。
- **update** の後にもう 1 つの **update** があると、2 つのオペレーションは 1 つのオペレーションに集約され、結果のオペレーションは **update** になる。結果のオペレーションの内容は、最初のオペレーション結果に次のオペレーションの相違点を上書きした結果となる。

オペレーションのその他の組み合わせでは、コンパイル・ステータスが無効になります。

例 1

これはログ順のローごとの変更例です。この例では、`T` は **create table** `T(k int , c int)` コマンドによって以前に作成されたテーブルです。

```
1. insert T values (1, 10)
2. update T set c = 11 where k = 1
3. delete T where k = 1
4. insert T values (1, 12)
5. delete T where k =1
6. insert T values (1, 13)
```

HVAR では、1 の **insert** と 2 の **update** を組み合わせて **insert** `T values (1, 11)` に変換できます。変換結果の **insert** と 3 の **delete** は、相殺されるので削除できます。4 の **insert** と 5 の **delete** は削除できます。コンパイルされた最終的な HVAR オペレーションは、6 の最後の **insert** になります。

```
insert T values (1, 13)
```

例 2

ログ順のローごとの変更のもう 1 つの例です。

```
1. update T set c = 14 where k = 1
2. update T set c = 15 where k = 1
3. update T set c = 16 where k = 1
```

HVAR では、1 と 2 の **update** を 2 の **update** にまとめることができます。2 と 3 の updates は 3 の単一の **update** にまとめることができ、これが k = 1 という最終的なロー変更になります。

Replication Server は最終的な変更を保管するインメモリ・データベース内の **insert**、**delete**、および **update** テーブルを使用して、レプリケート・データベースに適用する最終的なロー変更を保管します。最終的なロー変更が複写テーブル別およびオペレーションの種類別 (**insert**、**update**、または **delete**) にソートされると、バルク・インタフェースに渡す準備が整います。HVAR は **insert** オペレーションを複写テーブルに直接ロードします。Adaptive Server は **update** と **delete** のバルク・オペレーションをサポートしないので、HVAR は **update** と **delete** オペレーションをテンポラリ・ワークテーブルにロードします。テンポラリ・ワークテーブルは HVAR によってレプリケート・データベース内に作成されます。次に、HVAR は **join-update** または **join-delete** オペレーションをレプリケート・テーブルに対して実行して、最終的な結果を生成します。ワークテーブルは動的に作成され削除されます。

例 2 では、次の処理によってコンパイル結果が `update T set c = 16 where k = 1` になります。

1. HVAR は `#rs_uT(k int, c int)` ワークテーブルを作成します。
2. この文で、HVAR がワークテーブルに対して **insert** を実行します。
   ```
   insert into #rs_uT(k, c) location 'idemo.db' {select * from rs_uT}
   ```
3. HVAR は、**join-update** を実行します。
   ```
   update T set T.c=#rs_uT.c from T,#rs_uT where T.k=#rs_uT.k
   ```

HVAR が大量のトランザクションを 1 つのグループにまとめるので、バルク・オペレーション処理が向上し、複写スループットとパフォーマンスも向上します。HVAR サイズを設定パラメータで調整することによって、HVAR がバルク適用のためにグループにまとめるデータの量を制御できます。

HVAR はロー変更を変更がログされた順序で適用しませんが、データ・ロスはありません。その理由は、以下のとおりです。

- 異なったデータ・ローでは、ロー変更が適用される順序は結果に影響しません。

- 同じローでは、コンパイル後、**delete** の後に **insert** を適用することによって、整合性を維持します。

最終的な変更のデータベース

Replication Server には、最終的な変更を保管するデータベースがあります。これは、インメモリ・レポジトリとして機能し、トランザクションの最終的なロー変更、つまりコンパイルしたトランザクションを保管します。

各トランザクションに対して 1 つの最終的な変更のデータベース・インスタンスがあります。最終的な変更を保管するデータベース内の各複写テーブルには最高 3 つの追跡テーブルがあります。最終的な変更を保管するデータベースとその中のテーブルを点検することによって HVAR 複写のモニタと問題のトラブルシューティングを行うことができます。

最終的な変更のデータベースのモニタリング

sysadmin cdb コマンドを使用して、最終変更データベースをモニタリングし、最終変更データベース・インスタンスにアクセスします。

『Replication Server リファレンス・マニュアル』の「Replication Server コマンド」の「**sysadmin cdb**」を参照してください。

HVAR の処理と制限事項

HVAR は元のコミット順を維持しながら、トランザクションの最終的なロー変更のみを適用します。それによって中間的なロー変更は省略されますが、トランザクションの整合性は保証されます。

このアプローチには次のような問題が伴います。

- HVAR によるメモリの大量消費を回避するため、Replicaton Server は大規模トランザクションを HVAR ではなく連続複写モードを通じて処理し適用します。大規模トランザクションのスレッショルドは、**dsi_sqt_max_cache_size** で設定できる SQT キャッシュのサイズと、**dsi_compile_max_cmds** と **dsi_cdb_max_size** で制御できる最終変更データベースのサイズで決まります。
- **insert** トリガが起動されません。これは、HVAR プロセスが最終的な新しいローのバルク・ロードをテーブルに対して直接行うからです。Replication Server がコンパイルの最終結果をレプリケート・データベースに適用すると、**update** と **delete** の各トリガは引き続き起動します。ただし、Replication Server がコンパイルし最終結果には含まれないロー変更が、それらのトリガから確認できなくなります。トリガが検出できるのは、最後のロー・イメージのみです。Replication Server を使用して、ユーザが変更したテーブルの任意のカラムにそのユーザを関連付けるトリガ・ロジックのあるテーブル・スキーマで、`last_update_user` カラムを使用してユーザ更新を監査するとします。userA がテーブルの `colA` と `colC` を変更した後に、userB が `colB` と `colD` を変更した場合、トリガが起動すると、トリガ・ロジックが検出できるのはテーブルを

最後に変更したユーザのみです。したがって、トリガ・ロジックはこれら 4 つのカラムを変更したユーザとして userB を関連付けます。ロー変更を個別に検出する必要がある同様のロジックを含むトリガを定義する場合、そのテーブルの HVAR コンパイルを無効にする必要がある場合があります。

- HVAR はロー変更をロー変更がログに記録された順序と同じ順序では適用しません。複写テーブルにログ順に変更を適用するには、そのテーブルに対して HVAR コンパイルを無効にします。
- 複写テーブルに参照制約がある場合、その参照制約を複写定義の中で指定してください。制約エラーを避けるために、HVAR は複写定義に従ってテーブルをロードします。
- HVAR が有効の場合、Replication Server では、カスタム・ファンクション文字列および並列 DSI 逐次化メソッド (デフォルトの **wait_for_commit** メソッドを除く) をサポートしません。HVAR はカスタム・ファンクション文字列をコンパイルできないコマンドとして扱います。
- 以下を検出すると、Replication Server はログ順にローごとに行う連続複写に戻ります。
  - コンパイルできないコマンド ― ストアド・プロシージャ、SQL 文、データ定義言語 (DDL) トランザクション、システム・トランザクション、Replication Server の内部マーカ。
  - コンパイルできないトランザクション ― コンパイルできないコマンドを含んでいるトランザクション。
  - コンパイルできないテーブル ― HVAR が無効にされているテーブル、変更されたファンクション文字列を持つテーブル、HVAR がコンパイルできないテーブルとの参照制約関係があるテーブル。
- 複写定義に **replicate minimal columns** 句が含まれていない場合、HVAR はプライマリ・キー **update** を **delete**、次いで **insert** に自動的に変更します。プライマリ・キー更新は、以下のいずれかです。
  - プライマリ・キーがテーブルの複写定義で定義されているテーブルのプライマリ・キーに影響を及ぼす更新。または
  - 複写定義がない場合は、任意のカラム (`text` カラムと `image` のカラムを除く) に影響を及ぼす更新。この場合、複写定義でプライマリ・キー定義が具体的に定義されていないので、Replication Server はすべてのカラムがプライマリ・キーの一部であると想定します。
- 複写定義に **replicate minimal columns** 句があり、かつ HVAR がコンパイル中のグループにプライマリ・キー・カラムを変更する **update** コマンドが含まれている場合、HVAR はグループの残りの部分について、実行時にコンパイルできないものとしてテーブルを自動的に認識します。このテーブルに適用される **update** オペレーションがコンパイルできないのは、HVAR が **update** を **delete** と **insert** で構成されるオペレーションのペアに変換できないからです。HVAR は、自己が処理しているトランザクション・グループ内で、コンパイルできないプ

ライマリ・キーの **update** オペレーションに出会う前に処理したオペレーションをすべて、最終変更データベースにコンパイルできます。ただし、HVAR はトランザクション・グループ内では、初回のコンパイルできないプライマリ・キー **update** およびそれに続くオペレーションすべてをコンパイルできないものとしてマーク付けします。コンパイルできないというテーブルのステータスは一時的なものであり、HVAR が処理中の同じトランザクション・グループの期間中しか続きません。

- HVAR はトランザクションのグループ化を停止できるパラメータ (**dsi_partition_rule** など) を無視します。
- HVAR 処理中にエラーが発生すると、Replication Server はコンパイルをリトライします。リトライでは、コンパイルが失敗したトランザクションを特定できるまで、トランザクション・グループを小さなグループに分割していきます。特定されたトランザクションは連続複写を使って適用されます。
- パフォーマンス上の利点を実現するには、プライマリ・データベースとレプリケート・データベースを同期させ、エラー発生による Replication Server への余分な処理オーバヘッドを避けるようにします。**dsi_command_convert** を **i2di,u2di** に設定して、データを同期できますが、これも処理オーバヘッドを発生させます。データベースを同期する場合は、**dsi_command_convert** を **none** にリセットします。
- HVAR はロー・カウントの検証を行って複写の整合性を確認します。ロー・カウントの検証はコンパイルに基づいて行われます。予期されるロー・カウントはコンパイル後のロー数です。
- 複写定義の中に `identity` データ型のカラムがある場合、Replication Server はレプリケート・データベース内で次のコマンドを実行します。
  - **set identity_insert** *table_name* **on** (ID カラムの挿入前) および **set identity_insert** *table_name* **off** (ID カラムの挿入後)。
  - **set identity_update***table_name* **on** (ID カラムの更新前) および **set identity_update***table_name* **off** (ID カラムの更新後)。

参照：

- 参照制約のあるテーブル (282 ページ)

HVAR を有効にする

dsi_compile_enable を使用して、レプリケート・データベースへのコネクションで HVAR を有効にします。

dsi_compile_enable を off に設定した場合、Replication Server はログ順、ローごとの連続複写モードを使用します。たとえば、テーブル上のすべてのオペレーションをログ順に複写する必要があるトリガがテーブルにあるためコンパイルを使用できない場合のように、最終ロー変更を複写すると問題が発生する場合、問題のテーブルで **dsi_compile_enable** を off に設定します。

注意： **dsi_compile_enable** を on に設定すると、Replication Server は **dsi_cmd_prefetch** と **dsi_num_large_xact_threads** を無効にします。

指定したデータベースのみに影響するように、データベース・レベルで HVAR を有効にして設定するには、次のように入力します。

```
alter connection to data_server.database
set dsi_compile_enable to 'on'
go
```

HVAR をサーバまたはテーブル・レベルで有効にして設定することもできます。

- サーバ・レベル ― Replication Server へのすべてのデータベース・コネクションに影響します。

  ```
  configure replication server
  set dsi_compile_enable to 'on'
  ```

- テーブル・レベル ― 指定した複写テーブルのみに影響します。テーブル・レベルとデータベース・レベルの両方でパラメータを指定している場合は、テーブル・レベルのパラメータがデータベース・レベルのパラメータよりも優先されます。テーブル・レベルでパラメータを指定しなければ、データベース・レベルのパラメータの設定が適用されます。テーブルにパラメータを設定するには、**alter connection** と **for replicate table named** 句を使用します。次に例を示します。

  ```
  alter connection to data_server.database
  for replicate table named dbo.table_name
  set dsi_compile_enable to 'on'
  ```

  for replicate table name 句を使用すると、コネクション設定がテーブル・レベルで変更されます。設定の変更は指定したテーブルのすべてのサブスクリプションからの複写データと複写定義に適用されます。

  注意： テーブル・レベルの設定には、**alter connection** しか使用できません。これは Replication Server が **for** 句を **create connection** に対してサポートしていないためです。

dsi_compile_enable を実行した後、レプリケート・データベースへのコネクションをサスペンドしてレジュームします。

HVAR 設定パラメータ

Replication Server は、Sybase が推奨するいくつかのパラメータのデフォルト値を自動的に設定します。これらのパラメータの値を変更すると、レプリケーション・パフォーマンスを微調整できます。

変更するパラメータごとに個別の **alter connection** コマンドを実行する必要があります。**alter connection** を入力した後は、2 つ以上のパラメータを入力しないでください。

HVAR は、Sybase が推奨する **dsi_cdb_max_size**、**dsi_compile_max_cmds**、**dsi_bulk_threshold**、**dsi_command_convert**、および **dsi_compile_retry_threshold** のデフォルト値を自動的に設定します。ただし、複写環境のパフォーマンスを調整するために独自の値を指定することもできます。

パラメータの詳細な説明については、『Replication Server リファレンス・マニュアル』の「Replication Server コマンド」の「**alter connection**」を参照してください。

dsi_bulk_threshold

dsi_bulk_threshold は、特定のコマンド・タイプのテーブルでコンパイルが行われた後の最終的なロー変更コマンド数を指定します。その数に達したら、それがトリガになって、Replication Server はそのテーブルの同じコマンド・タイプにバルク・コピー・インを使用します。

デフォルトの最終的なロー変更コマンド数は 20 です。

例：

```
alter connection to SYDNEY_DS.pubs2
set dsi_bulk_threshold to '15'
go
```

dsi_cdb_max_size

dsi_cdb_max_size は、トランザクションが DSI SQT キャッシュを超えない、またはトランザクション内のコマンド数が **dsi_compile_max_cmds** を超えない場合に、HVAR がコンパイルできるトランザクションの最大サイズをメガバイトで指定します。

HVAR がコンパイルしている現在のグループでトランザクションのサイズが **dsi_compile_max_cmds** に達すると、HVAR は新しいグループを開始します。読み込むデータがなくなると、グループが **dsi_cdb_max_size** で設定した最大サイズに達していなくも、HVAR は現在のトランザクションのセットを現在のグループにグループ化する処理を終了します。

デフォルトは 1024MB です。

例：

```
alter connection to SYDNEY_DS.pubs2
set dsi_cdb_max_size to '2048'
go
```

dsi_compile_max_cmds

dsi_compile_max_cmds は、トランザクションが DSI SQT キャッシュを超えない、またはトランザクション・サイズが **dsi_cdb_max_size** を超えない場合に、HVAR がコンパイルできるトランザクションの最大サイズをコマンド数で指定します。

Replication Server は、連続したログ順のレプリケーション・モードでコンパイルできないトランザクションを複写します。

HVAR がコンパイルしている現在のグループでコマンド数が **dsi_compile_max_cmds** に達すると、HVAR は新しいグループを開始します。読み込むデータがなくなると、グループが **dsi_compile_max_cmds** で設定した最大コマンド数に達していなくも、HVAR は現在のトランザクションのセットを現在のグループにグループ化する処理を終了します。

デフォルトは 10,000 コマンドです。

例：

```
alter connection to SYDNEY_DS.pubs2
set dsi_compile_max_cmds to ‘50000’
go
```

dsi_compile_retry_threshold

dsi_compile_retry_threshold は、グループ内のコマンド数に対するスレッショルド値を指定します。失敗したトランザクションを含むグループ内のコマンド数が **dsi_compile_retry_threshold** の値より小さい場合、Replication Server は HVAR モードでそのグループのリトライ処理を行わないので、処理時間を節約してパフォーマンスを向上できます。代わりに、Replication Server は連続複写モードに切り替わります。連続複写モードでは、プライマリ・データベースのログ順に従って変更がレプリケート・データベースに送信されます。

デフォルトは 100 コマンドです。

例：

```
alter connection to SYDNEY_DS.pubs2
set dsi_compile_retry_threshold to '200'
go
```

dsi_command_convert

dsi_command_convert – 複写コマンドの変換方法を指定します。変換の種類は次のオペレーションの組み合わせによって指定されます。

- **d** – delete
- **i** – insert
- **u** – update
- **t** - truncate
- **none** – オペレーションなし

dsi_command_convert に対するオペレーションの組み合わせには、**i2none**、**u2none**、**d2none**、**i2di**、**t2none**、**u2di** があります。変換前のオペレーションは "2" の前に、変換後のオペレーションは "2" の後ろにあります。次に例を示します。

- **d2none** – **delete** コマンドを複写しない。このオプションでは、**rs_delete** ファンクション文字列をカスタマイズする必要はありません (**delete** オペレーションを複写しない場合)。
- **i2di**、**u2di** – **insert** と **update** の両方を **delete** とその後の **insert** に変換する。これはオートコレクションと同等のオペレーションです。**dsi_row_count _validation** を off にすることによってロー・カウントの検証を無効にする場合、複写時に重複キー・エラーを避け、データベースの自動同期ができるようにするために、**dsi_command_convert** を **i2di,u2di** に設定するようおすすめします。
- **t2none** – **truncate table** コマンドを複写しない。

dsi_command_convert のデフォルトは **none** です。これは、コマンドの変換がないことを意味します。

例：

```
alter connection to SYDNEY_DS.pubs2
set dsi_command_convert to 'i2di,u2di'
go
```

リトライ・メカニズムの強化

リトライ・メカニズムの強化を使用すると、Replication Server によるコンパイルおよび一括適用の回数が減らされるため、レプリケーションのパフォーマンスが向上します。

HVAR はできるだけ多くのコンパイル可能なトランザクションをグループ化して、グループ内のトランザクションをまとめた最終的な変更としてコンパイルしてから、レプリケート・データベースでバルク・インタフェースを使用してその変更をレプリケート・データベースに適用しようとします。HVAR の処理結果から発生するレプリケーションのトランザクションが失敗すると、HVAR はリトライ・メカニズムを呼び出します。グループ内のトランザクションが失敗すると、HVAR はそのグループを同じサイズの 2 つのグループに分割し、コンパイルとバルク適用を各グループに対して試みます。リトライ・メカニズムは失敗したトランザクションを特定し、Replication Server がエラー・アクションのマッピングを実行できるようにします。また、DSI が停止する場合もあるので、失敗したトランザクションの前にあるすべてのトランザクションが適用されます。

HVAR 内の最終的な変更を保管するデータベースは、トランザクションの最終的なロー変更、つまりコンパイルしたトランザクションを保管するインメモリ・レポジトリとして機能します。最終的な変更を保管するデータベースの内容は、複数のプライマリ・トランザクションからのコマンドを集約したものであり、HVAR ではログ順に適用されません。したがって、リトライ・メカニズムがないと失敗したトランザクションを特定する方法がありません。グループ内のトランザクションが失敗したら、リトライ・メカニズムはそのグループを分割してコンパイルとバルク適用を繰り返します。このような連続したリトライ・プロセスはパフォーマンスの低下の原因となります。

リトライ・メカニズムの強化によって、HVAR でトランザクションが失敗したグループが検出された場合にグループが 3 等分され、失敗したトランザクションを含むグループの特定がより効率的に行われるようになりました。

さらに **dsi_compile_retry_threshold** パラメータを使用してグループ内のコマンド数にスレッショルド値を指定できます。失敗したトランザクションを含むグループ内のコマンド数が **dsi_compile_retry_threshold** の値より小さい場合、Replication Server は HVAR モードでそのグループのリトライ処理を行わないので、処理時間を節約してパフォーマンスを向上できます。代わりに、Replication Server は連続複写モードに切り替わります。連続複写モードでは、プライマリ・データベースのログ順に従って変更がレプリケート・データベースに送信されます。

メモリ消費の制御

HVAR でのメモリ消費を低減するため、コンパイル可能なグループのサイズを制御します。

メモリ消費は、最終的な変更を保管するデータベース、および構造で保存されるデータなどの Replication Server データ構造です。最終的な変更が保管されるデータベースは、インメモリ・データ構造です。最終的な変更が保存されるデータベースのメモリ消費は、Replication Server が多数のカラムを持つテーブルや、大きな `text` および `image` データ型値を持つテーブルに適用されたコマンドをコンパイルすると、劇的に増加することがあります。たとえば、100 のカラムを持つテーブルで 1,000,000 のローをコンパイルすると、10 のカラムを持つテーブルで同数のローをコンパイルするのと比べて約 10 倍のメモリが消費されます。他のプロセスやモジュールに使用できるメモリが不足すると、レプリケーションのパフォーマンスが低下します。

トランザクションが DSI SQT キャッシュ・サイズより大きい場合、Replication Server はそのトランザクションにコンパイル不可のマークを付けます。トランザクションが DSI SQT キャッシュに収まる場合、Replication Server は **dsi_cdb_max_size** および **dsi_compile_max_cmds** の値をトランザクションのサイズに対してチェックします。Replication Server で、最終変更を保管するデータベースのサイズに **dsi_cdb_max_size** よりも大きいサイズが必要と見なされた場合、またはトランザクションに **dsi_compile_max_cmds** よりも多くのコマンドが含まれていると、Replication Server はトランザクションにコンパイル不可のマークを付けます。Replication Server は、連続したレプリケーション・モードにコンパイルできない大きなトランザクションを適用します。連続したレプリケーション・モードを使用することで、大きなトランザクションに対応した 1 つの大きな最終変更保管データベースが作成されるのを防ぎ、メモリ消費を低減することができます。

Replication Server はできるだけ多くのコンパイル可能なトランザクションをコンパイル可能なグループに分けようとします。また、Replication Server は、**dsi_cdb_max_size** と **dsi_compile_max_cmds** をコンパイル可能なグループのスレッ

ショルドに使用します。グループが **dsi_cdb_max_size** または **dsi_compile_max_cmds** で設定したサイズに達すると、Replication Server はトランザクションをグループにコンパイルするのを停止し、コンパイル可能な各グループを単一のトランザクションとしてレプリケート・データベースに適用します。

HVAR の SQT メモリ消費の制御

HVAR でトランザクション・プロファイリング中に DSI SQT キャッシュでパックされていないコマンドが消費する最大メモリ量を制御します。

SQT スレッドは、HVAR のトランザクション・プロファイリング処理によってアンパックされたコマンドが使用し、DSI SQT キャッシュによって参照されるメモリをモニタします。

Replication Server が HVAR を使用して複写を行っている場合、DSI スレッドで使用される最大メモリ量は **dsi_sqt_max_cache_size**、**sqt_max_prs_size**、および **dsi_cdb_max_size** の合計です。**dsi_sqt_max_cache_size**、**sqt_max_prs_size**、および **dsi_cdb_max_size** に小さい値を設定するとメモリ消費は低減しますが、レプリケーション・パフォーマンスは低下します。最適なメモリ消費とパフォーマンスを実現するには、複写環境をチューニングします。パラメータの設定については、『Replication Server リファレンス・マニュアル』の「Replication Server コマンド」を参照してください。

最終的な変更を保管するデータベースのサイズ予測とトランザクション・プロファイリング

トランザクションが DSI SQT キャッシュ・サイズより大きい場合でも、Replication Server はそのトランザクションにコンパイル不可のマークを付けません。

DSI SQT キャッシュ・サイズを制約として持たない場合、最終的な変更を保管するデータベースのサイズを予測する機能が向上し、連続レプリケーション・モードに切り替える必要がなくなり、トランザクション・プロファイリング処理がより効率的になります。

次の場合にのみ、Replication Server はコンパイル不可としてトランザクションにフラグを付けます。

- トランザクションにあるコマンドの数が **dsi_compile_max_cmds** を超える、または
- トランザクションの最終的な変更を保管するデータベースの予測サイズが **dsi_cdb_max_size** を超える

パフォーマンスをさらに向上させるには、Replication Server は最終的な変更を保管するデータベースのサイズを予測する回数を 100 コマンドに 1 回に減らします。

HVAR のフル・インクリメンタル・コンパイル

フル・インクリメンタル・コンパイルにより、High-Volume Adaptive Replication (HVAR) のレプリケーション・パフォーマンスが向上しますが、これは多くのコマンドを含む大規模なコンパイル可能なトランザクションの処理中のメモリ消費が低減された結果です。フル・インクリメンタル・コンパイルでは、Replication Server は、連続レプリケーション・モードに戻す必要はなく、より効率的な HVAR モードを使用してラージ・トランザクションのコンパイルとレプリケートを行います。

フル・インクリメンタル・コンパイルでは、**insert**、**delete**、または **update** の混合オペレーションを含む大規模なトランザクションをコンパイルできます。Replication Server では、フル・インクリメンタル・コンパイルを使用して大規模なコンパイル可能なトランザクションをレプリケート・データベースに適用します。その際、最終的な変更を保管する複数のインメモリ・データベース・インスタンスを使用します。フル・インクリメンタル・コンパイルは、ラージ・トランザクションをセグメントのシーケンスに分割します。各セグメントはコマンド・グループで構成されます。

Replication Server は各セグメントをコンパイルし、1 つのセグメントを格納するために最終的な変更を保管する専用データベースを作成します。Replication Server は、最終的な変更を保管するデータベース・インスタンスに、セグメントをレプリケート・データベースに送信し適用するよう指示します。この後、Replication Server は、最終的な変更を保管するデータベース・インスタンスを閉じ、そのデータベースが消費していたメモリを解放します。Replication Server は、次のトランザクション・セグメントに対して別の最終的な変更を保管するデータベース・インスタンスを作成し、最終的な変更を保管するデータベース・インスタンスをすべてのセグメントに対して順序どおりに引き続き作成してから閉じます。

このため、最終的な変更を保管する大規模なデータベース・インスタンスに対して単一のメモリの大部分を消費してラージ・トランザクションを保持するのではなく、フル・インクリメンタル・コンパイルによってメモリ要件は、トランザクションのセグメントのみを含んだ単一の最終的な変更を保管する小さなデータベース・インスタンスが消費するメモリに低減されます。フル・インクリメンタル・コンパイルは、使用した最終的な変更を保管するデータベース・インスタンスの数でメモリ要件を除算します。たとえば、フル・インクリメンタル・コンパイルが最終的な変更を保管する 10 のデータベース・インスタンスを使用してラージ・トランザクションを適用する場合、メモリ要件はフル・インクリメンタル・コンパイルを伴わない要件の約 10 分の 1 です。

スモール・トランザクションのコンパイル中、Adaptive Server は **update** と **delete** のバルク・オペレーションをサポートしないので、HVAR は **update** と **delete** オペレーションをテンポラリ・ワークテーブルにロードします。テンポラリ・ワークテーブルは HVAR によってレプリケート・データベース内に作成されます。次に、

HVAR は **join-update** または **join-delete** オペレーションをレプリケート・テーブルに対して実行して、最終的な結果を生成します。HVAR はワークテーブルを動的に作成して削除します。ただし、ラージ・トランザクションのコンパイルに対してフル・インクリメンタル・コンパイルをサポートするには、HVAR はテンポラリ・オブジェクトでなく通常のテーブルを使用してレプリケート・データ・サーバの **tempdb** データベースにワークテーブルを作成します。

デフォルトでは、Replication Server で HVAR のフル・インクリメンタル・コンパイルは有効になっていません。

データベースのサポート

High Volume Adaptive Replication to Adaptive Server データベースのフル・インクリメンタル・コンパイルは有効にできます。

HVAR のフル・インクリメンタル・コンパイルの有効化

High Volume Adaptive Replication (HVAR) to Adaptive Server レプリケート・データベースのフル・インクリメンタル・コンパイルを有効にするには、**RSFEATURE_HQ_INCR_CMPL_ON** トレース・フラグを on に設定します。

isql にログインし、**RSFEATURE_HQ_INCR_CMPL_ON** トレース・フラグを on に設定し、HVAR のフル・インクリメンタル・コンパイルを有効にします。

```
trace {"on"|"off"},rsfeature,rsfeature_hq_incr_cmpl_on
```

必要に応じて、このトレース・フラグを Replication Server 設定ファイルの終わりに入力することもできます。

```
trace=rsfeature,rsfeature_hq_incr_cmpl_on
```

トレース・フラグのアクティブ化を解除するには、この行を設定ファイルから削除します。

デフォルトで、**RSFEATURE_HQ_INCR_CMPL_ON** は off に設定されます。トレース・フラグの設定を上書きするには、設定が設定ファイルで異なる場合でも、**trace** コマンドを Replication Server の現在のセッションの **isql** で使用します。ただし、Replication Server が再起動すると、設定ファイルの設定が実行されます。

メモリ制御パラメータおよび Replication Server の処理

レプリケーション・モードおよびレプリケーション・アクションは、メモリ制御パラメータに対して設定する値に応じて異なります。

Replication Server の処理

1. Replication Server はアウトバウンド・キューからトランザクションを読み取り、最終的な変更を保管するデータベースのサイズを推定します。

2. トランザクションに **insert**、**update**、および **delete** コマンドのみが含まれている場合、Replication Server はコンパイル可能としてトランザクションにフラグを付けます。
3. 次の場合、Replication Server はコンパイル不可としてトランザクションにフラグを付けます。
   - トランザクションにあるコマンドの数が **dsi_compile_max_cmds** を超える
   - トランザクションの最終的な変更を保管するデータベースの予測サイズが **dsi_cdb_max_size** を超える、または
   - トランザクション・サイズが DSI SQT キャッシュよりも大きい

   Replication Server は、連続モードのログ順でコンパイルできないトランザクションを処理します。
4. Replication Server は、トランザクションがコンパイル可能かどうかを確認すると、連続したコンパイル可能なトランザクションをコンパイル可能なグループに集約します。ただし、Replication Server は 2 つのスレッショルドに基づいてコンパイル可能なグループのサイズを増やす処理を停止します。
   - Replication Server によって、処理中のコンパイル可能なトランザクションのグループ内のコマンド数が新しいトランザクション内のコマンド数に追加されると **dsi_compile_max_cmds** スレッショルド値を超えることが計算されると、Replication Server はグループを閉じてディスパッチし、新しいトランザクションを新しい空のグループに追加します。そうでない場合、Replication Server は新しいコンパイル可能なトランザクションをグループに追加します。
   - 新しいトランザクションを新しいグループに集約することで発生する次の最終的な変更を保管するデータベースの予測サイズが **dsi_cdb_max_size** を超えると、Replication Server はグループを閉じてディスパッチし、新しいトランザクションを新しい空のグループに追加します。そうでない場合、Replication Server は新しいコンパイル可能なトランザクションをグループに追加します。
5. アウトバウンド・キューにコンパイル可能なトランザクションがなくなったら、Replication Server は即座に処理中のグループを閉じてディスパッチします。Replication Server は新しいトランザクションを待たずに、アウトバウンド・キューに入ります。

dsi_cdb_max_size の異なる値への設定

以下は、Replication Server が 2 つのテーブルで 100,000 の更新によってトランザクションを適用しているのを示しています。テーブル 1 には約 4GB のメモリを必要とする 100 のカラムがあり、テーブル 2 にはメモリの約 10 分の 1 の 400MB を必要とする 10 のカラムがあります。

dsi_cdb_max_size 値 (MB)	テーブル名	複写プロセスへの影響
1024 (デフォルト)	テーブル 1	Replication Server は、連続したログ順のレプリケーション・モードでトランザクションを適用する。
1024 (デフォルト)	テーブル 2	前提条件：Replication Server の **memory_limit** を、400MB の最終的な変更を保管するデータベースを作成できる大きさの値に設定する。 Replication Server は、HVAR を使用してトランザクションを適用する。
4096	テーブル 1	Replication Server は、連続したログ順のレプリケーション・モードでトランザクションを適用する。
4096	テーブル 2	前提条件：Replication Server の **memory_limit** を、400MB の最終的な変更を保管するデータベースを作成できる大きさの値に設定する。 Replication Server は、HVAR を使用してトランザクションを適用する。

参照制約のあるテーブル

参照制約 (外部キーその他の検査制約など) のあるテーブルの指定には複写定義を使用できます。それによって HVAR にそれらのテーブルの存在が通知されます。

通常は、参照元のテーブルには同じプライマリ・データベース内の参照先テーブルに対する参照制約が含まれています。HVAR では複数のプライマリ・データベースからの参照先テーブルをサポートするよう参照制約が拡張されています。

各プライマリ・データベースに対する複写定義内で参照元テーブルを指定できます。ただし、複数の参照制約が互いに競合する場合は、Replication Server によってランダムにテーブルが 1 つ選択されます。

参照：

- HVAR の処理と制限事項 (270 ページ)

複写定義の作成と変更

参照制約のあるテーブルを指定するには、**create replication definition** コマンドで **references** パラメータを指定します。

create replication definition

```
...
(column_name [as replicate_column_name]
    ...
    [map to published_datatype]] [quoted]
```

```
        [references [table_owner.]table_name [(column_name)]] …)
   ....]
```

参照元のテーブルの追加と変更には、**alter replication definition** コマンドで **references** パラメータを指定します。参照を削除するには、**null** オプションを使用します。

alter replication definition

```
   .....
add column_name [as replicate_column_name]
[map to published_datatype] [quoted]
[references [table_owner.]table_name [(column_name)]
...
| alter columns with column_name references
{[table_owner.]table_name [(column_name)] | NULL}
[, column_name references {[table_owner.]table_name [(column_name)]
| NULL}
     ...
```

alter replication definition と **create replication definition** (**reference** 句あり) の両方で、Replication Server は以下のように動作します。

- **reference** 句をカラム・プロパティとして扱う。各カラムはテーブルを 1 つだけ参照できる。
- **reference** 句内の *column_name* パラメータに指定したカラム名を処理しない。
- 循環参照になる参照制約を許可しない。たとえば、元の参照先テーブルは元の参照元テーブルへの参照制約を持つことはできない。

複写プロセスでは、HVAR は次のようにロードします。

- 参照先テーブルへの挿入の後で複写定義で指定した参照元テーブルに挿入する。
- 複写定義で指定したテーブルでの削除の後で参照先テーブルで削除する。

場合によっては、両方のテーブルでの更新が競合によって失敗することがあります。HVAR が複写処理のリトライをしないようにして、パフォーマンスの低下を防ぐには、以下を行います。

- 更新を削除と挿入に変換するように、**dsi_command_convert** を "u2di" に設定して複写の更新を停止する。
- **dsi_compile_enable** を off にして、影響を受けたテーブルがコンパイルされるのを避ける。

カスタム・ファンクション文字列を持つテーブルと、コンパイルできない既存テーブルへの参照制約を持つテーブルは HVAR でコンパイルできません。これらのテーブルにマークを付けることによって、HVAR は参照制約エラーによって発生するトランザクションのリトライを避け、複写処理を最適化できます。

HVAR 情報の表示

設定パラメータ・プロパティとテーブル参照の情報を表示できます。

設定パラメータ・プロパティの表示

admin config を使用して、例に示されているようなデータベース・レベルとテーブル・レベルの設定パラメータを表示します。

- データベース・レベル
  - NY_DS データ・サーバ (`NY_DS.nydb1`) の `nydb1` データベースへのコネクションに使用するデータベース・レベルの設定パラメータをすべて表示するには、次のように入力します。
    ```
    admin config, "connection", NY_DS, nydb1
    ```
  - `NY_DS.nydb1` へのコネクションで **dsi_compile_enable** が **on** であることを確認するには、次のように入力します。
    ```
    admin config, "connection", NY_DS, nydb1,dsi_compile_enable
    ```
  - **dsi_compile_enable** など、名前の一部に "enable" があるデータベース・レベルの設定パラメータをすべて表示するには、次のように入力します。
    ```
    admin config, "connection", NY_DS, nydb1,"enable"
    ```

    注意： "enable" は Replication Server の予約語なので、引用符で囲む必要があります。『Replication Server リファレンス・マニュアル』の「トピック」の「予約語」を参照してください。
- テーブル・レベル

  dsi_command_convert を使用して NY_DS データ・サーバの `nydb1` データベースにある `tb1` テーブルで **d2none** を設定した後、すべての設定パラメータを表示するには、次のように入力します。
  ```
  admin config, "table", NY_DS, nydb1
  ```

『Replication Server リファレンス・マニュアル』の「Replication Server コマンド」の「**admin config**」を参照してください。

テーブル参照の表示

テーブル参照の情報と RTL の情報を表示するには、**rs_helprep** を使用します。これは、Replication Server システム・データベース (RSSD) 上で実行できます。

create replication definition を使用して作成した **authors_repdef** 複写定義に関する情報を表示するには、次のように入力します。

```
rs_helprep authors_repdef
```

『Replication Server リファレンス・マニュアル』の「RSSD ストアド・プロシージャ」の「**rs_helprep**」を参照してください。

Replication Server 15.5 のシステム・テーブル・サポート

Replication Server では rs_tbconfig テーブルをテーブル・レベルの設定パラメータの保管に使用し、rs_columns テーブルの ref_objowner カラムと ref_objname カラムを参照制約のサポートに使用します。

テーブルの詳細については、『Replication Server リファレンス・マニュアル』の「Replication Server システム・テーブル」を参照してください。

混合バージョンのサポートと下位互換性

HVAR では、複写定義で指定されている参照制約を複写できるのは、アウトバウンド・ルートのバージョンが 15.5 以降の場合のみです。

アウトバウンド・ルートのバージョンが 15.5 より古くても HVAR は機能します。しかし、バージョン 15.5 以降の場合は、参照制約情報を Replication Server で使用できません。

連続複写モードはサポートされているすべてのバージョンの Replication Server のデフォルト複写モードです。HVAR を使用できるのは Replication Server 15.5 以降のみです。

DSI 効率の向上

dsi_cmd_prefetch を有効にすると、データ複写の遅延時間が短縮されます。これにより、Replication Server が **ct_results** ルーチンを通して複写データ・サーバからの結果を待つ時間が短縮され、その結果データ・サーバが Replication Server を待つ時間が短縮されます。

dsi_cmd_prefetch の動作の仕組みは以下のとおりです。

- Replication Server が、複写データ・サーバから返された現在のバッチの結果を処理する前に、複写データ・サーバ向けの次のコマンド・バッチを準備できるようにします。
- DSI エグゼキュータ (DSI/E) スレッドと DSI スケジューラ (DSI/S) スレッドとの間の同時実効性を向上させます。

dsi_cmd_prefetch を on に設定します。その際、**alter connection** または **create connection** を使用します。

たとえば、**dsi_cmd_prefetch** を SYDNEY_DS データ・サーバの pubs2 データベースへのコネクションに対して有効にするには、次のように入力します。

```
alter connection to SYDNEY_DS.pubs2
set dsi_cmd_prefetch to 'on'
```

デフォルト値は off

dsi_cmd_prefetch は、動的パラメータです。パラメータを有効にした後、データベース・コネクションのサスペンドとレジュームを行わなくても、変更は反映されます。

データ・サーバのパフォーマンスを高めるように調整する場合、**dsi_cmd_prefetch** を有効にすると、パフォーマンスがさらに向上する可能性があります。

注意： **dsi_compile_enable** を 'on' に設定すると、**dsi_cmd_prefetch** に設定した値は無視される。

RepAgent エグゼキュータ・スレッドの効率の向上

NRM スレッドを使用してログ転送言語 (LTL: Log Transfer Language) コマンドを正規化してパックし、それと並行して RepAgent エグゼキュータ・スレッドによる解析を行うことによって、Replication Server でのパフォーマンスを向上させることができます。

Replication Server に NRM スレッドを有効にするよう指示すると、ある 1 本のスレッドが RepAgent エグゼキュータ スレッドから分離して、NRM スレッドになります。NRM スレッドとの並列処理によって RepAgent エグゼキュータ・スレッドは応答時間を短縮します。

NRM スレッドを有効にした後、RepAgent エグゼキュータ スレッドから NRM スレッドへのメッセージ･キューに使用できるメモリを指定できます。

NRM スレッドの有効化

nrm_thread を on に設定 (**configure replication server** を使用) して、NRM スレッドを有効にします。

次のように入力します。

```
configure replication server
set nrm_thread to 'on'
```

デフォルト値は off

nrm_thread は、サーバ・レベルのパラメータです。パラメータ値を変更した後に、Replication Server を再起動してください。

RepAgent エグゼキュータが使用可能なメモリの指定

nrm_thread を on に設定した後、**exec_nrm_request_limit** パラメータに **configure replication server** または **alter connection** を付けて使用して、NRM スレッドのメッセージ・キューについて RepAgent エグゼキュータ・スレッドが使用できる合計メモリ量を指定します。

メッセージ・キュー上でコマンドが使用できる合計メモリ量が、**exec_nrm_request_limit** で指定した値より大きい場合、RepAgent エグゼキュータ・

スレッドはスリープに入り、メモリが空くまで待ちます。NRM スレッドがメッセージ・キュー上でコマンドを処理するにつれて、RepAgent エグゼキュータ・スレッドのためにメモリが解放されていきます。

たとえば、**exec_sqm_nrm_request_limt** を SYDNEY_DS データ・サーバにある pubs2 データベースへのコネクションのために、1 GB に設定するには、次のように入力します。

```
alter connection to SYDNEY_DS.pubs2
set exec_nrm_request_limit to '1073741824'
```

exec_nrm_request_limit 値:

- デフォルト値は
  - 32 ビット版 – 1,048,576 バイト (1MB)
  - 64 ビット版 – 8,388,608 バイト (8MB)
- 最大 – 2,147,483,647 バイト (2GB)
- 最小 - 16,384 バイト (16KB)

exec_nrm_request_limit の設定を変更した後、Replication Agent をサスペンドしてレジュームする。レジュームとサスペンドを行うには、以下のようにします。

- RepAgent for Adaptive Server の場合、Replication Server で **sp_stop_rep_agent** の次に **sp_start_rep_agent** を実行します。
- サポートされている非 ASE データベースの Replication Agent の場合、Replication Agent で **suspend** の次に **resume** を実行します。

ディストリビュータ・スレッドの読み込み効率の向上

ディストリビュータ (DIST) スレッドを有効にしてステーブル・キュー・スレッド (SQT) キャッシュから SQL 文を直接読み取ります。これにより、SQT からの負荷、および SQT と DIST の間の依存性が減少し、SQT と DIST の両方の効率が向上します。

この強化機能を使用するには、**dist_direct_cache_read** パラメータに **configure replication server** を付けて実行します。

次のように入力します。

```
configure replication server
set dist_direct_cache_read to 'on'
```

デフォルトでは、**dist_direct_cache_read** は 'off' に設定されています。このパラメータを無効にすると、ディストリビュータ・スレッドは、メッセージ・キュー経由で SQT から SQL 文を要求します。その結果、インバウンドとアウトバウンドのキューに競合が発生します。

dist_direct_cache_read は、サーバ・レベル・パラメータです。パラメータを有効または無効にした後は、Replication Server を再起動する必要があります。

メモリ割り付けの強化

Replication Server でメモリ割り付けを大きな単位で行うには、**mem_reduce_malloc** パラメータに **configure replication server** を付けて使用します。

これにより、必要とされるメモリ割り付け回数を削減し、Replication Server のパフォーマンスを向上させます。

次のように入力します。

```
configure replication server
set mem_reduce_malloc to 'on'
```

デフォルトでは、**mem_reduce_malloc** は 'off' に設定されています。

mem_reduce_malloc は、動的パラメータです。パラメータ設定を変更しても、データベース・コネクションのサスペンドとレジュームを行う必要はありません。

キュー・ブロック・サイズの増加

キュー・ブロック・サイズを増やしてレプリケーション・パフォーマンスを改善します。

キュー・ブロック・サイズはステーブル・キュー構造で使用される連続メモリ・ブロックのバイト数です。キュー・ブロック・サイズを大きく設定すると、Replication Server で、1 つのブロックでより多くのトランザクションを処理できるようになります。キュー・ブロック・サイズをデフォルトの 16KB から 32KB、64KB、128KB、または 256KB に増やすことができます。パフォーマンスの向上はトランザクション・プロファイルと環境にも依存します。

注意： キュー・ブロック・サイズの増加機能を使用するには、REP_HVAR_ASE という名前の Advanced Services Options ライセンスが必要です。

推奨事項

次のことを行うことを強くお勧めします。

- キュー・ブロック・サイズを増やす前に十分なメモリがあることを確認する。
- その複写システムに最適な値を決めるために異なったキュー・ブロック・サイズを試す。

制限

- キュー・ブロック・サイズの変更を実行している最中に Replication Server にデータが流れ込まないようにしてください。
- サブスクリプションのマテリアライゼーション、マテリアライゼーション解除、またはルートの作成や破棄を実行している最中にキュー・ブロック・サイ

ズは変更できません。Replication Server は処理を続けますが、キュー・ブロック・サイズの変更はエラー・メッセージで終了します。
- いったんキュー・ブロック・サイズを変更する手順を開始すると、Replication Server はその処理が完了するまで、別のキュー・ブロック・サイズの変更コマンドを受け付けません。
- RSSD で直接キュー・ブロック・サイズを変更する別の手順を使用しないでください。異なった手順の使用によってキュー・ブロック・サイズの設定に一貫性がなくなり、Replication Server が停止する場合があります。

注意： キュー・ブロック・サイズの変更後はすべてのキューが空になります。

キュー・ブロック・サイズを変更する

キュー・ブロック・サイズの変更は Replication Server の設定における主要な変更であり、Replication Server へのすべてのコネクションに影響します。ログ転送をサスペンドして、Replication Server をクワイス状態にする必要があります。

Replication Server のキュー・ブロック・サイズの変更手順では、「アップストリーム」は Replication Server にデータを送るすべての複写システム・コンポーネントを、「ダウンストリーム」は Replication Server からデータを受け取るコンポーネントを意味します。

1. データの整合性を維持するために、キュー・ブロック・サイズを変更する前に設定対象の Replication Server にデータが流れ込むのを止めます。
   a) すべての Replication Agent から設定を変更する Replication Server へのログ転送をサスペンドします。
   b) Replication Agent からのアップストリームのログ転送をすべてサスペンドします。
   c) すべてのアップストリーム Replication Server をクワイス状態にします。
   d) 設定を変更する Replication Server への受信ルートをすべてサスペンドします。
   e) 設定を変更する Replication Server をクワイス状態にします。
2. **configure replication server** と一緒に **set block_size to** '*value*' 句を使用して、設定対象の Replication Server のキュー・ブロック・サイズを設定します。

   このコマンドは、次の処理を実行します。
   - 進行中のサブスクリプション・マテリアライゼーションが存在しないことを確認します。
   - すべてのログ転送がサスペンドされていることを確認します。
   - すべての受信ルートがサスペンドされていることを確認します。
   - Replication Server がクワイス状態であることを確認します。
   - キューをパージします。

- `rs_locater` RSSD システム・テーブル内の値がゼロになって、キュー・ブロック・サイズ変更手順を開始したときにレプリケート・データベースに適用されていない可能性のあるトランザクションを Replication Agent が再送信できるようになります。
- キュー・ブロック・サイズが入力された値に設定されます。
- (省略可能) **with shutdown** オプションを含めた場合は、Replication Server が停止します。キュー・ブロック・サイズの変更は Replication Server を再起動すると有効になります。停止によって Replication Server は確実にすべてのメモリをクリアします。

3. キュー・ブロック・サイズを大きい値に変更すると、小さいブロック・サイズ値で作成したロー・パーティションが削除されるか、削除後に再作成されます。

   ブロック・サイズの変更後にパーティションを作成した場合にのみ、パーティションの正しいセグメント数が登録されます。
4. データ・フローのレジューム:

   a) **with shutdown** オプションを使用した場合は、Replication Server を再起動します。
   b) Replication Agent からのログ転送をレジュームします。
   c) 受信ルートをすべてレジュームします。
5. すべてのダウンストリーム Replication Server の RSSD とデータ・サーバでデータ・ロスを調べます。通常は、設定を変更した Replication Server の RSSD でデータ・ロスがあります。設定を変更した Replication Server の RSSD からデータを受け取るレプリケート RSSD でのデータ・ロスは無視します。

   データ・サーバでデータ・ロスを修復する手順に従います。RSSD でデータ・ロスがあった場合、影響を受けた Replication Server のログに次のようなメッセージが表示されます。

```
E. 2010/02/12 14:12:58. ERROR #6067 SQM(102:0 primaryDS.rssd) - /
sqmoqid.c(1071)
Loss detected for replicateDS.rssd from primaryDS.RSSD
```

   replicateDS は、レプリケート・データ・サーバの名前で、*primaryDS* は、プライマリ・データ・サーバの名前です。

単純な複写システムでキュー・ブロック・サイズを増やす

単純な複写システムで、プライマリ Replication Server とレプリケート Replication Server のキュー・ブロック・サイズを設定します。

複写システムの構成は次のとおりです。

- プライマリ・データベース – `pdb`
- レプリケート・データベース – `rdb`

- プライマリ Replication Server ― PRS
- プライマリ Replication Server の RSSD – `pRSSD`
- レプリケート Replication Server ― RRS
- レプリケート Replication Server の RSSD – `rRSSD`

```
          pRSSD       rRSSD
            |           |
pdb -----> PRS -----> RRS -----> rdb
```

この例では、RSSD は、Replication Server システム・データベース (RSSD) として機能する Adaptive Server と Embedded Replication Server システム・データベース (ERSSD) として機能する SQL Anywhere® の両方を意味します。すべてのコマンドの完全な構文、例、使用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。

1. プライマリ Replication Server を設定する

a) すべての Replication Agent からのログ転送をサスペンドします。プライマリ Replication Server で次のコマンドを実行します。

```
suspend log transfer from all
```

b) プライマリ Replication Server をクワイス状態にします。

```
admin quiesce_force_rsi
```

c) プライマリ Replication Server のキュー・ブロック・サイズを 64KB に設定します。

```
configure replication server
set block_size to '64'
```

(省略可能) ブロック・サイズの設定で **with shutdown** オプションを使用して、プライマリ Replication Server を停止します。次に例を示します。

```
configure replication server
set block_size to '64' with shutdown
```

d) トランザクション・ログを調べて、プライマリ Replication Server がマテリアライズ中でないこと、ログ転送とルートがサスペンドされていること、プライマリ Replication Server がクワイス状態であることを確認します。

e) プライマリ Replication Server を停止した場合は、再起動します。『Replication Server 管理ガイド第 1 巻』の「複写システムの管理」の「Replication Server の起動」を参照してください。

f) プライマリ Replication Server のトランザクション・ログを調べて、ブロック・サイズが変更されたことを確認します。

g) Replication Agent がプライマリ Replication Server に接続できるように、ログ転送をレジュームします。プライマリ Replication Server で次のコマンドを実行します。

```
resume log transfer from all
```

h) レプリケート Replication Server のログ・ファイルでデータ・ロスに関する情報を調べます。レプリケート Replication Server で **ignore loss** コマンドを実行して、プライマリ Replication Server の RSSD からレプリケート Replication Server の RSSD へのデータ・ロスを無視します。

```
ignore loss from PRS.pRSSD to RRS.rRSSD
```

2. レプリケート Replication Server を設定する

a) すべての Replication Agent からのログ転送をサスペンドします。プライマリ Replication Server とレプリケート Replication Server で次のコマンドを実行します。

```
suspend log transfer from all
```

b) プライマリ Replication Server をクワイス状態にします。

```
admin quiesce_force_rsi
```

c) レプリケート Replication Server へのルートを持つすべての Replication Server でルートをサスペンドします。

```
suspend route to RRS
```

d) レプリケート Replication Server をクワイス状態にします。

```
admin quiesce_force_rsi
```

e) レプリケート Replication Server のブロック・サイズを 64KB に設定します。

```
configure replication server
set block_size to '64'
```

(省略可能) **with shutdown** オプションを使用して、レプリケート Replication Server を停止します。次に例を示します。

```
configure replication server
set block_size to '64' with shutdown
```

f) トランザクション・ログを調べて、レプリケート Replication Server がマテリアライズ中でないこと、ログ転送とルートがサスペンドされていること、レプリケート Replication Server がクワイス状態であることを確認します。

g) レプリケート Replication Server を停止した場合は、再起動します。

h) レプリケート Replication Server のトランザクション・ログを調べて、ブロック・サイズが変更されたことを確認します。

i) Replication Agent がレプリケート Replication Server に接続できるように、ログ転送をレジュームします。レプリケート Replication Server で次のコマンドを実行します。

```
resume log transfer from all
```

j) Replication Agent がプライマリ Replication Server に接続できるように、ログ転送をレジュームします。プライマリ Replication Server で次のコマンドを実行します。

```
resume log transfer from all
```

k) サスペンドしたルートをレジュームします。
```
resume route to RRS
```
l) プライマリ Replication Server とレプリケート Replication Server のログ・ファイルでデータ・ロスに関する情報を調べます。レプリケート RSSD がプライマリ RSSD に複写されている場合は、プライマリ Replication Server で **ignore loss** コマンドを実行して、プライマリ RSSD とレプリケート RSSD の間のデータ・ロスを無視します。
```
ignore loss from RRS.rRSSD to PRS.pRSSD
```

参照：
- ロスの無視 (425 ページ)

中間ルートを持つ複写システムでキュー・ブロック・サイズを増やす

次の中間ルートを持つ複写システムで、プライマリ Replication Server のキュー・ブロック・サイズを設定します。

複写システムの構成は次のとおりです。

- プライマリ・データベース – `pdb`
- レプリケート・データベース – `rdb`
- プライマリ Replication Server ― PRS
- プライマリ Replication Server の RSSD – `pRSSD`
- レプリケート Replication Server ― RRS
- レプリケート Replication Server の RSSD – `rRSSD`
- 中間 Replication Server ― IRS
- 中間 Replication Server の RSSD – `iRSSD`

```
          pRSSD       iRSSD       rRSSD
            |           |           |
pdb -----> PRS -----> IRS -----> RRS -----> rdb
```

この例では、RSSD は、Replication Server システム・データベース (RSSD) として機能する Adaptive Server と Embedded Replication Server システム・データベース (ERSSD) として機能する SQL Anywhere の両方を意味します。すべてのコマンドの完全な構文、例、使用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。

1. すべての Replication Agent からのログ転送をサスペンドします。プライマリ Replication Server で次のコマンドを実行します。
```
suspend log transfer from all
```

2. PRS をクワイス状態にします。
```
admin quiesce_force_rsi
```
3. プライマリ Replication Server のブロック・サイズを 64KB に設定します。
```
configure replication server
set block_size to '64'
```
(省略可能) ブロック・サイズの設定で **with shutdown** オプションを使用して、プライマリ Replication Server を停止します。次に例を示します。
```
configure replication server
set block_size to '64' with shutdown
```
4. トランザクション・ログを調べて、プライマリ Replication Server がマテリアライズ中でないこと、ログ転送とルートがサスペンドされていること、プライマリ Replication Server がクワイス状態であることを確認します。
5. プライマリ Replication Server を停止した場合は、再起動します。『Replication Server 管理ガイド第 1 巻』の「複写システムの管理」の「Replication Server の起動」を参照してください。
6. プライマリ Replication Server のトランザクション・ログを調べて、ブロック・サイズが変更されたことを確認します。
7. Replication Agent がプライマリ Replication Server に接続できるように、ログ転送をレジュームします。プライマリ Replication Server で次のコマンドを実行します。
```
resume log transfer from all
```
8. 中間 Replication Server とレプリケート Replication Server のログ・ファイルでデータ・ロスに関する情報を調べます。中間 Replication Server で **ignore loss** コマンドを 2 回実行して、プライマリ Replication Server の RSSD からレプリケート Replication Server の RSSD、およびプライマリ Replication Server の RSSD から中間 Replication Server の RSSD へのデータ・ロスを無視します。
```
ignore loss from PRS.pRSSD to RRS
go
ignore loss from PRS.pRSSD to IRS.iRSSD
```

参照：
- ロスの無視 (425 ページ)

マルチパス・レプリケーション

複数のレプリケーション・パスを使用して、レプリケーションのスループットとパフォーマンスを向上させ、競合を低減します。

シングルパス・レプリケーション環境では、トランザクションはプライマリ・データベースからレプリケート・データベースに連続して複写されるため、プラ

イマリ・データベースのトランザクションのコミット順が確保され、このためレプリケート・データベースとプライマリ・データベースの一貫性が保たれます。トランザクションをレプリケート・データベースに適用する逐次モードは、複数のアプリケーションがプライマリ・データベースで並列してそれぞれのトランザクションが実行されていたり、複数のプライマリ・データベースから到着するトランザクションがある場合でも変更されません。

同じプライマリ・データベースから生じたすべてのトランザクションを直列化することなく、テーブルのサブセット内でデータの一貫性を維持できる複写環境があります。この環境の典型例は、異なるデータのセットにアクセスする異なるアプリケーションで 1 つのプライマリ・データベースが変更される場合です。特定のアプリケーション内で変更されたテーブルのサブセット内の異なるデータのセットは、引き続き連続して複写されます。異なるテーブルのサブセットのデータは並列して複写することができます。

マルチパス・レプリケーションでは、さまざまなストリームを介したデータのレプリケーションをサポートすると同時に、パス内でのデータ整合性を維持しますが、さまざまなパス間でのコミット順には従いません。

レプリケーション・パスは、Replication Server とプライマリ・データベースまたはレプリケート・データベースの間のコンポーネントとモジュールをすべて含んでいます。マルチパス・レプリケーションでは、プライマリ・データベースから 1 つまたは複数の Replication Server への複数の Replication Agent コネクションのために、複数のプライマリ・レプリケーション・パスを作成できます。また、1 つまたは複数の Replication Server からレプリケート・データベースへのコネクションのために、複数のレプリケート・パスを作成できます。マルチパス・レプリケーションは、ウォーム・スタンバイ環境と Multi-Site Availability (MSA) 環境で設定できます。トランザクションを Replication Server 間の専用ルートで伝達して、共有ルート上での輻輳を回避できます。また、プライマリ・データベースから Replication Server を経由してレプリケート・データベースに至るエンドツーエンドのレプリケーション・パスをオブジェクト (テーブルやストアド・プロシージャなど) 専用にすることができます。

ライセンス
マルチパス・レプリケーションは、Advanced Services Option の一部としてライセンスされます。『Replication Server インストール・ガイド』の「インストールの計画」の「ライセンスの取得」を参照してください。

システムの稼働条件
Replication Server では、プライマリ・データ・サーバが Adaptive Server 15.7 以降の Adaptive Server データベース間でのマルチパス・レプリケーションをサポートします。マルチパス・レプリケーション・システムの Adaptive Server 以外のデータ

ベースの場合、『Replication Server 異機種間複写ガイド』で、次を参照してください。

- 「レプリケート・データ・サーバとしての Sybase IQ」の「Sybase IQ へのマルチパス・レプリケーション」
- 「異機種間におけるマルチパス・レプリケーション」

マルチパス・レプリケーション・クイック・スタート

エンドツーエンドの複写のために、2 つのプライマリとレプリケートのパスで構成されるマルチパス・レプリケーション・レプリケーション・システムをセットアップします。

1. 2 つの複写パスを介して複写するテーブルまたはストアド・プロシージャを 2 セット作成または選択します。
2. **rs_init** を使用して、プライマリとレプリケートの各 Adaptive Server データベースを複写システムに追加します。
3. マルチスレッド RepAgent を有効にします。
   プライマリ Adaptive Server で、次を入力します。
   ```
   sp_config_rep_agent primary_database_name, 'multithread rep
   agent', 'true'
   ```
4. RepAgent への複写パス数を設定します。
   たとえば、2 つのパスを有効にするには、次を入力します。
   ```
   sp_config_rep_agent primary_database_name, 'max number of
   replication paths', '2'
   ```
5. プライマリ データベースから Replication Server への代替複写パスを作成します。
   a) *alternate_path_name* という名前の代替物理 RepAgent 複写パスを作成します。
      プライマリ Adaptive Server で、次を入力します。
      ```
      sp_replication_path "primary_database_name", 'add',
      "alternate_path_name", "repserver_name",
      "repserver_user", "repserver_password"
      ```
   b) Replication Server からプライマリ・データベースへの対応する代替プライマリ・コネクションを作成し、同じ RepAgent の複写パス名 (*alternate_path_name*) を使用して、代替物理 RepAgent の複写パスにバインドします。
      Replication Server で次のように入力します。
      ```
      create alternate connection to
      primary_dataserver.primary_database
      named primary_dataserver.alternate_path_name
      set error class to rs_sqlserver_error_class
      set function string class to rs_sqlserver_function_class
      set username to primary_db_maintenance_user
      ```

```
set password to primary_db_maintenance_password
with primary only
```

複写システムには、2 つのプライマリ複写パス (デフォルトと *alternate_path_name*) が含まれます。

6. 同じ代替複写パス名 (*alternate_path_name*) を使用して、Replication Server からレプリケート・データベースへの代替レプリケート・コネクションを作成します。

```
create alternate connection to
replicate_dataserver.replicate_database
named replicate_dataserver.alternate_path_name
```

複写システムには、2 つのレプリケート複写パス (デフォルトと *alternate_path_name*) が含まれます。

7. テーブルやストアド・プロシージャなど 1 セットのオブジェクトを代替複写パスにバインドします。

```
sp_replication_path pdb, 'bind', "table",
"[table_owner].table_name", "alternate_path_name"
```

他のセットのオブジェクトは、デフォルトの複写パスを使用します。オブジェクトを代替複写パスにバインドすることしかできません。代わりに、代替複写パスにバインドしないすべてのオブジェクトはデフォルト・パスを使用します。

8. プライマリ・データベースに複写定義を作成します。
たとえば、`authors` テーブルに **authors_rep** 複写定義を作成します。

```
create replication definition authors_rep
with primary at primary_dataserver.primary_database
with all tables named 'authors'
...
go
```

デフォルトのプライマリ・コネクションと代替プライマリ・コネクションが異なる Replication Server に存在する場合、それぞれの Replication Server に複写定義を作成します。

9. デフォルトのプライマリ・コネクションとデフォルトのレプリケート・コネクションに対してサブスクリプションを作成します。

```
create subscription subscription_default_path for
replication_definition
with primary at primary_dataserver.primary_database
with replicate at replicate_dataserver.replicate_database
```

10. 代替プライマリ・コネクションと代替レプリケート・コネクションに対してサブスクリプションを作成します。

```
create subscription subscription_alternate_path for
replication_definition
with primary at primary_dataserver.alternate_path_name
with replicate at replicate_dataserver.alternate_path_name
```

並列トランザクション・ストリーム

マルチパス・レプリケーションは、トランザクションが並行ストリームに分割され、異なるストリーム全体で順番にコミットされない限り、複写パフォーマンスを向上させることができます。

トランザクションを並列レプリケーション・パスに分割して輻輳を削減することにより、複写のパフォーマンスを向上させることができます。トランザクション属性または派生データ値などの並列化ルールに従って、トランザクションを分割できます。たとえば、次の方法を使用できます。

- テーブルまたはストアド・プロシージャなどの特定オブジェクトにパスを割り当てます。オブジェクトをパスにバインドする場合、Replication Agent はパスを介してそのオブジェクトに実行する複写可能なアクションを、複数のレプリケーション・パス設定で定義する Replication Servers に送信します。Adaptive Server の RepAgent および Replication Agent for Oracle では、この複写分散モードがサポートされています。
- プライマリ・データベースでクライアント・コネクションのセッション ID 別にトランザクションを分割します。Adaptive Server の RepAgent は、クライアント・コネクションによるトランザクションの分散をサポートしています。
- 各パスへの Replication Server を使用します。
- オブジェクトをパスにバインドするか、Replication Server 間に専用ルートを作成して、優先度の高い複写に対し専用パスまたは輻輳の少ないパスを割り当てます。

マルチパス・レプリケーションの分散モード

マルチパス・レプリケーション環境では、さまざまな分散モードを使用して並列レプリケーションとレプリケーション・パフォーマンスの向上を達成できます。具体的には、プライマリ・データベースから始まる使用可能なプライマリ・レプリケーション・パスを介してプライマリ・データベースのレプリケーション負荷を分散します。

複数のレプリケーション・パスを使用すると、以下のいずれかを選択できます。

- **オブジェクトのバインド別分散** – 複数のオブジェクト (テーブルやストアド・プロシージャなど) を特定のレプリケーション・パスにバインドすることで、これらオブジェクトの並列レプリケーションを有効にします。
- **コネクション別分散** – Adaptive Server RepAgent はさまざまなクライアント・プロセスから発生したトランザクションを使用可能なレプリケーション・パスに割り当てます。

デフォルト・モードは、オブジェクト・バインド別分散です。RepAgent では、一度に複数の分散モードは使用できません。

参照：
- 分散モードの設定 (318 ページ)

オブジェクトのバインド別分散
マルチパス・レプリケーション環境では、テーブルやストアド・プロシージャなどのオブジェクトを 1 つまたは複数のプライマリ・レプリケーション・パスにバインドできます。

オブジェクトをパスにバインドする場合、RepAgent はパスを介してそのオブジェクトに実行する複写可能なアクションを、複数のレプリケーション・パス設定で定義する Replication Servers に送信します。オブジェクトをパスにバインドしない場合は、デフォルトのパスを使用してオブジェクトをデフォルト・パスで定義した Replication Server に送信します。オブジェクトはデフォルト・パスにバインドできません。このため、オブジェクトをデフォルト・パスを介して送信する場合は、何もする必要はありません。バインドされたオブジェクトはレプリケーション中、常に同じパスに従います。

データベースのサポート
Replication Server では、Adaptive Server バージョン 15.7 以降のプライマリ・データベースのマルチパス・レプリケーションについて、オブジェクト・バインド別分散をサポートしています。

コネクション別分散
マルチパス・レプリケーション環境では、データベースから始まる使用可能なプライマリ・レプリケーション・パスを介して複数のクライアント・コネクションからプライマリ・データベースへのレプリケーション負荷を分散できます。

プライマリ・データベースへの各クライアント・コネクションと各サーバ・プロセスには、ユニークなシステム・プロセス ID (spid) が付いています。Adaptive Server RepAgent は、データベース・トランザクション・ログに格納された spid の値を使用して、特定のクライアント・コネクションまたはサーバ・プロセスで実行したトランザクションを識別します。クライアントが切断して再接続すると、クライアントの spid が変更されます。

コネクション別分散では、Adaptive Server RepAgent がさまざまなクライアント・プロセスから発生したトランザクションを使用可能なレプリケーション・パスに割り当てます。時間の経過とともに、使用可能なパス全体でデータ分散のバランスが取れていく傾向があります。使用可能な RepAgent パスがさらにあり、クライアント・プロセスの数が多い場合、レプリケーション・パフォーマンスが向上し、レプリケーション負荷分散はより均一化します。

コネクション別分散を設定する場合、RepAgent は spid と使用可能なレプリケーション・パスの数に従ってレプリケーション・パスを介してトランザクションを分散します。特定の spid によって実行されたトランザクションはすべて、同じレ

プリケーション・パスを必ず使用します。時間の経過とともに、spid の分散は均一化する傾向があるため、各パスに割り当てられた spid の数はほぼ同じになります。3 人のユーザ (user1、user2、user3 の spid がそれぞれ 0、1、5) と 3 つのレプリケーション・パス (デフォルト、PDS.pdb1_conn1、PDS.pdb1_conn2) があるとします。この例では次のようになります。

- user1 spid 0 によって行われるトランザクションはすべてデフォルトのコネクションを経由する。
- user2 spid 1 によって行われるトランザクションはすべて PDS.pdb1_conn1 コネクションを経由する。
- user3 spid 5 によって行われるトランザクションはすべて PDS.pdb1_conn2 コネクションを経由する。

user1 によって生成されたトランザクションは PDS.pdb1_conn1 または PDS.pdb1_conn2 を介して複写できず、user2 によって生成されたトランザクションはデフォルトまたは PDS.pdb1_conn1 コネクションを介して複写できません。

新しいパスを使用できるときに、RepAgent を再起動して新しいパスを認識しない場合、RepAgent は引き続きトランザクションを既存のパスに分散します。RepAgent を再起動すると、RepAgent はすべてのパスの設定を再ロードし、パスの数と送信先への変更が有効になります。トランザクションは別のパスを経由して送信されるか、異なる送信先 Replication Server に移動します。ただし、トランザクションが異なるパスを使用する場合は、重複するトランザクションが発生することがあります。RepAgent がコネクション別分散を使用している場合は、プライマリ・データ・サーバのレプリケーション・パスの送信先 Replication Server を変更しないことをおすすめします。

コネクション別分散を有効にすると、RepAgent はバインドを使用せず、バインドは効果がありません。**sp_replication_path** を使用するときに定義したオブジェクトのバインドを引き続き表示し、確認することができます。次の値を変更する場合は、RepAgent を再起動し、Replication Server をクワイス状態にする必要があります。

- 分散モード – 現在の分散モードがコネクションである場合、RepAgent は起動中にレプリケーション・パスをロードするときにオブジェクトのバインドを処理しません。
- レプリケーション・トポロジ – RepAgent を再起動し、Replication Server をクワイス状態にしない場合、データが消失または重複する場合があります。

データベースのサポート
Replication Server では、Adaptive Server バージョン 15.7 ESD #1 以降のプライマリ・データベースのマルチパス・レプリケーションについて、コネクション別分散をサポートしています。

コネクション別分散に対する制限事項
コネクション・モード別分散には、いくつかの制限事項があります。

- コネクション別分散では、トランザクションはコネクション間で順番に送信されません。1 つのコネクション上のトランザクションは、別のコネクション上の隣接するトランザクションと比べて、プライマリ・データベース・ログのトランザクション順とは異なる順序でレプリケート・データベースに送信されることがあります。
- コネクション別分散を使用する場合、均一したトランザクション分散と負荷バランスを確保できない恐れがあります。
  - RepAgent 送信者スレッドは、ラージ・トランザクションが一部の送信者に偏って分配されていると、満杯にある場合があります。
  - RepAgent 送信者はビジー状態であっても、トランザクションを送信側キューに入れることができます。
- コネクション別分散モデルは、ユーザまたはアプリケーション・セッションに関連付けられた SPID を使用して、複写データをパス全体に分散します。ユーザ・セッションに関連付けられた SPID が同じであれば、そのユーザまたはアプリケーションによるすべてのトランザクションで同じパスが使用されます。ユーザまたはアプリケーションが RepAgent として同じデータ・サーバに直接接続する場合、SPID は変更されず、トランザクションの逐次化の問題は発生しません。
  ただし、ユーザまたはアプリケーションが接続プールを持つ中間層のアプリケーション・サーバを使用して、RepAgent が接続するデータ・サーバ上のトランザクションを実行する場合は、トランザクションの実行に使用する SPID が変更されることがあります。たとえば、ユーザからの **insert** トランザクションの後に同じデータへの **update** トランザクションが続く場合、**insert** トランザクションの SPID は **update** トランザクションの SPID とは異なる場合があります。この状況は、中間層のアプリケーション・サーバが異なるコネクションを使用して各トランザクションを実行するために起こることがあります。この場合、RepAgent は異なるパス上の **update** と **insert** を送信し、**update** が **insert** の前にレプリケート・データベースに受信されると、トランザクションの逐次化の問題が発生することがあります。

デフォルトおよび代替コネクション

マルチパス・レプリケーションでは、コネクションには、デフォルトおよび 1 つ以上の代替コネクションが含まれます。

Replication Agent からのデータを受け入れるコネクションはプライマリ・コネクションで、データをデータベースに適用するコネクションはレプリケート・コネクションです。デフォルトまたは代替コネクションは、プライマリまたはレプリケート・コネクションのいずれかです。

デフォルトのコネクションは、データベースを複写ドメインに追加する際に、Replication Server から特定のプライマリ・データベースまたはレプリケート・データベースに作成するコネクションです。データ・サーバが Adaptive Server またはサポートされている ASE 以外のサーバであるかに応じて、**rs_init**、Replication Manager の Sybase Central プラグイン、**create connection**、または **create connection ... using profile** を使用して、デフォルトのコネクションを作成できます。

デフォルトのコネクションは、データ・サーバ名およびデータベース名を *dataserver.database* の形式で、コネクション名として使用します。ここで *dataserver* および *database* は、それぞれ実際のデータ・サーバ名およびデータベース名です。

必須のデフォルトのコネクションを作成後、複数の代替コネクションを作成できます。各代替コネクションには、それぞれユニークな名前が必要です。

代替コネクションを作成後、コネクションのプロパティを変更、またはコネクションを削除できます。すべてのコネクションのステータスを表示し、そのコネクションのサブスクリプションを作成することもできます。

代替コネクションを作成する場合、ユーザ ID が有効なユーザであることが必要です。Sybase IQ レプリケート・データベースにコネクションを作成する場合、デフォルトのコネクションおよび各代替コネクションに対し、コネクション・プロファイル、コネクション・プロファイルのバージョン、およびユニークなメンテナンス・ユーザ名を指定する必要があります。

レプリケート・データベースへの複数コネクション

Replication Server からレプリケート・データベースに複数のコネクションを作成します。

レプリケート・データベースに複数のコネクションを作成すると、レプリケーション・コネクションごとに 1 つのトランザクション・ストリームに対してサブスクリプションが作成されます。同じストリームのトランザクションは、プライマリのコミット順を順守します。別のストリームのトランザクションは並列適用され、プライマリのコミット順ではない場合があります。

デフォルトおよび代替レプリケート・コネクション
また、複数の Replication Server から同じレプリケーション・ドメイン内の同じレプリケート・データベースに対して、レプリケート・コネクションを作成することもできます。レプリケーション・ドメインの 1 つの Replication Server だけが、デフォルトのレプリケート・コネクションを所有および制御することができます。複数のデフォルトのレプリケート・コネクションをドメインの他の Replication Server から作成することはできません。他の Replication Server は代替レプリケート・コネクションのみを持つことができます。

代替レプリケート・コネクションを作成したら、コネクション・プロパティを変更したり、コネクションを削除したりすることができます。また、すべてのコネクションのステータスを表示し、そのコネクションのサブスクリプションを作成することもできます。

代替レプリケート・コネクションの作成

create alternate connection を使用して、Replication Server からレプリケート・データベースへの代替コネクションを作成します。

次のように入力します。

```
create alternate connection to dataserver.database
named conn_server.conn_db
[set error class [to] error_class
set function string class [to] function_class
set username [to] user
set password [to] pwd]
[set database_param [to] 'value']
```

構文の説明は次のとおりです。

- *dataserver* および *database* – レプリケート・データ・サーバとデータベースです。
- *conn_server.conn_db* – データ・サーバ名とコネクション名で構成される代替レプリケート・コネクション。
  - *conn_server* が *dataserver* と異なる場合は、interface ファイルに *conn_server* のエントリが必要です。
  - *conn_server* が *dataserver* と同じ場合は、*conn_db* が *database* と異なるようにしてください。
  - 各レプリケート・コネクション名は、複写システム内でユニークにしてください。
- **set function string class [to]** *function_class*、**set username [to]** *user*、および **set password [to]** *pwd* は **alter connection** と **create connection** の既存の句で、代替コネクションを作成するときに使用できます。
  - これらの句を省略すると、代替レプリケート・コネクションは、デフォルトのレプリケート・コネクションを使用して設定した値を継承します。

    - デフォルトの接続を制御する (コントローラ) Replication Server とは異なる (現在の) Replication Server に代替コネクションを作成するときに、これらの句を省略すると、現在の Replication Server はエラーを返します。
    - 代替コネクションは、同じ Replication Server が代替とデフォルトの両方のコネクションを制御する場合にのみ、デフォルト接続から値を継承できます。
    - 代替コネクションのメンテナンス・ユーザを設定しない場合は、デフォルト接続のメンテナンス・ユーザが継承されます。代替コネクションは、代替コネクションに指定した新しいメンテナンス・ユーザを使用します。
- *set param* – **alter connection** と **create connection** の既存の句で、オプションのコネクション・パラメータに使用できます。
    - 代替レプリケート・コネクションに対して設定した値は、デフォルト接続から継承された値またはデフォルト値で上書きされます。
    - 代替コネクションは、同じ Replication Server が代替とデフォルトの両方のコネクションを制御する場合にのみ、デフォルト接続から値を継承できます。

たとえば、`FINANCE_DS2.rdb_conn2` という名前の代替レプリケート・コネクションを `FINANCE_DS` データ・サーバの `rdb` レプリケート・データベースに作成する場合、次のように入力します。

```
create alternate connection to FINANCE_DS.rdb
named FINANCE_DS2.rdb_conn2
go
```

注意： FINANCE_DS と FINANCE_DS2 を `interfaces` または `sql.ini` ファイルで定義する必要があります。

代替レプリケート・コネクションの変更または削除

alter connection コマンドと **drop connection** コマンドを使用して、デフォルトまたは代替コネクションを変更または削除します。

コマンドで指定するデータ・サーバ名とデータベース名は、デフォルトまたは代替レプリケート・コネクション名にすることができます。

代替またはデフォルトのレプリケート・コネクションを設定するときに、**alter connection** で使用可能な設定パラメータを使用することができます。

たとえば、**dsi_max_xacts_in_group** を `TOKYO_DS.rdb_conn2` 代替レプリケート・コネクションに対して 40 に設定する場合、次のように入力します。

```
alter connection to TOKYO_DS.rdb_conn2
set dsi_max_xacts_in_group to '40'
go
```

レプリケート・コネクションに関する情報の表示

replicate パラメータを **admin show_connections** と一緒に使用して、すべてのレプリケート・コネクションに関する情報を表示します。

たとえば、FINANCE_DS データ・サーバと NY_DS データ・サーバでレプリケート・データベースを制御する Replication Server で、次のように入力します。

```
admin show_connections, 'replicate'
```

次のようなメッセージが表示されます。

```
Connection Name         Server        Database   User
-------------------     ----------    --------   ----------
FINANCE_DS.fin_rdb      FINANCE_DS    fin_rdb    rdb_maint
NY_DS.ny_rdb_conn2      NY_DS         ny_rdb     rdb_maint
```

FINANCE_DS.fin_rdb は、コネクション名がデータ・サーバとデータベース名の組み合わせに一致するため、Replication Server と FINANCE_DS データ・サーバの `fin_rdb` データベースの間のデフォルトのコネクションになります。

NY_DS.ny_db_conn2 は、コネクション名がデータ・サーバとデータベース名の組み合わせに一致しないため、Replication Server と NY_DS データ・サーバの `ny_rdb` データベースの間の代替コネクションになります。

オプションで、`rs_databases` システム・テーブルを使用して、Replication Server に対するデフォルトおよび代替コネクションの両方を一覧表示することもできます。

複数のレプリケート・コネクションを持つ複写システムの作成

デフォルトおよび代替のレプリケート・コネクションを作成し、対応するサブスクリプションを作成して、複数のレプリケート・コネクション複写システムを構築します。

前提条件

トランザクションを並列で実行できることを確認してから、レプリケート・トランザクションを 2 つのセットに分けます。

手順

このシナリオ例は、複数のレプリケート・コネクションを持つ複写システムを作成する際のモデルに使用できます。ここでは `PDS` プライマリ・データ・サーバ上の `pdb` プライマリ・データベースに、`T1` および `T2` テーブルと、対応する **repdef1** および **repdef2** 複写定義が含まれています。各テーブルに影響を与えるトランザクション・セットがあります。対応するサブスクリプションは、**sub1** および **sub2** です。`rdb` レプリケート・データベースは `RDS` レプリケート・データ・サーバにあり、プライマリ Replication Server およびレプリケート Replication Server は RS1 および RS1 となります。

1. RS1 で、**rs_init** または **create connection** を使用して、レプリケート・データベースに対するデフォルトのレプリケート・コネクションを作成します。

```
create connection to RDS.rdb
using profile ase_to_ase;standard
set username to rdb_maint
set password to rdb_maint_ps
go
```

2. RS1 で、RDS.rdb1 という名前の代替レプリケート・コネクションを rdb レプリケート・データベースに対して作成します。

```
create alternate connection to RDS.rdb
named RDS.rdb1
go
```

   オプションで、別の代替レプリケート・コネクションを RS2 からのレプリケート・データベースに対して作成します。RS2 で、次のように入力します。

```
create connection to RDS.rdb
named RDS.rdb2
set error class to rs_sqlserver_error_class
set function string class to rs_sqlserver_function_class
set username to rdb_maint
set password to rdb_maint_ps
go
```

3. **sub1** サブスクリプションを作成し、最初のトランザクション・セットでレプリケート・トランザクションに対するデフォルトのレプリケート・コネクションを指定します。

```
create subscription sub1 for repdef1
with replicate at RDS.rdb
go
```

4. **sub2** サブスクリプションを作成し、2 つ目のトランザクション・セットでレプリケート・トランザクションに対する代替レプリケート・コネクションを指定します。

```
create subscription sub2 for repdef2
with replicate at RDS.rdb2
go
```

プライマリ・データベースからの複数コネクション

RepAgent パスをプライマリ・データベースから Replication Server に関連付けることができるプライマリ・データベースへの Replication Server からの複数コネクションを作成し、管理します。

代替プライマリ・コネクションの作成

create alternate connection を使用して、Replication Server からプライマリ・データベースへの代替コネクションを作成します。

次のように入力します。

```
create alternate connection to dataserver.database
named conn_server.conn_db
[with {log transfer on | primary only}]
```

構文の説明は次のとおりです。

- *dataserver* および *database* – プライマリ・データ・サーバとデータベースです。
- *conn_server.conn_db* – データ・サーバ名とコネクション名で構成される代替プライマリ・コネクション名です。
  - *conn_server* が *dataserver* と同じ場合は、*conn_db* が *database* と異なるようにしてください。
  - *conn_server.conn_db* は、Replication Agent と Replication Server 間のコネクション名が一致している必要があります。
  - 各プライマリ・コネクション名は、複写システム内でユニークにしてください。
- **with log transfer on** – *dataserver.database* で指定したデータベースへの代替プライマリ・コネクションと代替レプリケート・コネクションを、両方とも *conn_server.conn_db* で指定した名前で作成するように Replication Server に指示します。
- **primary only** – *conn_server.conn_db* で指定した名前で、プライマリ・データベースへの代替プライマリ・コネクションのみを作成するように Replication Server に指示します。

たとえば、SALES_DS.pdb_conn2 という名前の代替プライマリ・コネクションを SALES_DS データ・サーバの pdb データベースに作成するには、次のように入力します。

```
create alternate connection to SALES_DS.pdb
named SALES_DS.pdb_conn2
with primary only
go
```

代替プライマリ・コネクションの変更または削除

既存の **alter connection** コマンドと **drop connection** コマンドをそれぞれ使用して、デフォルトまたは代替プライマリ・コネクションを変更または削除します。

たとえば、**alter connection** を使用して、*dataserver.database* で指定するプライマリ・データベースへのデフォルトのプライマリ・コネクションを有効または無効にします。

```
alter connection to dataserver.database
set primary only [on|off]
```

レプリケート・コネクションを有効にするには、off に設定します。

プライマリ・コネクションに関する情報の表示

primary パラメータを **admin show_connections** と一緒に使用して、すべてのプライマリ・コネクションに関する情報を表示します。

たとえば、SALES_DS データ・サーバでプライマリ・データベースを制御する Replication Server で、次のように入力します。

```
admin show_connections, 'primary'
```

次のようなメッセージが表示されます。

```
Connection Name     Server         Database     User
----------------    ----------     ---------    ----------
SALES_DS.pdb        SALES_DS       pdb          pdb_maint
SALES_DS.pdb_conn2  SALES_DS       pdb          pdb_maint
```

SALES_DS.pdb は、コネクション名がデータ・サーバとデータベース名の組み合わせに一致するため、Replication Server と SALES_DS データ・サーバの pdb データベースの間のデフォルトのコネクションになります。

SALES_DS.pdb_conn2 は、コネクション名がデータ・サーバとデータベース名の組み合わせに一致しないため、Replication Server と SALES_DS データ・サーバの pdb データベースの間の代替コネクションになります。

オプションで、rs_databases システム・テーブルを使用して、Replication Server に対するデフォルトおよび代替コネクションの両方を一覧表示することもできます。

複写定義およびサブスクリプション

複写定義とサブスクリプションを使用して、複数の代替コネクション間のレプリケーションを定義します。

代替コネクションの複写定義とサブスクリプション

プライマリ・データベースについて作成された複写定義は、複写定義を管理する Replication Server とプライマリ・データベース間のすべてのプライマリ・コネクション、デフォルト・コネクション、および代替コネクションに適用されます。したがって、プライマリ・データベースへの最後のプライマリ・コネクションを削除する前に、プライマリ・データベースのすべての複写定義を削除する必要があります。

システム・バージョン 1570 では、データベースに対してのみ複写定義とパブリケーションを作成できます。**with primary at** 句 (**create replication definition** コマンド) について指定する名前は、プライマリ・データベースの名前である必要があります。

プライマリ・データベースと Replication Server との間のすべてのプライマリ・コネクションが、すべての複写定義を共有しているので、どのプライマリ・コネクションがデータ・ソースであり、どのレプリケート・コネクションがレプリケーション先なのかをサブスクリプションで指定してください。対応するデフォルトまたは代替コネクション名を **create subscription** の **with primary** および **with replicate** 句で指定します。**with primary** 句を使用してコネクション名を指定しないと、Replication Server でプライマリ・データベースへのデフォルトのプライマリ・コネクションに対するサブスクリプションが作成されます。

```
create subscription sub_name
for {table_repdef | func_repdef | publication pub |
database replication definition db_repdef}
with primary at data_server.database
with replicate at data_server.database
[where {column_name | @param_name}
       {< | > | >= | <= | = | &} value
[and {column_name | @param_name}
     {< | > | >= | <= | = | &} value]...]
[without holdlock | incrementally | without materialization]
[subscribe to truncate table]
[for new articles]
```

代替コネクションがサポートされていない Replication Server のバージョンからアップグレードすると、すべてのサブスクリプションはアップグレードされた Replication Server のデフォルトのプライマリ・コネクションおよびデフォルトのレプリケート・コネクションに対して定義されたままとなります。

例 1 – 代替プライマリ・コネクションのサブスクリプション

LON_DS.pdb_conn2 代替プライマリ・コネクションで、NY_DS.rdb がデフォルトのレプリケート・コネクションとなる LON_DS プライマリ・データ・サーバに対する **sub_conn2** サブスクリプションを **repdef_conn2** 複写定義に対して作成するには、次のように入力します。

```
create subscription sub_conn2 for repdef_conn2
with primary at LON_DS.pdb_conn2
with replicate at NY_DS.rdb
without materialization
go
```

例 2 – 代替レプリケート・コネクションのサブスクリプション

NY_DS.rdb_conn2 代替レプリケート・コネクションで **sub_conn2** サブスクリプションを **repdef_conn2** 複写定義に対して作成するには、次のように入力します。

```
create subscription sub_conn2 for repdef_conn2
with replicate at NY_DS.rdb_conn2
without materialization
go
```

コネクション間でのサブスクリプションの移動

alter subscription を使用すると、再マテリアライズせずに、同じ Replication Server を使用する同じレプリケート・データベースのレプリケート・コネクション間のサブスクリプションを移動することができます。

alter subscription をレプリケートの Replication Server で実行します。

```
alter subscription sub_name
for {table_repdef|func_repdef|{{publication pub|
database replication definition db_repdef}
with primary at primary_dataserver_name.primary_database_name}}
move replicate from ds_name.db_name
to ds_name1.db_name1
```

サブスクリプションを *ds_name.db_name* レプリケート・コネクションから *ds_name1.db_name1* レプリケート・コネクションに移動します。

たとえば、**rep1** 複写定義の **sub1** サブスクリプションを `RDS.rdb1` コネクションから `RDS.rdb2` コネクションに移動するには、次のように入力します。

```
alter subscription sub1 for rep1
move replicate from RDS.rdb1
to RDS.rdb2
```

プライマリ Replication Server のバージョンが 1570 より前の場合は、**alter subscription** を使用できません。代わりに、目的のコネクションでサブスクリプションを削除して再作成してください。

同じパスを使用してレプリケートしなければならない複数のサブスクリプションを移動するには、プライマリ・コネクションのログ転送をサスペンドし、サブスクリプションをすべて移動してからログ転送を再開してください。

複数のプライマリ・レプリケーション・パス

プライマリ・データベースから 1 つまたは複数の Replication Server への複数のプライマリ・レプリケーション・パスを作成し、レプリケーションのスループットを向上させて競合を避けるか、データを別の Replication Server にルーティングします。

各プライマリ・レプリケーション・パスは、プライマリ・データベースから Replication Server への RepAgent パスおよび Replication Server からプライマリ・データベースへの関連プライマリ・コネクションで構成されます。テーブルやストアド・プロシージャなどのオブジェクトを 1 つまたは複数のパスにバインドできます。

物理パスは、パスにバインドするデータを受信する Replication Server と、同じ Replication Server に接続する RepAgent 送信者スレッドを定義します。同じコネクション名を使用して、プライマリ・データベースから Replication Server への RepAgent 物理パスを Replication Serve からプライマリ・データベースへの対応するコネクションに関連付けます。

論理パスは、データを複数の Replication Server に分散するために、単一の名前の下で 1 つまたは複数の物理パスをまとめます。テーブルを複数の送信先にレプリケートする必要がある場合は、テーブルを各送信先の物理パスにバインドするのでなく、関連する物理パスをグループ化する論理パスにバインドします。

複数のプライマリ・レプリケーション・パスの作成

プライマリ・データベースから 1 つまたは複数の Replication Server へのプライマリ・レプリケーション・パスを複数作成できます。各プライマリ・パスは、RepAgent パスと関連するプライマリ・コネクションで構成されます。

1. マルチスレッド RepAgent と複数のレプリケーション・パスの RepAgent を有効にします。
2. **rs_init** を使用してデフォルトのプライマリ・コネクションと RepAgent レプリケーション・パスを作成します。
3. 各代替プライマリ・レプリケーション・パスを作成します。各パスは、代替 RepAgent レプリケーション・パスにリンクされた代替プライマリ・コネクションで構成されます。
4. レプリケーション・パスからレプリケーションの分散モードを設定します。
   - コネクション別分散
   - オブジェクトのバインド別分散

マルチスレッド RepAgent および RepAgent の複数のパスを有効にします。

マルチスレッド RepAgent を有効にし、プライマリ・データベースから追加パスを使用するように設定します。

デフォルトで、Adaptive Server RepAgent はプライマリ・データベース・ログをスキャンして、LTL を生成し、生成した LTL を Replication Server に送信する単一のスレッドで構成されます。マルチスレッド RepAgent では、スキャンおよび送信アクティビティは別々のスレッドで実行されます。

マルチスレッド RepAgent を使用する場合は、データを Replication Server に送信するデフォルトのパスが常に存在します。RepAgent は、最初に RepAgent をデータベースで有効にするときにデフォルトのパスを作成します。

複数のレプリケーション・パスは、Adaptive Server 15.7 以降で生成される SQL 文の複写トランザクションのみを処理します。

1. RepAgent が使用可能なメモリを設定する

   複数の RepAgent 送信者スレッドを有効にして設定する前に、Adaptive Server の RepAgent スレッドに十分なメモリを確保しておく必要があります。
2. マルチスレッド RepAgent を有効にする

   RepAgent スキャナと送信者アクティビティに対して別々のスレッドを使用するマルチスレッド RepAgent を有効または無効にします。
3. 送信バッファ数を設定する

   マルチスレッド RepAgent のスキャナと送信者タスクが使用できる送信バッファの最大数を設定します。
4. RepAgent のレプリケーション・パスの最大数を設定する

   RepAgent がプライマリ・データベースからデータのレプリケートに使用できるパスの最大数を設定します。RepAgent は、各 RepAgent パスに対して 1 つの RepAgent 送信者スレッドを生成します。

5. 設定パラメータ設定の表示
   Adaptive Server ストアド・プロシージャを使用して、RepAgent 設定パラメータの設定と RepAgent マルチスレッドと複数のパスのステータスに関する他の情報を表示します。

使用可能なメモリの RepAgent への設定

複数の RepAgent 送信者スレッドを有効にして設定する前に、Adaptive Server の RepAgent スレッドに十分なメモリを確保しておく必要があります。

Adaptive Server の RepAgent スレッド専用のメモリ・プールのデフォルトのサイズは 4096 ページです。

1. 現在の RepAgent スレッド・プールのサイズと、他の RepAgent スレッド・パラメータの設定を表示します。プライマリ Adaptive Server で、次を入力します。

```
sp_configure 'Rep Agent Thread administration'
go
```

   次のようなメッセージが表示されます。

```
Group: Rep Agent Thread Administration

Parameter       Default  Memory  Config  Run    Unit      Type
Name                     Used    Value   Value

enable rep      0        0       1       1      switch    dynamic
agent threads

replication     4096     8194    4096    4096   memory    dynamic
agent memory                                    pages(2k)
size
```

   この例は、**enable rep agent threads** がスイッチのオン/オフを切り換える動的パラメータであることを示します。動的パラメータの変更に、RepAgent の再起動は必要ありません。
2. Adaptive Server が RepAgent スレッド・プールに割り付けるメモリを変更します。
   たとえば、プール・サイズを 8194 ページに設定するには、プライマリ Adaptive Server で次のように入力します。

```
sp_configure 'replication agent memory size', 8194
go
```

   次のようなメッセージが表示されます。

```
Group: Rep Agent Thread Administration

Parameter       Default  Memory  Config  Run    Unit      Type
Name                     Used    Value   Value

replication     4096     16430   8194    8194   memory    dynamic
agent memory                                    pages(2k)
```

```
size
(1 row affected)
Configuration option changed. ASE need not be rebooted since the
option is dynamic.
Changing the value of 'replication agent memory size' to '8194'
increases the amount of memory ASE uses by 8236 K.
```

マルチスレッド RepAgent の有効化

RepAgent スキャナと送信者アクティビティに対して別々のスレッドを使用するマルチスレッド RepAgent を有効または無効にします。

プライマリ Adaptive Server にログインし、次のように入力します。

```
sp_config_rep_agent dbname, 'multithread rep agent', {'true' |
'false'}
```

ここで、*dbname* は Adaptive Server プライマリ・データベースです。

マルチスレッド RepAgent を有効にするには、true に設定します。デフォルトは false です。変更内容を有効にするには、RepAgent を再起動する必要があります。

送信バッファの数の設定

マルチスレッド RepAgent のスキャナと送信者タスクが使用できる送信バッファの最大数を設定します。

送信バッファの数は、マルチスレッド RepAgent を有効にするとき、またはマルチパス・レプリケーション用に RepAgent を有効にして設定するプロセスを完了した後でも設定できます。

プライマリ Adaptive Server で、次を入力します。

```
sp_config_rep_agent dbname, 'number of send buffers',
'num_of_send_buffers'
```

ここで、*dbname* は Adaptive Server プライマリ・データベースです。

たとえば、pdb1 データベースで送信バッファの数を 40 に設定するには、次のように入力します。

```
sp_config_rep_agent pdb1, 'number of send buffers', '40'
```

number of send buffers のデフォルトは 50 バッファです。設定できる値の範囲は、1 ～ MAXINT (2,147,483,647) の間です。パラメータは動的で、RepAgent の再起動は必要ありません。

各送信バッファのサイズは同じで、**send buffer size** RepAgent パラメータを使用して設定できます。『Replication Server リファレンス・マニュアル』の「Adaptive Server コマンドとシステム・プロシージャ」の「**sp_config_rep_agent**」を参照してください。

RepAgent 用のレプリケーション・パスの最大数の設定

RepAgent がプライマリ・データベースからデータのレプリケートに使用できるパスの最大数を設定します。RepAgent は、各 RepAgent パスに対して 1 つの RepAgent 送信者スレッドを生成します。

プライマリ Adaptive Server で、次を入力します。

```
sp_config_rep_agent dbname, 'max number replication paths', 'max number replication paths value'
```

ここで、*dbname* は Adaptive Server プライマリ・データベースです。

たとえば、*max number replication paths* を `pdb1` データベースで 3 に設定するには、次のように入力します。

```
sp_config_rep_agent pdb1, 'max number replication paths', '3'
```

max number replication paths が 1 より大きい場合、RepAgent はパス専用にバインドしないすべての複写オブジェクトのデフォルトのパスを引き続き使用します。

max number replication paths がパスにバインドされる複写オブジェクトを持つパスの数より少ない場合は、RepAgent はエラーを報告し終了します。

設定パラメータ設定の表示

Adaptive Server ストアド・プロシージャを使用して、RepAgent 設定パラメータの設定と RepAgent マルチスレッドと複数のパスのステータスに関する他の情報を表示します。

- **sp_config_rep_agent** – **sp_config_rep_agent** で設定するパラメータの設定を表示するデータベース名のみを指定します。
- **sp_help_rep_agent** – RepAgent のステータスに関する追加情報を表示するには、次のように指定します。
  - **send** – RepAgent に割り当てた送信バッファの数を表示する。
  - **config** – RepAgent の複数のパスの設定パラメータに関する情報を表示する。
  - **process** – **multithread rep agent** を有効にしたときの複数の Rep Agent プロセスに関する情報を表示する。
- **sp_who** – Adaptive Server で稼働中の RepAgent プロセスとスレッドに関する情報を表示します。

sp_config_rep_agent および **sp_help_rep_agent** の詳細については、『Replication Server リファレンス・マニュアル』の「Adaptive Server コマンドとシステム・プロシージャ」を参照してください。

『Adaptive Server リファレンス・マニュアル：プロシージャ』の「システム・プロシージャ」の「**sp_who**」を参照してください。

プライマリ・データベースの代替レプリケーション・パスの作成

add パラメータを **sp_replication_path** と一緒に使用し、RepAgent レプリケーション・パスを Replication Server からのプライマリ・コネクションと関連付けることによって、プライマリ・データベースと Replication Server の間に代替物理パスを作成します。

前提条件

rs_init を使用して、プライマリ・データベースと Replication Server の間にデフォルトのレプリケーション・パスを作成します。

手順

PDS プライマリ・データ・サーバー、pdb データベース、RS1 および RS2 Replication Server で構成されるレプリケーション・システム例を使用して、プライマリ・データベース上に 2 つの代替レプリケーション・パスを作成し、デフォルトのプライマリ・レプリケーション・パスを含む合計 3 つのプライマリ・レプリケーション・パスを作成します。

1. pdb と pdb_1 という RS2 の間に、代替プライマリ・レプリケーション・パスを作成します。
   a) pdb と RS2 の間に、pdb_1 という名前の代替物理レプリケーション・パスを作成します。
   PDS で次のように入力します。
   ```
   sp_replication_path "pdb", 'add', "pdb_1", "RS2", "RS2_user",
   "RS2_password"
   ```
   b) RS2 から pdb に、pdb_1 という名前の対応する代替プライマリ・コネクションを作成します。
   RS2 で次のように入力します。
   ```
   create alternate connection to PDS.pdb
   named PDS.pdb_1
   set error class to rs_sqlserver_error_class
   set function string class to rs_sqlserver_function_class
   set username to pdb1_maint
   set password to pdb1_maint_ps
   with primary only
   ```
2. pdb と pdb_2 という RS1 の間に、もう 1 つのプライマリ・レプリケーション・パスを作成します。
   a) pdb と RS1 の間に、pdb_2 という名前の代替物理レプリケーション・パスを作成します。
   PDS で次のように入力します。

```
sp_replication_path "pdb", 'add', "pdb_2", "RS1", "RS1_user",
"RS1_password"
```

b) RS1 から pdb に、pdb_2 という名前の対応する代替プライマリ・コネクションを作成します。
RS1 で次のように入力します。

```
create alternate connection to PDS.pdb
named PDS.pdb_2
with primary only
```

Replication Server 定義の削除

drop パラメータを **sp_replication_path** と一緒に使用して、送信先としての Replication Server を、デフォルトのプライマリ・レプリケーションのパスでない物理レプリケーションのパスから削除します。

パスにバインドされているオブジェクトがある場合、デフォルトのプライマリ・レプリケーション・パスを削除したり、プライマリ・レプリケーション・パスを削除することはできません。
送信先としての RS1 を削除するには、PDS で次のように入力します。

```
sp_replication_path 'pdb', 'drop', "RS1"
```

論理プライマリ・レプリケーション・パスの作成

add パラメータと **logical** パラメータを **sp_replication_path** と併用して論理プライマリ・レプリケーション・パスを作成し、複数の Replication Server への物理パスにバインドされているデータやオブジェクト・バウンドを分散させることができます。

前提条件

論理プライマリ・レプリケーション・パスをサポートするために、関連した物理プライマリ・レプリケーション・パスを作成します。

手順

dt1 次元テーブルを pdb_1 にバインドするなどのように、レプリケーション・オブジェクトをバインドした場合、dt1 は常に pdb_1 から RS2 へと移動します。レプリケーション・システム例と 3 つの物理プライマリ・レプリケーション・パス—デフォルト、pdb_1、pdb_2 を使用して logical_1 という名前の論理レプリケーション・パスを作成し、pdb_2 から RS2 へ dt1 を分散させることができます。

注意： デフォルトのパスを論理パスに追加することはできません。

1. logical_1 論理パスを作成し、物理プライマリ・レプリケーション・パスとして pdb_1 を追加します。
   PDS で次のように入力します。

```
sp_replication_path 'pdb', 'add', 'logical', 'logical_1', 'pdb_1'
```

logical_1 は pdb_1 から RS1 にのみデータを送信します。

2. pdb_2 を logical_1 の物理プライマリ・レプリケーション・パスとして追加します。
   PDS で次のように入力します。

```
sp_replication_path 'pdb, 'add', 'logical', 'logical_1', 'pdb_2'
```

   logical_1 は pdb_1 から RS1 に、また pdb_2 から RS2 にデータを送信します。

論理プライマリ・レプリケーション・パスの要素の削除

drop パラメータと **logical** パラメータを **sp_replication_path** と一緒に使用して、論理レプリケーション・パスから要素を削除します。

次の例では、logical_1 論理パスに pdb_1 および pdb_2 という物理パスが含まれており、logical_1 の要素と呼ばれています。この論理パスから要素を削除することができます。

警告！ 既存の論理パスからパスを削除したり、既存の論理パスにパスを追加した場合、送信先のセットが変更され、変更前に複写オブジェクトが送られていた送信先に複写オブジェクトが到達しなくなることがあります。

1. logical_1 から pdb_1 を削除します。

```
sp_replication_path 'pdb', 'drop', 'logical', 'logical_1',
'pdb_1'
```

   これで、logical_1 論理パスに、pdb_2 物理パスのみが含まれるようになります。logical_1 にバインドされているすべてのオブジェクトは pdb_2 からのみ複写されるようになります。

2. logical_1 から pdb_2 を削除します。

```
sp_replication_path 'pdb', 'drop', 'logical', 'logical_1',
'pdb_2'
```

   最後の要素を削除して、論理パスにバインドされているオブジェクトがない場合、論理パスが要素なしで存在することはできないため、Replication Server で最後の要素と論理パス全体が一緒に削除されます。

論理パスの削除

drop パラメータと **logical** パラメータを **sp_replication_path** と一緒に使用して、論理レプリケーション・パス全体を削除します。

論理パスを全部削除する場合、コマンドに Replication Server や要素を指定しないでください。論理パスで最後の要素を削除すると、Replication Server は論理パス全体を削除します。

注意： オブジェクトがバインドされている場合は、論理パスや物理パスを削除できません。

logical_1 論理パスを削除するには、次のように入力します。

```
sp_replication_path 'pdb', 'drop', 'logical', 'logical_1'
```

分散モードの設定

プライマリ Adaptive Server データベースから複数のプライマリ・レプリケーション・パスを介してレプリケーションの分散モードを設定します。

前提条件

プライマリ Adaptive Server から Replication Server へのデフォルト・コネクションと代替コネクションを作成し、マルチスレッド RepAgent を有効にします。

手順

オブジェクト・バインド別分散からコネクション別分散に変更すると、RepAgent はすべてのオブジェクトのバインドを無視し、警告を表示します。オブジェクト・バインド別分散に戻して RepAgent を再起動すると、バインドが保持されます。

1. 分散モードを設定します。
   sp_config_rep_agent *データベース*、'multipath distribution model'、{'connection' | 'object'}

   構文の説明は次のとおりです。
   - **multipath distribution model** – **sp_config_rep_agent** の分散モード・パラメータ
   - **connection** – モードをコネクション別分散に設定
   - **object** – モードをオブジェクト・バインド別分散に設定 (デフォルト)
2. Replication Server をクワイス状態にし、RepAgent を再起動します。

   『Replication Server 管理ガイド 第 1 巻』の「複写システムの管理」にある「Replication Server のクワイス」の「複写システムのクワイス」を参照してください。

レプリケーション・パスへのオブジェクトのバインド

bind パラメータを **sp_replication_path** と一緒に使用して、オブジェクトを物理または論理プライマリ・レプリケーション・パスに関連付けます。バインドされたオブジェクトはレプリケーション中、常に同じパスに従います。

オブジェクトをプライマリ・レプリケーション・パスにバインドするには、次のように入力します。

```
sp_replication_path dbname, 'bind', "object_type",
"[table_owner].object_name", "path_name"
```

構文の説明は次のとおりです。

- *object_type* – **table** または **sproc** (ストアド・プロシージャ)。

- *[table_owner.]object_name* – テーブル名またはストアド・プロシージャ名です。

  注意： オブジェクトがテーブルの場合にテーブル所有者を指定しなければ、dbo、すなわちデータベース所有者が所有しているテーブルにのみバインドが適用されます。

- *path_name* – 物理パスまたは論理パスの名前です。

次にバインド例を示します。

- t1 テーブルを pdb_2 レプリケーション・パスに：

```
sp_replication_path pdb, 'bind', "table", "t1", "pdb_2"
```

- owner1 が所有する t2 テーブルを pdb_2 レプリケーション・パスに：

```
sp_replication_path pdb, 'bind', "table", "owner1.t2", "pdb_2"
```

- **sproc1** ストアド・プロシージャを pdb_2 レプリケーション・パスに：

```
sp_replication_path pdb, 'bind', "sproc", "sproc1", "pdb_2"
```

- dt1 次元テーブル・オブジェクトを論理パスのすべての場所に：

```
sp_replication_path pdb, 'bind', "table", "dt1", "everywhere"
```

オプションで、アスタリスク "*" またはパーセント "%" ワイルドカード文字、または両方の組み合わせを *object_name* で使用して、パスにバインドする名前の範囲または一致する文字を指定します。たとえば、さまざまなワイルドカード文字の組み合わせと一致する名前のテーブルを pdb_2 レプリケーション・パスにバインドするには、以下のコマンドを実行します。

- `sp_replication_path pdb, 'bind', 'table', 'a*', "pdb_2"`
- `sp_replication_path pdb, 'bind', 'table', 'au%rs', "pdb_2"`
- `sp_replication_path pdb, 'bind', 'table', 'a*th%s', "pdb_2"`
- `sp_replication_path pdb, 'bind', 'table', 'authors%', "pdb_2"`

レプリケーション・パスからのオブジェクトのバインド解除

unbind パラメータを **sp_replication_path** と一緒に使用して、バインドされたオブジェクトと物理または論理レプリケーション・パス間の関連付けを削除します。

オブジェクトと 1 つ以上のプライマリ・レプリケーションのパスとのバインドを削除するには、次のように入力します。

```
sp_replication_path dbname, 'unbind', "object_type", "object_name",
{"path_name"|all}
```

構文の説明は次のとおりです。

- *object_type* – オブジェクトの種類で、**path**、**table**、または **sproc** (ストアド・プロシージャ) を指定します。
- *[table_owner.]object_name* – テーブル名、ストアド・プロシージャ、またはバインドを解除するパスです。

注意： オブジェクトがテーブルの場合にテーブル所有者を指定しなければ、dbo、すなわちデータベース所有者が所有しているテーブルにのみバインドが適用されます。

- *path_name* | **all** – 物理パス名か論理パス名、またはすべてのパスを指定します。**path** を *object_type* と指定し、パス名を *object_name* と入力し、**all** オプションを指定した場合、指定したパス名からすべてのオブジェクトがバインド解除されます。

次に例を示します。

- `t1` テーブルのバインドを pdb_2 レプリケーション・パスから削除するには、次のコマンドを実行します。

```
sp_replication_path pdb, 'unbind', "table", "t1", "pdb_2"
```

- `t1` テーブルのすべてのバインドを削除するには、次のコマンドを実行します。

```
sp_replication_path pdb, 'unbind', "table", "t1", "all"
```

- pdb_2 レプリケーション・パスに対するすべてのオブジェクトのバインドを削除するには、次のコマンドを実行します。

```
sp_replication_path pdb, 'unbind', 'path', 'pdb_2', "all"
```

SQL 文および DDL 文のオブジェクトのバインドとレプリケーション

デフォルトのパスまたはパスにバインドされていない任意のオブジェクトのすべてのパスに SQL 文および DDL 文を送信することができます。

SQL 文のレプリケーションと DDL のレプリケーションを使用すると、テーブルなどのオブジェクトを特定のレプリケーション・パスにバインドしたときに、そのオブジェクトが含まれるすべての SQL または DDL 文で指定したレプリケーション・パスが使用されます。SQL 文または DDL 文は、パスにバインドしていないオブジェクトが文に含まれる場合、デフォルトのレプリケーション・パスか、すべてのレプリケーション・パスを使用します。

ddl path for unbound objects を **sp_config_rep_agent** と一緒に使用して、バインドされていないオブジェクトの SQL 文または DDL 文をすべてのパスまたはデフォルトのパスに送信します。

```
sp_config_rep_agent dbname, 'ddl path for unbound objects', {'all' |
'default'}
```

デフォルト設定は **all** です。

オブジェクトのバインドとデータベースの再同期

マルチパス・レプリケーションは、使用可能なすべてのレプリケーション・パスに、**resync**、**resync purge**、および **resync init** データベース再同期マーカを送信します。

『Replication Server 管理ガイド第 2 巻』の「複写システム・リカバリ」の「Adaptive Server のレプリケート・データベースの再同期」の「データベースの再同期を設

定する」の「データベース再同期マーカを Replication Server に送信する」を参照してください。

オブジェクトのバインドと rs_ticket

マルチパス・レプリケーションは、使用可能なすべてのレプリケーション・パスから **rs_ticket** の実行結果を送信します。関連するデータを入手するには、データをフィルタする必要があります。

『Replication Server リファレンス・マニュアル』の「RSSD ストアド・プロシージャ」の「**rs_ticket**」を参照してください。

レプリケーション・パスでの設定値の変更

config パラメータを **sp_replication_path** と一緒に使用して、代替レプリケーション・パスのパラメータ値を設定します。

代替レプリケーション・パスでは、パスワードとユーザ ID のみを変更できます。

代替パスのパラメータの値を変更するには、次のように入力します。

```
sp_replication_path dbname, 'config', "path_name",
 "config_parameter_name", "config_value"
```

ここで *config_parameter_name* は **rs_username** または **rs_password** です。

次に変更例を示します。

- `pdb_1` が RS1 に接続する際に使用するユーザ名を "RS1_user" に変更する場合は、PDS で次のように入力します。

  ```
  sp_replication_path pdb, 'config', "pdb_1", "rs_username",
  "RS1_user"
  ```

- `pdb_1` が RS1 に接続するために使用するパスワードを "january" に変更するには、PDS で次のように入力します。

  ```
  sp_replication_path pdb, 'config', "pdb_1", "rs password",
  "january"
  ```

sp_config_rep_agent を使用して、デフォルトのレプリケーション・パスのパラメータを設定します。『Replication Server リファレンス・マニュアル』の「Adaptive Server コマンドとシステム・プロシージャ」の「**sp_config_rep_agent**」を参照してください。

レプリケーション・パスに関する情報の表示

プライマリ・データベースで **list** パラメータを **sp_replication_path** と一緒に使用すると、バインドおよびレプリケーション・オブジェクトに関する情報が表示されます。

```
sp_replication_path dbname, 'list', ['object_type'], ['object_name']
```

- *object_type* – オブジェクトの種類を指定します。種類は次のとおりです。**path**、**table**、**sproc** (ストアド・プロシージャ)。
- *object_name* – 特定のオブジェクトのバインド関係を表示します。オブジェクト名を指定する場合は、*object_type* を指定する必要があります。

例 1

バインドされたすべてのオブジェクトのパスの関係を表示する場合は、*object_type* または *object_name* を指定しないでください。

```
sp_replication_path 'pdb','list'
go
```

次のようなメッセージが表示されます。

```
Binding               Type     Path
--------------------- -------- --------------------
dbo.dt1               T        everywhere
dbo.sproc1            P        pdb_1
dbo.sproc1            P        pdb_2
dbo.t1                T        pdb_2
dbo.t2                T        pdb_1

(5 rows affected)

Logical Path                  Physical Path
----------------------------- ---------------------
everywhere                    pdb_1
everywhere                    pdb_2

(2 rows affected)
Physical Path                  Destination
------------------------------ -------------------
pdb_1                           RS2
pdb_2                           RS1

(2 rows affected)
(return status = 0)
```

例 2

バインドされたテーブルすべてに関する情報を表示するには、次のコマンドを実行します。

```
sp_replication_path 'pdb','list','table'
go
```

次のようなメッセージが表示されます。

```
Binding               Type     Path
--------------------- -------- --------------------
dbo.dt1               T        everywhere
dbo.t1                T        pdb_2
dbo.t2                T        pdb_1

(3 rows affected)
(return status = 0)
```

例 3

すべてのストアド・プロシージャに関する情報を表示するには、次のコマンドを実行します。

```
sp_replication_path 'pdb','list','sproc'
go
```

次のようなメッセージが表示されます。

```
Binding               Type     Path
--------------------- -------- --------------------
dbo.sproc1            P        pdb_2
dbo.sproc1            P        pdb_1
dbo.sproc2            P        pdb_1

(3 rows affected)
(return status = 0)
```

例 4

sproc1 ストアド・プロシージャに関する情報のみを表示するには、次のコマンドを実行します。

```
sp_replication_path 'pdb','list','sproc','sproc1'
go
```

次のようなメッセージが表示されます。

```
Binding               Type     Path
--------------------- -------- --------------------
dbo.sproc1            P        pdb_2
dbo.sproc1            P        pdb_1

(2 rows affected)
(return status = 0)
```

例 5

すべてのレプリケーション・パスに関する情報を表示するには、次のコマンドを実行します。

```
sp_replication_path 'pdb','list','path'
go
```

次のようなメッセージが表示されます。

```
Path                  Type     Binding
--------------------- -------- ----------------------
everywhere            T        dbo.dt1
pdb_1                 P        dbo.sproc1
pdb_1                 T        dbo.t2
pdb_2                 P        dbo.sproc1
pdb_2                 T        dbo.t1

(5 rows affected)
Logical Path                   Physical Path
------------------------------ -------------------------
everywhere                     pdb_1
```

```
everywhere                     pdb_2

(2 rows affected)
Physical Path                  Destination
------------------------------ ----------------------
pdb_1                          RS2
pdb_2                          RS1

(2 rows affected)
(return status = 0)
```

例 6

"everywhere" 論理レプリケーション・パスに関する情報のみを表示するには、次のコマンドを実行します。

```
sp_replication_path 'pdb','list','path','everywhere'
go
```

次のようなメッセージが表示されます。

```
Path                 Type     Binding
-------------------- -------- -------------------------
everywhere           T        dbo.dt1

(1 rows affected)
Logical Path                   Physical Path
------------------------------ -------------------------
everywhere                     pdb_1
everywhere                     pdb_2

(2 rows affected)
Physical Path                  Destination
------------------------------ -------------------------
pdb_1                          RS2
pdb_2                          RS1

(2 rows affected)
(return status = 0)
```

注意： 論理パスの基になる物理パスも表示されます。

例 7

pdb_1 物理パスに関する情報のみを表示するには、次のコマンドを実行します。

```
sp_replication_path 'pdb','list','path','pdb_1'
go
```

次のようなメッセージが表示されます。

```
Path                 Type     Binding
-------------------- -------- --------------------
pdb_1                P        dbo.sproc1
pdb_1                T        dbo.t2

(2 rows affected)
Physical Path                   Destination
```

```
------------------------------ --------------------
pdb_1                          RS2

(1 rows affected)
(return status = 0)
```

MSA 環境での複数のレプリケーション・パスの作成

複写定義とサブスクリプションを使用して、レプリケート・データベースとプライマリ・データベースのプライマリ・コネクションのレプリケート・コネクションをバインドし、MSA 環境に 2 つの完全なレプリケーション・パスを作成します。

1. トランザクションを 2 つのセットに分け、このトランザクションが並列実行できるようにします。

   たとえば、トランザクションをテーブルやストアド・プロシージャなどの 2 つのオブジェクトのセットに分割することができます。
2. プライマリ・データベースに対してデフォルトのプライマリ・コネクションを作成し、レプリケート・データベースに対してデフォルトのレプリケート・コネクションを作成します。
3. プライマリ・データベースに対して代替プライマリ・コネクションを作成し、レプリケート・データベースに対して代替レプリケート・コネクションを作成します。
4. マルチスレッドの RepAgent と RepAgent 用の 2 つのレプリケーション・パスを有効にし、このオブジェクトをレプリケーション・パスにバインドします。
5. プライマリ・データベースに複写定義を作成します。

   デフォルトのプライマリ・コネクションと代替プライマリ・コネクションが異なる Replication Server に存在する場合、それぞれの Replication Server に複写定義を作成します。
6. デフォルトのプライマリ・コネクションとデフォルトのレプリケート・コネクションに対してサブスクリプションを作成します。
7. 代替プライマリ・コネクションと代替レプリケート・コネクションに対してサブスクリプションを作成します。

ウォーム・スタンバイ環境での複数のレプリケーション・パス

代替コネクションと代替論理コネクションを使用するか、エンドツーエンドのレプリケーション・パスを構築して、ウォーム・スタンバイ環境のレプリケーションのパフォーマンスを改善することができます。

ウォーム・スタンバイ環境での代替論理コネクションの作成

create alternate logical connection を使用して、ウォーム・スタンバイ環境での既存のデフォルト論理コネクションの代替論理コネクションを作成します。

さまざまな Replication Server を使用して、デフォルトの論理コネクションと代替論理コネクションを制御できます。アクティブとスタンバイの両方のデータベースが複数の Replication Agent の使用をサポートしている必要があります。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場合、代替、デフォルトなどすべての論理コネクションで **switch active command** を実行します。ウォーム・スタンバイのプロセスは、すべてのパスの切り替えが終わると完了します。

ウォーム・スタンバイのペアを管理する Replication Server で、次のように入力します。

```
create alternate logical connection to LDS.ldb
named conn_lds.conn_ldb
```

構文の説明は次のとおりです。

- *LDS* および *ldb* – 論理データ・サーバ名およびデータベース名
- *conn_lds.conn_ldb* – 代替論理コネクション名のデータ・サーバおよびデータベース・コネクションのコンポーネント。

ウォーム・スタンバイ環境での代替コネクションの作成

アクティブなデータベースの代替プライマリ・コネクションを作成するか、代替論理コネクションのスタンバイ・データベースに対して代替レプリケート・コネクションを作成します。

ウォーム・スタンバイのペアを管理する Replication Server で、次のように入力します。

```
create alternate connection to ds_name.db_name
named conn_server.conn_db
...
[as {active|standby} for conn_lds.conn_ldb]
```

構文の説明は次のとおりです。

- *ds_name.db_name* – データ・サーバ名およびアクティブまたはスタンバイのデータベース名。
- *conn_server.conn_db* – 代替のアクティブまたはスタンバイ・コネクションで、データ・サーバ名とコネクション名で構成されます。受信される Replication Agent コネクションをサポートするには、*conn_ds* が *ds_name* と同じである必要があります。
- *conn_lds.conn_ldb* – 代替論理コネクション名のデータ・サーバおよびデータベース・コネクションのコンポーネント。

- **active | standby** – アクティブなデータベースに代替コネクションを作成するか、スタンバイ・データベースに代替コネクションを作成するかを指定します。

ウォーム・スタンバイ環境での複数のレプリケーション・パスの作成

論理コネクションを使用して、スタンバイ・データベースとアクティブなデータベースのプライマリ・コネクションのレプリケート・コネクションをバインドし、ウォーム・スタンバイ環境でアクティブなデータベースとスタンバイ・データベース間に 2 つの完全なレプリケーション・パスを作成します。

1. トランザクションを 2 つのセットに分け、2 つのセットのトランザクションが並列実行できるようにします。
   たとえば、トランザクションをテーブルやストアド・プロシージャなどの 2 つのオブジェクトのセットに分割することができます。
2. マルチスレッドの RepAgent とアクティブなデータベースおよびレプリケート・データベースの両方の RepAgent の 2 つのレプリケーション・パスを有効にして、そのオブジェクトをレプリケーション・パスにバインドします。
3. 論理コネクションを作成します。『Replication Server リファレンス・マニュアル』の「**create logical connection**」を参照してください。
4. **rs_init** を使用して、アクティブなデータベースとスタンバイ・データベースをレプリケーション・システムに追加します。
5. 代替論理コネクションを作成します。
6. 代替論理コネクションに代替のアクティブなコネクションを作成します。
7. **admin who** を使用して、REP AGENT スレッドを調べ、ウォーム・スタンバイのペアのアクティブなデータベースへのデフォルトのコネクションと代替コネクションがアクティブであることを確認します。
   たとえば、次のコマンドを確認します。
   ```
   31 REP AGEN   Awaiting Command     TOKYO_DS.pubs2
   ```
8. 代替論理コネクションに代替スタンバイ・コネクションを作成します。

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、複数のレプリケーション・パスを使用して、ウォーム・スタンバイ環境ですべてのレプリケーション・パスを切り替える必要があります。

切り替える手順は、代替とデフォルトのレプリケーション・パスで同じです。

1. すべての代替レプリケーション・パスを切り替える。
2. デフォルトのレプリケーション・パスを切り替える。

参照：

• アクティブ ASE データベースとスタンバイ ASE データベースの切り替え (105 ページ)

専用ルート

専用ルートは、特定のプライマリ・コネクションのトランザクションのみを分配します。レプリケート Replication Server に専用ルートを作成して、優先順位の高いトランザクションを複写するか、特定のプライマリ・コネクションのために輻輳の少ないパスを維持できます。

共有ルートは、プライマリ Replication Server から始まるすべてのプライマリ・コネクションのためにトランザクションを分配する、プライマリ Replication Server とレプリケート Replication Server の間です。共有ルートは特定のコネクションにバインドされません。専用ルートにバインドされないコネクションは、使用可能な任意の有効共有ルートを使用します。

専用ルートを作成できるのは、以下の条件が満たされた場合だけです。

- プライマリ Replication Server から送信先 Replication Server への共有ルートが存在し、この共有ルートが直接ルートである。Replication Server 間に間接ルートしかない場合、専用ルートは作成できません。
- 共有ルートが有効であり、サスペンドされていない。
- 共有ルートのバージョンが 1570 またはそれ以降である。

専用ルートの作成

create route および **with primary at** 句を使用して、専用ルートを作成します。

たとえば、`NY_DS.pdb1` プライマリ・コネクションのために、RS_NY プライマリ Replication Server と RS_LON レプリケート Replication Server との間の専用ルートを作成するには、RS_NY で次のように入力します。

```
create route to RS_LON
 with primary at NY_DS.pdb1
 go
```

特定のコネクションのために専用ルートを作成すると、そのコネクションから送信先 Replication Server へのトランザクションはすべて、その専用ルートを通るようになります。

専用ルートを管理するコマンド

create route、**drop route**、**resume route**、および **suspend route** を使用して、専用ルートを管理およびモニタします。

コマンドに **with primary at** *dataserver.database* 句を含めると、専用ルートを指定できます。ここで、*dataserver.database* は、プライマリ・コネクションの名前です。

『Replication Server リファレンス・マニュアル』の「Replication Server コマンド」で、「**create route**」、「**drop route**」、**suspend route**」、および「**resume route**」を参照してください。

コマンド	構文	コマンドおよびパラメータの変更

create route	create route to *dest_replication_server* {
	 with primary at *dataserver.database* |
	 set next site [to] *thru_replication_server* |
	 [set username [to] *user*]
	 [set password [to] *passwd*]
	 [set *route_param* to '*value*'
	  [set *route_param* to '*value*']...]
	 [set *security_param* to '*value*'
	  [set *security_param* to '*value*']...]}
		専用ルートの作成時にユーザ ID を使用する場合、ユーザ ID は有効なユーザであることが必要です。

コマンド	構文	コマンドおよびパラメータの変更
drop route	`drop route to dest_replication_server [with primary at dataserver.database] [with nowait]`	専用ルートを削除してから、共有ルートを削除してください。 専用ルートが削除されると、指定プライマリ・コネクションから送信先 Replication Server へのトランザクションは、共有ルートを通るようになります。 **警告！ with nowait** 句は、最後の手段としてのみ使用してください。 この句を使用すると、ルートのアウトバウンド・キューにトランザクションが含まれている場合でも、ルートが強制的に削除されます。その結果、Replication Server はプライマリ・コネクションからトランザクションを破棄する可能性があります。この句は、専用ルートが送信先 Replication Server と通信できない場合でも、専用ルートを削除するように、Replication Server に指示します。 句を使用する場合、以前の送信先サイトで **sysadmin purge_route_at_replicate** を使用して、送信先のシステム・テーブルからサブスクリプションおよびルート情報を削除します。 『Replication Server 管理ガイド 第 1 巻』の「ルートの管理」の「ルートの削除」の「**drop route** コマンド」を参照してください。
suspend route	`suspend route to dest_replication_server [with primary at dataserver.database]`	

コマンド	構文	コマンドおよびパラメータの変更
resume route	`resume route to *dest_replication_server* [with primary at *dataserver.database*] [skip transaction with large message]`	

専用ルート情報の表示

admin who を使用して、Replication Server 間の専用ルートの情報を表示します。

次の例では、NY_DS.pdb1 プライマリ・コネクションのために、RS_NY プライマリ Replication Server と RS_LON レプリケート Replication Server との間に専用ルートがあります。2 つの Replication Server に **admin who** と入力すると、次のように表示されます。

- RS_LON では次のように表示されます。

```
Spid Name       State               Info
45 SQT          Awaiting Wakeup     103:1  DIST NY_DS.pdb1
13 SQM          Awaiting Message    103:1 NY_DS.pdb1
32 REP AGENT    Awaiting Command    NY_DS.pdb1
16 RSI          Awaiting Wakeup     RS_LON
11 SQM          Awaiting Message    16777318:0 RS_LON
55 RSI          Awaiting Wakeup     RS_LON(103) /* Dedicated RSI
thread */
53 SQM          Awaiting Message    16777318:103 RS_LON(103) /
*Dedicated RSI outbound queue */
```

- RS_NY では次のように表示されます。

```
Spid Name      State                Info
37 RSI USER   Awaiting Command    RS_NY(103) /*Dedicated RSI user
*/
32 RSI USER    Awaiting Command    RS_NY
```

『Replication Server リファレンス・マニュアル』の「Replication Server コマンド」の「**admin who**」を参照してください。

複数のレプリケーション・パス用の Adaptive Server モニタリング・テーブル

Adaptive Server モニタリング・テーブルを使用して、複写中に Adaptive Server プライマリ・データベースの RepAgent に関係した複数のパスを用いて Adaptive Server の状態の統計的スナップショットを撮影します。このテーブルを使用して、Adaptive Server のパフォーマンスを解析できます。

テーブル	説明
monRepLogActivity	Replication Agent で更新されたモニタ・カウンタからの情報を提供する。
monRepScanners	RepAgent スキャナ・タスクについての統計情報を提供する。
monRepScannersTotal-Time	RepAgent スキャナ・タスクが費やす時間に関する情報を提供する。
monRepSenders	RepAgent の送信者タスクに関する処理情報を提供する。

コネクション別の分散を選択した場合、monRepSenders Adaptive Server モニタリング・テーブルの追加フィールドを使用して、データ分散の統計的スナップショットを撮影します。

表 24 : monRepSenders

カラム名	説明
NumberOfCommandsPro-cessed	各 RepAgent 送信者スレッドが LTL を生成するために処理するコマンド (**insert**、**delete**、**begin trans**、**commit trans** など) の数。
AvgBytesPerCmd	NumberOfBytesSent と NumberOfCommand-sProcessed の比率。

Adaptive Server Enterprise の『パフォーマンス&チューニング・シリーズ：モニタリング・テーブル』の「モニタリング・テーブルの概要」の「Adaptive Server のモニタリング・テーブル」を参照してください。

代替プライマリ・コネクションおよびレプリケート・コネクションでのシステム・テーブルのサポート

Replication Server では、rs_databases テーブルの各プライマリ・コネクションとレプリケート・コネクションごとにローが作成されますが、これには特定のローのプライマリ・コネクションまたはレプリケート・コネクションを一意に識別するためのカラム conn_id があります。

Replication Server は dsname カラムと dbname カラムを使用して、コネクション名ごとに代替コネクションを識別し、データ・サーバ名およびデータベース名ごとにデフォルトのプライマリ・コネクションまたはレプリケート・コネクションを識別します。dbid は、コネクションの接続先となるデータベースの ID を識別します。ローがデフォルトのコネクションを対象とする場合、connid は dbid と等しくなります。ローが代替コネクションを対象とする場合、connid は dbid と等

しくなくなります。『Replication Server リファレンス・マニュアル』の「Replication Server システム・テーブル」を参照してください。

マルチプロセッサ・プラットフォーム

Replication Server を対称型マルチプロセッサ (SMP) プラットフォームまたはシングルプロセッサ・プラットフォーム上で実行できます。これは、Replication Server のマルチスレッド化アーキテクチャで両方のハードウェア構成がサポートされているからです。マルチプロセッサ・プラットフォームでは、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。

シングルプロセッサ・プラットフォームでは、Replication Server のスレッドは順番に実行されます。

Replication Server は、Open Server アプリケーションです。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。どちらも 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ています。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。

Replication Server がシングルプロセッサ・モードの場合、サーバ全体の相互排他ロック (mutex) によって逐次スレッドが実行されます。逐次スレッドの実行によって、グローバル・データ、サーバ・コード、システム・ルーチンが保護され、スレッドセーフであることが保証されます。

Replication Server がマルチプロセッサ・モードの場合、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・サポートの有効化

Replication Server をマルチプロセッサ・マシン上で実行するかどうかを指定するには、**configure replication server** コマンドに **smp_enable** オプションを指定して使用します。

次のように入力します。

```
configure replication server set smp_enable to 'on'
```

smp_enable を on に設定するとマルチプロセッサ・サポートが指定され、**smp_enable** を off に設定するとシングルプロセッサ・サポートが指定されます。デフォルトは on です。

smp_enable は静的オプションです。**smp_enable** のステータスの変更後には、Replication Server を再起動する必要があります。

スレッドのステータスをモニタするためのコマンド

Replication Server スレッドのステータスは、**admin who** コマンドまたは **sp_help_rep_agent** ストアド・プロシージャを使用して確認できます。

- **admin who** — Replication Server スレッドに関するすべての情報が得られます。
- **admin who_is_up** または **admin who_is_down** — 実行中または実行されていない Replication Server スレッドをリストします。
- **sp_help_rep_agent** — RepAgent スレッドと RepAgent User スレッドに関する情報が得られる。

参照：

- Replication Server の検証とモニタリング (5 ページ)

パフォーマンスのモニタリング

Replication Server では、パフォーマンスをモニタするためにモニタとカウンタを用意しています。

参照：

- カウンタを使ったパフォーマンスのモニタリング (339 ページ)

キュー・セグメントの割り付け

Replication Server がステーブル・キューのセグメントを割り付けるディスク・パーティションは、選択することができます。ステーブル・キューの割り付け先を選択することによって、負荷分散と読み込み／書き込みの分散機能が向上します。

Replication Server は、他のサイトに向けられたメッセージをパーティションに格納します。パーティション上の領域をステーブル・キューに割り付けて、セグメントと呼ばれる 1MB のチャンクで処理します。各ステーブル・キューには、別の Replication Server またはデータ・サーバに配信されるメッセージが保持されます。送信されるまでの間、データはキューに保持されます。

Replication Server の初期パーティションを割り当てるには、**rs_init** を使用します。データベースの数と、Replication Server がメッセージを配信するリモート Replication Server の数によっては、パーティションの追加が必要になる場合もあります。

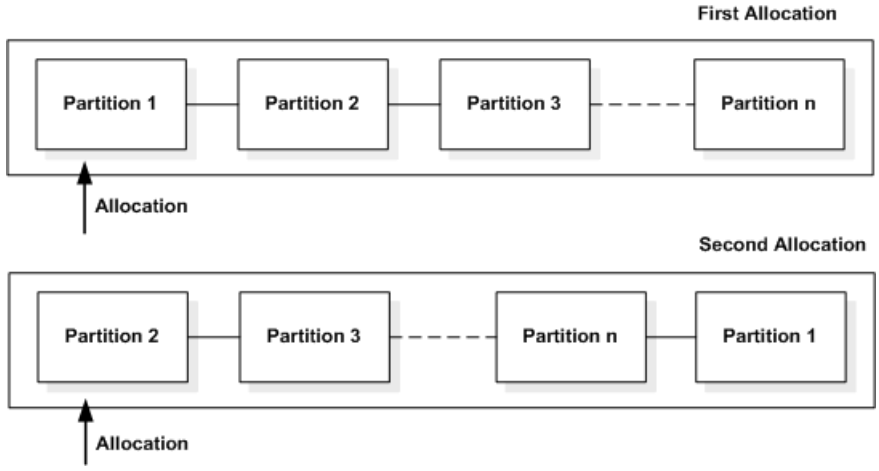
Replication Server は、さまざまなサイズのパーティションをいくつでも持つことができます。パーティション・サイズの合計が、Replication Server がキューに格納できるトランザクションの容量です。

デフォルトの割り付けメカニズム

Replication Server のデフォルトでは、順序付けされたパーティション・リストの中の最初のパーティションにキュー・セグメントが割り付けられます。

最初のパーティションが満杯になると、最初のパーティションが最後のパーティションになり、その次のキュー・セグメントが、新たに先頭になったパーティションに割り付けられます。デフォルトの方法を使用する場合、セグメントの循環割り付けが自動的に実行され、ユーザが制御することはできません。

図 22 : デフォルトの割り付けメカニズム



ディスク割り付けの選択

セグメント割り付けを選択するには、**alter connection** コマンドまたは **alter route** コマンドに **set disk_affinity** オプションを指定して使用します。

次に構文を示します。

```
alter connection to dataserver.database
    set disk_affinity to [ 'partition' | 'off' ]
```

```
alter route to replication_server
    set disk_affinity to [ 'partition' | 'off' ]
```

partition は、コネクションまたはルートの次のセグメントを割り付けるパーティションの論理名です。

割り付けられたパーティションが満杯になった場合や削除された場合などに、Replication Server がその割り付けを上書きできるため、各割り付け命令は「ヒント」と呼ばれます。Replication Server がヒントを上書きした場合、セグメントの割り付けはデフォルトの割り付けメカニズムに従って行われます。

Replication Server は、キューに新しいセグメントを割り付けるたびに割り付けヒントがあるかどうかをチェックします。各ヒントは `rs_diskaffinity` システム・テーブルに格納されます。各パーティションは複数のヒントを持つ場合もありますが、各ステーブル・キューはヒントを 1 つしか持てません。

ディスク割り付けを使用してパフォーマンスを向上させる方法は、サイトのアーキテクチャなどの特性によって異なります。全体のスループットを向上する 1 つの方法は、処理の遅いステーブル・キューには処理速度の速いデバイスを関連付けることです。

また、すべてのコネクションが所定どおりになった後で、新しいパーティションを追加した場合は、既存のパーティションが満杯になるまで、新しいパーティションは使用されません。割り付けヒントを追加して、新しいパーティションを使用するようにコネクションを強制できます。

ステーブル・キューへのディスク・パーティションの割り付け

ステーブル・キューには、キューごとに異なるディスク・パーティションを割り付けることができます。

たとえば、それぞれのデータベース・コネクションでサイズの異なるパーティションを使用可能にできます。この例では、10MB と 20MB のパーティションを Replication Server に追加し、TOKYO_DS データ・サーバと SEATTLE_DS データ・サーバに割り付けヒントを指定します。

1. パーティション `P1` と `P2` (`/dev/rds0a` という名前のデバイス上に存在) を Replication Server で使用できるようにします。

   次のように入力します。
   ```
   create partition P1 on '/dev/rds0a' with size 20
   ```
   および
   ```
   create partition P2 on '/dev/rds0a' with size 10
   ```
2. TOKYO_DS データ・サーバと SEATTLE_DS データ・サーバのコネクションをサスペンドするため、次のコマンドを入力します。

   次のように入力します。
   ```
   suspend connection to TOKYO_DS
   ```
   および
   ```
   suspend connection to SEATTLE_DS
   ```

3. TOKYO_DS データ・サーバと SEATTLE_DS データ・サーバへのコネクションに対して割り付けヒントを指定します。

   以下を入力します。

   ```
   alter connection to TOKYO_DS.db1
   set disk_affinity to 'P1'
   ```

   および

   ```
   alter connection to SEATTLE_DS.db5
   set disk_affinity to 'P2'
   ```

4. TOKYO_DS データ・サーバと SEATTLE_DS データ・サーバへのコネクションをレジュームします。

   以下を入力します。

   ```
   resume connection to TOKYO_DS
   ```

   および

   ```
   resume connection to SEATTLE_DS
   ```

ヒントとパーティションの削除

割り付けヒントを削除するには、**alter connection** コマンドまたは **alter route** コマンドに **set disk_affinity to 'off'** パラメータを指定して使用します。

次に例を示します。

```
alter connection to TOKYO_DS.db1
set disk_affinity to 'P1' to 'off'
```

このコマンドを実行すると、`rs_diskaffinity` テーブルから `P1` に対する割り付けヒントが削除されます。

Replication Server のパーティションを削除するには、**drop partition** コマンドを使用します。削除するパーティションの `rs_diskaffinity` テーブルに 1 つ以上の割り付けヒントがある場合、Replication Server は割り付けヒントを削除するようマーク付けしますが、そのパーティションに格納されたすべてのデータが正常に配信されてパーティションが削除されるまで、割り付けヒントは削除されません。

RMS のハートビート機能

遅延時間情報を表示するには、コマンド・ライン・サービスで Replication Monitoring Services (RMS) のハートビート機能を使用します。

ハートビート機能では、**rs_ticket** ストアド・プロシージャを使用して、遅延時間情報を生成します。遅延時間は、プライマリ・データベースからレプリケート・データベースに移動するトランザクションに要する時間です。RMS は、プライマ

リ・データベースで **rs_ticket** を指定した間隔で実行します。生成された遅延時間情報は、レプリケート・データベースのテーブルに格納されます。

RMS には、ハートビート・プロセスを設定し、レプリケート・データベースからその遅延時間情報を取得するためのコマンドが用意されています。ハートビート機能は RMS からのみ使用できます。「**get heartbeat**」と「**get heartbeat tickets**」(『Replication Server リファレンス・マニュアル』の「Replication Monitoring Services API」) を参照してください。

カウンタを使ったパフォーマンスのモニタリング

Replication Server には、複写処理のさまざまな時点および領域でパフォーマンスをモニタできる、数百種類のカウンタがあります。

デフォルトでは、常にアクティブであるいくつかのカウンタを除いて、アクティブ化するまでは、アクティブにはなっていません。

RepAgent カウンタを使用してパフォーマンスをモニタリングする方法の詳細については、『Replication Server 管理ガイド 第 1 巻』の「RepAgent の管理と Adaptive Server のサポート」の「RepAgent のパフォーマンスをモニタするためのカウンタの使用」を参照してください。

カウンタ値を表示するためのコマンド

いくつかのコマンドを使用して、現在のカウンタの値およびその他のパフォーマンス情報をいつでも表示できます。

次を使用できます。

- **admin stats** - 指定したカウンタの現在の値を表示する。
- **admin stats, backlog** - Replication Server ステーブル・キュー内の現在のバックログを表示する。
- **admin stats, { tps | cps | bps }** - 1 秒あたりのトランザクション数、1 秒あたりのコマンド数、または 1 秒あたりのバイト数で、スループットを表示する。
- **admin stats, { md | mem | mem_in_use }** - メッセージおよびメモリについての情報を表示する。

カウンタ値を RSSD に保存 (フラッシュ) し、標準の Transact-SQL 文または **rs_dump_stats** ストアド・プロシージャを使用して、平均および比率を計算して表示することもできます。

モジュール

Replication Server では、モジュールは、連携して動作し特定のサービスを実行するコンポーネントのグループです。

たとえば、ステーブル・キュー・マネージャ (SQM) は、ステーブル・キュー・サービスを提供する、論理的に関連したコンポーネントで構成されています。

Replication Server には、各モジュールの各インスタンス (オカレンス) でのアクティビティを追跡できるカウンタが用意されています。

Replication Server では、1 つのインスタンスしか持たないモジュールがあります。それらのモジュールのインスタンスは、モジュール名のみで識別することができます。このタイプのモジュールの例を以下に示します。

- システム・テーブル・サービス (STS)
- コネクション・マネージャ (CM)

Replication Server では、複数のインスタンスを持つことができるモジュールもあります。モジュールの各インスタンスをユニークに識別するには、モジュール名とインスタンス ID の両方を含める必要があります。次のモジュールがこのタイプに該当します。

- Replication Server インタフェース (RSI)
- ディストリビュータ (DIST)
- データ・サーバ・インタフェース・スケジューラ・スレッド (DSI/S)

さらに他のモジュールでは、差別化するためにモジュール名、インスタンス ID、およびインスタンス値という 3 つの識別子を必要とします。次のモジュールがこのタイプに該当します。

- ステーブル・キュー・トランザクション・スレッド (SQT)
- ステーブル・キュー・マネージャ (SQM)
- データ・サーバ・インタフェース・エグゼキュータ・スレッド (DSIEXEC)

Replication Server モジュール

Replication Server には、共通して使用されるモジュールがいくつかあります。

Replication Server のコマンドを使用して、独立したモジュールのカウンタを直接指定できます。依存するモジュールのカウンタにアクセスするには、それらの親モジュールの名前を使用します。

表 25 : Replication Server モジュール

モジュール名	頭文字	独立／依存
Connection Manager (コネクション・マネージャ)	CM	独立
Distributor (ディストリビュータ)	DIST	独立
Data Server Interface (データ・サーバ・インタフェース)	DSI	独立
DSI Executor (DSI エグゼキュータ)	DSIEXEC	DSI に依存

モジュール名	頭文字	独立／依存
RepAgent thread (RepAgent スレッド)	REPAGENT	独立
Replication Server Interface (Replication Server インタフェース)	RSI	独立
RSI User (RSI ユーザ)	RSIUSER	独立
Replication Server Global (Replication Server グローバル)	SERV	独立
Stable Queue Manager (ステーブル・キュー・マネージャ)	SQM	独立
SQM Reader (SQM リーダ)	SQMR	SQM に依存
SQM Transaction Manager (SQM トランザクション・マネージャ)	SQT	独立
System Table Services (システム・テーブル・サービス)	STS	独立
Thread Synchronization (スレッド同期)	SYNC	独立
SYNC Element (SYNC 要素)	SYNCELE	SYNC に依存

カウンタ

各カウンタには、記述名と、RCL コマンドを入力するとき、および表示された情報を参照するときに、カウンタを識別するために使用する表示名があります。

Replication Server のカウンタについての詳細な情報およびステータス情報を表示するには、**rs_helpcounter** ストアド・プロシージャを使用します。

さまざまなカウンタが、さまざまな種類の情報を提供します。すべてのカウンタを個別のカテゴリに分類できるわけではありませんが、Replication Server は、カウンタの情報を表示するときに、以下のカテゴリを使用します。

- オブザーバ - ある一定時間におけるイベントの発生回数を収集する。たとえば、キューからメッセージが読み取られる回数を収集する。Replication Server は、発生回数と 1 秒あたりの発生回数をレポートする。
- モニタ - 指定した時刻または期間の計測値を収集する。たとえば、モニタは、トランザクションあたりのオペレーションの数を収集する。Replication Server は、監視数、最後に収集された値、最大値、平均値をレポートする。
- カウンタ - さまざまな計測値を収集する。経過時間を測定するカウンタおよびバイトの合計数を収集するカウンタは、このグループに属する。このカテゴリの場合、Replication Server は、監視数、合計値、最終値、最大値、平均、1 秒あたりの比率をレポートできる。

参照：
- カウンタに関する情報の表示 (351 ページ)

データ・サンプリング

データを収集するための複数のオプションがあります。データをサンプリングする時間を、長時間、短時間 (秒)、または 1 回の発生から選択できます。

次の 2 つのうち、いずれかの方法でカウンタ統計を収集できます。

- **display** オプションを指定して **admin stats** を実行する。これによって、Replication Server に、指定した一定時間の情報を収集し、その時間の終了時に収集した情報をコンピュータ画面に表示するように指示する。
- **admin stats** を **save** オプションを指定して実行する。これによって Replication Server に、指定した一定時間に、指定した監視数で情報を収集し、その情報を RSSD に保存するように指示する。

デフォルトでは、カウンタをオンにするまで、情報は収集されません。**admin stats** を実行すると、特定の時間カウンタをオンにできます。**stats_sampling** 設定パラメータを on に設定して、永続的にサンプリングをオンにすることもできます。

サンプル収集をオンにすると、すべてのカウンタがアクティブになります。ただし、対象となるカウンタまたはモジュールの統計のみを表示または保存できます。

コンピュータ画面に表示される統計は、1 回の監視期間中のイベント数と、平均や比率などの計算値を記録しています。統計が RSSD に送信されると、Replication Server は、連続する複数の監視期間中の監視数、合計数、最終値、最大値などのロー値を保存します。その後、これらの格納された値から平均および比率を計算できます。

特定時間の統計の収集

admin stats を使用して、特定時間の統計を収集します。

admin stats の構文は次のとおりです。

```
admin { stats | statistics } [, sysmon | "all"
        | module_name [, inbound | outbound ] [, display_name] ]
        [, server[, database ]       | instance_id ]
        [, display |, save [, obs_interval ] ]
        [, sample_period]
```

admin stats を使用して、以下を指定できます。

- サンプリングされるカウンタ

- 監視間隔およびサンプリング時間の長さ
- 統計を RSSD に保存するか、それともコンピュータ画面に表示するか

注意： **admin stats** は、**cancel** オプションもサポートしています。このオプションは、現在実行中のコマンドを中止します。

デフォルトでは、Replication Server は、サンプリング時間の監視結果が 0 (ゼロ) のカウンタをレポートしません。**configure replication server** を使用して、**stats_show_zero_counters** 設定を on に設定することで、この動作を変更できます。構文の詳細と使用方法については、『Replication Server リファレンス・マニュアル』の「Replication Server コマンド」を参照してください。

サンプリングされるカウンタの指定

すべてのカウンタまたはカウンタの 1 つのインスタンスのみを指定できます。

次のパラメータに **admin stats** を付けて使用して、カウンタを指定します。

- **sysmon** - Sybase によって、パフォーマンスとチューニングにとって最も重要であるとされているすべてのカウンタをサンプリングする。これはデフォルトの値。

  sysmon カウンタのリストを表示するには、次のように入力します。

  ```
  rs_helpcounter sysmon
  ```
- **"all"** - すべてのカウンタをサンプリングする。
- **module_name** - 特定のモジュールのすべてのカウンタをサンプリングする。
- **module_name, display_name** - 特定のカウンタのすべてのインスタンスをサンプリングする。カウンタのリストを取得するには、**sp_helpcounter** を使用する。
- **module_name, display_name, instance_id** - あるカウンタの特定のインスタンスをサンプリングする。インスタンスの数値 ID を見つけるには、**admin_who** を実行し、`Info` カラムを確認する。

  注意： インスタンス ID が指定されていて、モジュールが SQT または SQM である場合は、カウンタ・インスタンスのインバウンド・キューまたはアウトバウンド・キューによって提供される情報のいずれかを指定できます。

たとえば、1 秒間の **sysmon** カウンタの統計を収集して、情報をコンピュータ画面に送信するには、次のように入力します。

```
admin stats, sysmon, display, 1
```

参照：

- モジュール (339 ページ)

サンプリング時間の指定

サンプリング時間は、秒数で指定します。

Replication Server は、その秒数の間、指定されたカウンタの統計を収集して、画面または RSSD にレポートします。デフォルト値は 0 (ゼロ) 秒です。この場合、すべてのカウンタが現在の値をレポートします。

たとえば、すべてのカウンタの統計を 1 分間収集して、コンピュータ画面に表示するには、次のように入力します。

```
admin stats, "all", display, 60
```

統計をレポートする方法の指定

統計をコンピュータ画面または RSSD に送信できます。

統計のコンピュータ画面への表示

統計をコンピュータ画面に送信するには、**display** オプションを指定します。

この場合、Replication Server は、指定した時間の最後に 1 回監視を行います。監視された統計は、コンピュータ画面のみに送信されます。

たとえば、5 分間隔で、すべてのキューから、すべてのリーダによって読み込まれたブロック数をレポートするには、次のように入力します。

```
admin stats, sqm, blocksread, display, 300
```

admin stats を **display** オプションを使用して、0 以外の時間を指定して実行すると、Replication Server は以下を行います。

1. すべてのカウンタを 0 にリセットします。
2. すべてのカウンタをオンにします。
3. セッションを指定した時間スリープにします。
4. すべてのカウンタをオフにします。
5. 要求されたデータをレポートします。

統計の RSSD への保存

RSSD へ統計を保存するには、即座にセッションを返す **save** オプションを含めます。

RSSD へ統計を送信すると、指定したサンプリング時間中のそれぞれの監視間隔の長さを *obs_interval* で指定できます。*obs_interval* には、数値 (秒単位) を指定することも、引用符で囲まれた時刻フォーマット文字列 (hh:mm[:ss]) を指定することもできます。

たとえば、1 時間 30 分の間、20 秒間隔でサンプリングと RSSD への統計の保存を開始するには、次のように入力します。

```
admin stats, "all", save, 20, "01:30:00"
```

30 秒間隔で、コネクション 108 のアウトバウンド SQT の統計を 2 分間収集するには、次のように入力します。

```
admin stats, sqt, outbound, 108, save, 30, 120
```

Replication Server は、サンプリング時間を監視間隔で除算し、監視間隔の数を決定します。余りの秒数がある場合は、最後の監視間隔に加えられます。

表 26 : サンプリング時間と監視間隔

サンプリング時間 (*sample_period*)	監視間隔 (*obs_interval*)	監視間隔数
60 秒	15	4 (15 秒間隔)
75 秒	5	許可されない - 監視間隔は 15 秒以上
60 秒	30	2 (30 秒間隔)
130 秒	20	5 (20 秒間隔) および 1 (最後の 30 秒間隔)
10 秒	指定なし	1 (10 秒間隔)

admin stats を **save** オプションを指定して、0 以外の時間を設定して実行すると、Replication Server は、バックグラウンド・スレッドを開始してサンプリング・データを収集し、即座にセッションを返します。セッションが返されたら、**admin stats, status** コマンドを使用して、サンプリングの進行状況を確認できます。バックグラウンド・スレッドは、以下を実行します。

1. 設定パラメータ **stats_reset_rssd** が on に設定されている場合は、`rs_statrun` および `rs_statdetail` システム・テーブルをトランケートします。
2. すべてのカウンタをリセットします。
3. すべてのカウンタをオンにします。
4. 各監視期間の最後に、要求されたカウンタを RSSD に書き込みます。
5. すべてのカウンタをオフにします。

注意： 古いサンプリング・データを保持するには、設定パラメータ **stats_reset_rssd** を off に設定するか、`rs_statrun` と `rs_statdetail` から、必要な情報をすべてダンプしたことを確認してから、**save** オプションを指定して **admin stats** を実行します。 **rs_dump_stats** プロシージャを使用して、これらのテーブルから情報をダンプできます。

参照：

- rs_dump_stats ストアド・プロシージャの使い方 (349 ページ)

永続的な統計の収集

永続的なサンプリングをオンにするには、**stats_sampling** パラメータを使用して Replication Server を設定します。

次のように入力します。

```
configure replication server
  set stats_sampling to "on"
```

Replication Server は、ユーザが Replication Server を再設定してサンプリングをオフにするまで、データを収集し続けます。

```
configure replication server
   set stats_sampling to "off"
```

その後、コンピュータ画面でデータを表示する、または収集したデータを RSSD に送信するには、**admin stats** を使用します。

注意： **admin stats** の使用時は、**stats_sampling** が on になっていることに注意してください。**admin stats** を実行し、0 以外の時間を指定した場合、Replication Server はカウンタをクリアし、コマンドを実行して **stats_sampling** を off にします。

たとえば、連続する 24 時間で統計を収集し、結果をコンピュータ画面にレポートするには、次の手順に従います。

1 日目、午前 8 時

1. すべての統計をクリアします。
   次のように入力します。
   ```
   admin statistics, reset
   ```
2. サンプリングをオンにします。
   次のように入力します。
   ```
   configure replication server
   set stats_sampling to "on"
   ```

2 日目、午前 8 時

1. 統計を画面にダンプするために、サンプリングをオフにして、Replication Server が統計を収集しないようにします。
   次のように入力します。
   ```
   configure replication server
   set stats sampling to "off"
   ```
2. 統計を画面にダンプします。
   次のように入力します。
   ```
   admin statistics, "all"
   ```

3. すべての統計をクリアします。
   次のように入力します。
   ```
   admin statistics, reset
   ```
4. サンプリングをオンにします。
   次のように入力します。
   ```
   configure replication server
   set stats_sampling to "on"
   ```

3 日目、午前 8 時

1. 統計を画面にダンプするために、サンプリングをオフにして、Replication Server が統計を収集しないようにします。
   次のように入力します。
   ```
   configure replication server
   set stats sampling to "off"
   ```
2. 統計を画面にダンプします。
   次のように入力します。
   ```
   admin statistics, "all"
   ```
3. すべての統計をクリアします。
   次のように入力します。
   ```
   admin statistics, reset
   ```

画面での統計の表示

admin stats を使用して、1 回のサンプル実行の統計をコンピュータ画面に表示します。

1 つのカウンタ・インスタンス、1 つのカウンタ、特定のモジュールのすべてのカウンタ、通常最も便利な **sysmon** カウンタ、またはすべてのカウンタの統計を表示できます。

admin stats を使用して、サンプル実行を設定するときに、統計をコンピュータ画面に表示するかどうかを選択できます。

出力例および構文の詳細と使用方法については、『Replication Server リファレンス・マニュアル』の「Replication Server コマンド」の「**admin stats**」を参照してください。

参照：

- 特定時間の統計の収集 (342 ページ)

スループット率の表示

1 秒あたりのトランザクション数、1 秒あたりのコマンド数、または 1 秒あたりのバイト数で、現在のスループットを表示するには、**admin stats** を **tps**、**cps**、または **bps** オプションを指定して使用します。

1 秒あたりのトランザクション数
Replication Server は、最後にカウンタをリセットしてから処理されたトランザクション数と経過秒数に基づいてトランザクション率を計算します。データは、SQT、DIST、DSI モジュールなどの複数のモジュールから取得されます。

1 秒あたりのトランザクション数でスループットを表示するには、次のように入力します。

```
admin stats, tps
```

1 秒あたりのコマンド数
1 秒あたりのコマンド数は、最後にリセットしてから処理されたコマンド数と経過秒数から計算されます。データは、REPAGENT、RSIUSER、RSI、SQM、DIST、DSI モジュールから取得されます。

1 秒あたりのコマンド数でスループットを表示するには、次のように入力します。

```
admin stats, cps
```

1 秒あたりのバイト数
1 秒あたりのバイト数は、最後にリセットしてから処理されたバイト数と経過秒数から計算されます。データは、REPAGENT、RSIUSER、SQM、DSI、RSI モジュールから取得されます。

1 秒あたりのバイト数でスループットを表示するには、次のように入力します。

```
admin stats, bps
```

メッセージおよびメモリ使用状況に関する統計の表示

メッセージ数に関する情報を表示するには、**admin stats** を **md** オプションを指定して使用します。メモリの使用状況に関する情報を表示するには、**mem**、または **mem_in_use** オプションを指定して、**admin stats** を使用します。

- ディストリビュータと RSI ユーザに関連しているメッセージ配信の統計を表示するには、次のように入力する。
  ```
  admin stats, md
  ```
- 現在のセグメントの使用状況をセグメントのサイズに従って表示するには、次のように入力する。
  ```
  admin stats, mem
  ```

- 現在のメモリの使用状況をバイト数で表示するには、次のように入力する。
  ```
  admin stats, mem_in_use
  ```

ステーブル・キュー内のトランザクション数の表示

admin stats に **backlog** オプションを付けて使用して、分配を待っているインバウンド・ステーブル・キューとアウトバンド・ステーブル・キュー内のトランザクション数を表示します。

Replication Server は、データをセグメント数およびブロック数でレポートします。1 セグメントは 1MB、1 ブロックは 16K に相当します。データは、SQMRBacklogSeg カウンタと SQMRBacklogBlock カウンタから取得されます。

ステーブル・キューのバックログを表示するには、次のように入力します。

```
admin stats, backlog
```

RSSD に保存されている統計の表示

RSSD に保存されている統計を表示するために使用できるコマンドとプロシージャがいくつかあります。

RSSD に送信された統計は、以下のシステム・テーブルに格納されます。

- `rs_statcounters` - 各カウンタの詳細な情報が格納される。
- `rs_statdetail` - 各カウンタのサンプリング実行ごとの監視されたメトリックが格納される。
- `rs_statrun` - 各サンプリング実行を記述する。

これらのテーブルの詳細については、『Replication Server リファレンス・マニュアル』の「Replication Server システム・テーブル」を参照してください。

以下を使用して、これらのテーブルに格納されている統計を表示できます。

- **select** およびその他の Transact-SQL コマンド
- **rs_dump_stats**
- **rs_helpcounter** (`rs_statcounters` からの情報を表示)

rs_dump_stats ストアド・プロシージャの使い方

rs_dump_stats は、`rs_statrun` および `rs_statdetail` システム・テーブルの内容を、分析のためにスプレッドシートにロードできる CSV ファイルにダンプします。

構文と使用方法の詳細については、『Replication Server リファレンス・マニュアル』の「RSSD ストアド・プロシージャ」の「**rs_dump_stats**」を参照してください。

rs_dump_stats を使用するには、RSSD にログインして、このストアド・プロシージャを実行します。次に例を示します。

```
1> rs_dump_stats
2> go
```

rs_dump_stats のサンプル出力

注意： 出力の右側にあるコメントは、例を説明するために含まれています。これらは、**rs_dump_stats** の出力の一部ではありません。

```
Comment:  Sample of rs_dump_stats output
Nov  5 2005 12:29:18:930AM                          *Start time stamp*
Nov  5 2005 12:46:51:350AM                          *End time stamp*
16                                                  *No of observation
intervals*
1                                                   *No of min between
                                                     observations*
16384                                             *SQM bytes per block*
64                                               *SQM blocks per segment*
CM                                                  *Module name*
13                                                  *Instance ID*
-1                                                  *Instance value*
dCM                                                 *Module name*
CM: Outbound database connection request            *Counter external
name*
CMOBDBReq                                         *Counter display name*
13003          , , 13,  -1                        *Counter ID, instance
ID,
                                                        instance value*
ENDOFDATA                                           *EOD for counter*

CM: Outbound non-database connection requests       *Counter external
name*
CMOBNonDBReq                                      *Counter display name*
13004          , , 13,  -1                        *Counter ID, instance
ID,
                                                        instance value*
Nov  5 2005 12:29:18:930AM,  103,  103,  1,  1   *Dump ts, obs,
total,
                                                         last, max*
Nov  5 2005 12:30:28:746AM,  103,  103,  1,  1
Nov  5 2005 12:31:38:816AM,  107,  107,  1,  1
Nov  5 2005 12:32:49:416AM,  104,  104,  1,  1
Nov  5 2005 12:33:58:766AM,  114,  114,  1,  1
...
Nov  5 2005 12:46:51:350AM,  107,  107,  1,  1
ENDOFDATA                                           *EOD for counter*

CM: Outbound 'free' matching connections found      *Counter external
name*
CMOBFreeMtchFound                                 *Counter display name*
13005          , , 13,  -1                        *Counter ID, instance
ID,
                                                        instance value*
```

```
Nov  5 2005 12:29:18:930AM,  103,  103,  1,  1   *Dump ts, obs,
total,
                                                      last, max*

Nov  5 2005 12:30:28:746AM,  103,  103,  1,  1
...
Nov  5 2005 12:46:51:350AM,  2,  2,  1,  1
ENDOFDATA                                        *EOD for counter*
```

カウンタに関する情報の表示

rs_statcounters テーブルに格納されているカウンタについての詳細な情報を表示するには、**rs_helpcounter** システム・プロシージャを使用します。

- カウンタを持つモジュールのリストや **rs_helpcounter** プロシージャの構文を表示するには、次のコマンドを入力します。
  ```
  rs_helpcounter
  ```
- 指定されたモジュールのすべてのカウンタについての詳細な情報を表示するには、次のコマンドを入力します。
  ```
  rs_helpcounter module_name[, short | long ]
  ```
  short を入力すると、各カウンタの表示名、モジュール名、カウンタの説明が出力されます。
  long を入力すると、各カウンタの rs_statcounters のすべてのカラムが出力されます。
  2 番目のパラメータを入力しないと、各カウンタの表示名、モジュール名、外部名が出力されます。
- キーワードに一致するカウンタをすべてリストするには、次のコマンドを入力します。
  ```
  rs_helpcounter keyword [, short |, long ]
  ```
- 指定されたステータスを持つカウンタをリストするには、構文は次のようになります。
  ```
  rs_helpcounter { sysmon | internal | must_sample
          | no_reset | old | configure }
  ```

構文と使用方法の詳細については、『Replication Server リファレンス・マニュアル』の「RSSD ストアド・プロシージャ」の「**rs_helpcounter**」を参照してください。

カウンタのリセット

admin stats, reset コマンドを使用して、リセットできないカウンタを除いて、すべてのカウンタを 0 (ゼロ) にリセットします。

次のように入力します。

```
admin stats, reset
```

stats_sampling パラメータを使用するサンプリングが有効に設定されていなかった場合、カウンタ値は 0 です。0 以外のサンプリング時間を設定して **admin stats** を実行すると、カウンタが 0 に設定され、サンプリングがオンになります。サンプリング実行が完了すると、カウンタのサンプリングがオフになり、カウンタが 0 にリセットされます。サンプリング時間が 0 の場合は、現在のカウンタ値がレポートされます。

サンプリングが有効になっている場合は、**admin stats** は注意して使用してください。**stats_sampling** 設定を使用してサンプリングが有効になっている場合、カウンタ値は累積されています。**admin stats** を発行して、サンプリング時間を指定すると、Replication Server は、サンプリング実行後に、すべてのカウンタをクリアしてサンプリングを無効 (**stats_sampling** off) にします。

パフォーマンス・レポートの生成

rs_stat_populate ストアド・プロシージャと **rs_stat_genreport** ストアド・プロシージャを使用して、パフォーマンス・レポートを生成します。

Replication Server 15.1 にアップグレードした後に、次のスクリプトを RSSD にロードしてください。

```
$SYBASE/$SYBASE_REP/scripts/
rs_install_statreport_v1510_[ase|asa].sql
```

スクリプトをロードしたら、**rs_stat_populate** ストアド・プロシージャと **rs_stat_genreport** ストアド・プロシージャを実行して、次のパフォーマンス・レポートを生成します。

- Replication Server のパフォーマンスの概要 － DIST 処理や DSI 処理など、Replication Server に関する概要情報。
- Replication Server のパフォーマンス分析 － Replication Server の重要なカウンタに基づいたパフォーマンス分析およびチューニングのためのヒント。詳細な説明は、スクリプト・ファイルに記載されている。

- アクティブなオブジェクトの識別結果 ― アクティブなテーブルとプロシージャの名前、所有者名、実行時間などのリスト。

rs_stat_populate と **rs_stat_genreport** の詳細については、スクリプト・ファイルを参照してください。このファイルには、構文や例などが含まれています。

エラーと例外の処理

Replication Server のさまざまなエラー処理方法について学びます。

特定のエラーの解決に関する情報については、『Replication Server トラブルシューティング・ガイド』を参照してください。

一般的なエラー処理

Replication Server は、データ・サーバおよび他の Replication Server へのアクセスが可能であるときにはメッセージを渡し、コネクションがダウンしているときはメッセージをキューイングします。Sybase Central を使用して、複写システムのステータスをモニタでき、問題が発生したときにその問題を解決できます。

通常、ネットワークやデータ・サーバで短期間の障害が発生しても、特別なエラー処理やユーザの介入は必要ありません。障害が解決されると、複写システム・コンポーネントの動作は自動的にレジュームされます。ただし、長期間の障害が発生した場合、メッセージをキューイングできるだけのディスク領域が不足していたり、障害を回避するために複写システムを再設定しなければならないときには、ユーザの介入が必要になることがあります。

Replication Server のパーティションやプライマリ・データベースなど、一部のシステム・コンポーネントで障害が発生した場合も、複写システム・リカバリ手順によるユーザの介入が必要になります。

エラーに対する Replication Server の応答は、エラーの種類、エラーの発生元、Replication Server の設定によって異なります。Replication Server は次の作業を行います。

- エラー・ログ・ファイルにエラーのログを取る。
- 設定内容に応じてデータ・サーバ・エラーに応答する。
- データベース内でトランザクションがコミットに失敗した場合、手動での解析が可能となるよう、そのトランザクションを例外ログに書き込む。
- システムの再起動後に重複するトランザクションを検出する。

参照：

- 複写システム・リカバリ (381 ページ)

エラー・ログ・ファイル

Replication Server および RepAgent のトラブルシューティングに利用できる複写システムのエラー・ログ・ファイルについて学びます。

システム・テーブルに書き込まれたトランザクションで、省略されたものを表示するには、指定されたデータベースを管理する Replication Server 用の Adaptive Server にアクセスすることもできます。『Replication Server トラブルシューティング・ガイド』を参照してください。

Replication Server では、データ・サーバのエラーに対するエラー処理をユーザが設定できます。

参照：
- データ・サーバのエラー処理 (360 ページ)

Replication Server エラー・ログ

Replication Server エラー・ログは、Replication Server によって情報メッセージとエラー・メッセージが書き込まれるテキスト・ファイルです。

デフォルトでは、Replication Server のエラー・ログ・ファイルは `repserver.log` という名前で、Replication Server を起動したディレクトリにあります。エラー・ログ・ファイルの名前とロケーションは、Replication Server の起動時に **-E** コマンド・ライン・フラグを使用して、または Replication Server の実行ファイルで指定できます。

Replication Server エラー・ログのメッセージ・タイプ

Replication Server エラー・ログには、メッセージ・タイプがいくつかあります。各ログ・メッセージは、メッセージ・タイプを示す文字で始まります。

表 27 : Replication Server エラー・ログのメッセージ・タイプ

エラー・コード	説明
I	情報メッセージ。
W	まだエラーにはなっていないが、注意が必要な状態に対する警告。たとえば、リソースの不足など。
E	今後の処理を妨げないエラー。たとえば、使用できないサイトなど。

エラー・コード	説明
H	機能停止した Replication Server のスレッドがあることを示す。たとえば、ネットワーク・コネクションが失われているなど。
F	致命的エラー。重大なエラーによって Replication Server が終了したことを示す。たとえば、間違った設定で Replication Server を起動するなど。
N	内部エラー。Replication Server の異常によって発生する。このエラーが発生した場合は、Sybase 製品の保守契約を結んでいるサポート・センタへの報告が必要。

情報メッセージ

Replication Server エラー・ログの情報メッセージです。

エラー・ログ内の情報メッセージのフォーマットは次のとおりです。

```
I. date: message
```

メッセージの先頭の文字 “I” は、情報メッセージであることを表します。エラーが発生したことを表すわけではありません。たとえば、サブスクリプションの削除時に、次のようなメッセージが出力されます。

```
I. 95/11/01 05:41:54. REPLICATE RS: Dropping
subscription authors_sub for replication definition
authors with replicate at <SYDNEY_DS.pubs2>

I. 95/11/01 05:42:02. SQM starting: 104:-2147483527
authors.authors_sub

I. 95/11/01 05:42:12. SQM Stopping: 104:-2147483527
authors.authors_sub

I. 95/11/01 05:42:20. REPLICATE RS: Dropped
subscription authors_sub for replication definition
authors with replicate at <SYDNEY_DS.pubs2>
```

エラー・メッセージと警告メッセージ

Replication Server のエラー・ログ内のエラーおよび警告メッセージです。

情報メッセージ以外のメッセージのフォーマットは、次のとおりです。

```
severity, date. ERROR #error_number thread_name(context) -
source_file(line) message
```

メッセージが警告の場合には、上記のフォーマットの “ERROR” が “WARNING” になります。

パラメータは次のとおりです。

- *severity* – Replication Server のエラー・ログのメッセージのタイプに応じて W、E、H、F、N となります。

- *date* – エラーが発生した日時。
- *error_number* – Replication Server のエラー番号。
- *thread_name* – エラーを受信した Replication Server スレッドの名前。Replication Server のスレッドの詳細については、『Replication Server 管理ガイド 第 1 巻』の「Replication Server の技術的概要」を参照してください。
- *context* – エラー発生時のスレッドのコンテキストに関する一部の情報が提供されます。
- *source_file* – エラーが報告された Replication Server のソース・コードのプログラム・ファイル。
- *line* – エラーが報告された Replication Server のソース・コードのプログラム・ファイルの行番号。
- *RS_language* – Replication Server メッセージの言語を指定します。
- *message* – Replication Server から出力されるメッセージの全文を *RS_language* 設定パラメータで指定された言語で指定します。一部のメッセージには、データ・サーバや、Replication Server で使用されるコンポーネント・ライブラリから出力されたメッセージも含まれます。

注意： Replication Server では、メッセージ中の何らかの情報が不明な場合には、疑問符 (?) が付けられます。たとえば、初期化中にエラーが発生し、Replication Server の内部構成が完了していない場合、収集されていない情報の代わりに疑問符が表示されます。

データ・サーバの Replication Server のエラー・ログ・エントリです。

```
E. 95/11/01 05:30:52. ERROR #1028 DSI(SYDNEY_DS.pubs2)
- dsiqmint.c(3522)Message from server:
 Message: 2812, State: 4, Severity: 16 --
 'Stored procedure 'upd_authors' not found.
```

```
H. 95/11/01 05:30:53. THREAD FATAL ERROR #5049
DSI(SYDNEY_DS.pubs2) - dsiqmint.c(3529)
The DSI thread for database 'SYDNEY_DS.pubs2' is being
shutdown because of error action mapped from data server
error '2812'. The error was caused by output command '1'
mapped from source command '2' of the transaction.
```

このメッセージは、Adaptive Server からエラー番号 2812 が返されたため、Replication Server で **stop_replication** アクションが実行されたことを示します。データ・サーバ・エラーに他のアクションを割り当てることができます。

Replication Server のエラー・ログ名の検索

現在の Replication Server のエラー・ログ・ファイル名を検索するには、**admin log_name** コマンドを使用します。

Replication Server は、ログ・ファイルのパスを表示します。次に UNIX 環境での例を示します。

```
Log File Name
 -----------------------------
 /work/sybase/SYDNEY_RS/SYDNEY_RS.log
```

新しい Replication Server ログ・ファイルへの変更

新しいエラー・ログ・ファイルを開始するには、**admin set_log_name** コマンドを使用します。

このコマンドを実行すると、現在のログ・ファイルがクローズし、新しいログ・ファイルがオープンします。以降のメッセージは、新しいログ・ファイルに書き込まれます。

たとえば、UNIX では、次のようなコマンドを入力します。

```
admin set_log_name, '/work/sybase/SYDNEY_RS/951101.log'
```

Replication Server が新しいログ・ファイルの作成およびオープンに失敗した場合は、前のログがアクティブなままになります。

RepAgent エラー・ログ・メッセージ

RepAgent のエラー・メッセージ、トレース・メッセージ、情報メッセージのログは、すべて Adaptive Server のエラー・ログ・ファイルに記録されます。

各メッセージでは、エラーのログを取った RepAgent を文字列 RepAgent (*dbid*) で識別できます。この文字列は、メッセージの 1 行目に表示されます。*dbid* は、エラーのログを取った RepAgent のデータベース ID 番号です。

これは情報メッセージです。

```
RepAgent(dbid): Recovery of transaction log is
complete. Please load the next transaction log dump and
then start up the Rep Agent Thread with
sp_start_rep_agent, with 'recovery' specified.
```

Adaptive Server のエラー・ログは、テキスト・ファイルです。メッセージは、Adaptive Server に指定された言語で出力されます。Adaptive Server のトランザクション・ログから複写オブジェクトを転送してコマンドに変換するときに発生するエラー・メッセージと情報メッセージが、RepAgent によって記録されます。RepAgent のエラーの範囲は、通常、9200 ～ 9299 です。

Adaptive Server では、エラーの重大度とリカバリ性に応じたアクションが実行されます。一部のエラーは情報提供のみを目的としたものですが、他のエラーの場合、Adaptive Server は成功するまでエラーの発生したオペレーションをリトライする場合もあります。また、重大度が高いエラーの場合は、RepAgent が停止することもあります。『Adaptive Server Enterprise トラブルシューティング』の「エラー・メッセージと詳細な解決方法」を参照してください。

RepAgent エラーのサンプル・メッセージ

RepAgent の一般的なエラー・メッセージと考えられる解決方法について説明します。

- 次は、RepAgent のログイン名が Replication Server に設定されていない場合の例です。

```
RepAgent(6): Failed to connect to Replication
Server. Please check the Replication Server,
username, and password specified to
sp_config_rep_agent. RepSvr = repserver_name, user =
RepAgent_username
```

```
RepAgent(6): This Rep Agent Thread is aborting due
to an unrecoverable communications or Replication
Server error.
```

  RepAgent のログイン名を Replication Server に追加するか、RepAgent のログイン名を変更する必要があります。

- 次は、RepAgent が Replication Server に接続できない場合の例です。

```
RepAgent(7): The Rep Agent Thread will retry the
connection to the Replication Server every 60
second(s). (RepSvr = repserver_name.)
```

  Replication Server のステータスをチェックします。Replication Server がダウンしている場合は、問題を解決してから再起動してください。または、考えられるネットワークの問題が解決されるまでお待ちください。

データ・サーバのエラー処理

Replication Server では、データ・サーバのエラーに対するエラー処理をユーザが設定できます。適切なエラー・クラスを指定されたコネクションに割り当て、割り当てられたエラー・クラスをカスタマイズします。エラー・アクションはデータ・サーバから返されるエラーと一致する必要があります。

エラー処理用の RCL コマンドとシステム・プロシージャ

エラーおよびエラー・クラスを管理する RCL コマンドと Adaptive Server システム・プロシージャがいくつかあります。

表 28 : エラー処理用の RCL コマンドとシステム・プロシージャ

コマンド	説明
assign action	1 つ以上のデータ・サーバ・エラーまたは Replication Server エラーに対して 1 つのエラー処理アクションを指定する。

コマンド	説明
alter error class	既存のエラー・クラスを変更する。
create error class	新しいエラー・クラスを作成する。
drop error class	既存のエラー・クラスを削除する。
alter connection	エラー・クラスと既存のデータベース・コネクションとを対応させる。
create connection	エラー・クラスと新しいデータベース・コネクションとを対応させる。
rs_helpclass	既存のエラー・クラス、ファンクション文字列クラス、プライマリ Replication Server の各名前を表示し、継承クラスの場合は親クラスも表示する Adaptive Server のストアド・プロシージャ。
rs_helperror	指定したデータ・サーバまたは Replication Server のエラー番号に割り当てられている Replication Server のエラー・アクションを表示する Adaptive Server のストアド・プロシージャ。

デフォルトのエラー・クラス

Replication Server は、デフォルトの Adaptive Server エラー・クラスとして **rs_sqlserver_error_class** を、デフォルトの Replication Server エラー・クラスとして **rs_repserver_error_class** を、ASE 以外のデータベースにデフォルトのエラー・クラスを提供します。これらのデフォルトのエラー・クラスは変更できません。

表 29 : ASE 以外のためのエラー・クラス

データベース	クラス名
IBM DB2	**rs_db2_error_class**
IBM UDB	**rs_udb_error_class**
Microsoft SQL Server	**rs_msss_error_class**
Oracle	**rs_oracle_error_class**
Sybase IQ	**rs_iq_error_class**

参照：

- エラー・クラスのプライマリ・サイトの指定 (363 ページ)

ASE 以外のデータベースのためのネイティブ・エラー・コード

Replication Server は、ASE 以外のレプリケート・サーバへのコネクションを確立するときに、コネクションで ASE 以外のレプリケート・サーバからネイティブ・エラー・コードが返されるオプションが有効になっているかどうかを検証します。

オプションが有効になっていない場合、Replication Server は、コネクションは機能しているが、エラー・アクションのマッピングが正確でない可能性があるという、警告メッセージをログに記録します。

Enterprise Connect™ Data Access (ECDA) Option for ODBC でレプリケート・サーバ用のオプションを設定するには、『Replication Server Options』の「Enterprise Connect Data Access Option for ODBC Users Guide for Access Services」の「Configuring the Access Service Library」の「Configuration Property Categories」の「Target Interaction Properties」の「**ReturnNativeError**」を参照してください。

エラー・クラスの作成

create error class は、独自のエラー・クラスを作成するときに使用します。

エラー・クラスとは、エラー・アクションの割り当てをグループ化するために使用される名前です。**create error class** により、テンプレートのエラー・クラスのエラー・アクションが新しいエラー・クラスにコピーされます。

1 つのエラー・クラスを定義すると、同じタイプのデータ・サーバで管理するすべてのデータベースでそのクラスを使用できます。たとえば、デフォルトの Adaptive Server のエラー・クラスである `rs_sqlserver_error_class` は、どの Adaptive Server データベースでも使用できます。データベースに特別なエラー処理要件がなければ、別のエラー・クラスを作成する必要はありません。

注意： 接続プロファイルを使用してコネクションを作成する場合は、接続プロファイルによってエラー・クラスが割り当てられます。接続プロファイルでは、特定のデータ・サーバに対してエラー・クラスが事前に定義されています。『Replication Server 管理ガイド 第 1 巻』の「データベース・コネクションの管理」の「データベースの複写準備」の「ASE 以外のサーバの複写準備」の「接続プロファイル」を参照してください。

エラー・クラスを作成するには、次のように入力します。

```
create [replication server] error class error_class
[set template to template_error_class]
```

replication_server オプションでは、Replication Server エラー・クラスを作成するかどうかを指定します。**set template to** オプション、およびエラー・クラスを作成するためのテンプレートとしてもう 1 つ別のエラー・クラスを使用することができます。

例

この例では、テンプレートのエラー・クラスを使用しないで、**pubs2_error_class** という名前のエラー・クラスを作成します。

```
create error class pubs2_error_class
```

この例では、**my_rs_err_class** Replication Server エラー・クラスを、デフォルトの Replication Server エラー・クラスである **rs_repserver_error_class** に基づいて作成します。

```
create replication server error class my_rs_err_class
set template to rs_repserver_error_class
```

この例では、**rs_oracle_error_class** をテンプレートとして、Oracle データベース用の **my_error_class** エラー・クラスを作成します。

```
create error class my_error_class
set template to rs_oracle_error_class
```

エラー・クラスのプライマリ・サイトの指定

プライマリ・サイトを指定してから、デフォルトのエラー・クラスを修正します。

`rs_sqlserver_error_class` と、その他のデフォルトの ASE 以外のためのエラー・クラスには、初期設定ではプライマリ・サイトがありません。サーバワイドなエラー・クラスは、クラスのプライマリ・サイトでしか作成できないため、**create error class** を使用して、Replication Server の 1 つをエラー・クラスのプライマリ・サイトとして指定します。

たとえば、Adaptive Server の場合は、プライマリ・サイトで **create error class rs_sqlserver_error_class** を実行します。その他すべての Replication Server に、プライマリ・サイトからの直接または間接のルートがあることを確認してください。

参照：

- エラー・クラスのプライマリ Replication Server の変更 (365 ページ)

エラー・アクションの割り当て

異なるエラー・アクションをデータ・サーバから返されるエラーに割り当てることができます。

データ・サーバから返されるすべてのエラーに対するデフォルトのエラー・アクションは **stop_replication** です。

これは、最も重大度の高いアクションでもあります。**suspend connection** コマンドを入力したときと同じように、データベースの複写がサスペンドされます。エラーに対する処理方法を変えて、重大度の低いアクションを割り当てるには、**assign action** コマンドを使用します。

参照：

- データ・サーバ・エラーへのアクションの割り当て (366 ページ)

エラー クラスの変更

alter error class は、テンプレートのエラー・クラスから変更対象のエラー・クラスにエラー・アクションをコピーし、同じエラー・コードを持つエラー・アクションを上書きします。

エラー・クラスを変更するには、次のように入力します。

```
alter [replication server] error class error_class
set template to template_error_class
```

replication_server オプションでは、Replication Server エラー・クラスを作成することを指定します。

たとえば、rs_sqlserver_error_class をテンプレートとして、Oracle データベース用の my_error_class を変更するには、次のように入力します。

```
alter error class my_error_class
set template to rs_sqlserver_error_class
```

新しいエラー・クラスの初期化

新しく作成したエラー・クラスは、システムが提供する rs_sqlserver_error_class などのエラー・クラスのエラー・アクションで初期化できます。

そのためには、**rs_init_erroractions** ストアド・プロシージャを使用します。

```
rs_init_erroractions new_error_class, template_class
```

たとえば、テンプレート・エラー・クラス rs_sqlserver_error_class をもとにエラー・クラス pubs2_error_class を作成するには、次のように入力します。

```
rs_init_erroractions pubs2_error_class, rs_sqlserver_error_class
```

次に、**assign action** コマンドを使用して個々のエラーに対するアクションを変更します。

エラー・クラスの削除

drop error class コマンドを実行すると、エラー・クラスとそのクラスに関連付けられているすべてのアクションが削除されます。

エラー・クラスは、削除されるときには、アクティブ・データベース・コネクションを介して使用中であってはなりません。**drop error class** の構文は次のとおりです。

```
drop [replication server] error class error_class
```

たとえば、pubs2_error_class という名前のエラー・クラスを削除するには、次のコマンドを使用します。

```
drop error class pubs2_error_class
```

rs_sqlserver_error_class またはデフォルトの ASE 以外のエラー・クラスは削除できません。

参照：

- デフォルトのエラー・クラス (361 ページ)

エラー・クラスのプライマリ Replication Server の変更

エラー・クラスのプライマリ・サイトを変更するには、**move primary** コマンドを使用します。

これは、プライマリ・サイトをある Replication Server から別の Replication Server に変更して、エラー・アクションを新しいルート経由で分配できるようにする場合に必要なことです。たとえば、エラー・クラスの現在のプライマリ・サイトである Replication Server を複写システムから削除する場合には、このコマンドを使用します。

move primary を実行する前に、次のルートが存在することを確認してください。

- 新しいプライマリ・サイトから、エラー・クラスを使用する各 Replication Server へのルート
- 現在のプライマリ・サイトから新しいプライマリ・サイトへのルート
- 新しいプライマリ・サイトから現在のプライマリ・サイトへのルート

エラー・クラスに対する **move primary** コマンドの構文は次のとおりです。

```
move primary
   of [replication server] error class class_name
   to replication_server
```

move primary コマンドは、エラー・クラスの新しいプライマリ・サイトとして指定する Replication Server で実行します。

パラメータは次のとおりです。

- **replication_server** オプションでは、Replication Server エラー・クラスのプライマリ Replication Server を変更することを指定します。データ・サーバのエラー・クラスを修正する場合は、このままにしておきます。
- *class_ name* ― 変更するプライマリ Replication Server のエラー・クラス名

- *replication_server* — エラー・クラスの新しいプライマリ Replication Server を指定します。

次のコマンドを実行すると、エラー・クラス pubs2_error_class のプライマリ・サイトは、コマンドを実行した場所である TOKYO_RS Replication Server に変更されます。

```
move primary of error class pubs2_error_class
 to TOKYO_RS
```

デフォルトのエラー・クラス rs_sqlserver_error_class には、ユーザが割り当てないかぎり、プライマリ・サイトである Replication Server は存在しません。プライマリ・サイトを指定してから、**assign action** コマンドを使用してデフォルトのエラー・アクションを変更します。

デフォルトのエラー・クラスにプライマリ・サイトを指定するには、Replication Server で次のコマンドを実行します。

```
create error class rs_sqlserver_error_class
```

このコマンドを実行した後、**move primary** コマンドを使用してエラー・クラスのプライマリ・サイトを変更することができます。

エラー・クラス情報の表示

rs_helpclass ストアド・プロシージャを使用して、プライマリ Replication Server と複写システム内の既存のエラー・クラスとファンクション文字列クラスの名前が表示されます。

次に例を示します。

```
rs_helpclass error_class
```

```
Error Class(es)              PRS for class
 --------------               ---------
rs_sqlserver_error_class    Not Yet Defined
```

『Replication Server リファレンス・マニュアル』の「RSSD ストアド・プロシージャ」の「**rs_helpclass**」を参照してください。

データ・サーバ・エラーへのアクションの割り当て

データ・サーバから Replication Server に返される可能性のあるエラーに対するアクションを指定するには、**assign action** コマンドを使用します。

```
assign action
  {ignore | warn | retry_log | log | retry_stop | stop_replication}
  for error_class
  to server_error1 [, server_error2]...
```

プライマリ・サイトでデフォルトのエラー・クラスを作成してから、**assign action** コマンドを使用してデフォルトのエラー・アクションを変更します。*data_server_error* パラメータは、データ・サーバのエラー番号です。

create connection コマンドと **alter connection** コマンドを使用して、エラー・クラスをレプリケート・データベースの特定のコネクションに割り当てることができます。

エラー・クラスを作成した Replication Server で、実行可能な 6 つのエラー・アクションのいずれかを入力します。**ignore** は最も重大度の低いアクションで、**stop_replication** は最も重大度の高いアクションです。エラー番号、エラー・メッセージ、該当するデフォルトのエラー・アクション、情報については、『Replication Server リファレンス・マニュアル』の「Replication Server コマンド」の「**assign action**」を参照してください。

1 つのトランザクションに複数のエラーが発生した場合、発生したエラーに対して割り当てられている最も重大度の高いアクションが 1 つ選択されて Replication Server によって実行されます。デフォルトのエラー・アクション **stop_replication** にエラーを返すには、このアクションを明示的に割り当て直す必要があります。

SQL 文の複写中に発生する可能性のある SQLDML のロー・カウント・エラーに Replication Server が対応する方法も指定できます。SQLDML のロー・カウント・エラーでは、SQL 文の複写後に、変更されたローの数がプライマリ・データベースとレプリケート・データベースで一致していません。Replication Server のデフォルトのエラー・アクションは、複写の停止です。デフォルトの Replication Server のエラー・クラスは `rs_repserver_error_class` です。

次に示すのは、ロー・カウント・エラー・メッセージの例です。

```
DSI_SQLDML_ROW_COUNT_INVALID 5186
Row count mismatch for the SQL Statement Replication
command executed on 'mydataserver.mydatabase'. The
command impacted 10 rows but it should impact 15 rows.
```

エラー・アクションの割り当て例

たとえば、Replication Server で Adaptive Server エラー 5701 と 5703 が無視されるようにするには、次のように指定します。

```
assign action ignore
 for rs_sqlserver_error_class
 to 5701, 5703
```

たとえば、Replication Server でロー・カウント・エラー (エラー番号 5186) が発生した場合に警告するには、次のように入力します。

```
assign action warn
   for rs_repserver_error_class to 5186
```

参照：
- データ・サーバ・エラーに対するエラー・アクション (368 ページ)
- デフォルトのエラー・クラス (361 ページ)

データ・サーバ・エラーに対するエラー・アクション

データ・サーバ・エラーに割り当てることができるエラー・アクションがいくつかあります。

表 30 : 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。リトライの回数は、**configure connection** コマンドで設定する。リトライ後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て例外ログにログを取ってから、次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。リトライの回数は、**configure connection** コマンドで設定する。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。これは **suspend connection** コマンドを使用するのと同じことである。デフォルトのアクションである。 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。

エラー番号に割り当てられたアクションの表示

エラー番号に割り当てられたアクションを表示するには、**rs_helperror** ストアド・プロシージャを使用します。

rs_helperror の構文は、次のとおりです。

```
rs_helperror server_error_number [, v]
```

server_error_number パラメータは、情報を表示するエラーのデータ・サーバ・エラー番号です。**v** パラメータを使用すると、「冗長」レポートを指定できます。このオプションを指定して **rs_helperror** を実行すると、Adaptive Server のエラー・

メッセージ・テキストがある場合は、それも表示されます。『Replication Server リファレンス・マニュアル』の「RSSD ストアド・プロシージャ」の「**rs_helperror**」を参照してください。

ロー・カウントの検証

Replication Server では、ロー・カウントの検証がデフォルトで有効になっているので、ロー・カウントの不一致などのさまざまなロー・カウント検証エラーに対して自動的にエラー・メッセージが表示され、デフォルト・エラー・アクションが実行されます。Replication Server エラー・クラスを設定してさまざまなエラー・アクションを有効にできます。

データ・サーバ・エラー・クラスと Replication Server エラー・クラスの 2 つのエラー・クラス・タイプにコネクションが関連付けられます。コネクションの Replication Server のエラー・クラスとの関連付けは、Replication Server がデフォルトの Replication Server エラー・アクションに対してオーバーライドする Replication Server エラー・クラスをクエリする前に行われる必要があります。コネクションとの関連付けは 1 つの Replication Server エラー・クラスにしかできません。ただし、1 つの Replication Server エラー・クラスを複数のコネクションに関連付けることはできます。Replication Server エラー・クラスとコネクションを関連付けるには、**create connection** コマンドと **alter connection** コマンドの **set replication server error class** パラメータを使用します。

Replication Server がエラーに応答するとき、そのコネクションに関連付けられた Replication Server エラー・クラスをまず最初に探します。Replication Server エラー・クラスが見つからなかったときは、そのサーバに指定されているデフォルトの **rs_repserver_error_class** エラー・クラスが使用されます。

注意： Replication Server は、カスタム・ファンクション文字列内のそのようなコマンドに対するロー・カウントの検証を無視します。

ロー・カウントの検証を制御する

dsi_row_count_validation を使用して、ロー・カウントの検証を無効にします。

同期されていないテーブル・ローがあり、デフォルトのエラー・アクションとメッセージをバイパスする場合、**dsi_row_count_validation** を **off** に設定してロー・カウントの検証を無効にできる。

デフォルトでは、**dsi_row_count_validation** は **on** に設定されているので、ロー・カウントの検証は有効になっています。

configure replication server は **dsi_row_count_validation** をサーバ・レベルで設定してすべてのレプリケート・データベース・コネクションに適用するときに使用し、**alter connection** は指定したデータベースとデータ・サーバへのコネクションに対してそのパラメータを設定するときに使用します。次に例を示します。

- すべてのデータベース・コネクションに対してロー・カウントの検証を無効にするため、次のように入力します。

```
configure replication server
set dsi_row_count_validation to 'off'
```

  configure replication server を **dsi_row_count_validation** を指定して実行した後、Replication Server へのすべてのデータベース・コネクションをサスペンドしてレジュームする必要があります。設定の変更はデータベース・コネクションをレジュームした後で有効になります。
- 特定のコネクションのロー・カウント検証を有効にする (次の例では SYDNEY_DS データ・サーバの `pubs2` データベースを指定) 場合、次のように入力します。

```
alter connection to SYDNEY_DS.pubs2
set dsi_row_count_validation to 'on'
```

  特定のコネクションに対して **dsi_row_count_validation** を設定した場合は、データベース・コネクションをサスペンドしてレジュームする必要はありません。パラメータはただちに有効になります。ただし、新しい設定は、このコマンドを実行した後で Replication Server が処理する複写オブジェクトのバッチに影響します。設定の変更は Replication Server が現在処理している複写オブジェクトのバッチには影響しません。

ロー・カウントの検証エラー・メッセージにテーブル名が表示される

ロー・カウントの検証エラー・メッセージにテーブル名が表示される。

使用しているものに応じて次のようにします。

- 連続モードのログ順、ローごとの複写 — Replication Server はテーブル名、テーブル所有者名、およびそのトランザクション・エラーの元となった出力コマンドを特定する番号をログに記録して表示します。Replication Server はテーブル名の最初の 30 バイトしかログに記録しません。
  DSI_CHECK_ROW_COUNT_FULL_NAME トレースを有効にすると、表示されるテーブル名の最大の長さが 255 バイトになります。
- High Volume Adaptive Replication (HVAR) または Real-Time Loading (RTL) — Replication Server は HVAR と RTL のコンパイルの結果できる内部の **join-update** 文と **join-delete** 文をログに記録して表示します。HVAR または RTL が HVAR と RTL の処理の一部としてコマンドを既にコンパイルした後なので、トランザクション・エラーの原因になったコマンド自体を取得することはできません。表示できる **join-update** 文と **join-delete** 文の最大の長さは 128 バイトです。これには末尾の `"...¥0"` も含まれます。

この例は次のもので構成されます。

- プライマリ・サイト ― pdb1 というプライマリ・データベース。3 カラム 3 ローからなる ThisTableHasANameLongerThan30Characters という名前のテーブルがあります。

id	name	age
1	John	40
2	Paul	38
3	George	37

- レプリケート・サイト ― rdb1 というプライマリ・データベース。ローが 2 つあり、id カラムの値がそれぞれ 1 と 3 になっている ThisTableHasANameLongerThan30Characters というテーブルがあります。

次のコマンドをこの pdb1 に対して実行します。

```
update ThisTableHasANameLongerThan30Characters set age = 20
```

エラー・メッセージは複写モードによって異なります。

- 連続モードのログ順、ローごとの複写では次のようになります。

```
I. 2010/06/07 01:30:21.DSI received Replication Server
error #5185 which is mapped to WARN by error action mapping.
W. 2010/06/07 01:30:21.WARNING #5185 DSI EXEC(103(1)
ost_replnx6_61.rdb1) - /dsiexec.c(11941)
Row count mismatch for the command executed on
'ost_replnx6_61.rdb1'.The command impacted 0 rows but it
should impact 1 rows.
I. 2010/06/07 01:30:21.The error was caused by output
command #3 of the failed transaction on table
'dbo.ThisTableHasANameLongerThan30C'.
```

注意： テーブル名はデフォルトの 30 バイトにトランケートされます。

エラー・メッセージが表示できるテーブル名の最大の長さを 255 バイトにするために **DSI_CHECK_ROW_COUNT_FULL_NAME** 追跡を on にした場合、エラー・メッセージの最後の行に完全なテーブル名が表示されます。

```
I. 2010/06/07 02:22:55.The error was caused by output
command #3 of the failed transaction on table
'dbo.ThisTableHasANameLongerThan30Characters'.
```

- HVAR または RTL の複写では次のようになります。

```
W. 2010/06/07 02:06:56.WARNING #5185 DSI EXEC(103(1)
ost_replnx6_61.rdb1) - i/hqexec.c(4047)
```

```
Row count mismatch for the command executed on
'ost_replnx6_61.rdb1'.The command impacted 1 rows but it
should impact 2 rows.
I. 2010/06/07 02:06:56.(HQ Error):update
ThisTableHasANameLongerThan30Characters set age = w.age
from ThisTableHasANameLongerThan30Characters
t,#rs_uThisTab...
I. 2010/06/07 02:06:57.The DSI thread for database
'ost_replnx6_61.rdb1' is shutdown.
```

例外処理

Replication Server が送信したトランザクションが失敗すると、Replication Server はそのトランザクションを RSSD の例外ログに記録します。サイトの複写システム管理者は、例外ログのトランザクションの問題を解決する必要があります。

トランザクションが失敗する理由には、重複キー、カラム値のチェック、ディスク領域不足などのエラーなどがあります。不十分なパーミッション、バージョン管理における競合、無効なオブジェクト参照などによっても、トランザクションは拒否されます。

トランザクションが省略されると矛盾が発生し、システムに悪影響を与えるため、例外ログ内に記録されたトランザクションを定期的にチェックし、問題を解決してください。トランザクションに対する最適な解決方法は、トランザクションを生成したクライアント・アプリケーションによって異なります。たとえば、失敗したトランザクションが、現金引き出しなどの現実のイベントに対応している場合は、トランザクションを何らかの方法で適用しなければなりません。

トランザクションの省略による影響の詳細については、『Replication Server トラブルシューティング・ガイド』を参照してください。

参照：

- 例外ログへのアクセス (374 ページ)

失敗したトランザクションの処理

失敗したトランザクションをユーザが手動で処理する必要がある場合の推奨手順について説明します。

データベース・コネクションのサスペンド

データベースやログ・ファイルの領域不足などの一時的な障害が原因で、データ・サーバからトランザクションが拒否され始めた場合は、エラーを解決するまでデータベース・コネクションをサスペンドできます。

データベース・コネクションをサスペンドしないと、Replication Server によってトランザクションがデータベースの例外ログに書き込まれます。このようなトラン

ザクションの問題は手動で解決する必要があるため、エラー状態を解決するまでコネクションを停止すると時間を節約できます。

データベース・コネクションがサスペンドされている間、Replication Server によりトランザクションがステーブル・キュー内に格納されます。コネクションがレジュームされると、格納されていたトランザクションがデータ・サーバに送信されます。

Replication Server からデータベースへのトランザクションの流れを止めるには、次のように **suspend connection** を使用します。

```
suspend connection to data_server.database
```

このコマンドを実行するには、**sa** パーミッションが必要です。また、このコマンドは、データベースを管理する Replication Server で入力します。

問題の分析と解決

トランザクションが失敗した原因を明らかにして、修正または調整をした後、トランザクションを再送信します。

1. 例外ログからトランザクションのリストを取得します。
2. トランザクションを調べて、障害の原因と最適な解決方法を判断します。
3. その判断に従ってトランザクションの問題を解決します。たとえば、パーミッションの問題を解決してから、トランザクションを再送信します。
4. 例外ログから解決済みのトランザクションを削除します。

たとえば、メンテナンス・ユーザに付与されているパーミッションが不十分なためにトランザクションが失敗した場合は、メンテナンス・ユーザに必要なパーミッションを付与してからトランザクションをリトライします。

参照：

- 例外ログへのアクセス (374 ページ)
- 例外ログからのトランザクションの削除 (376 ページ)

コネクションのレジューム

サスペンドされているデータベース・コネクションのトランザクションの流れを再起動するには、**resume connection** コマンドを使用します。

suspend connection コマンドを使用して意図的にコネクションをサスペンドした場合にも、エラー・アクションの結果として Replication Server によってコネクションがサスペンドされた場合にも、同じコマンドを使用します。

```
resume connection to data_server.database
[skip transaction]
```

このコマンドを実行するには、**sa** パーミッションが必要です。また、このコマンドは、データベースを管理する Replication Server で入力します。

キュー内の最初のトランザクションを無視するように Replication Server に指示するには、**skip transaction** 句を使用します。コネクションをレジュームするたびにトランザクションが失敗し続ける場合、この作業が必要な場合があります。

例外ログへのアクセス

Replication Manager は、例外ログのトランザクションを表示および管理するためのグラフィカル・インタフェースを提供します。

例外ログでのトランザクションの表示

例外ログのすべてのトランザクションの概要リストを表示するには、**rs_helpexception** ストアド・プロシージャを使用します。

```
rs_helpexception [transaction_id, [, v]]
```

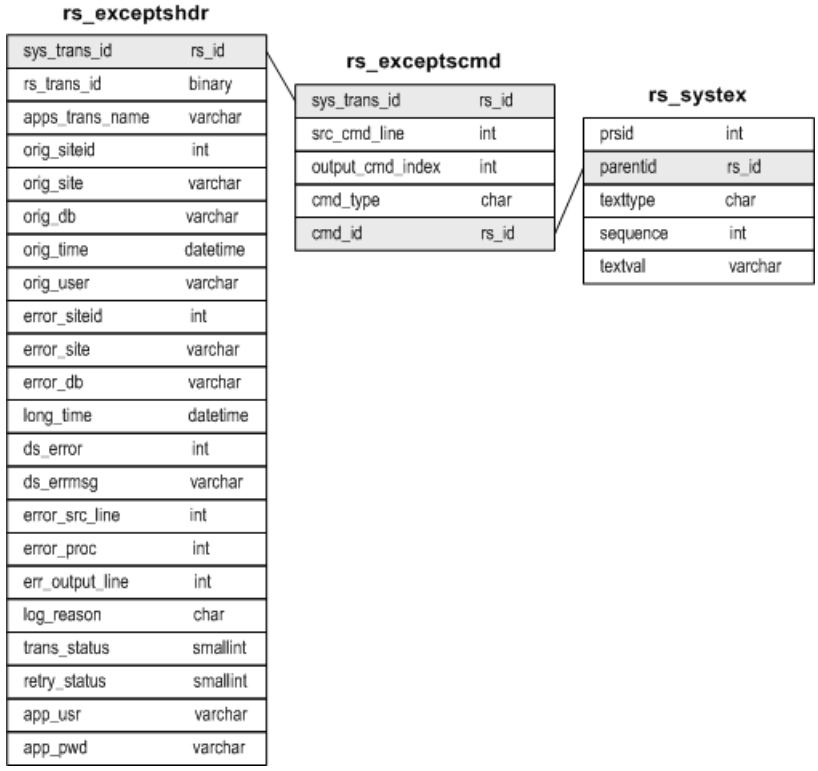
「冗長」レポートを要求するために、有効な *transaction_id* と **v** を指定して **rs_helpexception** を実行すると、トランザクションについての詳細な説明が表示されます。例外ログのすべてのトランザクションの *transaction_id* 番号を表示するには、パラメータを指定せずに **rs_helpexception** を実行します。

例外ログのシステム・テーブルに対するクエリの実行

例外に関する情報を取得するために、`rs_exceptshdr` システム・テーブルと `rs_exceptscmd` システム・テーブルを `sys_trans_id` カラムでジョインできます。

また、`rs_exceptscmd` システム・テーブルと `rs_systext` システム・テーブルをジョインして、トランザクションのテキストを取得することもできます。そのためには、`rs_exceptscmd` の `cmd_id` カラムを `rs_systext` の `parentid` カラムにジョインします。

図 23 : 例外ログのシステム・テーブル



`rs_exceptshdr` システム・テーブルには、例外ログのトランザクションに関する次のような記述情報があります。

- ユーザが割り当てたトランザクション名
- トランザクションを開始したサイトとデータベース
- トランザクションを送信したオリジン・サイトのユーザ
- トランザクションが例外ログに記録された原因となったエラーに関する情報

特定のデータベースに対して例外処理されたトランザクションのリストを取得するには、次のようなクエリを使用します。

```
select * from rs_exceptshdr
 where error_site = 'data_server'
 and error_db = 'database'
 order by log_time
```

特定のシステム・トランザクション ID に対応するトランザクションのソース・テキストと出力テキストを取得するには、次のクエリを使用します。

```
select t.texttype, t.sequence,
 t.textval
 from rs_systext t, rs_exceptscmd e
```

```
 where e.sys_trans_id = sys_trans_id
and t.parentid = e.cmd_id
 order by e.src_cmd_line, e.output_cmd_index,
 t.sequence
```

これらの Replication Server システム・テーブルの全カラムのリストについては、『Replication Server リファレンス・マニュアル』の「Replication Server システム・テーブル」を参照してください。

例外ログからのトランザクションの削除

ストアド・プロシージャを使用して、RSSD 例外ログからトランザクションを削除します。

- **rs_delexception** – 例外ログから 1 つのトランザクションを削除します。次に構文を示します。

  ```
  rs_delexception [transaction_id]
  ```

  パラメータを指定せずに **rs_delexception** を実行すると、例外ログのトランザクションの概要が表示されます。有効な *transaction_id* を指定して **rs_delexception** を実行すると、トランザクションが削除されます。トランザクションの *transaction_id* を調べるには、パラメータを指定しないで **rs_helpexception** または **rs_delexception** を実行します。

  たとえば、ID 番号 1234 のトランザクションを削除するには、次のように入力します。

  ```
  rs_delexception 1234
  ```

- **rs_delexception_id** – トランザクション ID によって指定された範囲のトランザクションを削除します。次に構文を示します。

  ```
  rs_delexception_id transaction_id_start [,transaction_id_end]
  ```

  たとえば、ID 番号 1234 ～ 9800 のトランザクションをすべて削除するには、次のように入力します。

  ```
  rs_delexception_id 1234, 9800
  ```

- **rs_delexception_date** – トランザクションの日付によって指定された範囲のトランザクションを削除します。次に構文を示します。

  ```
  rs_delexception_date transaction_date_start
  [,transaction_date_end]
  ```

  たとえば、開始の日付が "10/01/2010" ～ "10/31/2010" の範囲にあるトランザクションを例外ログから削除するには、次のように入力します。

  ```
  rs_delexception_date "10/01/2010", "10/31/2010"
  ```

- **rs_delexception_range** – 元のサイトまたはユーザ、または送信先サイトで指定した範囲のトランザクションを削除します。次に構文を示します。

  ```
  rs_delexception_range
  {{"origin"|"org"}, "origin_data_server.origin_database" |
  , {"destination"|"dest"},
  ```

```
"destination_data_server.destination_database" |
, "user", "origin_user"}
```

たとえば、SYDNEY_DS データ・サーバの south_db データベースが開始したトランザクションを例外ログから削除するには、次のように入力します。

```
rs_delexception_range "org", "SYDNEY_DS.south_db"
```

完全な使用方法とその他の例については、『Replication Server リファレンス・マニュアル』の「RSSD ストアド・プロシージャ」のストアド・プロシージャの説明を参照してください。

RM と RMS を使用した複写環境のモニタリングの詳細については、『Replication Server 管理ガイド 第 1 巻』の「Sybase Central での複写環境の管理」の「複写環境の設定」の「Replication Server オブジェクトの管理」の「キュー」の「キュー・データの表示」を参照してください。

DSI による重複検出

DSI は、コミットされた、または例外ログに書き込まれた最後のトランザクションを記録して、システムの再起動後に重複を検出できるようにします。各トランザクションは、ユニークなオリジン・データベース ID と、各トランザクションで増加するオリジン・キュー ID によって識別されます。

各オリジン・データベースでコミットされた最後のトランザクションは、データ・サーバのファンクション文字列クラスに定義されたファンクション文字列の実行によって、データ・サーバに記録されます。システム定義クラスの場合、これは、**commit** コマンドのファンクション文字列内 (**rs_commit** ファンクション) で実行されます。すべてのファンクション文字列クラスは、各オリジン・データベースの origin_qid と secondary_qid を返す **rs_get_lastcommit** ファンクションをサポートしています。secondary_qid は、サブスクリプション・マテリアライゼーションまたはマテリアライゼーション解除に使用されるキューの ID です。

各オリジンから例外ログに書き込まれた最後のトランザクションの origin_qid と secondary_qid は、rs_exceptslast システム・テーブルに記録されます。ただし、**sysadmin log_first_tran** コマンドによって明示的に記録されたトランザクションは、このシステム・テーブルには記録されません。これらのトランザクションは、ログが記録されますが、省略されません。

DSI は、起動または再起動すると、**rs_get_lastcommit** ファンクションによって返された origin_qid と rs_exceptslast システム・テーブルに格納された値を取得します。これら 2 つのうち大きい方の値よりも小さい値の origin_qid を持つトランザクションがキュー内にある場合、DSI はそれらをすべて重複トランザクションとみなし、無視します。

データ・サーバまたは rs_exceptslast システム・テーブルに格納された origin_qid の値が誤って修正された場合は、重複していないトランザクションが無視されたか、または重複するトランザクションが再適用された可能性があります。システム内でこれらのエラーが発生している疑いがある場合は、格納されている値とデータベースのステーブル・キュー内のトランザクションに指定されている値とを比較し、値の妥当性を確認します。値が誤っている場合は、それらの値を直接修正してください。

キュー内のトランザクションをダンプする方法の詳細については、『Replication Server トラブルシューティング・ガイド』を参照してください。

システム・トランザクションの重複検出

システム・トランザクション実行の失敗を検出し解決する方法について説明します。

truncate table コマンドとサポートされている特定の DDL コマンドのログは取られませんが、これらのコマンドをスタンバイ・データベースやレプリケート・データベースに複写できます。各 DDL コマンドについては、『Adaptive Server Enterprise リファレンス・マニュアル』を参照してください。

Replication Server はこれらのコマンドをシステム・トランザクションとしてコピーし、2つの完了したトランザクションの間に **truncate table** または同様のコマンドを「差し込み」ます。最初のトランザクションの実行は、レプリケート・データベースの rs_lastcommit テーブルの secondary_qid カラムと origin_qid カラムに記録されます。Replication Server によって 2 番目のトランザクションが記録される場合、そのシステム・トランザクションは完了しており、secondary_qid カラムがクリアされます。

システム障害が発生した後、次のようなエラー・メッセージは、システム・コマンドが完了しなかったことを示します。コネクションは停止します。

```
5152 DSI_SYSTRAN_SHUTDOWN,"There is a system
transaction whose state is not known. DSI will be
shutdown."
```

システム・トランザクション内のコマンドがレプリケート・データベースで実行されたかどうかを確認してください。

- コマンドが実行された、またはユーザ自身がコマンドを実行する場合、コネクションをレジュームしたときに、キュー内の最初のトランザクションを省略します。レプリケート Replication Server で次のように入力します。

  ```
  resume connection to data_server.database
  skip transaction
  ```

- コマンドが実行されていない場合には、問題を解決してから、キュー内の最初のコマンドを実行できます。レプリケート Replication Server で次のように入力します。

```
resume connection to data_server.database
execute transaction
```

skip transaction を含めるか、または **execute transaction** 句を **resume connection** で実行する必要があります。そうしないと、Replication Server で `secondary_qid` が正しく再設定されず、エラー・メッセージが再び表示されます。

参照：

- サポートされている DDL コマンドとシステム・プロシージャ (76 ページ)

複写システム・リカバリ

Replication Server は、ほとんどの障害に対する耐性があり、自動的にリカバリしますが、障害によってはユーザの介入が必要になるものもあります。障害、および失われたデータおよび破壊されたデータをリカバリし、それを前の状態にリストアすることによって、複写システムの整合性を保つように設計されているリカバリ手順を指定する方法について説明します。

そのため、バックアップおよびリカバリを前提として複写システムを設計、インストール、管理します。また、ダンプが定期的に実行されていることや、リカバリ処理用の適切なツールが使用され、設定値も適切であることが前提となります。

リカバリについては、リカバリ対象のデータベース (RSSD など) がある Replication Server のことを「現在の」Replication Server と呼びます。現在の Replication Server への直接ルートまたは間接ルートを持つ Replication Server のことを「アップストリーム」の Replication Server と呼びます。現在の Replication Server からの直接ルートまたは間接ルートを持つ Replication Server のことを「ダウンストリーム」の Replication Server と呼びます。

たとえば、複写だけによるデータベースの回復が不可能なプライマリ・データベースとレプリケート・データベース間の複写の遅延時間がある場合、複写環境にレプリケート・データベースを再同期することができます。

参照：

- コーディネート・ダンプの作成 (390 ページ)
- Adaptive Server のレプリケート・データベースの再同期 (431 ページ)

リカバリ手順の使用方法

リカバリ手順を使用するときは、実行したリカバリ手順を書き留めたり、チェック・マークを付けるなどしてください。これらの情報は、Sybase 製品の保守契約を結んでいるサポート・センタに問い合わせる場合、どのリカバリ手順までが実行済みであるかをサポート担当者が把握するのに役立ちます。

それぞれの障害条件に、対応する障害現象とリカバリ手順があります。

警告！ リカバリ手順は、各手順の対象となっている障害だけに使用してください。データ複写の障害などの複写システムの問題には、これらのリカバリ手順を実行しないでください。リカバリ手順を、その手順の対象となっている障害とは別の障害で使用しようとすると、問題がより複雑化し、さらに抜本的なリカバリ対処方法が必要になります。

問題の診断と解決については、『Replication Server トラブルシューティング・ガイド』を参照してください。

参照：
- パーティションのロスまたは障害からのリカバリ (391 ページ)
- トランケートされたプライマリ・データベース・ログからのリカバリ (396 ページ)
- プライマリ・データベース障害からのリカバリ (399 ページ)
- RSSD 障害からのリカバリ (402 ページ)
- Adaptive Server のレプリケート・データベースの再同期 (431 ページ)

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

バージョン 12.0 以降の Replication Server が、バージョン 12.0 以降の Adaptive Server の Sybase フェールオーバ機能をサポートする方法について説明します。

Sybase フェールオーバを使用すると、バージョン 12.0 以降の 2 つの Adaptive Server をコンパニオンとして設定できます。プライマリ・コンパニオンの Adaptive Server に障害が発生した場合、そのサーバのデバイス、データベース、コネクションをコンパニオンの Adaptive Server が引き継ぐことができます。

高可用性システムは、非対称型にも対称型にも設定できます。

「非対称型設定」は、物理的に異なるマシンに置かれていても、同じシステム・デバイス、システム・データベース、master データベース、ユーザ・データベース、ユーザ・ログインを共有する 2 つの Adaptive Server で構成されます。これらの 2 つのサーバが接続され、1 つのサーバが停止した場合に、もう 1 つのサーバがその負荷を引き継ぐことができます。コンパニオンの Adaptive Server は、「ホット・スタンバイ」として機能し、フェールオーバが発生するまで処理を実行することはありません。

「対称型設定」も、別のマシン上で動作する 2 つの Adaptive Server で構成されますが、それぞれの Adaptive Server は、独自にシステム・デバイス、システム・データベース、master データベース、ユーザ・データベース、ユーザ・ログイン情報を持ち、完全に機能しています。フェールオーバが発生した場合、どちらの Adaptive Server も、もう一方の Adaptive Server のコンパニオンとして動作することができます。

どちらの設定の場合でも、2 台のマシンはデュアル・アクセス用に設定されます。これにより、ディスクは両方のサーバで参照およびアクセスできるようになります。

複写システムでは、Replication Server は Adaptive Server に何度も接続します。このシステムでは、Replication Server で開始された Adaptive Server へのデータベース・コネクションのフェールオーバ・サポートを有効または無効にできます。フェールオーバ・サポートを有効にすると、障害が発生した Adaptive Server に接続された Replication Server はコンパニオン・マシンに自動的に切り替わり、ネットワーク・コネクションが再確立されます。

Adaptive Server Enterprise のマニュアルの「高可用性システムにおける Sybase フェールオーバの使用」を参照してください。

参照：

- Sun Cluster 2.2 での高可用性 (463 ページ)

Replication Server のフェールオーバ・サポートの有効化

使用するシステムの Replication Server ごとにフェールオーバ・サポートを有効にします。フェールオーバ・サポートは、RSSD (Replication Server システム・データベース) コネクションに対して 1 回、指定した Replication Server から Adaptive Server へのその他のすべてのデータベース・コネクションに対して 1 回有効にします。

RSSD コネクション以外では、個々のコネクションごとにフェールオーバ・サポートを有効にすることはできません。

Replication Server のフェールオーバ・サポートのデフォルトは、Replication Server から Adaptive Server へのすべてのコネクションに対して、「off」です。

複写を続ける場合、すべてのコネクションに対するフェールオーバ・サポートを有効にする必要があります。しかし、コンパニオン・サーバの容量を超える負荷がかかった場合には、コネクション・フェールオーバを無効にすることもできます。

Replication Server での Sybase フェールオーバの動作

Replication Server から Adaptive Server への Sybase フェールオーバを設定するには、Adaptive Server でコネクション・フェールオーバを許可するよう設定されている必要があります。

Adaptive Server がフェールオーバ・コンパニオン・モードの場合、プライマリ・コンパニオンに障害が発生すると、セカンダリ・コンパニオンがその負荷を引き継ぎます。RSSD に対する更新を必要とする不完全なトランザクションやオペレーションは失敗します。Replication Server は既存のコネクションについてはリトライしますが、新しいコネクションはフェールオーバされます。

データ・サーバ・インタフェース (DSI) コネクションの場合、DSI は短時間スリープした後、失敗したトランザクションをリトライします。

RSSD に対するコネクションの場合、フェールオーバ中に実行されるユーザのコマンドは成功しません。内部オペレーション (たとえば、ロケータとディスク・セグメントに対する更新) は失敗しないはずです。RSSD オブジェクトの複写は、DSI で扱われます。

非同期コマンド (サブスクリプション、ルート指定、スタンバイのコマンドなど) は拒否されたり、エラーになったりすることがあります。またコマンドが受け入れられたのに完了しなかった場合は、リカバリが必要になることがあります。たとえば、create subscription コマンドが受け入れられたのに、サブスクリプションが作成中のままである場合があります。

注意： 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。フェールオーバ・サポートは、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に対しても同じ働きをします。

Sybase のフェールオーバ・サポートの稼働条件

フェールオーバ・サポートの稼働条件には、いくつかあります。

- フェールオーバ・サポートを有効にするには、フェールオーバ用に設定されたバージョン 12.0 以降の Adaptive Server に Replication Server が接続している必要があります。
- RSSD とユーザ・データベースのフェールオーバが、Adaptive Server を介して直接設定されている必要があります。
- フェールオーバ・サポートは、Adaptive Server のフェールオーバのみに応答します。つまり Replication Server のフェールオーバはサポートされていません。
- Adaptive Server は、RepAgent スレッドをフェールオーバし、フェールオーバ／フェールバック後に RepAgent スレッドを Replication Server に再接続します。
- 各 Replication Server が独自のコネクションを設定します。

RSSD コネクションに対するフェールオーバ・サポートの有効化

Replication Server をインストールした後に、RSSD コネクションに対してフェールオーバ・サポートを有効にする設定ファイルを編集します。

また、**rs_init** を使用すると、新しい Replication Server をインストールするときにフェールオーバ・サポートを有効にすることもできます。『Replication Server 設定ガイド』の「rs_init による Replication Server の設定とデータベースの追加」を参照してください。

1. テキスト・エディタを使用して、Replication Server の設定ファイルを開きます。

デフォルトのファイル名は、Replication Server の名前に拡張子 `.cfg` が付いたものです。設定ファイルは 1 行 1 エントリになっています。

2. `RSSD_ha_failover=no` 行を検索し、`RSSD_ha_failover=yes` に変更します。

   `RSSD_ha_failover=no` を設定することによって、RSSD コネクションに対するフェールオーバ・サポートを無効にすることができます。

   これらの変更はすぐに反映されます。つまり、フェールオーバ・サポートを有効にするために Replication Server を再起動する必要はありません。

RSSD 以外のデータベース・コネクションに対するフェールオーバ・サポートの有効化

configure replication server と **ha_failover** を使用して、Replication Server から Adaptive Server への新しいデータベース・コネクションに対するフェールオーバ・サポートを有効にします。

Adaptive Server Enterprise のマニュアルの「高可用性システムにおける Sybase フェールオーバの使用」を参照してください。

1. 必要に応じて、Replication Server を起動します。

   『Replication Server 管理ガイド第 1 巻』の「複写システムの管理」の「Replication Server の起動」を参照してください。

2. Replication Server にログインします。

   ```
   isql -Uuser_name -Ppassword -Sserver_name
   ```

   この *user_name* には、管理者権限が必要です。**-S** フラグを使用して、Replication Server の名前を指定します。

   ログインが受け入れられると、**isql** から次のプロンプトが表示されます。

   ```
   1>
   ```

3. **ha_failover** を設定します。

   ```
   configure replication server
   set ha_failover to 'on'
   ```

データ・ロスを防ぐための複写システムの設定

リカバリ不可能なデータベース・エラーの発生時にデータ・ロスを防ぐための推奨手段について説明します。これらの手段を適切に使用すれば、システムのリカバリ手順を使用して複写データをリストアできます。

リカバリのためのセーブ・インターバル

Replication Server は送信元 Replication Server からのメッセージを格納して、送信先 Replication Server に転送するように設計されています。ステーブル・キューの再構築後にオンライン・メッセージをリカバリできる可能性を高めるために、Replication Server 間のルートに対して分単位でセーブ・インターバルを設定できます。

セーブ・インターバルとは、メッセージが転送後に格納されている時間です。Replication Server からの物理または論理データベース・コネクションのセーブ・インターバルを設定して、Replication Server が DSI アウトバウンド・キューにメッセージを保存できるようにすることもできます。

ルートまたはコネクションに対する現在のセーブ・インターバルを検索するには、**admin who, sqm** コマンドを使用します。`Save_Int:Seg` カラムには 2 つの値が保持されています。コロンの前の値がセーブ・インターバルです。コロンの後ろの値は、ステーブル・キュー内に最初に保存されたセグメントです。

Replication Server 間のルート

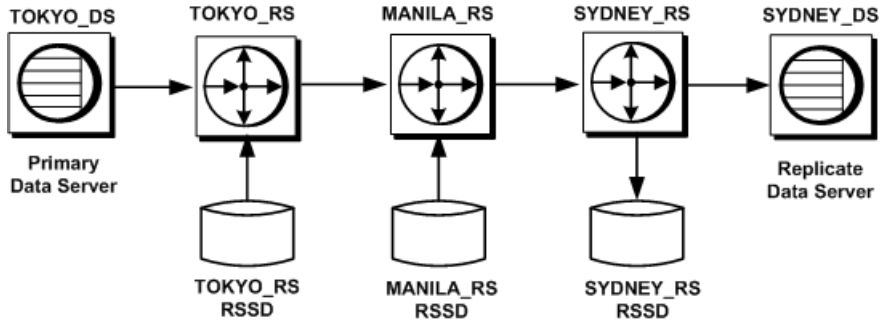
メッセージのリカバリのために、Replication Servers 間のルートにセーブ・インターバルを設定することができます。

Replication Server がルートをサスペンドしている場合、あるいはネットワークまたはデータ・サーバのコネクションがダウンしている場合、メッセージのバックログが Replication Server のステーブル・キューに蓄積されていることがあります。これらのメッセージをリカバリできる可能性は時間とともに低くなります。送信元 Replication Server がそれらのステーブル・キューからメッセージをすでに削除していたり、データベース・ログがすでにトランケートされていることがあるからです。

Replication Server 間の個々のルートに対して *save_interval* パラメータを設定した場合、個々の Replication Server はルート内の次のサイトがメッセージを受信したことを確認した後、最小時間だけメッセージを保持できます。これらのメッセージが利用できると、キューが再構築された後、オンライン・メッセージをリカバリできる可能性が高くなります。

たとえば、次の図では、Replication Server TOKYO_RS は MANILA_RS への直接ルートを管理し、MANILA_RS は SYDNEY_RS への直接ルートを管理しています。

図 24 : セーブ・インターバルの例



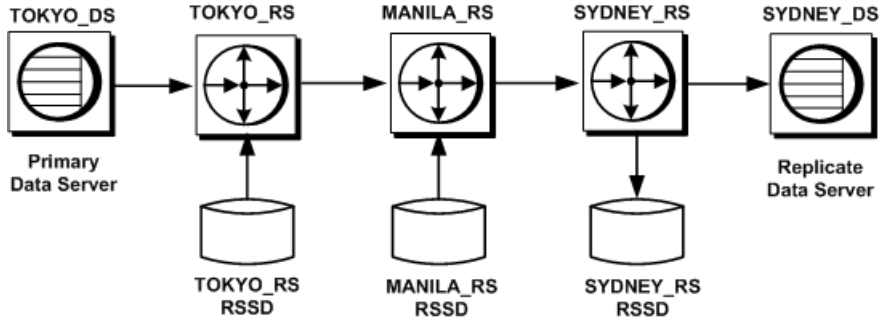
TOKYO_RS は、MANILA_RS がメッセージを受信した後、一定期間メッセージを保持します。MANILA_RS でパーティションの障害が発生した場合は、バックログされたメッセージの TOKYO_RS からの再送信が必要になります。MANILA_RS も、SYDNEY_RS が障害からリカバリできるようにメッセージを保持しておくことができます。

ステーブル・キュー・セグメントに格納されているすべてのメッセージが *save_interval* に設定された時間を経過すると、Replication Server がそのセグメントを削除するので、再び利用できるようになります。

ルートに対するセーブ・インターバルの設定

ルートに対してセーブ・インターバルを設定するには、*save_interval* パラメータを持つ **alter route** コマンドを送信元 Replication Server で実行します。

図 25 : セーブ・インターバルの例



たとえば、TOKYO_RS という Replication Server が MANILA_RS 宛のすべてのメッセージを 1 時間保存するように設定するには、次のように入力します。

```
alter route to MANILA_RS
    set save_interval to '60'
```

デフォルトでは、*save_interval* は 0 (分) に設定されています。低容量システムの場合は、このデフォルト値はリカバリのための許容可能な設定です。その理由は、Replication Server が送信先のサーバから受信確認を受け取った直後にメッセージを削除しないからです。むしろ、メッセージは一定のまとまりで定期的に削除されます。

ただし、Replication Server からの分配を受け取るサイトの容量とアクティビティに対応できるようにして、データベースやパーティションの障害から完全にリカバリする可能性を高めるには、*save_interval* の設定を変更する必要があります。

ステーブル・キューでパーティション障害が発生した場合に備えて、使用しているシステムをリストアするのに十分な時間が確保されるように設定してください。また、バックログされたメッセージに対して割り付けられているパーティションのサイズも考慮してください。パーティションは、追加のメッセージを保持するのに十分な大きさにします。

キュー領域の条件を調べるのに役立つ情報については、『Replication Server デザイン・ガイド』の容量計画のガイドラインを参照してください。

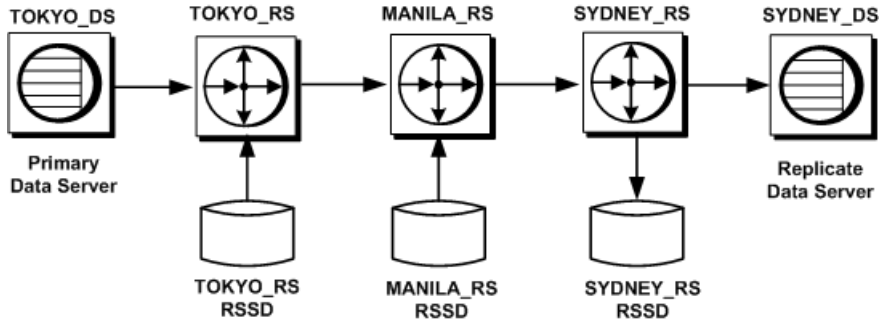
Replication Server とデータ・サーバ間のコネクション

メッセージのリカバリのために、Replication Servers 間のコネクションにセーブ・インターバルを設定することができます。

Replication Server とデータ・サーバおよびデータベース間の物理コネクションまたは論理コネクションに対して *save_interval* を設定する場合は、Replication Server が DSI キューにトランザクションを保存できるようにします。**sysadmin restore_dsi_saved_segments** を使用すると、バックログ・トランザクションをリストアできます。『Replication Server リファレンス・マニュアル』の「Replication Server コマンド」の「**sysadmin restore_dsi_saved_segments**」を参照してください。

データベースがトランザクション・ダンプとデータベース・ダンプから直前の状態にロードされた後で、保存されているこれらのトランザクションを使用してそのデータベースを再度同期できます。

図 26 : セーブ・インターバルの例



たとえば、この図のように、Replication Server SYDNEY_RS に接続しているレプリケート・データ・サーバ SYDNEY_DS で障害が発生した場合、この SYDNEY_DS は、SYDNEY_RS の DSI キューに保存されたメッセージを取得して、レプリケート・データベースをリストア後に再度同期できます。

いくつかのレプリケート・データを保持するデータベースや、適用されるファンクションを受け取るデータベースのウォーム・スタンバイを設定するために *save_interval* を使用することもできます。

コネクションに対するセーブ・インターバルの設定

データベース・コネクションに対してセーブ・インターバルを設定するには、*save_interval* パラメータを持つ **alter connection** コマンドを送信元 Replication Server で実行します。

たとえば、Replication Server SYDNEY_RS が、レプリケート・データ・サーバ SYDNEY_DS 宛のすべてのメッセージを 1 時間保存するように設定するには、次のように入力します。

```
alter connection to SYDNEY_DS.pubs2
    set save_interval to '60'
```

デフォルトでは、*save_interval* は 0 (分) に設定されています。

論理コネクションに対する DSI キューとマテリアライゼーション・キューのセーブ・インターバルも設定できます。

参照：

- 論理コネクションのセーブ・インターバルの設定 (130 ページ)

RASD のバックアップ

ルートの変更やサブスクリプションの追加などの任意の複写 DDL に続けて RSSD のダンプを実行します。

RSSD を最新の状態にリカバリできない場合、リカバリは複雑になります。使用する手順は、最後のダンプを行ったあと、RSSD のアクティビティがどの程度あったかに依存します。

参照：

- ダンプから RSSD をリカバリするプロシージャ (403 ページ)

コーディネート・ダンプの作成

バックアップをリストアしてプライマリ・データベースをリカバリする必要がある場合、他のサイトにある影響を受けるデータベース内のレプリケート・データがプライマリ・データと一貫性があることも確認してください。

複数のデータ・サーバでリストア後の一貫性を保つため、Replication Server には、複写システム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・ダンプをコーディネートするための方法が用意されています。

プライマリ・データベースからデータベース・ダンプまたはトランザクション・ダンプを開始します。RepAgent 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・サイトに分配されるようにします。この方法によって、すべてのデータを特定の一貫性ポイントまで確実に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が格納されたデータベースだけで実行できます。プライマリ・データベース内からコーディネート・ダンプを開始します。

ダンプをコーディネートする処理は、次のように行われます。

- それぞれのサイトの複写システム管理者は、関与するデータベースに割り当てられた各ファンクション文字列クラスで、**rs_dumpdb** システム・ファンクションと **rs_dumptran** システム・ファンクションのファンクション文字列を作成します。このファンション文字列では、**dump database** コマンドと **dump transaction** コマンドかそれに相当するコマンドを実行して、`rs_lastcommit` システム・テーブルを更新するストアド・プロシージャを呼び出す必要があります。例については、『Replication Server リファレンス・マニュアル』を参照してください。
- ファンクション文字列の作成と修正ができる、派生クラスなどのファンクション文字列クラスを使用している必要があります。

- それぞれのレプリケート・サイトの複写システム管理者は、**alter connection** コマンドを使用して、Replication Server がコーディネート・ダンプを有効にするように設定します。
- ダンプがプライマリ・データベースで開始された場合、RepAgent は **dump database** コマンドまたは **dump transaction** コマンドのログ・レコードを Replication Server に転送します。
- Replication Server は、データベース内に複写テーブルのサブスクリプションを持つサイトに **rs_dumpdb** ファンクション呼び出しまたは **rs_dumptran** ファンクション呼び出しを分配します。
- レプリケート・サイトの **rs_dumpdb** ファンクション文字列と **rs_dumptran** ファンクション文字列は、各レプリケート・サイトで、カスタマイズされたストアド・プロシージャを実行します。

参照：
- ファンクション文字列クラスの管理 (31 ページ)

パーティションのロスまたは障害からのリカバリ

Replication Server は、障害パーティションまたはパーティション・ロスを検出すると、そのパーティションを使用しているステーブル・キューを停止し、障害についてのメッセージのログを取ります。Replication Server を再起動しても問題は解決されません。破損したパーティションを削除し、ステーブル・キューを再構築する必要があります。

完全にリカバリできるかどうかは、キューからクリアしたメッセージの量と、障害が起こったあとにどれだけ早くリカバリ手順を適用したかによって異なります。Replication Server が複写システムでの最小遅延時間内に修復されれば、メッセージのキューが再構築されたときに、最新のメッセージを失うだけで済みます。

プライマリ Replication Server でパーティションに障害が発生した場合、通常はオフライン・データベース・ログを使用して、失われたメッセージをメッセージの送信元から再送信できます。レプリケート Replication Server でパーティションに障害が発生した場合は、アップストリーム Replication Server のステーブル・キューからリカバリする必要があります。

場合によっては、オフライン・ログを使用することが、メッセージをリカバリできる唯一の方法であることがあります。Replication Server がルートまたはコネクションをサスペンドした場合、あるいはネットワークまたはデータ・サーバのコネクションがダウンしている場合、バックログが Replication Server のステーブル・キューに蓄積されていることがあります。バックログを対象としたセーブ・インターバルの設定を指定しないかぎり、これらのメッセージをリカバリできる可能性は時間の経過とともに低くなります。送信元 Replication Server がそれらの

ステーブル・キューからメッセージをすでに削除していたり、データベース・ログがすでにトランケートされていることがあるからです。

注意： リカバリのためにセーブ・インターバルを設定することができます。

参照：

- リカバリのためのセーブ・インターバル (386 ページ)

パーティション・ロスまたは障害の現象と関連するリカバリ手順

パーティション・ロスまたはパーティション障害へのリカバリ手順の使用が必要な状況と参照項目を説明します。

表 31 : パーティション・ロスまたは障害の現象と関連するリカバリ手順

現象	使用するリカバリ手順
Replication Server がステーブル・キューのロス、損傷、障害を検出した。	パーティションのロスまたは障害からのリカバリ。
障害の起きた Replication Server 内にバックログが存在しており、直前のサイトで保存されたメッセージが十分でないため、メッセージ・ロスが発生した。	オフライン・データベース・ログからのメッセージ・リカバリ。
メッセージ・ロスの発生に加えて、データベース・ログがトランケートされている。セカンダリ・トランケーション・ポイントが正しくないか、**dbcc settrunc('ltm', 'ignore')** コマンドが実行されて、RepAgent によって Replication Serer に転送されていないログ・レコードがトランケートされている。	データベース・ログをリカバリするために、データベース・ログからトランケートされたメッセージのリカバリを使用する。その後、ステーブル・キューを再構築し、消失したメッセージをリカバリするには、オフライン・データベース・ログからのメッセージ・リカバリを使用する。

参照：

- パーティションのロスまたは障害からのリカバリ (393 ページ)
- オフライン・データベース・ログからのメッセージのリカバリ (394 ページ)
- トランケートされたプライマリ・データベース・ログからのメッセージのリカバリ (397 ページ)

パーティションのロスまたは障害からのリカバリ

Replication Server がステーブル・キューのロス、損傷、障害を検出した場合、Replication Server のパーティション・ロスまたはパーティション障害からリカバリします。

1. Replication Server にログインし、障害が発生したパーティションを削除します。

   ```
   drop partition logical_name
   ```

   Replication Server は、使用中のパーティションをすぐには削除しません。パーティションが損傷を受けていない場合は、パーティション内のすべてのメッセージが配信され、削除されてから、パーティションが削除されます。『Replication Server リファレンス・マニュアル』の「Replication Server コマンド」の「**drop partition**」を参照してください。
2. 障害のあるパーティションが、Replication Server でアクセスできる唯一のパーティションであった場合は、新しいパーティションを追加し、置き換えます。

   ```
   create partition logical_name
   on 'physical_name' with size size
   [starting at vstart]
   ```

   『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。
3. パーティションが損傷を受けている場合は、ステーブル・キューを再構築します。

   ```
   rebuild queues
   ```

   このコマンドによってパーティション上のすべてのステーブル・キューが削除されると、Replication Server は障害が発生したパーティションをシステムから削除し、残りのパーティションを使用して、オンラインでステーブル・キューを再構築します。
4. キューが再構築されたら、Replication Server ログにロス検出メッセージがあるかどうかを確認します。
5. Replication Server がメッセージ・ロスを検出した場合、次のいずれかを実行します。
   - オフライン・データベース・ログからのメッセージ・リカバリの実行
   - ロスが検出された Replication Server のデータベースに対して **ignore loss** コマンドを実行することによって、Replication Server にロスを無視させる。

次のステップ

Replication Server にメッセージ・ロスを無視するように指定し、ルートの一部である Replication Server のキューを再構築した場合は、送信先でサブスクリプション

を再作成するか、**-r** フラグを指定して **rs_subcmp** プログラムを実行し、プライマリ・データとレプリケート・データを一致させてください。

参照：
- オンラインでのキューの再構築 (418 ページ)
- ステーブル・キュー再構築後のロス検出 (421 ページ)
- オフライン・データベース・ログからのメッセージのリカバリ (394 ページ)

オフライン・データベース・ログからのメッセージのリカバリ

パーティション障害後に、オフライン・ログからのメッセージをリカバリします。

オンライン・ログで、リカバリに必要なデータが不足している場合は、古いバージョンのプライマリ・データベースを別のデータベースにロードして、そのデータベースに対して RepAgent を起動します。RepAgent は別のデータベースにアクセスしていますが、リカバリ対象のメッセージの格納先であるデータベースから送信しているかのように、メッセージを送信します。

1. **-M** フラグを指定して、Replication Server をスタンドアロン・モードで再起動します。
2. ステーブル・キューの再構築 Replication Server にログインして、次のように入力します。

   ```
   rebuild queues
   ```
3. 各サイトで Replication Server のログを検査して「Checking Loss」メッセージを探し、エラー・ログ・メッセージに記載されている日時に基づいてロードするダンプを決定します。
4. テンポラリ・リカバリ・データベースに対して、RepAgent を有効にするには、次のように入力します。

   ```
   sp_config_rep_agent temp_dbname, 'enable', ¥
   'rs_name', 'rs_user_name', 'rs_password'
   ```

   『Replication Server 管理ガイド 第 1 巻』の「RepAgent の管理と Adaptive Server のサポート」の「RepAgent の準備」を参照してください。
5. データベース・ダンプと最初のトランザクション・ログ・ダンプをテンポラリ・リカバリ・データベース内にロードします。
6. テンポラリ・データベースに対して、RepAgent をリカバリ・モードで起動します。

   ```
   sp_start_rep_agent temp_dbname, 'recovery', ¥
   'connect_dataserver', 'connect_database', ¥
   'rs_name', 'rs_user_name', 'rs_password'
   ```

   connect_dataserver と *connect_database* には、元のプライマリ・データ・サーバとデータベースを指定します。

RepAgent は、テンポラリ・リカバリ・データベースのトランザクション・ログ内のデータを元のプライマリ・データベースに転送します。RepAgent は、トランザクション・ログのスキャンが終了すると、停止します。

7. RepAgent がテンポラリ・データベースのトランザクション・ログをリプレイしたかどうかを確認します。次の方法のいずれかを使用します。
   - Adaptive Server のログに次のようなメッセージがあるかどうかを確認します。

     ```
     Recovery of transaction log is complete. Please
     load the next transaction log dump and then start
     up the Rep Agent Thread with sp_start_rep_agent,
     with 'recovery' specified.
     ```

     適切なアクションを実行します。
   - Adaptive Server から、次のコマンドを実行します。

     ```
     sp_help_rep_agent dbname, 'recovery'
     ```

     このプロシージャは、RepAgent のリカバリ・ステータスを表示します。リカバリ・ステータスが「not running」または「end of log」である場合は、リカバリが完了したことを示します。次のトランザクション・ログ・ダンプをロードできます。リカバリ・ステータスが「initial」または「scanning」である場合は、ログがリプレイされていないか、リプレイが完了していないことを示します。
8. 最後のデータベース・ダンプの実行後に、別のリカバリ手順を実行した場合は、トランザクション・ログ・ダンプのロード後にデータベース生成番号を変更しなければならない場合があります。
9. 他にもロードするトランザクション・ログ・ダンプがある場合は、ダンプごとに次の 3 つの手順を繰り返します。
   a) 次のトランザクション・ログ・ダンプをロードします (正しい順序でダンプをロードしてください)。
   b) リカバリ・モードで RepAgent を再起動します。
   c) 完了メッセージがあるかどうか Adaptive Server ログを調べます。または **sp_help_rep_agent** を使用します。
10. Replication Server のログでロス検出メッセージを確認します。

    すべてのメッセージを検索できる状態までさかのぼってデータベースをロードするのに失敗していなければ、ロスは検出されません。
11. Replication Server をノーマル・モードで再起動します。
12. 元のプライマリ・データ・サーバとデータベースに対して RepAgent をノーマル・モードで再起動します。

参照：

- オンラインでのキューの再構築 (418 ページ)

- ロードするダンプの決定 (427 ページ)
- データベース生成番号の決定 (428 ページ)
- ステーブル・キュー再構築後のロス検出 (421 ページ)

オンライン・データベース・ログからのメッセージのリカバリ

プライマリ・データベースのオンライン・ログに残っているメッセージをリカバリします。

1. クライアントのアクティビティをすべて停止します。
2. プライマリ・データベースの RepAgent をリカバリ・モードで再起動します。

   これによって、RepAgent はログを最初からスキャンし、すべてのメッセージを検索します。

トランケートされたプライマリ・データベース・ログからのリカバリ

Replication Server がメッセージを受け取る前に、プライマリ・トランザクション・ログがトランケートされたことによって発生する障害からリカバリします。

RepAgent、(プライマリ・データベースを管理する) Replication Server、またはそれらの間のネットワークが長期間ダウンして、RepAgent や Replication Server がトランザクション・ログからレコードを読み込めない場合に、この障害が発生しやすくなります。セカンダリ・トランケーション・ポイントは移動できないので、Adaptive Server がログをトランケートできず、プライマリ・データベースのトランザクション・ログが満杯になります。この場合、**sp_stop_rep_agent**、次に **dbcc settrunc (ltm, ignore)** を実行して、セカンダリ・トランケーション・ポイントを削除できます。

障害が起こったコンポーネントが正常に戻ったときには、Replication Server のメッセージが失われます。失われたメッセージのステータスによって、次のいずれかの手順を実行します。

- まれに、メッセージがプライマリ・データベースのオンライン・ログにある場合には、オンライン・データベース・ログからのメッセージをリカバリします。
- まれに、メッセージがオンライン・データベース・ログからトランケートされている場合には、データベース・ログからのトランケートされているメッセージをリカバリします。
  ここで説明する手順では、直前のデータベース・ダンプとトランザクション・ログ・ダンプをテンポラリ・リカバリ・データベース内にロードする必要があります。次に、RepAgent をこのデータベースに接続し、トランケートされたログを Replication Server に転送します。失われたログ・レコードがリカバリさ

れたら、通常のプライマリ・データベースを使用してシステムを再起動できます。
テンポラリ・リカバリ・データベースを使用すると、ログがトランケートされたあともプライマリ・データベースを使い続けているクライアントからトランザクションをリカバリできます。

注意： テンポラリ・データベースは、メッセージのリカバリ専用に使用してください。テンポラリ・データベースに変更を加えると、次のトランザクション・ログ・ダンプをロードできなくなります。また、元のプライマリ・データベースのトランザクション・ログがダンプされ、再度トランケートされる前にリカバリが完了するように、元のプライマリ・データベースのアクティビティを制限してください。

参照：
- オンライン・データベース・ログからのメッセージのリカバリ (396 ページ)

トランケートされたプライマリ・データベース・ログからのメッセージのリカバリ

オフライン・トランザクション・ログをリプレイして、プライマリ・データベース・ログからトランケートされているメッセージをリカバリします。

1. **sysusages** テーブルが、元のデータベースとテンポラリ・データベースの両方で同じになるように、テンポラリ・データベースを作成します。

   そのためには、テンポラリ・データベースを作成するときに、元のデータベースを作成したときと同じ **create database** および **alter database** コマンドのシーケンスを使用する必要があります。
2. Replication Server を停止します。
3. **-M** フラグを指定して、Replication Server をスタンドアロン・モードで再起動します。
4. Replication Server にログインし、リカバリ対象の各プライマリ・データベースに対して **set log recovery** コマンドを実行します。

   このコマンドによって、Replication Server はデータベースに対するロス検出モードにします。Replication Server は、次のようなメッセージをログに取ります。

   ```
   Checking Loss for DS1.PDB from DS1.PDB
       date=Nov-01-1995 10:35am
       qid=0x01234567890123456789
   ```
5. **allow connections** コマンドを実行して、Replication Server がリカバリ・モードの他の Replication Server や RepAgent からのコネクションだけを受け入れることができるようにします。

注意： スクリプトを使用し、ノーマル・モードで自動的に RepAgent を再起動してこの Replication Server に接続しようとすると、Replication Server はコネク

ションを拒否します。正しいオフライン・ログを参照している間にリカバリ・モードで RepAgent を再起動してください。この手順によって、古いトランザクション・ログを再送してから、現在のトランザクションを処理できます。

6. データベース・ダンプをテンポラリ・プライマリ・データベース内にロードします。
7. 最初または次のトランザクション・ログ・ダンプをテンポラリ・プライマリ・データベース内にロードします。
8. テンポラリ・データベースに対して、RepAgent をリカバリ・モードで起動します。

```
sp_start_rep_agent temp_dbname, 'recovery',
'connect_dataserver', 'connect_database',
'repserver_name', 'repserver_username',
'repserver_password'
```

connect_dataserver と *connect_database* には、元のプライマリ・データ・サーバとデータベースを指定します。

RepAgent は、テンポラリ・リカバリ・データベースのトランザクション・ログ内のデータを元のプライマリ・データベースに転送します。RepAgent は、現在のトランザクション・ログのスキャンが終了すると、停止します。

9. 次のどちらかを実行して、RepAgent がテンポラリ・データベースのトランザクション・ログをリプレイしたかどうかを確認します。

   - Adaptive Server のログで次のメッセージを確認します。

     ```
     Recovery of transaction log is complete. Please
     load the next transaction log dump and then start
     up the Rep Agent Thread with sp_start_rep_agent,
     with 'recovery' specified.
     ```

     その後、適切な作業を行います。
   - **admin who_is_down** を実行します。

     RepAgent が「down」をレポートする場合は、次のトランザクション・ログをロードします。

10. すべてのトランザクション・ログが処理されるまで、手順 7 ～ 9 までを繰り返します。

    これで、プライマリ・データベースから通常の複写をレジュームする準備が整いました。

11. スタンドアロン・モードになっている Replication Server を停止します。
12. **rs_zeroltm** を実行して、ロケータ情報をクリアする必要がある場合があります。

```
rs_zeroltm data_server, database
dbcc settrunc('ltm', 'valid')
```

13. Replication Server をノーマル・モードで再起動します。

14. **sp_start_rep_agent** を使用して、プライマリ・データ・データベースと RSSD に対して RepAgent を再起動します。
15. 最後のデータベース・ダンプの実行後に、別のリカバリ手順を実行した場合は、トランザクション・ログ・ダンプのロード後にデータベース生成番号を変更しなければならない場合があります。

参照：

- データベースに対するログ・リカバリの設定 (425 ページ)
- データベース生成番号の決定 (428 ページ)

プライマリ・データベース障害からのリカバリ

プライマリ・データベースに障害が発生し、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ない場合は、データベースを直前の状態にロードして、レプリケート・サイトでの一貫性を回復するためのリカバリ手順に従ってください。

ほとんどの場合、データベース障害はコミットされたトランザクションを失うことなくリカバリされます。再起動時にデータベースがリカバリすれば、特別な Replication Server のリカバリ手順を踏む必要はありません。この場合、Replication Server はデータベースとのハンドシェークを実行し、トランザクションが失われたり重複したりすることなく、複写システムをリカバリします。

プライマリ・データベースの障害からは、次の 2 つの方法で回復できます。

- プライマリ・ダンプのみを使用したリカバリです。

  コーディネート・ダンプがない場合は、障害が発生したプライマリ・データベースをロードしてから、レプリケート・データベースとリストア後のプライマリ・データベースとの一貫性を確認できます。
- コーディネート・ダンプを使用したリカバリです。

  プライマリ・データベースとレプリケート・データベースのコーディネート・ダンプがある場合は、これらを使用して複写システム内のすべてのデータベースをロードし、一貫性のある状態にすることができます。

ダンプからのプライマリ・データベースのロード

複写システムのプライマリ・データベースのみをロードする場合、データベースをロードして直前の状態に戻し、レプリケート・データベースとの矛盾を解決します。

1. プライマリ Replication Server にログインしてプライマリ・データベースのデータベース生成番号を取得するには、次のように入力します。

```
admin get_generation, data_server, database
```

後の手順で必要になる、この番号を記録しておきます。

2. プライマリ・データベースの RepAgent を停止するには、次のように入力します。

```
sp_stop_rep_agent database
```

3. プライマリ・データベースへの DSI コネクションをサスペンドして、排他的に使用できるようにします。
4. 最新または直前の状態にデータベースをロードします。

   この手順では、最新のデータベース・ダンプと、それ以降のすべてのトランザクション・ログ・ダンプをロードする必要があります。

   手順については、『Adaptive Server Enterprise システム管理ガイド』を参照してください。
5. DSI コネクションのレジューム
6. トランザクション・ログをダンプします。

   次のように入力します。

```
use database
go
dbcc settrunc('ltm', 'ignore')
go
dump tran database with truncate_only
go
dbcc settrunc('ltm', 'valid')
go
```

7. リストアされたプライマリ・データベース内で **dbcc settrunc** コマンドを実行して、生成番号を 1 つ大きい番号に設定します。

   たとえば、手順 1 の **admin get_generation** コマンドから返された番号が 0 である場合は、次のコマンドを入力します。

```
use database
go
dbcc settrunc('ltm', 'gen_id', 1)
```

8. ロケータ情報をクリアします。

   次のように入力します。

```
rs_zeroltm data_server, database
```

9. プライマリ・データベースに対して RepAgent を起動します。そのためには、次のコマンドを実行します。

```
sp_start_rep_agent database
```

10. レプリケート・サイトで、サブスクリプションごとに **rs_subcmp** プログラムを実行します。**-r** フラグを使用して、レプリケート・データとリストアされたプライマリ・データを一致させるか、またはすべてのサブスクリプションを削除してから再作成します。

また、『Replication Server 管理ガイド 第 1 巻』の「サブスクリプションの管理」の「サブスクリプション情報の取得」の「サブスクリプションの一貫性の確認」の「**rs_subcmp** を使用した矛盾の検索と訂正」、および 『Replication Server リファレンス・マニュアル』の「実行プログラム」の「**rs_subcmp**」を参照してください。

コーディネート・ダンプからのロード

プライマリ・データベースとレプリケート・データベースのコーディネート・ダンプがある場合は、この手順を使用します。手順は、プライマリ・データベースおよびすべてのレプリケート・データベースをロードして同一の状態にします。

1. ダンプからプライマリ・データベースをロードするプロシージャの手順 1 ～ 10 を実行します。
2. リストアが必要なレプリケート・データベースへのコネクションをサスペンドします。
3. 各レプリケート・データベースの場合、管理している Replication Server にログインし、データベースへのコネクションをサスペンドします。

   次のように入力します。

   ```
   suspend connection to data_server.database
   ```
4. リストアされたプライマリ・データベースの状態と一致するコーディネート・ダンプからレプリケート・データベースをロードします。
5. 各レプリケート・データベースの場合、管理している Replication Server にログインし、**sysadmin set_dsi_generation** コマンドを実行し、レプリケート・データベースの生成番号を手順 1 で使用したデータベースの生成番号と同じに設定します。

   次のように入力します。

   ```
   sysadmin set_dsi_generation, 101,
       primary_data_server, primary_database,
       replicate_data_server, replicate_database
   ```

   primary_data_server と *primary_database* の各パラメータには、ロードに使用するプライマリ・データベースを指定します。*replicate_data_server* と *replicate_database* の各パラメータには、ロードに使用するレプリケート・データベースを指定します。

   この方法で生成番号を設定すると、キュー内に存在する可能性のある古いメッセージがレプリケート・データベースに適用されることがありません。
6. 各レプリケート・データベースの場合、管理している Replication Server にログインし、データベースの DSI を再起動するために、データベースへのコネクションをレジュームします。

   次のように入力します。

```
resume connection to data_server.database
```

7. プライマリ Replication Server をノーマル・モードで再起動します。
8. プライマリ・データベースの RepAgent をノーマル・モードで再起動します。

次のステップ

障害発生時にマテリアライズ中のサブスクリプションがあった場合は、それらのサブスクリプションを削除してから再作成してください。

参照：

- ダンプからのプライマリ・データベースのロード (399 ページ)

RSSD 障害からのリカバリ

RSSD を最新の状態にリカバリできない場合は、RSSD 障害のリカバリ手順は複雑なものになります。この場合、古いデータベース・ダンプとトランザクション・ログ・ダンプから RSSD をロードしなければなりません。

注意： プラットフォーム間の **dump** と **load** または **bcp** などのコマンドを使用した RSSD データベースのマイグレーションはできません。移行するには、新しいプラットフォームでの複写システムの再構築が必要です。

RSSD をリカバリする手順は、プライマリ・データベースをリカバリする手順と似ています。ただし、RSSD には複写システム自体に関する情報が含まれるため、必要な手順は多くなります。RSSD システム・テーブルは、複写システム内のステーブル・キューおよび他の RSSD の状態と密接に関連しています。

Replication Server の RSSD に障害が発生した場合には、必要なリカバリの範囲を最初に判断する必要があります。そのために、次のアクションのうちの 1 つ以上を実行します。

- RSSD が使用可能になったら、Replication Server にログインし、**admin who_is_down** コマンドを実行します。RSSD が動作していなかった間に、Replication Server の一部のスレッドが停止している可能性があります。
  - インバウンド・キューまたはアウトバウンド・キューの SQM スレッドか、RSI アウトバウンド・キューがダウンしている場合は、Replication Server を再起動します。
  - DSI スレッドがダウンしている場合は、関連するデータベース宛のコネクションをレジュームします。
  - RSI スレッドがダウンしている場合は、送信先データベースへのルートをレジュームします。

- **sp_help_rep_agent** システム・プロシージャを使用して、接続している RepAgent がすべて実行されているかチェックしてください (RSSD の停止に起因するエラーが発生したために RepAgent が停止している場合があります)。必要に応じて、RepAgent を再起動します。
- RSSD を最新の状態にリカバリできない場合は、古いデータベース・ダンプとトランザクション・ログ・ダンプから RSSD をロードする必要があります。

参照：
- ダンプから RSSD をリカバリするプロシージャ (403 ページ)

ダンプから RSSD をリカバリするプロシージャ

RSSD のリカバリ手順は、最後の RSSD ダンプを行った後、RSSD のアクティビティがどの程度あったかによって異なります。RSSD 障害の重大度には 4 つのレベルがあり、レベルごとにリカバリの必要条件が異なります。

表 32 : RSSD 障害からリカバリするプロシージャ

RSSD の最後のダンプ後のアクティビティ	使用するリカバリ手順
DDL コマンドは実行していない。	RSSD の基本的なリカバリ手順
DDL コマンドは実行したが、新しいルートやサブスクリプションは作成されていない。	サブスクリプション比較の手順
DDL コマンドは実行したが、新しいルートは作成されていない。	サブスクリプション比較の手順
新しいルートが作成された。	統合解除／再統合の手順

参照：
- RSSD の基本的なリカバリ手順の使用 (403 ページ)
- サブスクリプション比較の手順の使用 (406 ページ)
- サブスクリプション再作成の手順の使用 (413 ページ)
- 統合解除と再統合の手順の使用 (417 ページ)

RSSD の基本的なリカバリ手順の使用

最後の RSSD ダンプ以降に DDL コマンドを実行していない場合は、RSSD をリストアします。RCL の DDL コマンドには、ルート、複写定義、サブスクリプション、ファンクション文字列、ファンクション、ファンクション文字列クラス、エラー・クラスの作成、変更、削除のためのコマンドがあります。

ここで説明する手順の一部は、他の RSSD リカバリ手順でも使用します。

警告！ このリカバリ手順を完了するまでは、DDL コマンドを実行しないでください。

1. 該当の Replication Server に接続している RepAgent をすべて停止します。
2. RSSD に障害が発生したので、該当の Replication Server は停止しているはずです。何らかの理由で停止していない場合は、Replication Server にログインし、**shutdown** コマンドを実行して停止させます。

   注意： Replication Server のステーブル・キューにメッセージが残っている場合があります。これらのキューのデータは、あとの手順でキューを再構築するときに削除されます。
3. 最新の RSSD データベース・ダンプとすべてのトランザクション・ダンプをロードして、RSSD をリストアします。
4. **-M** フラグを指定して、Replication Server をスタンドアロン・モードで再起動します。

   この時点では、ステーブル・キューが RSSD の状態と一致していないため、Replication Server はスタンドアロン・モードで起動してください。Replication Server がスタンドアロン・モードで起動された場合は、ステーブル・キューの読み取りは自動的には開始されません。
5. Replication Server にログインして、RSSD の生成番号を取得します。

   次のように入力します。

   ```
   admin get_generation, data_server, rssd_name
   ```

   この例では、生成番号として 100 が返されるとします。
6. Replication Server では、キューを再構築します。

   次のように入力します。

   ```
   rebuild queues
   ```
7. 該当の Replication Server に接続しているすべての RepAgent (RSSD の RepAgent を除く) をリカバリ・モードで起動します。

   次のように入力します。

   ```
   sp_start_rep_agent dbname, recovery
   ```

   各 RepAgent が、現在のログを読み終わったというメッセージを Adaptive Server のログに出力するまで待ちます。
8. Replication Server ログにロス・メッセージがないかどうかチェックします。また、現在の Replication Server を起点とする直接ルートを持つすべての Replication Server 内のログにもロス・メッセージがないかどうかをチェックします。

- すべてのルートが障害時にアクティブであった場合は、実際のデータ・ロスは発生していないはずです。
- ただし、ロスの検出で、実際のロスが見つかる場合があります。プライマリ・データベースでデータベース・ログがトランケートされている場合には、再構築のプロセスでリカバリする情報が十分になかったため、実際のデータ・ロスが検出されることがあります。実際のデータ・ロスが検出されたら、トランケートされたプライマリ・データベース・ログからリカバリするための手順を使用して、古いダンプからデータベース・ログを再ロードします。

9. 該当の Replication Server が管理するすべてのプライマリ・データベースの

次のように入力します。

```
sp_stop_rep_agent dbname
```

10. Replication Server を停止します。

11. セカンダリ・トランケーション・ポイントを移動します。

リストアされた RSSD の Adaptive Server で **dbcc settrunc** コマンドを実行します。

```
use rssd_name
go
dbcc settrunc('ltm', 'ignore')
go
dump tran rssd_name with truncate_only
go
begin tran commit tran
go 40
```

注意： **begin tran commit tran go 40** コマンドは、Adaptive Server のログを次のページに移動します。

12. ロケータ情報をクリアします。

次のように入力します。

```
rs_zeroltm rssd_server, rssd_name
go
```

13. リストアされた RSSD 用の Adaptive Server で **dbcc settrunc** コマンドを実行して、手順 5 で **admin get_generation** コマンドによって返された番号よりも 1 つ大きな生成番号を設定します。

次のように入力します。

```
dbcc settrunc ('ltm', 'gen_id', generation_number)
go
dbcc settrunc('ltm', 'valid')
go
```

必要があればこの RSSD のリカバリ手順に戻ることができるように、この生成番号と現在の時刻を記録してください。または、生成番号を設定してからデータベースをダンプしてもよいでしょう。

14. Replication Server をノーマル・モードで再起動します。

サブスクリプション比較またはサブスクリプション再作成の手順の一部としてこの手順を実行している場合は、アップストリーム RSI アウトバウンド・キューに、**rs_subcmp** ですでに適用されている該当の Replication Server の RSSD へのトランザクションが含まれている可能性があります。この場合は、Replication Server の起動後のエラー・ログに、重複した挿入に関する警告が含まれる場合があります。これらの警告は無視しても問題はありません。

15. RSSD および ユーザ・データベースの RepAgent をノーマル・モードで再起動します。

RSSD のリカバリ手順 (サブスクリプション比較またはサブスクリプション再作成) の一部としてこの手順を実行した場合は、該当の Replication Server を起点とするルートを持つすべての Replication Server で、RSSD ロスが検出されたというメッセージが表示されます。

参照：

- オンラインでのキューの再構築 (418 ページ)
- トランケートされたプライマリ・データベース・ログからのリカバリ (396 ページ)
- ステーブル・キュー再構築後のロス検出 (421 ページ)

サブスクリプション比較の手順の使用

最後のトランザクション・ダンプの作成以降に何らかの DDL コマンドを実行していても、新しいサブスクリプションやルートは作成していない場合は、ここで説明する RSSD のリカバリ手順を実行してください。

RCL の DDL コマンドには、ルート、複写定義、サブスクリプション、ファンクション文字列、ファンクション、ファンクション文字列クラス、エラー・クラスの作成、変更、削除のためのコマンドがあります。

警告！ このリカバリ手順を完了するまでは、DDL コマンドを実行しないでください。

この手順を実行すると、障害が発生した RSSD とアップストリーム RSSD との一貫性が回復します。アップストリーム Replication Server がない場合は、最新のデータベースおよびトランザクション・ダンプとの一貫性が回復します。さらに、ダウンストリーム RSSD とリカバリ後の RSSD との一貫性も回復します。

最後のトランザクション・ダンプ以降に該当の Replication Server で DDL コマンドが実行された場合は、そのコマンドを再実行しなければならない場合があります。

警告！ 複写システム内の Replication Server がすべて同じバージョン・レベルではない混合バージョン環境で操作している場合には、この手順が失敗することがあります。

1. 障害が発生した RSSD をリカバリするための準備として、RSSD の基本的なリカバリの手順 1 ～ 4 を実行します。
2. すべてのアップストリーム RSSD をリカバリする準備として、**admin quiesce_force_rsi** コマンドを各アップストリーム Replication Server で実行します。
   - これによって、該当の Replication Server からのコミット済みトランザクションがすべて確実に適用されてから、**rs_subcmp** プログラムを実行できます。
   - 該当の Replication Server と最も離れたアップストリーム Replication Server から順番にこのコマンドを実行します。
   - RSSD の変更が適用されていることを確認してください。適用されていれば、RSSD DSI アウトバウンド・キューは空になります。
   - 該当の Replication Server の直接のアップストリーム Replication Server である Replication Server は、クワイス状態にできません。
3. すべてのダウンストリーム RSSD をリカバリする準備として、**admin quiesce_force_rsi** コマンドを各ダウンストリーム Replication Server で実行します。
   - これによって、該当の Replication Server 宛のコミット済みトランザクションがすべて確実に適用されてから、**rs_subcmp** プログラムを実行できます。
   - 該当の Replication Server からの直接のダウンストリーム Replication Server から順番にこのコマンドを実行します。
   - RSSD の変更が適用されていることを確認してください。適用されていれば、RSSD DSI アウトバウンド・キューは空になります。
4. **rs_subcmp** プログラムを使用して、障害が発生した RSSD とすべてのアップストリーム RSSD とを一致させます。
   - 最初に、一致を指定しないで **rs_subcmp** を実行し、どのようなオペレーションが実行されるかを確認します。一致の準備ができたら、**-r** フラグを使用してレプリケート・データとプライマリ・データとを一致させます。
   - **rs_subcmp** は、メンテナンス・ユーザでないと実行できません。メンテナンス・ユーザに関する詳細については、『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。
   - 各インスタンスで、障害が発生した RSSD をレプリケート・データベースとして指定します。
   - 各インスタンスで、各アップストリーム Replication Server の RSSD をプライマリ・データベースとして指定します。

- 該当の Replication Server と最も離れたアップストリーム Replication Server から始めて、該当の Replication Server へのルート (直接ルートまたは間接ルート) を持つ他のすべての Replication Server に対してダウンストリームにプログラムを実行します。
- 次の各 RSSD システム・テーブルを一致させます。`rs_articles`、`rs_classes`、`rs_columns`、`rs_databases`、`rs_erroractions`、`rs_functions`、`rs_funcstrings`、`rs_objects`、`rs_publications`、`rs_systext`、`rs_whereclauses`
- 複写 RSSD テーブル上で **rs_subcmp** を実行する場合、**select** 文の **where** 句と **order by** 句には複写されるすべてのローを指定してください。

  これで、障害が発生した RSSD がリカバリされます。

5. **rs_subcmp** プログラムを使用して、すべてのダウンストリーム RSSD と前の手順でリカバリされた現在の Replication Server の RSSD とを一致させます。
   - 最初に、一致を指定しないで **rs_subcmp** を実行し、どのようなオペレーションが実行されるかを確認します。一致の準備ができたら、**-r** フラグを使用してレプリケート・データとプライマリ・データとを一致させます。
   - **rs_subcmp** は、メンテナンス・ユーザでないと実行できません。メンテナンス・ユーザに関する詳細については、『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。
   - 各インスタンスで、リカバリされた RSSD をプライマリ・データベースとして指定します。
   - 各インスタンスで、各ダウンストリーム Replication Server の RSSD をレプリケート・データベースとして指定します。
   - 直接のダウンストリーム Replication Server から始めて、該当の Replication Server を起点とするルート (直接ルートまたは間接ルート) を持つ他のすべての Replication Server に対してダウンストリームにプログラムを実行します。
   - 次の各 RSSD システム・テーブルを一致させます。`rs_articles`、`rs_classes`、`rs_columns`、`rs_databases`、`rs_erroractions`、`rs_functions`、`rs_funcstrings`、`rs_objects`、`rs_publications`、`rs_systext`、`rs_whereclauses`
   - 複写 RSSD テーブル上で **rs_subcmp** を実行する場合、**select** 文の **where** 句と **order by** 句では、複写されるすべてのローが選択されるようにしてください。

     これで、すべてのダウンストリーム RSSD が完全にリカバリされます。
6. リカバリ対象の Replication Server が ID サーバである場合は、Replication Server とその RSSD 内のデータベース ID をリストアします。

a) Replication Server ごとに、rs_databases および rs_sites システム・テーブルで ID の存在を確認します。
b) 消失したローがある場合は、リカバリ対象の RSSD の rs_idnames システム・テーブルに適切なローを挿入します。
c) 複写システムに削除したはずのデータベースまたは Replication Server がある場合は、その ID をリカバリ対象の RSSD の rs_idnames システム・テーブルから削除します。
d) rs_ids システム・テーブルの一貫性が保たれていることを確認します。

   現在の Replication Server の RSSD で実行するには、このストアド・プロシージャを次のように入力します。

```
rs_mk_rsids_consistent
```

7. リカバリ対象の Replication Server が ID サーバではなく、かつデータベース・コネクションが最後のトランザクション・ダンプ以降にリカバリ対象の Replication Server で作成された場合は、ID サーバの RSSD の rs_idnames システム・テーブルから、そのデータベース・コネクションに対応するローを削除します。
8. 基本的な RSSD のリカバリ手順 5 ～ 14 に従って実行します。
9. RSSD のリカバリを完了するために、最後のトランザクション・ダンプ以降に該当の Replication Server で実行された DDL コマンドをすべて再実行します。

参照：

- 複写 RSSD システム・テーブルで rs_subcmp を使用する select 文 (409 ページ)
- RSSD の基本的なリカバリ手順の使用 (403 ページ)

複写 RSSD システム・テーブルで rs_subcmp を使用する select 文

RSSD のリカバリ手順の実行中に複写 RSSD テーブルで **rs_subcmp** を実行する場合は、**select** 文の **where** 句と **order by** 句で、各システム・テーブルに複写される必要のあるローをすべて選択してください。

リストされた **select** 文の *sub_select* では、次の副次選択文を表しており、現在の Replication Server の送信元 Replication Server であるすべてのサイト ID を選択します。

```
(select source_rsid from rs_routes
    where
      (through_rsid = PRS_site_ID
      or through_rsid = RRS_site_ID)
    and
    dest_rsid = RRS_site_ID)
```

PRS_site_ID は、プライマリ RSSD を管理する Replication Server のサイト ID です。*RRS_site_ID* は、**rs_subcmp** オペレーションの対象となるレプリケート RSSD を管理する Replication Server のサイト ID です。

rs_columns、rs_databases、rs_funcstrings、rs_functions、rs_objects システム・テーブルでは、rowtype = 1 のローは複写ローです。**rs_subcmp** を使用して比較する必要があるローは、複写ローだけです。

primary_keys は、各システム・テーブルのユニーク・インデックスです。テーブルの詳細については、『Replication Server リファレンス・マニュアル』の「Replication Server システム・テーブル」を参照してください。

注意： これらは、**select** 文の一般的な形式を示します。混合バージョン環境では、これらの **select** 文の調整が必要なことがあります。

表 33 : rs_subcmp プロシージャの select 文の一般的形式

RSSD テーブル名	select 文
rs_articles	`select * from rs_articles,rs_objects where rs_objects.prsid in sub_select and rs_articles.objid = rsobjects.objid order by articleid`
rs_classes	`select * from rs_classes where prsid in sub_select order by classid`
rs_columns	`select * from rs_columns where prsid in sub_select and rowtype = 1 order by objid, basecolnum, colname, colnum, version`
rs_databases	`select * from rs_databases where prsid in sub_select and rowtype = 1 order by dbid, dbname, dsname, ldbid, ltype, ptype`
rs_erroractions	`select * from rs_erroractions where prsid in sub_select order by ds_errorid, errorclassid`
rs_funcstrings	`select * from rs_funcstrings where prsid in sub_select and rowtype = 1 order by fstringid`
rs_functions	`select * from rs_functions where prsid in sub_select and rowtype = 1 order by funcid, funcname, objid`

RSSD テーブル名	select 文
rs_objects	select * from rs_objects where prsid in *sub_select* and rowtype = 1 order by active_inbound, dbid, has_baserepdef, objid, objname, objtype, phys_tablename, phys_objowner, version
rs_publications	select * from rs_publications where prsid in *sub_select* order by pubid
rs_systext	select * from rs_systext where prsid in *sub_select* and texttype in ('O', 'S') order by parentid, texttype, sequence
rs_whereclauses	select * from rs_whereclauses,rs_articles,rs_objects where rs_objects.prsid in *sub_select* and rs_articles.objid = rsobjects.objid and rs_whereclauses.articleid = rs_artilces.articleid order by wclauseid

クラスとシステム・テーブル

クラス・テーブルとシステム・テーブルでのサブスクリプション比較の手順の影響と、RSSD を一貫性のある状態にする方法について説明します。

システム提供ファンクション文字列クラスとエラー・クラスは、初めは指定されたプライマリ・サイトを持っていません。つまり、これらのサイト ID は 0 です。rs_default_function_class クラスと rs_db2_function_class クラスは修正できないので、指定されたプライマリ・サイトを持つことはできません。rs_sqlserver_function_class クラスと rs_sqlserver_error_class クラスには、プライマリ・サイトを割り当て、修正することができます。派生ファンション文字列クラスのプライマリ・サイトは、その親クラスと同じです。

リカバリ対象の Replication Server が最後のトランザクション・ダンプ以降にファンクション文字列クラスまたはエラー・クラスのプライマリ・サイトになった場合は、**rs_subcmp** プロシージャはダウンストリーム RSSD 内の孤立したローを検出します。

この場合は、rs_classes、rs_erroractions、rs_funcstrings、rs_systext の各システム・テーブルに対して、**rs_subcmp** をもう一度実行しま

す。**prsid = 0** を設定して、これらのテーブルに必要なデフォルト値を設定しなおしてください。

次は、rs_classes テーブルへの **select** 文の適用例です。

```
select * from rs_classes where prsid = 0
    order by primary_keys
```

例
RSSD を一貫性のある状態にします。

複写システムに次の Replication Server サイトが存在するものとします。矢印 (→) はルートを意味します。サイト B は障害が発生したサイトであり、間接ルートは存在しません。

- A > B
- C > B
- C > D
- B > E

各 Replication Server のサイト ID は次のとおりです。

- A = 1
- B = 2
- C = 3
- D = 4
- E = 5

この例で、各 RSSD を一貫性のある状態にするには、rs_classes、rs_columns、rs_databases、rs_erroractions、rs_funcstrings、rs_functions、rs_objects、rs_systext システム・テーブルに対して、次の作業を説明の順番どおりに実行します。

アップストリーム RSSD との一致
アップストリーム RSSD とシステム・テーブルとを一致させます。

1. サイト B をレプリケート・サイト、サイト A をプライマリ・サイトとし、**where** 句に prsid = 1 と指定して、**rs_subcmp** を前述のシステム・テーブルに対して実行します。

   次の例は、rs_columns に対する **select** 文を示します。

   ```
   select * from rs_columns where prsid in
       (select source_rsid from rs_routes
       where
           (through_rsid = 1 or through_rsid = 2)
            and dest_rsid = 2)
   ```

```
        and rowtype = 1
       order by objid, colname
```

2. サイト B をレプリケート・サイト、サイト C をプライマリ・サイトとし、**where** 句に `prsid = 3` と指定して、**rs_subcmp** を前述のシステム・テーブルに対して実行します。

   次の例は、`rs_columns` に対する **select** 文を示します。

```
select * from rs_columns where prsid in
      (select source_rsid from rs_routes
       where
        (through_rsid = 3 or through_rsid = 2)
         and dest_rsid = 2)
       and rowtype = 1
      order by objid, colname
```

ダウンストリーム RSSD との一致

ダウンストリーム RSSD とシステム・テーブルとを一致させます。

サイト B をプライマリ・サイト、サイト E をレプリケート・サイトとし、**where** 句に `prsid = 2` と指定して、**rs_subcmp** を前述のシステム・テーブルに対して実行します。

次の例は、`rs_columns` に対する **select** 文を示します。

```
select * from rs_columns where prsid in
      (select source_rsid from rs_routes
       where
        (through_rsid = 2 or through_rsid = 5)
         and dest_rsid = 5)
       and rowtype = 1
      order by objid, colname
```

『Replication Server リファレンス・マニュアル』の「実行プログラム」の「**rs_subcmp**」および『Replication Server リファレンス・マニュアル』の「Replication Server システム・テーブル」を参照してください。

サブスクリプション再作成の手順の使用

最後のトランザクション・ダンプ以降に新しいサブスクリプションまたは他の DDL を作成しており、新しいルートは作成していない場合、失われたサブスクリプションを再作成する必要がある RSSD をリストアします。

RCL の DDL コマンドには、ルート、複写定義、サブスクリプション、ファンクション文字列、ファンクション、ファンクション文字列クラス、エラー・クラスの作成、変更、削除のためのコマンドがあります。

警告！ このサブスクリプション再作成リカバリ手順を完了するまでは、DDL コマンドを実行しないでください。

このタスクは、障害が発生した RSSD とアップストリーム RSSD との一貫性を回復させます。アップストリーム Replication Server がない場合は、最新のデータベースおよびトランザクション・ダンプとの一貫性が回復します。さらに、ダウンストリーム RSSD とリカバリ後の RSSD との一貫性も回復します。また、RSSD のロスが原因で一貫性のなくなったサブスクリプションも削除または再作成します。

ただし、この手順では、次を実行します。

最後のトランザクション・ダンプ以降に該当の Replication Server で DDL コマンドが実行された場合は、そのコマンドを再実行しなければならない場合があります。

『Replication Server リファレンス・マニュアル』の「実行プログラム」の「**rs_subcmp**」および 『Replication Server リファレンス・マニュアル』の「Replication Server システム・テーブル」を参照してください。

1. 障害が発生した RSSD をリカバリするための準備として、RSSD の基本的なリカバリの手順 1 ～ 4 を実行します。
2. すべてのアップストリーム Replication Server とダウンストリーム Replication Server の RSSD をリカバリする準備として、サブスクリプション比較の手順 2 ～ 3 を実行します。
3. 前の手順の影響を受けるすべてのアップストリーム Replication Server とダウンストリーム Replication Server を停止します。**shutdown** コマンドを使用します。
4. **-M** フラグを指定して、すべてのアップストリーム Replication Server とダウンストリーム Replication Server をスタンドアロン・モードで再起動します。

   これらの Replication Server をスタンドアロン・モードで再起動すると、Replication Server に接続しているすべての RepAgent が自動的に停止します。
5. 障害が発生した RSSD をすべてのアップストリーム RSSD と一致させるには、サブスクリプション比較の手順 4 を実行します。

   これで、障害が発生した RSSD がリカバリされます。
6. すべてのダウンストリーム RSSD を該当の Replication Server の RSSD と一致させるには、サブスクリプション比較の手順 5 を実行します。
7. リカバリ対象の Replication Server が ID サーバである場合、その Replication Server の RSSD 内の ID をリストアするには、サブスクリプション比較の手順 6 を実行します。
8. リカバリ対象の Replication Server が ID サーバではなく、かつ最後のトランザクション・ダンプ以降にリカバリ対象の Replication Server でデータベース・コネクションが作成されている場合は、サブスクリプション比較の手順 7 を実行します。

9. 該当の Replication Server の rs_subscriptions システム・テーブルに、サブスクリプション名、複写定義名またはパブリケーション名、関連するデータベース名について問い合わせます。
   - 該当の Replication Server が管理するプライマリ・データに対するサブスクリプションを持つすべての Replication Server、または該当の Replication Server がサブスクリプションを持つプライマリ・データのあるすべての Replication Server にも問い合わせます。
   - rs_subscriptions システム・テーブルを問い合わせるには、**rs_helpsub** ストアド・プロシージャを使用します。
10. rs_subscriptions システム・テーブルの各ユーザ・サブスクリプションに対して、手順 9 で取得した情報を使用して **check subscription** コマンドを実行します。
    - 該当の Replication Server と、該当の Replication Server が管理するプライマリ・データに対するサブスクリプションを持つすべての Replication Server、または該当の Replication Server がサブスクリプションを持つプライマリ・データのあるすべての Replication Server で、このコマンドを実行します。
    - ステータスが VALID 以外であるサブスクリプションは、次の方法で削除または再作成します。
11. プライマリである該当の Replication Server のサブスクリプションが VALID になっていない各 Replication Server で次のことを実行します。
    - subid を書き留めてから、プライマリ rs_subscriptions システム・テーブルから該当のローを削除します。
    - rs_subscriptions システム・テーブルに格納されている subid を使用して、rs_rules システム・テーブル内の対応するローを見つけ、これらのローも削除します。

    rs_subscriptions と rs_rules の各システム・テーブルに対して次の作業を実行します。
    - サブスクリプションが、プライマリ・テーブルに存在し、(削除されたことが原因で) レプリケート・テーブルに存在しない場合は、プライマリ・テーブルからそのサブスクリプション・ローを削除します。
    - サブスクリプションがレプリケート・テーブルに存在し、プライマリ・テーブルに存在しない場合は、レプリケート・テーブルからそのサブスクプリション・ローを削除します。この手順の残りの作業を完了してから、手順 17 ～ 19 に従って、サブスクリプションを再作成します。
    - プライマリ・テーブルにもレプリケート・テーブルにもサブスクリプションが存在するものの、いずれかのサイトで VALID でない場合は、両方のテーブルから該当のローを削除します。この手順の残りの作業を完了してから、手順 17 ～ 19 に従って、サブスクリプションを再作成します。

12. 該当の Replication Server が VALID でないユーザ・サブスクリプションを持つ各プライマリ Replication Server で、次のことを実行します。
    - `subid` を書き留めてから、プライマリ `rs_subscriptions` システム・テーブルから該当のローを削除します。
    - `rs_subscriptions` システム・テーブルに格納されている `subid` を使用して、`rs_rules` システム・テーブル内の対応するローを見つけ、これらのローも削除します。

    `rs_subscriptions` と `rs_rules` の各システム・テーブルに対して次の作業を実行します。
    - サブスクリプションがプライマリ・テーブルに存在し、レプリケート・テーブルに存在しない場合は、プライマリ・テーブルからそのサブスクプリション・ローを削除します。この手順の残りの作業を完了してから、手順 17 ～ 19 に従って、サブスクリプションを再作成します。
    - サブスクリプションが、レプリケート・テーブルに存在し、(削除されたことが原因で) プライマリ・テーブルに存在しない場合は、レプリケート・テーブルからそのサブスクリプション・ローを削除します。
    - プライマリ・テーブルにもレプリケート・テーブルにもサブスクリプションが存在するものの、いずれかのサイトで VALID でない場合は、両方のテーブルから該当のローを削除します。この手順の残りの作業を完了してから、手順 17 ～ 19 に従って、サブスクリプションを再作成します。
13. プライマリ Replication Server とレプリケート Replication Server の両方で、手順 17 ～ 19 に従って削除されるサブスクリプションに対する既存のすべてのマテリアライゼーション・キューについて、**sysadmin drop_queue** コマンドを実行します。
14. 該当の Replication Server によって管理されるプライマリ・データへのサブスクリプションを持っているか、該当の Replication Server がサブスクリプションを持つプライマリ・データのあるすべての Replication Server と RepAgent をノーマル・モードで再起動します。
15. 基本的な RSSD のリカバリ手順 5 ～ 13 を実行します。
16. 最後のトランザクション・ダンプ以降、該当の Replication Server で実行された DDL コマンドをすべて再実行します。
17. 各複写定義に対して、オートコレクション機能を使用可能にします。
18. バルク・マテリアライゼーション・メソッドまたは非マテリアライゼーション・メソッドを使用して、失われたサブスクリプションを再作成します。

    define subscription、**activate subscription**、**validate subscription**、**check subscription** の各コマンドを使用して、バルク・マテリアライゼーションを実行します。

19. 再作成されたサブスクリプションごとに、次の 2 つの方法のいずれかを使用して、プライマリ・データとレプリケート・データの一貫性を回復します。
    - **with purge** オプションを指定して **drop subscription** コマンドを実行し、サブスクリプションを削除します。次にサブスクリプションを再作成します。
    - **-r** フラグを指定して **rs_subcmp** プログラムを実行し、レプリケート・サブスクリプション・データとプライマリ・サブスクリプション・データを一致させます。

参照：
- RSSD の基本的なリカバリ手順の使用 (403 ページ)
- サブスクリプション比較の手順の使用 (406 ページ)

統合解除と再統合の手順の使用

RSSD の最後のダンプ以降にルートを作成した場合は、統合解除と再統合の手順に従う必要があります。

1. 該当の Replication Server を複写システムから削除します。

   『Replication Server 管理ガイド 第 1 巻』の「Replication Server の削除」を参照してください。
2. Replication Server を再インストールします。

   Replication Server の再インストールの詳細については、使用しているプラットフォームの『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。
3. Replication Server のルートとサブスクリプションを再作成します。

   『Replication Server 管理ガイド 第 1 巻』の「ルートの管理」および『Replication Server 管理ガイド 第 1 巻』の「サブスクリプションの管理」を参照してください。

リカバリ・サポート作業

標準リカバリ作業により、複写システム内の非常に重要なデータをユーザが操作および識別できるようになります。

各リカバリ作業は、リカバリ手順でそれらの作業が必要な場合にのみ実行するようにしてください。

リカバリ・サポート作業には、次のものがあります。

- ステーブル・キューの再構築

- ステーブル・キュー再構築後の Replication Server ロス検出メッセージのチェック
- Replication Server をログ・リカバリ・モードにすることとデータベースのログ・リカバリの設定
- データベースのログ・リカバリを設定した後の Replication Server ロス検出メッセージのチェック
- ロードするダンプとログの決定
- データベース生成番号の調整

参照：
- ステーブル・キューの再構築 (418 ページ)
- ステーブル・キュー再構築後のロス検出 (421 ページ)
- データベースに対するログ・リカバリの設定 (425 ページ)
- ログ・リカバリ設定後のロス検出 (426 ページ)
- ロードするダンプの決定 (427 ページ)
- データベース生成番号の調整 (428 ページ)

ステーブル・キューの再構築

rebuild queues コマンドでは、すべての既存のキューが削除され、再構築されます。このコマンドで、ステーブル・キューを個別に再構築することはできません。

状況に応じて、オンラインまたはオフラインでキューを再構築できます。一般的には、最初にキューをオンラインで再構築して、失われたステーブル・キュー・メッセージを検出します。失われたメッセージがある場合は、最初に Replication Server をスタンドアロン・モードにして、オフライン・ログからデータをリカバリすることによって、それらのメッセージを取得できます。

『Replication Server リファレンス・マニュアル』の「Replication Server コマンド」の「**rebuild queues**」を参照してください。

オンラインでのキューの再構築

オンラインでの再構築中、Replication Server はノーマル・モードで動作します。すべての RepAgent および他の Replication Server は、自動的に再構築中の Replication Server からコネクションを切断されます。

コネクションの試行は拒否され、次のメッセージが表示されます。

```
Replication Server is Rebuilding
```

Replication Server と RepAgent は、**rebuild queues** コマンドが完了するまで定期的にコネクションをリトライします。このコマンドが完了すると、コネクションは確立されます。

キューがクリアされると、再構築が完了します。次に、Replication Server は、クリアされたメッセージを以下の送信元から取得しようとします。

- 再構築された Replication Server 宛の直接ルートを持つ他の Replication Server。他の Replication Server からのセーブ・インターバルを設定している場合は、リカバリできる確率が高くなります。
- Replication Server が管理するプライマリ・データベースのデータベース・トランザクション・ログ。

ロス検出メッセージがある場合は、これらのメッセージのステータスを確認する必要があります。失敗の状況に応じて、失われたメッセージを送信元から取り出せなかった場合は、オフライン・ログを使用してキューを再構築することが必要な場合もあります。または、Replication Server に失われたメッセージを無視するよう指示することもできます。

参照：
- オフライン・データベース・ログからのキューの再構築 (419 ページ)
- ステーブル・キュー再構築後のロス検出 (421 ページ)

オフライン・データベース・ログからのキューの再構築

オフライン・データベース・ログからデータをリカバリできます。

Replication Server をスタンドアロン・モードで再起動したあとに、**rebuild queues** コマンドを使用します。スタンドアロン・モードで **rebuild queues** を実行すると、Replication Server はリカバリ・モードになります。

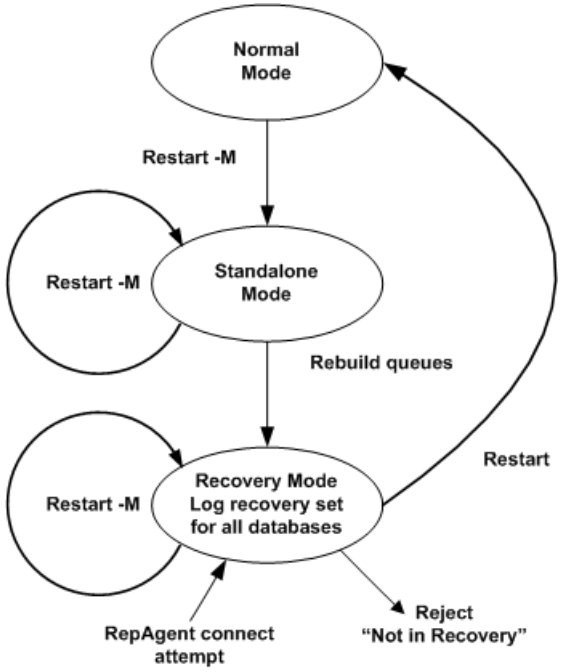
リカバリ・モードでは、リカバリ・モードの RepAgent だけが Replication Server に接続できます。リカバリ・モードではない RepAgent が接続しようとすると、Replication Server は接続を拒否し、次のエラー・メッセージを表示します。

```
Rep Agent not in recovery mode
```

RepAgent を自動的に再起動して Replication Server に接続させるスクリプトを使用する場合は、**for_recovery** オプションを使用して RepAgent を起動する必要があります。RepAgent はノーマル・モードでは接続できません。

この図は、**rebuild queues** コマンドを使用して、ノーマル・モード、スタンドアロン・モード、リカバリ・モードの順にモードを切り替える過程を示します。

図 27 : rebuild queues コマンドを使用してリカバリ・モードにする方法



参照：

- Replication Server スタンドアロン・モード (420 ページ)

Replication Server スタンドアロン・モード

スタンドアロン・モードでは、キューへのメッセージの書き込みやキューからのメッセージの読み取りがないため、ステーブル・キューの内容を確認できます。

Replication Server をスタンドアロン・モードで起動するには、**-M** フラグを指定します。スタンドアロン・モードは、Replication Server の状態が静的になるため、Replication Server の状態を確認する場合に便利です。

Replication Server のスタンドアロン・モードは、次の点がノーマル・モードと異なります。

- 受信コネクションは受け入れられません。RepAgent または Replication Server がスタンドアロン・モードでこの Replication Server に接続しようとすると、「`Replication Server is in Standalone Mode` (この Replication Server はスタンドアロン・モードです)」というメッセージが表示されます。
- 送信コネクションは開始されません。スタンドアロン・モードの Replication Server は、他の Replication Server に接続しようとしません。

- 適用されていないメッセージが DSI キュー内に存在する場合も、DSI スレッドは開始されません。
- ディストリビュータ (DIST) スレッドは開始されません。DIST スレッドは、インバウンド・キューからのメッセージの読み取り、サブスクリプション解析の実行、アウトバウンド・キューへのメッセージの書き込みを行うスレッドです。

ステーブル・キュー再構築後のロス検出

Replication Server は、ステーブル・キューの再構築後にリカバリできなかったメッセージがあるかどうかを判断するために、ロスを検出します。

Replication Server のロス検出メッセージを確認することによって、すべてのデータをシステムにリストアするためにユーザが何をすべきかを判断できます。

Replication Server は、ステーブル・キューの再構築後に、次の 2 つのタイプのロスを検出します。

- SQM ロス – 2 つの Replication Server 間のデータ・ロスのことですが、次のダウンストリーム・サイトで検出されます。
- DSI ロス – Replication Server と、Replication Server が管理するレプリケート・データベースの間のデータ・ロスのことです。

すべてのデータが使用可能であれば、ユーザの介入は必要がなく、複写システムは通常のオペレーションに戻ることができます。たとえば、ルートまたはコネクションのセーブ・インターバルとして設定した時間が障害発生期間よりも長いことがわかっている場合は、ユーザが何もしなくてもすべてのメッセージをリカバリできます。しかし、セーブ・インターバルが設定されていない場合や、設定時間が短すぎる場合は、一部のメッセージが失われている可能性があります。

注意： ロスを検出した Replication Server は、それ以降送信元からメッセージを受け取りません。ロスが検出されると、送信元はステーブル・キューをトランケートしなくなります。

たとえば、レプリケート・データ・サーバ DS2.RDB がプライマリ・データ・サーバ DS1.PDB からのデータを失ったことを Replication Server RS2 が検出すると、Replication Server RS1 はロスの処理方法が決定されるまでキューをトランケートできなくなります。このため、RS1 のステーブル領域が足りなくなることがあります。ロスが検出されるまでの間 (`Checking Loss` メッセージが表示されたあと）、送信元と送信先のロスを無視できます。

---

2 つの Replication Server 間の SQM ロス

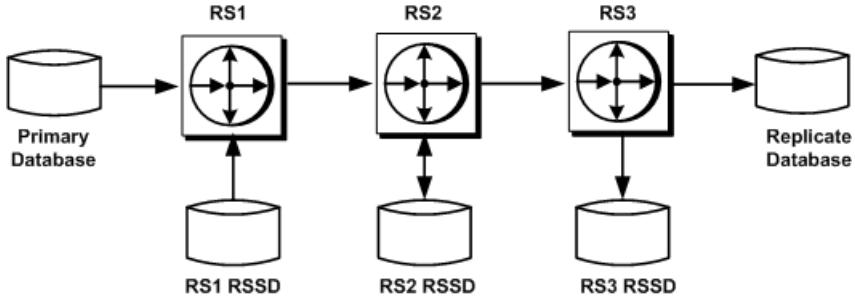
Replication Server が 2 つの Replication Server 間のデータ・ロスを検出する方法について説明します。

リカバリ手順でステーブル・キューを再構築するたびに、Replication Server は、分配元であるサイトからのバックログ・メッセージを要求します。Replication Server

は、プライマリ・データベースを管理している場合、プライマリ・データベースの RepAgent にオンライン・トランザクション・ログの最初からメッセージを送信するように命令します。バックログ・メッセージが、再び空のステーブル・キューに再送信されます。

Replication Server は、Replication Server からの直接ルートを持つ再構築中のサイトのロス検出モードを有効にします。図では、ユーザが Replication Server RS2 のキューを再構築すると、Replication Server RS3 によってロスが検出されます。同様に、ユーザが Replication Server RS1 のキューを再構築すると、RS2 によってロスが検出されます。

図 28 : 複写システムにおけるロス検出の例



RS2 で **rebuild queues** コマンドを実行すると、更新情報が RS2 経由で RS3 に到達するようになっているすべてのプライマリ・データベースに対して、RS3 によってロス検出が実行されます。RS3 は、これらの各データベースのメッセージのログを取ります。RS3 でキューを再構築した場合は、RS3 を起点とするルートが存在しないため、SQM ロス検出は実行されません。

Replication Server は重複メッセージの照合によってロスを検出します。RS3 が、**rebuild queues** コマンドの実行前に受け取ったメッセージを再び受け取った場合は、失われたメッセージはありません。RS3 が **rebuild queues** コマンドの実行後に初めて受け取ったメッセージが、今までに受け取ったことのないメッセージである場合は、メッセージが失われているか、ステーブル・キュー内にメッセージがなかったかのいずれかです。

RS3 は、ステーブル・キューに特定の送信元からのメッセージがない場合でも、比較の基準となる重複メッセージがないため、それらのメッセージが失われているとみなします。セーブ・インターバルより短いインターバルのハートビートを作成することによって、偽りのメッセージ・ロス検出を防ぐことができます。これによって、ステーブル・キュー内には常に少なくとも 1 つのメッセージが確実に存在するようになります。

SQM の例

RS3 は、再構築後の RS2 を対象に SQM ロス検出を実行するときに、次の Checking Loss メッセージの例と類似したログ・ファイル・メッセージのログを記録します。これらのメッセージは、ロス検出の処理が開始されたことを意味します。それ以降のメッセージは、検出結果とともにログに記録されます。各メッセージには、送信元とそれに対応する送信先が含まれています。

1 番目の例は、RS2 の RSSD から RS3 の RSSD への間のロスを RS3 がチェックしていることを意味するメッセージです。

```
Checking Loss for DS3.RS3_RSSD from DS2.RS2_RSSD
date=Nov-01-95 10:15 am
qid=0x01234567890123456789
```

2 番目の例は、RS1 の PDB プライマリ・データベースから RS3 の RDB レプリケート・データベースへの間のロスを RS3 がチェックしていることを意味するメッセージです。

```
Checking Loss for DS3.RDB from DS1.PDB
date=Nov-01-95 11:00am
qid=0x01234567890123456789
```

3 番目の例は、RS1 の RSSD から RS3 の RSSD への間のロスを RS3 がチェックしていることを意味するメッセージです。

```
Checking Loss for DS3.RS3_RSSD from DS1.RS1_RSSD
date=Nov-01-95 10:00am
qid=0x01234567890123456789
```

RS3 はロスを検出したかどうかをレポートします。ロス検出テストの結果は次の例のようにレポートされます。

```
No Loss for DS3.RS3_RSSD from DS2.RS2_RSSD
```

```
Loss Detected for DS3.RDB from DS1.PDB
```

```
No Loss for DS3.RS3_RSSD from DS1.RS1_RSSD
```

Replication Server とそのデータベースの間の DSI ロス

Replication Server と、Replication Server が管理するレプリケート・データベースの間のデータ・ロスを Replication Server が検出する方法について説明します。

Replication Server キュー内のメッセージには、他の Replication Server ではなくデータベース宛に送信されるものがあります。DSI は、ステーブル・キューのロス検出と同様の方法でロス検出を実行します。

起点とするルートが設定されていない Replication Server でキューを再構築しても、SQM ロス検出は実行されません。ただし Replication Server は DSI ロス検出を実行して、DSI ロス検出メッセージの有無を確認します。

DSI の例

Replication Server RS2 の DSI は、RS2 の RSSD に対して次のメッセージを生成します。

```
DSI: detecting loss for database DS2.RS2_RSSD from origin
DS1.RS1_RSSD
date=Nov-01-95 10:58pm
qid=0x01234567890123456789
```

保持されたメッセージが直前のサイトから到達し始めると、送信元から最初に送られたメッセージを DSI がすでに検出しているかどうかに従って、DSI はロスを検出します。DSI がロスを検出しない場合には、次のようなメッセージが生成されます。

```
DSI: no loss for database DS2.RS2_RSSD from origin DS1.RS1_RSSD
```

DSI がロスを検出した場合は、次のようなメッセージが生成されます。

```
DSI: loss detected for database DS2.RS2_RSSD from origin DS1.RS1_RSSD
```

ロスの処理

Replication Server がロスを検出した場合、SQM または DSI 宛のコネクションでは、それ以上メッセージが受け取られなくなります。

たとえば、RS3 は、PDB データベースから RDB データベースへの間で SQM メッセージのロスを検出すると、PDB データベースから RDB データベースへのそれ以降のメッセージをすべて拒否します。

ロスのリカバリ

ロスをリカバリするには、3 つの方法のいずれかを選択する必要があります。

次の方法があります。

- 一部のメッセージが失われる可能性がある場合でも、ロスを無視して続行します。**-r** フラグを指定した **rs_subcmp** プログラムを含むサブスクリプション比較の手順を実行して、プライマリ・データとレプリケート・データを一致させます。
  また、『Replication Server 管理ガイド 第 1 巻』の「サブスクリプションの管理」、および『Replication Server リファレンス・マニュアル』の「実行プログラム」の「**rs_subcmp**」を参照してください。
- ロスを無視し、サブスクリプションを削除してから、再作成します。
- トランザクションをオフライン・ログからリプレイして、リカバリします (プライマリ Replication Server のロスの場合のみ有効)。この場合は、ロスを無視しません。

参照：
- サブスクリプション比較の手順の使用 (406 ページ)

ロスの無視
ignore loss コマンドは、特定の状況で実行します。

ignore loss は次の状況で実行します。

- サブスクリプションを再作成するか、ログをリプレイして、失われたメッセージをリカバリすることを選択した場合、コマンドを実行します。
- ロスをレポートした Replication Server でメッセージの受け取りを強制的に再開させるために SQM ロスに対してコマンドを実行します。たとえば、Replication Server RS3 で検出された DS1.PDB からのロスを無視するには、RS3 で次のコマンドを入力します。
  ```
  ignore loss from DS1.PDB to DS3.RDB
  ```
- ロスの検出場所が Replication Server のデータベースである場合、DSI ロスに対してコマンドを実行します。たとえば、DS2.RS2_RSSD でオリジン DS1.RS1_RSSD からのロスがレポートされた場合にこのロスを無視するには、RS2 で次のコマンドを入力します。
  ```
  ignore loss from DS1.RS1_RSSD to DS2.RS2_RSSD
  ```
- 2 つの Replication Server を連続して再構築するときに、ルートの送信先の Replication Server で検出された SQM ロスと DSI ロスの両方に対してコマンドを実行します。
  この場合は、**ignore loss** コマンドを、SQM ロスに対して 1 回、DSI ロスに対して 1 回、合計 2 回実行します。送信先 Replication Server で、DSI ロスを無視するために実行する **ignore loss** コマンドは、直前のサイトからの SQM ロスを無視するために使用するコマンドと同一です。

データベースに対するログ・リカバリの設定
ログ・リカバリを手動で設定する手順は、トランケート済みのプライマリ・データベース・ログからオフラインでリカバリする手順や、プライマリ・データベースとレプリケート・データベースをダンプからリストアする手順の一部です。

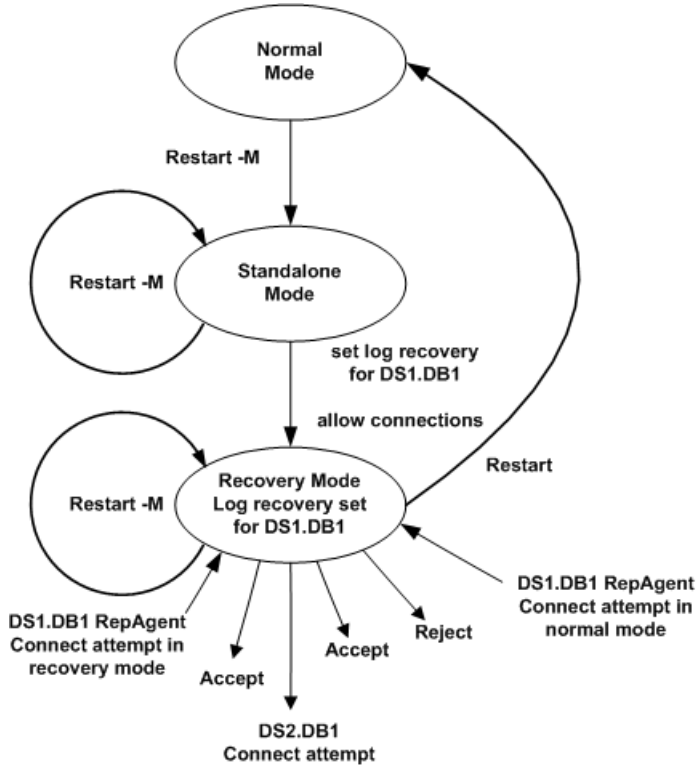
キューをオフラインで再構築する手順では、すべてのデータベースに対してログ・リカバリが自動的に設定されますが、ログ・リカバリを手動で設定すると、ステーブル・キューを再構成せずにそれぞれのデータベースをリカバリできます。

set log recovery コマンドは、Replication Server を特定のデータベースのログ・リカバリ・モードに設定します。このコマンドは、Replication Server をスタンドアロン・モードにしてから実行してください。ログ・リカバリ・モードの対象として設定されたデータベースにのみ RepAgent を接続するには、**allow connections** コマンドを実行します。このコマンドは、Replication Server をリカバリ・モードにします。

この図は、**set log recovery** コマンドと **allow connections** コマンドを使用して、ノーマル・モード、スタンドアロン・モード、リカバリ・モードの順にモードを切り替える過程を示します。

set log recovery コマンドを使用して指定されたデータベースについて、Replication Server は、リカバリ・モードの他の Replication Server または RepAgent からのコネクションだけを受け入れます。その後、トランザクション・ダンプをテンポラリ・リカバリ・データベースにリカバリします。

図 29 : allow connections コマンドを使用してリカバリ・モードにする方法



ログ・リカバリ設定後のロス検出

プライマリ・データベースにテンポラリ・リカバリ・データベースを適用している間に、Replication Server がプライマリ・データベースとそのプライマリ・データベースを管理する Replication Server 間で SQM ロスを検出する場合があります。

すべてのデータが使用可能であれば、ユーザの介入は必要がなく、複写システムは通常のオペレーションに戻ることができます。Replication Server は次のようなメッセージのログを取ります。

```
No Loss Detected for DS1.PDB from DS1.PDB
```

メッセージが十分にない場合、Replication Server は次のようなロス検出メッセージのログを取ります。

```
Loss Detected for DS1.PDB from DS1.PDB
```

ignore loss コマンドを実行してロスを無視するか、リカバリ手順を初めから繰り返すかを決定する必要があります。ロスを無視するには、プライマリ Replication Server で次のコマンドを入力します。

```
ignore loss for DS1.PDB from DS1.PDB
```

ロス検出メッセージを受け取った場合は、データベースの再ロードに失敗して、すべてのメッセージを取得できる状態にさかのぼることができなかったことを意味します。ロードするダンプを正しく決定する必要があります。

参照：

- ロードするダンプの決定 (427 ページ)

ロードするダンプの決定

トランザクション・ログ・ダンプをロードするときには、ロス検出時に表示される `Checking Loss` メッセージを常に確認してください。

複数のメッセージが存在する場合は、日時が古い方のメッセージを選択して、どのダンプをロードするかを決定します。

たとえば、Replication Server が次のメッセージを生成すると、2011 年 11 月 1 日午後 10 時 58 分の直前に生成されたダンプをロードします。

```
Checking Loss for DS3.RDB from DS1.PDB
date=Nov-01-2011 10:58pm
qid=0x01234567890123456789
```

メッセージの `date` は、Replication Server が受け取った最後のメッセージがオリジン・キューによって生成された、ログ内の最も古いオープン・トランザクションの日時です。メッセージに示された日時よりも前の `timestamp` を持つ、最新のトランザクション・ダンプを特定します。さらに、このトランザクション・ダンプの前に生成された、完全なデータベース・ダンプを検索します。

オリジン・キュー ID すなわち `qid` は、RepAgent によって作成され、トランザクション・ログ内のログ・レコードを識別します。`date` は、`qid` 内に `timestamp` として埋め込まれます。Replication Server は、`timestamp` を Adaptive Server の RepAgent の日付に変換します。

Sybase 以外のデータ・サーバの複写エージェントも `qid` 内に `timestamp` を埋め込む場合があります。Replication Server は、Sybase 以外のデータ・サーバの `timestamp` を 20 ～ 27 バイト単位で変換します。実際のバイト数は、Replication Agent によって異なります。

注意： データ・サーバが Adaptive Server でない場合は、メッセージ内の日付が意味をなさない場合があります。`qid` を 20 ～ 27 バイト単位でデコードして、ロード対象のダンプを識別してください。

データベース生成番号の調整

リカバリのためにデータベースをロードする場合は、使用しているリカバリ手順の指示に従って、データベース生成番号を変更することが必要になる場合があります。

複写システム内の各プライマリ・データベースには、データベース生成番号が格納されます。この番号は、プライマリ・データベースと、プライマリ・データベースを管理する Replicatin Server の RSSD に格納されます。

データベース生成番号の最大値は 65,535 です。どうしても必要なとき以外は、大きい番号にならないようにすることをおすすめします。

データベース生成番号をリセットする場合は、レプリケーション環境を再構築する必要があります。環境を再構築するには、データベース生成番号をリセットするプライマリ・データベースへのコネクションを削除し、コネクションを再確立し、プライマリ・データベースの複写設定を再構築します。

データベース生成番号の決定

データベース生成番号の調整を行う時期について説明します。

プライマリ・データベースの RepAgent は、Replication Server に渡すログ・レコードごとに作成する `qid` の上位 2 バイトに、データベース生成番号を設定します。

`qid` の残りのバイトは、データベース生成番号以外の、ログ内のレコードの場所を示す情報から構成され、Replication Server にレコードが渡されるたびに `qid` を増分させます。

`qid` 値を増やすための必要条件に基づいて、Replication Server は重複レコードを検出できます。たとえば、RepAgent が、再起動時に Replication Server によって処理されているログ・レコードを再送信するとします。Replication Server は、Replication Server が処理した最後のレコードより小さい `qid` を持ったレコードを受信する場合、そのレコードを重複したレコードとして扱い、無視します。

プライマリ・データベースを以前の状態にリストアする場合は、データベースが再ロードされた後に送信されるログ・レコードを Replication Server が無視しなくなるように、データベース生成番号を増分します。この手順は、プライマリ・データベースをダンプからロードしている場合、またはコーディネート・ダンプからロードしている場合にのみ適用されます。

ログ・レコードをリプレイする場合は、RepAgent がより大きな生成番号を持つ、再ロードされたログ・レコードを事前に送信したときにのみ、データベース生成番号を増分します。この状況は、最初に発生した障害に対してデータベースとロ

グを以前の状態にリストアしてから、次の障害に対してログをリプレイする必要がある場合にのみ発生します。

警告！ データベース生成番号は、リカバリ手順の一部としてのみ変更できます。その他の場合にデータベース生成番号を変更すると、レプリケート・データベースでデータの重複または欠落が発生することがあります。

参照：

- ダンプからのプライマリ・データベースのロード (399 ページ)
- コーディネート・ダンプからのロード (401 ページ)

ダンプとデータベース生成番号

ダンプを再ロードした後、データベース生成番号を調整する時期と方法について説明します。

データベース・ダンプを再ロードする場合、データベース生成番号は、リストア済みのデータベースに含まれます。データベース生成番号は、データベースを管理する Replication Server の RSSD にも格納されているため、RSSD 内の番号とリストア済みのデータベース内の番号が一致するよう更新しなければならない場合もあります。

ただし、トランザクション・ログを再ロードする場合は、データベース生成番号はリストアされたログには含まれません。次に、例としてデータベース内に発生したオペレーションの種類と対応するデータベース生成番号を示します。

表 34 : ダンプとデータベース生成番号

オペレーション	データベース生成番号
データベース・ダンプ D1	100
トランザクション・ダンプ T1	100
dbcc settrunc('ltm', 'gen_id', 101)	101
トランザクション・ダンプ T2	101
データベース・ダンプ D2	101

データベース・ダンプ D1 を再ロードすると、データベース生成番号 100 がリストアされます。トランザクション・ダンプ T1 を再ロードすると、データベース生成番号は 100 のまま変わりません。トランザクション・ダンプの再ロードによってデータベース生成番号は変更されないため、トランザクション・ダンプ T2 を再ロードしても 100 のまま変わりません。この場合は、**dbcc settrunc** コマンドを使用して、データベース生成番号を 101 に変更してから、RepAgent に トランザクション・ダンプ T2 をスキャンさせてください。

ただし、データベース・ダンプ D2 をロードしてから、複写を再開する場合は、データベース生成番号 101 がリストアされるため、番号を変更する必要はありません。

プライマリ・データベース生成番号のリセット
データベース生成番号のリセット方法について説明します。

この手順におけるプライマリ・データベースは、データベース生成番号をリセットするプライマリ・データベースのことです。

1. レプリケート Replication Server では、プライマリ・データベースへのコネクション用に定義された複写定義とパブリケーションを参照するすべてのサブスクリプションを削除します。
2. 手順 1 で削除したサブスクリプションが参照するすべてのパブリケーションを削除します。
3. 手順 2 で削除したパブリケーションが参照するすべてのアーティクルを削除します。
4. プライマリ Replication Server では、プライマリ・データベース・コネクションのすべての複写定義を削除します。
5. プライマリ Replication Server では、プライマリ・データベースへのコネクションと、プライマリ・データベースをサブスクライブするレプリケート・データベースへのすべてのコネクションを削除します。
6. プライマリ・データベースで、データベース生成番号を 0 に設定します。

   - Adaptive Server の場合:
     ```
     dbcc settrunc('ltm', 'gen_id', 0)
     ```
   - IBM DB2 UDB (UNIX および Windows 上)、Microsoft SQL Server、および Oracle の Replication Agent の場合
     ```
     pdb_gen_id 0
     ```
7. プライマリ Replication Server で、プライマリ・データベースへの新しいコネクションを作成し、レプリケート・データベースへのコネクションを作成します。
8. 削除したすべての複写定義、パブリケーション、アーティクル、およびサブスクリプションを再作成します。『Replication Server 管理ガイド 第 1 巻』の「Sybase Central での複写環境の管理」の「複写環境の設定」を参照してください。

ASE 以外のデータベースのサポート
プライマリ・データベースとして機能する ASE 以外のデータベースのうちサポートされているものはすべて、データベース生成番号をリセットできます。

サポートされているプライマリ・データベースについては、『Replication Server 異機種間複写ガイド』を参照してください。

Adaptive Server のレプリケート・データベースの再同期

Replication Server を使用すると、レプリケート・データベースを再同期してマテリアライズできます。また、プライマリ・データベースのクワイスを強いることなく、データの損失や整合性を失うリスクなしで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。

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」を参照してください。

データベースの再同期を設定する

データベースの再同期を設定するには、Replication Server と RepAgent の両方のコマンドとパラメータを使用します。

1. RepAgent をサスペンドして複写プロセスを停止します。
2. Replication Server を再同期モードにします。
   再同期モードになると、Replication Server はトランザクションをスキップします。さらに、プライマリ・データベースまたは信頼されたソースから取得したダンプを使ってレプリケート・データベースにデータを再移植する準備として、複写キューから複写データをパージします。
3. RepAgent を再開して、データベース再同期マーカを Replication Server に送信し、再同期処理が進行中であることを示します。
4. DSI がデータベース再同期マーカを受け取ったことを確認します。
5. プライマリ・データベースからダンプを取得します。

   Replication Server がプライマリ・データベース・ダンプが完了したことを示すダンプ・マーカを検出すると、Replication Server はトランザクションのスキップを停止し、どのトランザクションをレプリケート・データベースに適用するかを判定できるようになります。
6. DSI がデータベース・ダンプ・マーカを受け取ったことを確認します。

   注意： データベース・ダンプのマーカの送信は、再同期マーカに **init** 命令を付けて送信した場合には適用されません。
7. ダンプをレプリケート・データベースに適用します。
8. 複写をレジュームします。

Replication Server にトランザクションをスキップさせる

指定されたレプリケート・データベースで DSI アウトバウンド・キュー内のトランザクションを Replication Server にスキップさせるには、**skip to resync** パラメータ

を **resume connection** コマンドに付けて使用します。これは Replication Server が RepAgent からのデータベース・ダンプ・マーカを受け取るまで有効です。

レプリケート・データベース内のデータはダンプの内容によって置き換えられることになっているので、Replication Server はアウトバウンド・キュー内のレコードを処理しません。

『Replication Server リファレンス・マニュアル』の「Replication Server コマンド」の「**resume connection**」を参照してください。

次のコマンドを実行します。

```
resume connection to data_server.database skip to resync
marker
```

警告！ resume connection を **skip to resync marker** オプションを付けて間違ったコネクションで実行すると、レプリケート・データベースのデータが非同期されます。

skip to resync marker を設定すると、Replication Server は Replication Server ログ内またはデータベース例外ログ内でスキップされたトランザクションをログに記録しません。**skip [n] transaction** を設定すると、Replication Server はスキップされたトランザクションをログに記録します。

データベース再同期マーカを Replication Server に送信する

RepAgent を使用してデータベース再同期マーカを Replication Server に送信し、再同期処理が進行中であることを示します。

再同期モードで RepAgent を再開すると、RepAgent はデータベース再同期マーカを最初のメッセージとして Replication Server へ送信してから、SQL データ定義言語 (DDL: data definition language) またはデータ操作言語 (DML: data manipulation language) のトランザクションを送信します。同じプライマリ・データベースの複数のレプリケート・データベースはそれぞれに DSI アウトバウンド・キューがあるので、すべて同じ再同期マーカを受け取ります。

skip to resync marker パラメータでレジュームする各 DSI に対して、DSI が再同期マーカを受け取ったことが、DSI アウトバウンド・キューによって Replication Server システム・ログに記録されます。また、その時点からデータベース・ダンプ・マーカを受け取るまで DSI がコミットされたトランザクションを拒否することも記録されます。

Adaptive Server では、データベース再同期マーカの送信で各オプションをサポートするために、**resync**、**resync purge**、または **resync init** パラメータを指定して **sp_start_rep_agent** を使用します。

オプションを指定しないで再同期マーカを送信する
トランケーション・ポイントに変更がなく、RepAgent が最後に処理したところからトランザクション・ログの処理を続けることになっているときは、**sp_start_rep_agent** を使用してオプションを指定しないで再同期マーカを送信します。

構文： **sp_start_rep_agent** *database_name*, **'resync'**

各アウトバウンド DSI スレッドとキューはデータベース再同期マーカを受け取り処理します。再同期マーカを受け取ったとき、DSI は skip to resync マーカ要求に従って Replication Server システム・ログにレポートを送ります。その後、ダンプ・データベース・マーカを待つ間、DSI はコミットされたトランザクションを拒否します。このメッセージと、ダンプ・データベース・マーカを待つ動作の変更によって、レプリケート・データベースにダンプを適用できるようになります。

パージ命令付きで再同期マーカを送信する
再同期マーカを送信するために **purge** オプションを指定して **sp_start_rep_agent** を使用すると、新しいインバウンド・トランザクションを受け取る前にインバウンド・キュー内のすべてのオープン・トランザクションをパージして重複の検出をリセットするよう、Replication Server に指示できます。

構文： **sp_start_rep_agent** *database_name*, **'resync purge'**

プライマリ・データベースのトランケーション・ポイントが移動した場合は、**purge** オプションを使用します。これは次の操作を行った場合に発生します。

- トランケーション・ポイントを手動で変更する。
- RepAgent を無効にする。
- **dbcc dbrepair** などの Adaptive Server コマンドを実行する。

トランケーション・ポイントが変更されると、Replication Server のインバウンド・キュー内にあるオープン・トランザクションは、新しいセカンダリ・トランケーション・ポイントから送られたアクティビティと一致しないため、パージされる必要があります。変更されたトランケーション・ポイントが以前のオリジン・キュー ID (OQID) を持つレコードを送信する可能性があるので、Replication Server は重複検出をリセットします。以前のデータがキューからパージされると、Replication Server は RepAgent からのすべてのアクティビティを重複アクティビティとして扱いません。したがって、新しいアクティビティが拒否されることはありません。ダンプ・データベース・マーカを受け取るまで、Replication Server はアウトバウンド・キュー・コマンドを拒否し続けるので、パージ・オプションは DSI の処理を変更しません。

init コマンド付きで再同期マーカを送信する
再同期マーカを **init** コマンド付きで送信するには、init オプションを指定して **sp_start_rep_agent** を使用します。これによって、インバウンド・キュー内のすべてのオープン・トランザクションをパージして重複の検出をリセットし、アウトバウンド DSI をサスペンドするよう、Replication Server に指示できます。

構文： **sp_start_rep_agent** *database_name*, **'resync init'**

このオプションはプライマリ・データベースにレプリケート・データベースと同じダンプを再ロードするときに使用します。プライマリ・データベースから取得したダンプはないので、RepAgent はダンプ・データベース・マーカを送りません。再同期マーカの後に来るダンプ・データベース・マーカを待つ代わりに、**init** オプションは Replication Server が再同期マーカを処理したらすぐに DSI コネクションをサスペンドします。

DSI がサスペンドされたら、それ以降 DSI を通るすべてのアクティビティは新しいトランザクションのみになります。プライマリで使用したダンプをレプリケート・データベースに再ロードしたら、DSI をレジュームできます。

データベースのダンプを取得する
Adaptive Server の **dump database** コマンドを使用します。

Adaptive Server Enterprise の『システム管理ガイド 第 2 巻』の「バックアップおよびリカバリ・プランの作成」の「**dump** コマンドおよび **load** コマンドの使用方法」を参照してください。

ダンプ・データベース・マーカを Replication Server に送信する
RepAgent はプライマリ・データベースからダンプを取得すると、Replication Server にダンプ・データベース・マーカを自動的に生成して送信します。

注意： ダンプ・データベース・マーカの送信は、再同期マーカに **init** 命令を付けて送信した場合には適用されません。

レプリケート・データベースにダンプを適用したら、手動で DSI をレジュームできます。ダンプ・データベース・マーカが示すダンプ・ポイントの後にコミットされたトランザクションは、レプリケートされます。

DSI スレッド情報をモニタする

データベースの再同期中に DSI についての情報を提供するには、**admin who** コマンドを使用します。

状態	説明
SkipUntil Resync	**skip to resync** を実行した後 DSI がレジュームする。このステータスは DSI がデータベース再同期マーカを受け取るまで続く。
SkipUntil Dump	DSI がデータベース再同期マーカを受け取った。このステータスは DSI がその後のダンプ・データベース・マーカを受け取るまで続く。

再同期するデータベースにダンプを適用する

プライマリ・データベースのダンプをレプリケート・データベースに適用できるのは、関連するメッセージがシステム・ログに表示された後だけです。

- Replication Server が **purge** オプション付き、またはなしの再同期データベース・マーカと、ダンプ・データベース・マーカを受け取るときのメッセージ。

```
DSI for data_server.database received and processed
Resync Database Marker. Waiting for Dump Marker.
```

```
DSI for data_server.database received and processed
Dump Marker. DSI is now suspended. Resume after database has been
reloaded.
```

- Replication Server が **init** マーカ付きの再同期データベースを受け取るときのメッセージ。

```
DSI for data_server.database received and processed
Resync Database Marker. DSI is now suspended. Resume after
database has been reloaded.
```

再同期するデータベースにダンプをロードする方法の詳細については、『Adaptive Server Enterprise リファレンス・マニュアル：コマンド』の「コマンド」の「**load database**」を参照してください。

データベース再同期化シナリオ

データベースの再同期手順はそのシナリオによって異なります。再同期手順を完了すると、プライマリ・データベースとレプリケート・データベースはトランザクションの一貫性が保たれた状態になります。

手順を実行するには次の要件があります。

- 複写システム管理者であること。
- 正常に稼動する複写環境が存在すること。
- プライマリ・データベースからレプリケート・データベースへデータをコピーするためのメソッドやプロセスがあること。

Adaptive Server と Replication Server の RepAgent のコマンドと構文については、『Replication Server リファレンス・マニュアル』、および『Replication Server 管理ガ

イド 第 1 巻』の「RepAgent の管理と Adaptive Server のサポート」を参照してください。

1 つ以上のレプリケート・データベースをプライマリ・データベースから直接再同期する

1 つ以上のレプリケート・データベースを 1 つのプライマリ・データベースから再同期します。

この手順では、多少の違いはありますが、次のことを実行できます。

- プライマリ・データベースとレプリケート・データベース間の複写の遅延時間が、複写によるデータベースの回復が不可能で、複写データに基づくレポートの作成が実用的でなくなった場合に、レプリケート・データベースにデータを再移植する。
- プライマリ・データベースから信頼されたデータをレプリケート・データベースに再移植する。
- プライマリ・データベースが複数のレプリケート・データベースのソースになっている場合に、再同期を調整する。
- プライマリ・サイトが一対のウォーム・スタンバイ・データベースで構成されている論理データベースであり、それに 1 つまたは複数のレプリケート・データベースを再同期する場合に、再同期を調整する。ウォーム・スタンバイのペアでは、アクティブ・データベースがプライマリ・データベースとして、スタンバイがレプリケート・データベースとして機能する。したがって、1 つまたは複数のレプリケート・データベースからはアクティブ・データベース (プライマリ・サイトのウォーム・スタンバイ・ペアの 1 つ) がプライマリ・データベースに見える。

プライマリ・データベースから直接再同期する

レプリケート・データベースをプライマリ・データベースから直接再同期します。

1. RepAgent による複写プロセスを停止します。Adaptive Server で次のコマンドを実行します。

   ```
   sp_stop_rep_agent database
   ```

2. レプリケート・データベースとの Replication Server DSI コネクションをサスペンドします。

   ```
   suspend connection to dataserver.database
   ```

3. レプリケート・データベースのアウトバウンド・キューからデータを削除し、プライマリ・データベースの RepAgent からの再同期マーカを待機するように Replication Server に指示します。

   ```
   resume connection to data_server.database skip to
   resync marker
   ```

4. RepAgent に再同期モードで起動するよう指示し、再同期マーカを Replication Server に送信します。
   - トランケーション・ポイントが元の位置から移動していない場合は、Adaptive Server で次のコマンドを実行します。

     ```
     sp_start_rep_agent database, 'resync'
     ```
   - トランケーション・ポイントが元の位置から移動している場合は、Adaptive Server で次のコマンドを実行します。

     ```
     sp_start_rep_agent database, 'resync purge'
     ```
5. Replication Server システム・ログで次のメッセージを検索して、DSI が RepAgent から再同期マーカを受信して受け入れていることを確認します。

   ```
   DSI for data_server.database received and processed
   Resync Database Marker. Waiting for Dump Marker.
   ```

   注意： 複数のデータベースを再同期する場合は、再同期する各データベースの DSI コネクションが再同期マーカを受け入れていることを確認します。
6. プライマリ・データベース・コンテンツのダンプを取得します。詳細については、『Adaptive Server Enterprise リファレンス・マニュアル：コマンド』の「コマンド」の「**dump database**」を参照してください。Adaptive Server は自動的にダンプ・データベース・マーカを生成します。
7. Replication Server システム・ログで次のメッセージを検索して、Replication Server がダンプ・データベース・マーカを処理していることを確認します。

   ```
   DSI for data_server.database received and processed
   Dump Marker. DSI is now suspended. Resume after database has been
   reloaded.
   ```

   Replication Server がダンプ・マーカを受け取ると、DSI コネクションが自動的にサスペンドされます。
8. プライマリ・データベースのダンプをレプリケート・データベースに適用します。詳細については、『Adaptive Server Enterprise リファレンス・マニュアル：コマンド』の「コマンド」の「**load database**」を参照してください。
9. レプリケート・データベースにダンプを適用したら、次のコマンドを使用して DSI をレジュームします。

   ```
   resume connection to data_server.database
   ```

サードパーティ・ダンプ・ユーティリティを使用して再同期する

ディスク・スナップショット用ツールのようなサードパーティ・ダンプ・ユーティリティを使ってプライマリ・データベースをダンプした後、再同期を調整します。

サードパーティ・ツールでは、プライマリ・データベースとのやり取りをネイティブのデータベース・ダンプ・ユーティリティほど密接には行うことができま

せん。RepAgent がダンプ・データベース・マーカの生成に使用できるような記録をサードパーティ・ツールがプライマリ・データベースのトランザクション・ログに書き込まない場合は、独自のダンプ・データベース・マーカを生成して再同期処理を完了できるようにします。詳細については、サードパーティ・ツールのマニュアルを参照してください。

1. RepAgent による複写プロセスを停止します。Adaptive Server で次のコマンドを実行します。

   ```
   sp_stop_rep_agent database
   ```

2. レプリケート・データベースとの Replication Server DSI コネクションをサスペンドします。

   ```
   suspend connection to dataserver.database
   ```

3. レプリケート・データベースのアウトバウンド・キューからデータを削除し、プライマリ・データベースの RepAgent からの再同期マーカを待機するように Replication Server に指示します。

   ```
   resume connection to data_server.database skip to
   resync marker
   ```

4. サードパーティ・ユーティリティを使用してプライマリ・データベースのコンテンツのダンプを取得します。
5. サードパーティ・ツールからダンプまたは情報を取得したら、プライマリ・データベースからの情報に基づいてダンプ・ポイントを決定します。サードパーティ・ツールを使用する場合、ユーザはダンプ・ポイントを決定する責任があります。たとえば、ディスク複写ツールを使用する場合、プライマリ・データベースでアクティビティを一時的に停止してディスク・スナップショットから実行中のトランザクションを消去し、ダンプ・データベース・マーカとして「トランザクション・ログの末尾」ポイントを使用できます。
6. 手順 5 で取得したダンプの位置の末尾に RepAgent 用のマークを付けるには、プライマリ・データベースで **rs_marker** ストアド・プロシージャを実行します。

   ```
   rs_marker "dump database database_name 'current date' oqid"
   ```

   ここで、*current date* は `datetime` 形式の任意の値、*oqid* は任意の有効な 16 進値です。『Replication Server リファレンス・マニュアル』の「トピック」の「データ型」の「日付と時間のデータ型」の「日付および時間の値の入力フォーマット」を参照してください。

   たとえば、`rdb1` 上のダンプ位置の末尾を日付および時間値 "20110915 14:10:10" と *oqid* の値 0x0003 でマーク付けすることができます。

   ```
   rs_marker "dump database rdb1 '20110915 14:10:10' 0x0003"
   ```

RepAgent は、手順 6 でマーク付けしたポイントのダンプ・データベース・マーカを自動的に生成し、それを Replication Server に送信します。

7. 再同期モードで RepAgent を開始し、再同期マーカを Replication Server に送信します。
   - トランケーション・ポイントが元の位置から移動していなければ、Adaptive Server で次のコマンドを実行します。

     ```
     sp_start_rep_agent database, 'resync'
     ```
   - トランケーション・ポイントが元の位置から移動している場合は、Adaptive Server で次のコマンドを実行します。

     ```
     sp_start_rep_agent database, 'resync purge'
     ```
8. Replication Server システム・ログで次のメッセージを検索して、DSI が Replication Agent から再同期マーカを受信して受け入れていることを確認します。

   ```
   DSI for data_server.database received and processed
   Resync Database Marker. Waiting for Dump Marker.
   ```
9. Replication Server システム・ログで次のメッセージを検索して、Replication Server がダンプ・データベース・マーカを処理していることを確認します。

   ```
   DSI for data_server.database received and processed
   Dump Marker. DSI is now suspended. Resume after
   database has been reloaded.
   ```

   Replication Server がダンプ・マーカを受け取ると、DSI コネクションが自動的にサスペンドされます。
10. プライマリ・データベースのサードパーティ・ツールからのダンプをレプリケート・データベースに適用します。詳細については、Adaptive Server のマニュアルとサードパーティ・ツールのマニュアルを参照してください。
11. レプリケート・データベースにダンプを適用したら、次のコマンドを使用して DSI をレジュームします。

    ```
    resume connection to data_server.database
    ```

データベース再同期マーカに対するサポートがない場合に再同期する

RepAgent またはプライマリ・データベースが再同期マーカを自動生成するように更新されていない場合に再同期を調整します。

注意： この手順は Adaptive Server でしか実行できません。

1. レプリケート・データベースとの Replication Server DSI コネクションをサスペンドします。

   ```
   suspend connection to dataserver.database
   ```

2. レプリケート・データベースのアウトバウンド・キューからデータを削除し、プライマリ・データベースの RepAgent からの再同期マーカを待機するように Replication Server に指示します。

```
resume connection to data_server.database skip to
resync marker
```

3. システム・ログにオープン・トランザクションがないことを確認してから、プライマリ・データベースで **resync marker** を手動生成します。

```
execute rs_marker 'resync database'
```

4. Replication Server システム・ログで次のメッセージを検索して、DSI が RepAgent から再同期マーカを受信して受け入れていることを確認します。

```
DSI for data_server.database received and processed
Resync Database Marker. Waiting for Dump Marker.
```

5. プライマリ・データベース・コンテンツのダンプを取得します。

   Adaptive Server は自動的にダンプ・データベース・マーカを生成します。『Adaptive Server Enterprise リファレンス・マニュアル：コマンド』の「コマンド」の「**dump database**」を参照してください。

6. Replication Server システム・ログで次のメッセージを検索して、Replication Server がダンプ・データベース・マーカを処理していることを確認します。

```
DSI for data_server.database received and processed
Dump Marker. DSI is now suspended. Resume after database has been
reloaded.
```

   Replication Server がダンプ・マーカを受け取ると、DSI コネクションが自動的にサスペンドされます。

7. プライマリ・データベースのダンプをレプリケート・データベースに適用します。詳細については、『Adaptive Server Enterprise リファレンス・マニュアル：コマンド』の「コマンド」の「**load database**」を参照してください。

8. レプリケート・データベースにダンプを適用したら、次のコマンドを使用して DSI をレジュームします。

```
resume connection to data_server.database
```

プライマリ・データベースとレプリケート・データベースを同じダンプから再同期する

再同期を調整して、プライマリ・データベースとレプリケート・データベースを同じダンプまたはデータのコピーから再ロードします。プライマリ・データベースからダンプを取得しないので、ダンプ・データベース・マーカは必要ありません。

1. RepAgent による複写プロセスを停止します。トランケーション・ポイントを変更しないでください。

Adaptive Server で次のコマンドを実行します。

```
sp_stop_rep_agent database
```

2. レプリケート・データベースとの Replication Server DSI コネクションをサスペンドします。

```
suspend connection to data_server.database
```

3. レプリケート・データベースのアウトバウンド・キューからデータを削除し、プライマリ・データベースの RepAgent からの再同期マーカを待機するように Replication Server に指示します。

```
resume connection to data_server.database skip to
resync marker
```

4. ダンプを適用する前に RepAgent の設定を取得します。

注意： Adaptive Server はデータベース内に RepAgent が使用する接続設定およびその他の設定を保存します。別のデータベースから取得したダンプをプライマリ・データベースにロードすると、RepAgent はその設定を失い、設定はダンプを取得した元のデータベースの設定に一致するように変更されます。

5. プライマリ・データベースに外部ソースからのデータ・ダンプを適用します。ダンプを適用したら、RepAgent の設定をダンプを適用する前の設定にリセットします。
6. プライマリ・データベースのトランザクション・ログにレプリケート・データベース・テーブルに影響するオペレーションが含まれていないことを確認するために、プライマリ Adaptive Server データベースで次のコマンドを実行します。

```
rs_update_lastcommit 0, 0, 0, ""
go 100
```

7. プライマリ・データベースのトランザクション・ログの最後までトランケーション・ポイントを移動します。Adaptive Server で次のコマンドを実行します。

```
dbcc settrunc('ltm', 'end')
go
```

8. **init** 命令を付けて再同期モードで RepAgent を開始します。Adaptive Server で次のコマンドを実行します。

```
sp_start_rep_agent database, 'resync init'
```

9. Replication Server システム・ログで次のメッセージを検索して、DSI が RepAgent から再同期マーカを受信して受け入れていることを確認します。

```
DSI for data_server.database received and processed
Resync Database Marker. DSI is now suspended. Resume
after database has been reloaded.
```

Replication Server が **init** マーカ付きのデータベース再同期指示を受信して処理すると、DSI コネクションはサスペンドします。

10. レプリケート・データベースに外部ソースからのデータ・ダンプを適用します。
11. レプリケート・データベースにダンプを適用したら、レプリケート・データベースで DSI をレジュームして Replication Server がプライマリ・データベースからのトランザクションを適用できるようにします。

```
resume connection to data_server.database
```

ウォーム・スタンバイ・アプリケーションのアクティブ・データベースとスタンバイ・データベースの再同期

ウォーム・スタンバイ・ペアが単一プライマリ・データベースのレプリケート・サイトになっているときに、ウォーム・スタンバイ環境でアクティブ・データベースとスタンバイ・データベースを再同期します。

このシナリオでは、レプリケート・サイトはウォーム・スタンバイ・ペアです。ウォーム・スタンバイ・ペアは、アクティブ・データベースとスタンバイ・データベースから構成され、1 つの論理データベースとして機能します。

```
Primary ---> Replication ---> Replicate logical database

database     Server           [Active+Standby warm standby

                               pair]
```

この再同期化シナリオ手順は、ウォーム・スタンバイ・ペアのアクティブなレプリケート・データベースをプライマリ・データベースからのダンプに再同期するプロセスと、その次のウォーム・スタンバイ・ペアのスタンバイ・レプリケート・データベースをアクティブ・データベースからのダンプまたはプライマリ・データベースからの既存のダンプに再同期するプロセスの 2 段階の再同期プロセスです。

1. プライマリ・データベースの RepAgent とウォーム・スタンバイ・アクティブ・データベースの RepAgent による複写プロセスを停止します。

   Adaptive Server で次のコマンドを実行します。

   ```
   sp_stop_rep_agent database
   ```
2. アクティブ・データベースとスタンバイ・データベースとの Replication Server DSI コネクションをサスペンドします。

   ```
   suspend connection to dataserver.database
   ```
3. アクティブ・データベースとスタンバイ・データベースのアウトバウンド・キューからデータを削除し、プライマリ・データベースの RepAgent からの再同期マーカを待機するように Replication Server に指示します。

```
resume connection to data_server.database skip to
resync marker
```

4. 再同期モードでプライマリ・データベースの RepAgent を開始し、再同期マーカを Replication Server に送信します。
   - トランケーション・ポイントが元の位置から移動していなければ、Adaptive Server で次のコマンドを実行します。

     ```
     sp_start_rep_agent database, 'resync'
     ```
   - トランケーション・ポイントが元の位置から移動していれば、Adaptive Server で次のコマンドを実行します。

     ```
     sp_start_rep_agent database, 'resync purge'
     ```
5. Replication Server システム・ログで次のメッセージを検索して、アクティブ・データベースの DSI がプライマリ・データベース RepAgent から再同期マーカを受信して受け入れていることを確認します。

   ```
   DSI for data_server.database received and processed
   Resync Database Marker. Waiting for Dump Marker.
   ```
6. プライマリ・データベース・コンテンツのダンプを取得します。詳細については、『Adaptive Server Enterprise リファレンス・マニュアル：コマンド』の「コマンド」の「**dump database**」を参照してください。Adaptive Server は自動的にダンプ・データベース・マーカを生成します。
7. ダンプを適用する前に RepAgent の設定を取得します。

   注意： Adaptive Server はデータベース内に RepAgent が使用する接続設定およびその他の設定を保存します。別のデータベースから取得したダンプをプライマリ・データベースにロードすると、RepAgent はその設定を失い、設定はダンプを取得した元のデータベースの設定に一致するように変更されます。
8. Replication Server システム・ログでアクティブ・データベースから次のメッセージを検索して、アクティブ・データベースの Replication Server DSI がダンプ・データベース・マーカを処理していることを確認します。

   ```
   DSI for data_server.database received and processed
   Dump Marker. DSI is now suspended. Resume after database has been
   reloaded.
   ```
9. プライマリ・データベースのダンプをアクティブ・データベースに適用します。詳細については、『Adaptive Server Enterprise リファレンス・マニュアル：コマンド』の「コマンド」の「**load database**」を参照してください。

   ダンプを適用したら、RepAgent の設定をダンプを適用する前の設定にリセットします。
10. プライマリ・データベースのトランザクション・ログにレプリケート・データベース・テーブルに影響するオペレーションが含まれていないことを確認するために、プライマリ Adaptive Server データベースで次のコマンドを実行します。

```
rs_update_lastcommit 0, 0, 0, ""
go 100
```

11. アクティブ・データベースのトランザクション・ログの最後までトランケーション・ポイントを移動します。Adaptive Server で次のコマンドを実行します。

```
dbcc settrunc('ltm', 'end')
go
```

12. **init** 命令を付けて再同期モードで RepAgent を開始します。Adaptive Server で次のコマンドを実行します。

```
sp_start_rep_agent database, 'resync init'
```

13. Replication Server システム・ログで次のメッセージを検索して、スタンバイ・データベースの DSI がアクティブ・データベース RepAgent から再同期マーカを受信して受け入れていることを確認します。

```
DSI for data_server.database received and processed
Resync Database Marker. DSI is now suspended. Resume
after database has been reloaded.
```

    Replication Server が **init** マーカ付きのデータベース再同期指示を受信して処理すると、DSI コネクションはサスペンドします。

14. アクティブ・データベースのコンテンツのダンプを取得し、スタンバイ・データベースにダンプを適用します。ダンプにデータベース設定情報が含まれない場合は、手順 6 からプライマリ・データベースのダンプを適用することもできます。
15. アクティブ・データベースとスタンバイ・データベースの DSI をレジュームします。

```
resume connection to data_server.database
```

非同期プロシージャ

非同期ストアド・プロシージャと、テーブル複写定義に対応するストアド・プロシージャを複写する方法について説明します。この方法は、必要とするアプリケーションで使用できます。

ファンクション複写定義に対応する複写ストアド・プロシージャの詳細については、『Replication Server 管理ガイド 第 1 巻』の「複写ファンクションの管理」を参照してください。

複写ストアド・プロシージャに関する複写システム設計の問題の詳細については、『Replication Server デザイン・ガイド』を参照してください。

非同期プロシージャ配信の概要

非同期プロシージャ配信を使用すると、プライマリ・データベースまたはレプリケート・データベースでの複写用に指定された SQL ストアド・プロシージャを実行できます。

これらの SQL ストアド・プロシージャは、**sp_setreplicate** または **sp_setrepproc** システム・プロシージャを使って複写するようマーク付けされているため、複写ストアド・プロシージャとも呼ばれます。

分散アプリケーションの要件を満たすため、Replication Server は、適用ストアド・プロシージャと要求ストアド・プロシージャの 2 種類の非同期ストアド・プロシージャ配信を提供しています。

Adaptive Server による複写ストアド・プロシージャのロギング

複写ストアド・プロシージャの実行ログを記録するデータベースを Adaptive Server が決定する方法について説明します。

ストアド・プロシージャのログは、プロシージャ内のトランザクションが開始されたデータベースに記録されます。

- ユーザが明示的にトランザクションを開始しない場合、Adaptive Server は、ストアド・プロシージャが実行される前にユーザの現在のデータベース内でトランザクションを開始します。
- ユーザがデータベース内でトランザクションを開始し、他のデータベース内で複写ストアド・プロシージャを実行する場合、その実行ログはユーザがトランザクションを起動したデータベースに記録されます。

テーブル形式の複写ストアド・プロシージャ (**sp_setreplicate***proc_name*, **'true'** または **sp_setrepproc***proc_name*, **'table' で複写するようにマーク付けされている)** の実行ログ

が 1 つのデータベースのログに記録され、別のデータベースの複写テーブルを変更すると、そのテーブルの変更とプロシージャの実行のログは、それぞれ異なるデータベースに記録されます。このため、ストアド・プロシージャの実行の結果は 2 回複写できます。1 回目はストアド・プロシージャの実行自体が複写され、2 回目は別のデータベースに記録されているテーブルの変更が複写されます。

複写ストアド・プロシージャのロギングの制限
複写される Adaptive Server ストアド・プロシージャには、`text` データ型、`unitext` データ型、または `image` データ型のパラメータを指定できません。

『Adaptive Server Enterprise リファレンス・マニュアル』を参照してください。

混合モード・トランザクション
単一のトランザクションが 1 つ以上の要求ストアド・プロシージャを呼び出すことを混合モード・トランザクションと呼びます。混合モード・トランザクションは、適用ストアド・プロシージャを実行したり、データ修正言語を含んだりします。Replication Server は、すべての他のオペレーションを終了した後に、その要求ストアド・プロシージャを処理します。

すべての要求オペレーションは、独立した単一のトランザクション内でまとめて処理されます。これは、単一の Replication Server がプライマリ・データとレプリケート・データの両方を管理する場合に発生します。

適用ストアド・プロシージャ

Replication Server がプライマリ・データベースからレプリケート・データベースに配信する複写ストアド・プロシージャを、適用ストアド・プロシージャと呼びます。

プライマリ・データで最初に実行されたトランザクションをレプリケート・データベースに複写するには、適用ストアド・プロシージャ配信を使用します。データの変更はプライマリ・データベースで適用されてから、そのデータの複写定義にサブスクリプションを作成しているレプリケート・データベースへ分配されます。Replication Server は、メンテナンス・ユーザとしてレプリケート・データベースの複写ストアド・プロシージャを実行します。これは通常のデータ複写と同じです。

適用ストアド・プロシージャを使用すると、パフォーマンスが大幅に向上します。たとえば、組織内で非常に多くのローを変更する場合、ローを 1 つずつ複写するのではなく、複数のローを変更する適用ストアド・プロシージャを作成できます。また、適用ストアド・プロシージャを使用して、通常のサブスクリプションでは表現が困難なデータ・セットの変更を複写することもできます。詳細については、『Replication Server デザイン・ガイド』を参照してください。

適用ストアド・プロシージャは、そのストアド・プロシージャの最初の文でテーブルの更新を行うことによって設定します。送信先データベースでは、その更新されたローの更新前イメージと更新後イメージに対するサブスクリプションを作成している必要があります。適用ストアド・プロシージャは、1 つの複写テーブル内の 1 つのローだけを更新します。Replication Server は、このストアド・プロシージャで更新された最初のローを使用して、そのプロシージャに対するユーザ定義ファンクションの送信先を決定します。

このような適用ストアド・プロシージャの設定規則に従っていない場合、Replication Server はストアド・プロシージャをレプリケート・データベースに分配できません。適用ストアド・プロシージャの配信に失敗した場合、いくつかの警告状態と、それに対して Replication Server が行うアクションがあります。

参照：

- 警告状態 (452 ページ)

要求ストアド・プロシージャ

Replication Server がレプリケート・データベースからプライマリ・データベースに配信する複写ストアド・プロシージャを、要求ストアド・プロシージャと呼びます。

要求ストアド・プロシージャを使用すると、レプリケート・データベースからプライマリ・データベースにトランザクションを配信できます。

たとえば、リモートのクライアント・アプリケーションが、プライマリ・データを変更する必要があるとします。この場合、リモートのアプリケーションは要求ストアド・プロシージャをローカルで実行してプライマリ・データを変更します。Replication Server は、プライマリ・データベースのストアド・プロシージャと同じ名前を持つストアド・プロシージャをレプリケート・データベースで実行して、この要求ストアド・プロシージャをプライマリ・データベースに配信します。プライマリ・データベースのストアド・プロシージャは、トランザクションによって変更されるプライマリ・データを更新します。

Replication Server は、レプリケート・データベースでそのストアド・プロシージャを実行したユーザとして、プライマリ・データベースで複写ストアド・プロシージャを実行します。これにより、認可されたユーザだけがプライマリ・データを変更できるということが保証されます。

アプリケーション内では、Replication Server はプライマリ・データベース内で変更されたデータの一部または全部を複写します。その変更は、Replication Server が管理するレプリケート・データベースに、データ・ロー (挿入、削除、または更新オペレーション) またはストアド・プロシージャとして関連データのサブスクリプションとともに送信されます。このメカニズムにより、プライマリ・データベー

スとレプリケート・データベースにトランザクションの結果が短時間で反映されます。

警告！ プライマリ・データベースでは要求ストアド・プロシージャを実行しないでください。実行すると、レプリケート Replication Server がプライマリ・データベースで同じプロシージャを実行してしまうというループが発生します。

要求ストアド・プロシージャを使用すると、すべての更新が確実にプライマリ・データベースで実行され、Replication Server の基本的なプライマリ・コピー・データ・モデルが維持されます。同時に、複写システムはネットワーク障害やトラフィック超過の影響を受けません。プライマリ・データベースの障害やレプリケート・データベースからプライマリ・データベースへのネットワーク障害が発生しても、Replication Server はフォールト・トレラントの状態を維持します。Replication Server は、障害の発生したコンポーネントがオンラインに復帰するまで、配信されていない要求ストアド・プロシージャ呼び出しをキューイングします。コンポーネントのサービスが再開されると、Replication Server は配信を行います。

Replication Server の保証された要求ストアド・プロシージャ配信機能を使用すると、最新のすべての変更が反映されたデータの完全なコピーが 1 つ存在することによる、あらゆるメリットを活用できます。さらに、Replication Server では、レプリケート・データベースのアプリケーションをプライマリ・データベースから分離することで、可用性やパフォーマンスが向上します。

非同期プロシージャ配信に関する複写システム設計の問題の詳細については、『Replication Server デザイン・ガイド』を参照してください。

非同期ストアド・プロシージャを実装するための前提条件

適用ストアド・プロシージャまたは要求ストアド・プロシージャを実装するために必要な前提条件がいくつかあります。

- アプリケーション要件を満たすには、どのように非同期プロシージャ配信を使用すればよいかを理解してください。『Replication Server デザイン・ガイド』を参照してください。
- データベースにプライマリ・データがない場合 (要求ファンクションを使用する場合など) でも、ストアド・プロシージャ用 RepAgent を設定してください。詳細については、使用しているプラットフォーム用の『Replication Server インストール・ガイド』と『Replication Server 設定ガイド』を参照してください。
- Replication Server がデフォルトのファンクション文字列を生成しない場合には、ファンクション文字列クラスのユーザ定義ファンクションのファンクション文字列を作成してください。**alter function string** コマンドを使って、デフォルト

のファンクション文字列を、アプリケーションで必要なアクションを実行するファンクション文字列に置き換えることができます。

注意： デフォルトで生成されたファンクション文字列が提供されているファンクション文字列クラスの場合、Replication Server は、ストアド・プロシージャを実行するデフォルトのファンクション文字列を、ユーザ定義ファンクションと同じ名前で作成します。この項の作業では、Replication Server が適用ストアド・プロシージャまたは要求ストアド・プロシージャをこのようなクラスで処理すると想定します。その他のクラスの場合、ユーザ定義ファンクション文字列に対するファンクション文字列を作成する必要があります。

参照：
- ファンクション文字列とファンクション文字列クラス (38 ページ)

適用ストアド・プロシージャの実装

適用ストアド・プロシージャを実装する手順について説明します。

前提条件
非同期ストアド・プロシージャの前提条件を満たしていることを確認します。

手順

システム情報について RSSD に問い合わせるために使用するストアド・プロシージャについては、『Replication Server リファレンス・マニュアル』の「RSSD ストアド・プロシージャ」を参照してください。

1. レプリケート・テーブルを格納するレプリケート・データベースを設定します。レプリケート・テーブルは、プライマリ・テーブルの複写定義に一致する場合としない場合があります。
2. 必要に応じて、プライマリ Replication Server からプライマリ・テーブルの複写定義に対するサブスクリプションを持つレプリケート Replication Server へのルートを設定します。

   『Replication Server 管理ガイド 第 1 巻』の「ルートの管理」を参照してください。
3. 修正対象のテーブルを識別するプライマリ Replication Server に対して複写定義を特定または作成します。

   『Replication Server 管理ガイド 第 1 巻』の「複写テーブルの管理」を参照してください。

4. プライマリ・データベースで、**sp_setreplicate** システム・プロシージャまたは **sp_setreptable** システム・プロシージャを使用して、テーブルを複写するようマーク付けします。

   たとえば、`employee` テーブルの場合、次のコマンドのいずれかを入力します。

   - `sp_setreplicate employee, 'true'`

     ストアド・プロシージャやテーブルを指定する場合や、複写を実行する場合は、ガイドラインに従います。
   - `sp_setreptable employee, 'true'`

     sp_setreptable では、一重引用符はオプションです。『Replication Server 管理ガイド 第 1 巻』の「複写テーブルの管理」の「複写対象テーブルへのマーク付け」の「sp_setreptable システム・プロシージャの使用」を参照してください。
5. プライマリ・データベースでストアド・プロシージャを作成します。

   ストアド・プロシージャの最初の文には、プライマリ・テーブルの最初のローに対する update コマンドを指定します。次に例を示します。

```
create proc upd_emp
 @emp_id int, @salary float
 as
 update employee
 set salary = salary * @salary
 where emp_id = @emp_id
```

警告！ ストアド・プロシージャの最初の文が **update** 以外のオペレーションである場合、Replication Server は、レプリケート・データベースにストアド・プロシージャを分配できません。警告状態を確認してください。ストアド・プロシージャには、**dump transaction** コマンドや **dump database** コマンドを指定しないでください。ストアド・プロシージャが文レベル・エラーのあるコマンドを格納すると、レプリケート DSI にエラーが発生します。エラー・アクションによっては、DSI が停止する可能性があります。

6. プライマリ・データベースで、**sp_setreplicate** システム・プロシージャまたは **sp_setrepproc** システム・プロシージャを使用して、ストアド・プロシージャを複写するようマーク付けします。

   たとえば、次のいずれかのコマンドを入力します。

   - `sp_setreplicate upd_emp, 'true'`

     ストアド・プロシージャやテーブルを指定する場合や、複写を実行する場合は、ガイドラインに従います。
   - `sp_setrepproc upd_emp, 'table'`

『Replication Server 管理ガイド 第 1 巻』の「複写ファンクションの管理」の「ストアド・プロシージャへの複写のマーク付け」を参照してください。

7. レプリケート Replication Server で、プライマリ・データベースのストアド・プロシージャが更新するテーブルの複写定義に対してサブスクリプションを作成します。

   『Replication Server 管理ガイド 第 1 巻』の「サブスクリプションの管理」を参照してください。

   警告！ レプリケート・データベースでは、更新されたローの更新前イメージと更新後イメージに対してサブスクリプションを作成してください。レプリケート・データベースでこれらのサブスクリプションを作成していない場合は、Replication Server はレプリケート・データベースにストアド・プロシージャを分配しません。

8. プライマリ・データベースを対象とするストアド・プロシージャと同じ名前とパラメータを指定して、レプリケート・データベースを対象とするストアド・プロシージャを作成します。ただし、プロシージャは複写するようにマーク付けしないてください。

   次に例を示します。

   ```
   create proc upd_emp
    @emp_id int, @salary float
    as
    update employee
    set salary = salary * @salary
    where emp_id = @emp_id
   ```

9. ストアド・プロシージャに対する **execute** パーミッションをメンテナンス・ユーザに付与します。

   次に例を示します。

   ```
   grant execute on upd_emp to maint_user
   ```

10. プライマリ Replication Server で、更新するテーブルに対してストアド・プロシージャと同名のユーザ定義ファンクションを作成します。

    次に例を示します。

    ```
    create function employee_rep.upd_emp
     (@emp_id int, @salary float)
    ```

    1 つのテーブルのすべての複写定義に共有されるユーザ定義ファンクションは 1 つだけです。これらの複写定義の任意の名前を指定できます。

11. すべての手順で設定した Replication Server とデータベース・オブジェクトがすべて正しいロケーションにあることを確認します。

参照：

- 非同期ストアド・プロシージャを実装するための前提条件 (448 ページ)
- 複写用ストアド・プロシージャとテーブルの指定 (456 ページ)

警告状態

Replication Server の警告状態は、レプリケート・データベースで適用ストアド・プロシージャが配信されない場合に発生します。

適用ストアド・プロシージャの最初の文が更新以外のオペレーションである場合、またはレプリケート・データベースがサブスクリプションを作成していない場合、Replication Server は、適用ストアド・プロシージャをレプリケート・データベースへ配信できません。代わりに、警告が出されます。

Replication Server のアクションは、次の項目に基づいて決定されます。

- プライマリ・データベースの適用ストアド・プロシージャで指定された最初のオペレーション (更新を除く)。
- レプリケート・データベースのサブスクリプションにローの修正があるかどうか。また、その修正がサブスクリプションの更新前イメージまたは更新後イメージと一致するかどうか。

警告状態と Replication Server のアクション

- 条件：最初のローのオペレーションは挿入オペレーションである。
  アクション：Replication Server は適用ストアド・プロシージャではなく挿入オペレーションを分配します。
- 条件：最初のローのオペレーションは削除オペレーションである。
  アクション：Replication Server は適用ストアド・プロシージャではなく削除オペレーションを分配します。
- 条件：レプリケート Replication Server のサブスクリプションは、修正されたローの更新前イメージに一致するが、更新後イメージには一致しない。
  アクション：Replication Server は、ローの修正後のイメージではなく更新前イメージへのサブスクリプションを使って、ローに対する削除オペレーション (**rs_delete** システム・ファンクション) をレプリケート・データベースに分配します。
  例：`C1` というカラムの値が 1 である `T1` テーブルがあるとします。レプリケート・データベースには、`C1` = 1 である `T1` テーブルの複写定義へのサブスクリプションがあります。
  対応するストアド・プロシージャに、パラメータ 1 (更新前イメージ) と 2 (更新後イメージ) を指定して実行すると、レプリケート・データベースは値が 2 である更新後イメージにサブスクリプションを作成しません。このため、Replication Server はレプリケート・データベースに削除オペレーションを分配します。
- 条件：レプリケート Replication Server のサブスクリプションは、修正されたローの更新後イメージに一致するが、更新前イメージには一致しない。
  アクション：Replication Server は、ローの修正前のイメージではなく更新後イメージへのサブスクリプションを使って、ローに対する挿入オペレーション

(**rs_insert** システム・ファンクション) をレプリケート・データベースに分配します。
例：C1 というカラムの値が 1 である T1 テーブルがあるとします。レプリケート・データベースには、C1 = 2 である T1 テーブルの複写定義へのサブスクリプションがあります。
対応するストアド・プロシージャに、パラメータ 1 (更新前イメージ) と 2 (更新後イメージ) を指定して実行すると、レプリケート・データベースは値が 1 である更新前イメージにサブスクリプションを作成しません。このため、Replication Server はレプリケート・データベースに挿入オペレーションを分配します。

- 条件：レプリケート Replication Server のサブスクリプションは、修正されたローの更新前イメージとも更新後イメージとも一致しない。
  アクション：Replication Server は、オペレーションもストアド・プロシージャもレプリケート・データベースに分配しません。
  例：C1 というカラムの値が 1 である T1 テーブルがあるとします。レプリケート・データベースには、C1 > 2 である T1 テーブルの複写定義へのサブスクリプションがあります。
  対応するストアド・プロシージャに、パラメータ 1 (更新前イメージ) と 2 (更新後イメージ) を指定して実行すると、レプリケート Replication Server は値が 1 である更新前イメージまたは値が 2 である更新後イメージにサブスクリプションを作成しません。このため、Replication Server はレプリケート・データベースへの分配を実行しません。

要求ストアド・プロシージャの実装

要求ストアド・プロシージャを実装する手順について説明します。

前提条件
非同期ストアド・プロシージャの前提条件を満たしていることを確認します。

手順

システム情報について RSSD に問い合わせるために使用するストアド・プロシージャについては、『Replication Server リファレンス・マニュアル』の「RSSD ストアド・プロシージャ」を参照してください。

1. 必要に応じて、レプリケート Replication Server からプライマリ Replication Server (データを更新する場所) へのルートと、プライマリ Replication Server からレプリケート Replication Server (更新を送信する先) へのルートを設定します。

『Replication Server 管理ガイド 第 1 巻』の「ルートの管理」を参照してください。

2. プライマリ・データ・サーバにレプリケート Replication Server のユーザがログインするためのログイン名とパスワードを作成します。

   『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。

3. レプリケート Replication Server で、このユーザがプライマリ Replication Server においてストアド・プロシージャを実行するために必要なパーミッションを作成します。

   『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。

4. プライマリ Replication Server で、修正対象のテーブルを識別する複写定義を特定または作成します。

   複写定義を作成する方法については、『Replication Server 管理ガイド 第 1 巻』の「複写テーブルの管理」を参照してください。

   レプリケート Replication Server には、この複写定義に対するサブスクリプションがあります。

5. レプリケート・データベースで、更新を実行しないストアド・プロシージャを作成します。

   次に例を示します。

   ```
   create proc upd_emp
    @emp_id int, @salary float
    as
    print "Transaction accepted."
   ```

   ストアド・プロシージャに別のレプリケート・データベースのストアド・プロシージャと同じ名前を設定する場合は、ガイドラインに従ってユニークでないユーザ定義ファンクション名を指定してください。

6. レプリケート・データベースで **sp_setreplicate** システム・プロシージャまたは **sp_setrepproc** システム・プロシージャを使用して、ストアド・プロシージャを複写するようマーク付けします。

   たとえば、次のいずれかのコマンドを入力します。

   ```
   sp_setreplicate upd_emp, 'true'
   ```

   ストアド・プロシージャやテーブルを指定する場合や、複写を実行する場合は、ガイドラインに従います。

   または、

   ```
   sp_setrepproc upd_emp, 'table'
   ```

『Replication Server 管理ガイド 第 1 巻』の「複写ファンクションの管理」の「ストアド・プロシージャへの複写のマーク付け」を参照してください。

7. レプリケート・データベースを対象とするストアド・プロシージャと同じ名前を指定して、プライマリ・データベースを対象とするストアド・プロシージャを作成します。ただし、プロシージャには複写するようにマーク付けしないでください。このストアド・プロシージャは、プライマリ・テーブルを修正します。

   次に例を示します。

   ```
   create proc upd_emp
    @emp_id int, @salary float
    as
    update employee
    set salary = salary * @salary
    where emp_id = @emp_id
   ```

   注意： ファンクションのファンクション文字列を変更して異なる名前でストアド・プロシージャを実行することで、プライマリ・データベースとレプリケート・データベースでプロシージャの名前を変えることができます。

   ファンクションを別のストアド・プロシージャ名にマッピングできます。

8. このストアド・プロシージャを実行するレプリケート Replication Server ユーザに、このストアド・プロシージャに対するパーミッションを付与します。

   次に例を示します。

   ```
   grant all on upd_emp to public
   ```

9. プライマリ Replication Server で、更新するテーブルに対してストアド・プロシージャと同名のユーザ定義ファンクションを作成します。

   次に例を示します。

   ```
   create function employee_rep.upd_emp
    (@emp_id int, @salary float)
   ```

10. すべての手順で設定した Replication Server とデータベース・オブジェクトがすべて正しいロケーションにあることを確認します。

参照：

- 非同期ストアド・プロシージャを実装するための前提条件 (448 ページ)
- ユニークでないユーザ定義ファンクション名の指定 (461 ページ)
- 複写用ストアド・プロシージャとテーブルの指定 (456 ページ)
- ファンクションを別のストアド・プロシージャ名にマッピング (459 ページ)

複写用ストアド・プロシージャとテーブルの指定

Adaptive Server で **sp_setreplicate** システム・プロシージャを使用すると、データベース・テーブルとストアド・プロシージャを複写するようマーク付けできます。

また、**sp_setreptable** システム・プロシージャを使用してテーブルを複写するようマーク付けしたり、**sp_setrepproc** システム・プロシージャを使用してストアド・プロシージャを複写するようマーク付けしたりできます。この 2 つのシステム・プロシージャは **sp_setreplicate** システム・プロシージャの機能を拡張し、その代わりに使用できます。

sp_setreplicate システム・プロシージャの構文は次のとおりです。

```
sp_setreplicate [object_name [, {' true' | 'false' ]]
```

object_name には、テーブル名かストアド・プロシージャ名を指定できます。

"true" および "false" パラメータを指定すると、指定したオブジェクトの複写ステータスを変更できます。一重引用符はオプションです。

- **sp_setreplicate** にパラメータを指定しないと、データベース内のすべての複写オブジェクトがリストされます。
- **sp_setreplicate** にオブジェクト名だけを指定すると、そのオブジェクトの複写ステータスをチェックできます。このオブジェクトに対して複写を実行できる場合、Adaptive Server は **'true'** をレポートし、実行できない場合は **'false'** をレポートします。
- **sp_setreplicate** にオブジェクト名と **'true'** または **'false'** を指定すると、オブジェクトの複写を有効または無効にできます。**sp_setreplicate** を使用してオブジェクトの複写ステータスを変更するには、Adaptive Server のシステム管理者またはデータベース所有者である必要があります。

警告！ 複写ストアド・プロシージャでは、そのプロシージャを実行したデータベース内のデータのみを修正します。プロシージャが別のデータベース内のデータを修正する場合、Replication Server はその更新データとストアド・プロシージャを複写します。

ユーザ定義ファンクションの管理

ユーザ定義ファンクションを管理するコマンドについて説明します。

ユーザ定義ファンクションのファンクション文字列を変更すると、データベースのオペレーションをカスタマイズできます。また、ファンクション関連情報を表示できます。

コマンドを使用するために必要なパーミッションのリストについては、『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。

参照：

- データベース・オペレーションのカスタマイズ (15 ページ)

ユーザ定義ファンクションの作成

Replication Server に複写ストアド・プロシージャを登録するには、**create function** コマンドを使用します。

ストアド・プロシージャが実行されると、Replication Server はストアド・プロシージャを複写定義にマップします。複写定義には、ストアド・プロシージャと名前が一致するユーザ定義ファンクション名があります。

Replication Server は、複写定義のプライマリである Replication Server にファンクションを配信します。複写定義を所有する送信先 Replication Server は、ファンクションを受信すると、ストアド・プロシージャ・パラメータをユーザ定義ファンクションのコマンドにマップします。

create function コマンドの構文は次のとおりです。

```
create function replication_definition.function
([@parameter datatype [, @parameter datatype]...])
```

replication_definition には、既存の複写定義を指定する必要があります。

このコマンドは、次のガイドラインに従って使用してください。

- このコマンドは、複写定義を作成した Replication Server で実行してください。
- 予約済みのシステム・ファンクション名を使用しないでください。
- パラメータはカッコで囲んでください。パラメータを指定しないでファンクションを定義する場合でも、カッコを挿入してください。
- デフォルトで生成されたファンクション文字列が提供されているファンクション文字列クラスを使用しない場合は、ユーザ定義ファンクションを作成してから **create function string** コマンドを使用してファンクション文字列を追加してください。

次の例では、**Stock_receipt** というユーザ定義ファンクションを作成します。このファンクションを、`Items_rd` 複写定義に関連付けます。

```
create function Items_rd.Stock_receipt
 (@Location int, @Recpt_num int,
 @Item_no char(15), @Qty_recd int)
```

複写ストアド・プロシージャを実行すると、Replication Server によってそれが配信されます。

参照：
- ファンクション文字列の作成 (45 ページ)
- システム・ファンクションの概要 (19 ページ)

ユーザ定義ファンクションへのパラメータの追加

alter function コマンドを使用して、複写ストアド・プロシージャに追加する新しいパラメータに関する情報を Replication Server に通知します。

1. プライマリ・データ・サーバまたはレプリケート・データ・サーバでストアド・プロシージャを変更し、新しいパラメータにデフォルト値を指定します。
2. 予防のため、システムをクワイス状態にします。更新中にファンクションを変更すると、予期しない結果が発生することがあります。

   『Replication Server 管理ガイド 第 1 巻』の「複写システムの管理」の「Replication Server のクワイス」を参照してください。
3. **alter function** コマンドを使用してファンクション文字列を変更します。
4. デフォルトで生成されたファンクション文字列が提供されているファンクション文字列クラスを使用しない場合は、ファンクション文字列を変更して新しいパラメータを使用してください。

alter function コマンドの構文は次のとおりです。

```
alter function replication_definition.function
add parameters @parameter datatype
[, @parameter datatype]...
```

replication_definition はファンクションの複写定義名です。1 つのファンクションには最大 255 のパラメータを指定できます。

次の例では、`Volume` という `int` パラメータを `Tokyo_quotes` 複写定義の **New_issue** ファンクションに追加します。

```
alter function Tokyo_quotes.New_issue
 add parameters @Volume int
```

参照：
- ファンクション文字列の変更 (49 ページ)

ユーザ定義ファンクションの削除

ユーザ定義ファンクションを削除するには、**drop function** コマンドを使用します。

このコマンドを実行すると、ファンクション名と、そのファンクションのために作成されているすべてのファンクション文字列が削除されます。システム・ファンクションは削除できません。

ユーザ定義ファンクションを削除する前に、次の手順を実行してください。

1. **drop procedure** Adaptive Server コマンドを使用して、プライマリ・データベースでストアド・プロシージャを削除します。
   必要に応じて、**sp_setreplicate** システム・プロシージャまたは **sp_setrepproc** システム・プロシージャを使用して **'false'** を指定し、ストアド・プロシージャの複写を無効にします。
   ストアド・プロシージャやテーブルを指定する場合や、複写を実行する場合は、ガイドラインに従います。**sp_setrepproc** の使用の詳細については、『Replication Server 管理ガイド 第 1 巻』の「複写ファンクションの管理」の「ストアド・プロシージャへの複写のマーク付け」を参照してください。
2. 問題の発生を防ぐため、システムをクワイス状態にしてから **drop function** コマンドを実行します。更新中にファンクションを削除すると、予期しない結果が発生することがあります。
   『Replication Server 管理ガイド 第 1 巻』の「複写システムの管理」の「Replication Server のクワイス」を参照してください。

drop function コマンドの構文は次のとおりです。

```
drop function replication_definition.function
```

このコマンドを複写定義の作成場所である Replication Server で実行します。

次のコマンドを実行すると、前の項で作成した **Stock_receipt** ユーザ定義ファンクションが削除されます。

```
drop function Items_rd.Stock_receipt
```

参照：

- 複写用ストアド・プロシージャとテーブルの指定 (456 ページ)

ファンクションを別のストアド・プロシージャ名にマッピング

ユーザ定義ファンクションを別のストアド・プロシージャ名にマッピングする方法について説明します。

デフォルトで生成されたファンクション文字列が提供されているファンクション文字列クラスを使用するデータベースでユーザ定義ファンクションを作成すると、Replication Server はデフォルトのファンクション文字列を生成します。デフォルトで生成されたファンクション文字列によって、ユーザ定義ファンクションと同じ名前とパラメータのストアド・プロシージャが実行されます。

たとえば、デフォルトのファンクション文字列を使用している場合、ユーザ定義ファンクションと同じ名前とパラメータを指定してストアド・プロシージャをプライマリ・データベースで作成することによって、レプリケート・データベースで実行するように要求ストアド・プロシージャを設定できます。

ユーザ定義ファンクションを異なる名前のストアド・プロシージャにマップするには、**alter function string** コマンドを使用し、別名でストアド・プロシージャを実

行してストアド・プロシージャを配信するように Replication Server を設定します。これは、ファンクション文字列をカスタマイズできるファンクション文字列クラスで行うこともできます。

例

次の例は、ユーザ定義ファンクションを異なるストアド・プロシージャ名にマッピングする方法を示します。

1. プライマリ Adaptive Server に **upd_sales** ストアド・プロシージャがあり、このストアド・プロシージャが Adaptive Server の `sales` テーブルに対して更新を実行する場合のコマンドは次のとおりです。

```
create proc upd_sales
 @stor_id varchar(10),
 @ord_num varchar(10),
 @date datetime
 as
64 update sales set date = @date
 where stor_id = @stor_id
 and ord_num = @ord_num
```

2. Replication Server に **upd_sales** ストアド・プロシージャを登録するには、次のファンクションを作成します。ファンクション名には、`sales` テーブルの `sales_def` 複写定義と **upd_sales** 複写ストアド・プロシージャを指定します。

```
create function sales_def.upd_sales
 (@stor_id varchar(10), @date datetime)
```

3. レプリケート Adaptive Server の場合、ストアド・プロシージャ **upd_sales** の何も実行しないバージョンが同じ名前で作成されます。

```
create proc upd_sales
 @stor_id varchar(10),
 @ord_num varchar(10),
 @date datetime
 as
 print "Attempting to Update Sales Table"
 print "Processing Update Asynchronously"
```

4. **upd_sales** ではなく **real_update** という名前を持つ **upd_sales** ストアド・プロシージャを実行するには、次の手順に従います。
   - デフォルトで生成されたファンクション文字列を変更します。

```
alter function string sales_def.upd_sales
 for rs_sqlserver_function_class
 output rpc
 'execute real_update
 @stor_id = ?stor_id!param?,
 @date = ?date!param?'
```

   - プライマリ・データベース内に、**real_update** という名前のストアド・プロシージャを作成します。このストアド・プロシージャには 2 つのパラメータを指定できます。

ユニークでないユーザ定義ファンクション名の指定

ユーザ定義ファンクションが定義されている個々の複写定義を Replication Server が特定できるように、ユーザ定義ファンクション名は複写システム全体内でユニークなものにします。

同じプライマリ・テーブルに対して複数の複写定義を作成した場合、ユーザ定義ファンクションはそのテーブルの複写定義すべてに対して 1 つだけになります。

ユーザ定義ファンクション名がユニークでない場合、ストアド・プロシージャの最初のパラメータに *@rs_repdef* を指定して、ストアド・プロシージャが実行される際、このパラメータで複写定義名を渡す必要があります。

ユーザ定義ファンクションを作成する **create function** コマンドには *@rs_repdef* パラメータを定義しないでください。Replication Agent は、複写定義名を抽出して、これを LTL コマンドで送信します。この規則は Adaptive Server の RepAgent に適用されますが、その他のデータ・サーバの Replication Agent ではサポートされない場合があります。

例

次の例では、ユーザ定義ファンクションがユニークでなく、次のストアド・プロシージャを実行するときに複写定義名が *@rs_repdef* パラメータに渡されます。

```
create proc upd_sales
 @rs_repdef varchar(255),
 @stor_id varchar(10),
 @date datetime
 as
 print "Attempting to Update Sales Table"
 print "Processing Update Asynchronously"
```

Sun Cluster 2.2 での高可用性

Sun Cluster 2.2 の高可用性 (HA) 用に Sybase Replication Server を設定するためのバックグラウンド情報と手順について説明します。

Sybase Replication for Sun Cluster HA の概要

Sybase Replication for Sun Cluster HA を使用する場合、いくつかの想定を行います。

想定は、次のとおりです。

- Sybase Replication Server についての知識があること。
- Sun Cluster HA についての知識があること。
- Sun Cluster HA 2.2 がインストールされた 2 ノード・クラスタ・ハードウェア・システムを持っていること。

参照マニュアル

- 『Sun Cluster 2.2 Software Planning and Installation Guide』
- 『Sun Cluster 2.2 System Administration Guide』
- Configuring Sybase Adaptive Server Enterprise 12.0 Server for High Availability:Sun Cluster HA (ホワイト・ペーパーを参照)

用語

Sybase Replication Server for Sun Cluster HA で使う用語について説明します。

使用される用語は、次のとおりです。

- クラスタ ― アプリケーション、システム・リソース、データをユーザに提供するために単一のエンティティとして稼働する複数のシステムまたはノードです。
- クラスタ・ノード ― Sun Cluster の一部となる物理マシンです。物理ホストともいいます。
- データ・サービス ― ネットワーク上でクライアント・サービスを提供し、ディスクベースのデータへの読み込みおよび書き込みアクセスを実装するアプリケーションです。データ・サービスの例としては、Replication Server や Adaptive Server Enterprise などがあります。

- ディスク・グループ ― HA 構成において、2 台のサーバ間を 1 単位として移動するマルチホスト・ディスクのグループです。
- フォールト・モニタ ― データ・サービスを検査するデーモンです。
- 高可用性 (HA) ― ダウン時間が非常に少ないことです。HA を提供するコンピュータ・システムは、通常、99.999% の可用性 (予定外のダウン時間が、年間約 5 分) を実現しています。
- 論理ホスト ― ディスク・グループ、論理ホスト名、論理 IP アドレスを含むリソースのグループです。論理ホストは、クラスタ・マシン上の物理ホスト (またはノード) に存在します (または物理ホストが論理ホストのマスタとなります)。論理ホストは、クラスタ上の物理ホスト間を 1 単位として移動できます。
- マスタ ― Ethernet アドレスにマップされた論理アドレスを持つディスク・グループへ排他的な読み込みアクセスと書き込みアクセスを行うノードです。論理ホストの現在のマスタが、論理ホストのデータ・サービスを実行します。
- マルチホスト・ディスク ― 複数ノードからアクセスできるように設定されたディスクです。
- フェールオーバ ― ノード障害またはデータ・サービス障害によってトリガされるイベントです。このイベントでは、論理ホストと論理ホスト上のデータ・サービスが別のノードに移動します。
- フェールバック ― 計画されたイベントです。このイベントでは論理ホストとそのデータ・サービスが元のホストに戻されます。

テクノロジの概要

Sun Cluster HA は、クラスタ マシンと自動データ サービス フェールオーバの高可用性サポートを数秒で提供するハードウェア ベースおよびソフトウェア ベースのソリューションです。これは、ハードウェアの冗長性、ソフトウェアのモニタリング、再起動機能を加えることによって実現されています。

Sun Cluster は、システム管理者が高可用性 (HA) 環境の設定、管理、トラブルシューティングを行うためのクラスタ管理ツールを備えています。

Sun Cluster 構成では、以下のようなシングルポイント障害が発生してもシステムは問題なく稼働し続けます。

- サーバ・ハードウェア障害
- ディスク・メディア障害
- ネットワーク・インタフェース障害
- サーバ OS 障害

上記のいずれかの障害が発生すると、HA ソフトウェアによって論理ホストが他のノードにフェールオーバされ、新しいノードの論理ホスト上でデータ・サービスが再起動されます。

Sybase Replication Server は、クラスタ・マシンの論理ホスト上にデータ・サービスとして実装されています。Replication Server の HA フォールト・モニタは、Replication Server を定期的に検査します。Replication Server が応答を停止した場合、フォールト・モニタはローカルで Replication Server を再起動しようとします。設定した時間内に Replication Server が再び停止した場合は、Replication Server をセカンド・ノードで再起動できるように、フォールト・モニタが論理ホストにフェールオーバします。

Replication Server クライアントからは、元の Replication Server が再起動したように見えます。他の物理マシンに移動したことをユーザが意識することはありません。Replication Server は、物理マシンではなく論理ホストに結び付けられています。

データ・サービスとして、Replication Server には、Sun Cluster でコールバック・メソッドとして登録されている一連のスクリプトが含まれています。Sun Cluster は、フェールオーバの各段階で次のメソッドを呼び出します。

- FM_STOP ― フェールオーバされるデータ・サービスのフォールト・モニタを停止します。
- STOP_NET ― データ・サービス自体を停止します。
- START_NET ― 新しいノードでデータ・サービスを起動します。
- FM_START ― 新しいノードでデータ・サービスのフォールト・モニタを起動します。

各 Replication Server は、**hareg** コマンドを使って、データ・サービスとして登録されます。複数の Replication Server をクラスタで実行している場合、それぞれを登録する必要があります。各データ・サービスには、独立したプロセスとしてのフォールト・モニタがあります。

注意： **hareg** コマンドの詳細については、Sun Cluster の該当マニュアルを参照してください。

Replication Server の高可用性の設定

Sun Cluster で Replication Server を高可用性 (HA) に設定するために必要な作業について説明します (2 ノード構成のクラスタ・マシンを想定)。

システムには、次のコンポーネントが必要です。

- CPU、メモリなどのリソースが同じように構成された、同種の Sun Enterprise サーバが 2 台必要です。サーバはクラスタ・インターコネクトで構成されている必要があります。これによって、クラスタの可用性、同期、および整合性が保たれます。
- システムには、マルチホスト・ディスクのセットが装備されている必要があります。マルチホスト・ディスクには、可用性の高い Replication Server 用のデー

タ (パーティション) を保管します。ノードがマルチホスト・ディスク上のデータにアクセスできるのは、そのノードがディスクの属する論理ホストの現在のマスタである場合のみです。

- システムには、自動フェールオーバ機能が付いた Sun Cluster HA ソフトウェアがインストールされている必要があります。マルチホスト・ディスクのパス名は、そのシステム内でユニークである必要があります。
- ディスク障害からのデータ保護のために、ディスク・ミラーリング (Sybase では提供していません) を使用する必要があります。
- 論理ホストを設定する必要があります。Replication Server は論理ホスト上で動作します。
- Replication Server 用の論理ホストのマルチホスト・ディスク・グループに、パーティションを作成するのに十分なディスク領域があることを確認します。また、論理ホストの潜在的なマスタに Replication Server 用の十分なメモリがあることも確認します。

HA に対応した Replication Server のインストール

Replication Server のインストール時は、お使いのプラットフォームの『Replication Server インストール・ガイド』に記載されている作業の他に、いくつかの作業が必要です。

1. Sybase ユーザとして、共有ディスクまたはローカル・ディスクのいずれかに Replication Server をロードします。

   共有ディスクの場合、リリース・ディレクトリに両方のマシンから同時にアクセスすることはできません。ローカル・ディスクの場合、両方のマシンでリリース・ディレクトリのパスが同一であることを確認します。パスが同一でない場合は、シンボリック・リンクを使ってパスが同一になるようにします。

   たとえば、リリース・ディレクトリが node1 のパス `/node1/repserver` と、node2 のパス `/node2/repserver` にある場合、両方のノードでそれらのパスを `/repserver` にリンクさせます。そうすることによって、*$SYBASE* 環境変数がシステム全体で同一になります。
2. Replication Server、RSSD サーバ、プライマリ／レプリケート・データ・サーバのエントリを、両マシンの `$SYBASE` ディレクトリにある interfaces ファイルに追加します。

   interface ファイルの Replication Server に論理ホスト名を使用します。

注意： interfaces ファイルの代わりに LDAP ディレクトリ・サービスを使用する場合は、Replication Server の設定ファイルの DIRECTORY セクションに複数のエントリを指定します。最初のエントリへの接続が失敗すると、ディレクトリ・コントロール・レイヤ (DCL: Directory Control Layer) から 2 番目のエントリへの接続が試みられます。2 番目以降も同様です。DIRECTORY セクション内

のすべてのエントリへの接続が失敗すると、Open Client/Server により、デフォルトの interfaces ファイルを使用しないで接続しようとします。

LDAP ディレクトリ・サービスの設定の詳細については、使用しているプラットフォーム用の『Replication Server 設定ガイド』を参照してください。

3. RSSD サーバを起動します。
4. 使用しているプラットフォーム用の『Replication Server インストール・ガイド』に従って、Replication Server を論理ホストで現在マスタであるノードにインストールします。次のことを確認してください。
   a) 環境変数 SYBASE、SYBASE_REP、SYBASE_OCS を設定します。

      たとえば、次のように入力します。

      ```
      setenv SYBASE /REPSERVER1210
      setenv SYBASE_REP REP-12_1
      setenv SYBASE_OCS OCS-12_0
      ```

      /REPSERVER1210 は、リリース・ディレクトリです。
   b) Replication Server 用の実行ディレクトリを選択します。このディレクトリには Replication Server の実行ファイル、設定ファイル、ログ・ファイルが格納されます。

      実行ディレクトリは両方のノードに存在し、両方のノードでパスがまったく同一である必要があります (必要であればパスをリンクできます)。
   c) Replication Server のパーティション用にマルチホスト・ディスクを選択します。
   d) **rs_init** コマンドを `run` ディレクトリから開始します。

      次のように入力します。

      ```
      cd RUN_DIRECTORY
      $SYBASE/$SYBASE_REP/install/rs_init
      ```
5. Replication Server が起動していることを確認します。
6. Sybase ユーザとして、実行ファイルと設定ファイルを同一パスのもう一方のノードにコピーします。セカンド・ノード上の実行ファイルを編集します。設定ファイルとログ・ファイルの正しいパスが必ず含まれるようにします。リンクが使用されている場合は、特に注意が必要です。

注意： 実行ファイル名は `RUN_repserver_name` にします。*repserver_name* は、Replication Server の名前です。設定ファイルとログ・ファイルの名前はユーザが定義できます。

Replication Server をデータ・サービスとしてインストールする

Replication Server をデータ・サービスとしてインストールするには、特別な作業をいくつか行う必要があります。

1. root として、`/opt/SUNWcluster/ha/repserver_name` ディレクトリを両方のクラスタ・ノードに作成します。*repserver_name* は、Replication Server の名前です。

   各 Replication Server 用に、サーバ名をパスに含む固有のディレクトリを作成する必要があります。以下に示すスクリプトを Replication Server のインストール・ディレクトリ `$SYBASE/$SYBASE_REP/sample/ha` から次のディレクトリにコピーします。

   `/opt/SUNWcluster/ha/repserver_name`

   どちらのクラスタ・ノードでも *repserver_name* は、Replication Server の名前です。

   ```
   repserver_start_net
   repserver_stop_net
   repserver_fm_start
   repserver_fm_stop
   repserver_fm
   repserver_shutdown
   repserver_notify_admin
   ```

   スクリプトがもう 1 つの Replication Server データ・サービスの一部として、ローカル・マシン上に存在する場合は、スクリプト・ディレクトリへのリンクとして、代わりに次のディレクトリを作成できます。

   `/opt/SUNWcluster/ha/repserver_name`

2. `/var/opt/repserver` ディレクトリが両方のノードに存在しない場合は、root としてこれを作成します。

3. root として、Sun Cluster にデータ・サービスとしてインストールする各 Replication Server 用の `/var/opt/repserver/repserver_name` ファイルを両方のノードに作成します。*repserver_name* は、Replication Server の名前です。

   このファイルには、ブランクを入れずに次の形式で 2 行だけを記述してください。また、このファイルは、root 以外からは読み取れないようにします。

   ```
   repserver:logicalHost:RunFile:releaseDir:SYBASE_OCS:SYBASE_REP
   probeCycle:probeTimeout:restartDelay:login/password
   ```

   構文の説明は次のとおりです。

   - *repserver* - Replication Server 名です。

- *logicalHost* ― Replication Server が動作する論理ホストです。
- *RunFile* ― 実行ファイルのフル・パスです。
- *releaseDir* ― $SYBASE インストール・ディレクトリです。
- *SYBASE_OCS* ― コネクティビティ・ライブラリがある $SYBASE サブディレクトリです。
- *SYBASE_REP* ― Replication Server がある $SYBASE サブディレクトリです。
- *probeCycle* – フォールト・モニタによる 2 つの検査が開始される周期を設定する秒数です。
- *probeTimeout* ― ここで設定した秒数が経過すると、実行中の Replication Server の検査がフォールト・モニタによってアボートされ、タイムアウト条件が設定されます。
- *restartDelay* – Replication Server を 2 つ再起動する場合の再起動間の最小秒数です。フォールト・モニタは、Replication Server が再起動されてから *restartDelay* に設定されている秒数に満たないうちに再起動が必要な状態を再び検出すると、もう一方のホストへの切り替えをトリガします。これにより、データベースの再起動によって解決されない問題が解決されます。
- *login/password* - フォールト・モニタから Replication Server に ping が発行されるときに使用されるログイン名とパスワードです。

Replication Server をデータ・サービスとしてインストールした後に、検査用の *probeCycle*、*probeTimeout*、*restartDelay*、login/password を変更するには、SIGINT(2) をモニタ プロセス (repserver_fm) に送信し、メモリをリフレッシュします。

```
kill -2 monitor_process_id
```

4. root として、`/var/opt/repserver/repserver_name.mail` ファイルを両方のノードに作成します。*repserver_name* は、Replication Server の名前です。

   このファイルには、Replication Server 管理者の UNIX ログイン名をリストします。すべてのログイン名を、スペースで区切って 1 行に入力します。

   フォールト・モニタにユーザの介入が必要な問題が発生すると、このファイルにリストされている管理者に対して、メールが送信されます。

5. Replication Server を Sun Cluster のデータ・サービスとして登録します。

```
hareg -r repserver_name ¥
-b "/opt/SUNWcluster/ha/repserver_name" ¥
-m START_NET="/opt/SUNWcluster/ha/repserver_name/
repserver_start_net" ¥
-t START_NET=60 ¥
-m STOP_NET="/opt/SUNWcluster/ha/repserver_name/
repserver_stop_net" ¥
```

```
-t STOP_NET=60 ¥
-m FM_START="/opt/SUNWcluster/ha/repserver_name/
repserver_fm_start" ¥
-t FM_START=60 ¥
-m FM_STOP="/opt/SUNWcluster/ha/repserver_name/
repserver_fm_stop" ¥
-t FM_STOP=60 ¥
[-d sybase] -h logical_host
```

-d sybase は、RSSD が同一クラスタ の HA 環境にある場合に必要です。また、*repserver_name* は、Replication Server の名前であり、スクリプトのパスに含める必要があります。

6. データ・サービスを起動します。

   次のように入力します。**hareg -y** *repserver_name*

データ・サービスとしての Replication Server の管理

データ・サービスとしての Replication Server の起動および停止方法と、モニタリングおよびトラブルシューティングに役立つログについて説明します。

データ・サービスの起動と停止

Replication Server をデータ・サービスとして登録した後、Replication Server を起動および停止するコマンドについて説明します。

Replication Server をまだ起動していない場合は起動し、Replication Server 用のフォールト・モニタも起動するには、次のコマンドを入力します。

```
hareg -y repserver_name
```

Replication Server を停止するには、次のコマンドを入力します。

```
hareg -n repserver_name
```

Replication Server が停止されるか別の方法で強制終了された場合、フォールト・モニタによりその Replication Server が再起動またはフェールオーバされます。

Sun Cluster での HA のログ

いくつかのログは、デバッグに使用できます。

次を使用できます。

- Replication Server ログ ― ここには、Replication Server のメッセージのログが取られます。このログは、Replication Server からの情報メッセージやエラー・メッセージを調べるときに使用します。このログは、Replication Server の `Run` ディレクトリにあります。
- スクリプト・ログ ― ここには、データ・サービスの START スクリプトと STOP スクリプトのメッセージのログが取られます。このログは、スクリプトを実行した結果生成された情報メッセージやエラー・メッセージを調べるときに使用します。このログは、`/var/opt/repserver/harep.log` にあります。
- コンソール・ログ ― ここには、オペレーティング・システムのメッセージのログが取られます。このログは、ハードウェアからの情報メッセージやエラー・メッセージを調べるときに使用します。このログは、`/var/adm/messages` にあります。
- CCD ログ - ここには、Sun Cluster の構成の一部である CCD (Cluster Configurations Database) のメッセージのログが取られます。このログは、Sun Cluster 構成とその状態についての情報メッセージやエラー・メッセージを調べるときに使用します。このログは、`/var/opt/SUNWcluster/ccd/ccd.log` にあります。

リファレンス複写環境の実装

ご使用の環境で使用可能な製品を使って、Adaptive Server から Adaptive Server へ、または Oracle から Oracle へのリファレンス複写環境をすばやくセットアップできます。

リファレンス複写環境の実装

Replication Server には、ご使用の環境で使用可能な製品を使って、Adaptive Server から Adaptive Server へ、または Oracle から Oracle への複写のリファレンス実装をすばやくセットアップするためのツールセットが含まれています。

複写環境を実装すると、Replication Server のさまざまな機能を試すことができます。ツールセットを使用すると、次の手順を実行できます。

1. Replication Server、プライマリ・データベースおよびレプリケート・データベースを含む複写環境を構築します。
2. 複写環境を設定します。
3. プライマリ・データベースで単純なトランザクションを実行し、データベース・レベルの複写により、変更を複写します。
4. 手順 3 の複写処理から統計とモニタ・カウンタを収集します。
5. リファレンス複写環境をクリーンアップします。

リファレンス実装ツールセットはスクリプトで構成されており、`$SYBASE/refimp` にあります。

注意： リファレンス実装は、単一の Replication Server、プライマリ・データベース・サーバ、レプリケート・データベース・サーバを含む複写環境を構築します。複数の複写システム・コンポーネント用のリファレンス環境トポロジは設定できません。

プラットフォームのサポート

複写環境は、Linux on IBM p-Series (Linux on Power) 64 ビット版を除く、Replication Server がサポートしているすべてのプラットフォームに実装できます。リファレンス環境を Replication Server がサポートする Microsoft Windows プラットフォームでセットアップするには、Cygwin を使ってリファレンス実装スクリプトを実行する必要があります。

Cygwin Web サイト：http://www.cygwin.com を参照してください。

リファレンス実装のコンポーネント

リファレンス環境を実装するには、複写環境のサポートしているバージョンのコンポーネントが必要です。

Adaptive Server

Adaptive Server から Adaptive Server への複写のためのリファレンス実装環境の構築には、サポートされているバージョンの Replication Server と Adaptive Server が必要です。

表 35 : Adaptive Server のリファレンス実装でサポートされている製品コンポーネントのバージョン

Replication Server	Adaptive Server
15.5	15.0.3, 15.5

たとえば、Adaptive Server のリファレンス環境の構築には、Replication Server 15.5 と Adaptive Server のバージョン 15.0.3 または 15.5 が必要です。

Oracle

また、Oracle から Oracle への複写のためのリファレンス実装環境の構築には、サポートされているバージョンの Replication Server、Oracle、Replication Agent for Oracle、および ECDA Option for Oracle が必要です。

表 36 : Oracle のリファレンス実装でサポートされている製品コンポーネントのバージョン

Replication Server	Oracle	Replication Agent for Oracle	ECDA Option for Oracle
15.5	10.2	15.2	15.0 ESD #3

たとえば、Oracle 用のリファレンス実装環境の構築には、Replication Server 15.5、Oracle 10.2、Replication Agent 15.2、および ECDA Option for Oracle 15.0 ESD #3 が必要です。

リファレンス環境の前提条件

リファレンス環境を構築する際に注意が必要ないくつかの前提条件と情報があります。

1. Oracle の場合、Oracle リリース・ディレクトリでの **execute** パーミッションがあることを確認してください。たとえば、インスタンスを手動で作成できるかどうかを確認します。
2. Replication Server リリース・ディレクトリまたは Adaptive Server リリース・ディレクトリの `SYBASE.sh` ファイルの環境変数設定が正しいことを確認します。正しいことを確認できない場合は、このファイルを削除するか、名前を変更します。
3. ご使用の bash シェルで、UNIX の **grep**、**kill**、**awk**、および **ps** の各コマンドを使用できることを確認します。

リファレンス実装の手順では、Replication Server リリース・ディレクトリの `interfaces` ファイルを使用します。リファレンス実装の手順を実行する前にこのファイルが存在する場合、この手順の実行時にファイル名拡張子が追加されて既存のファイルがバックアップされます。

Oracle の場合、リファレンス実装の手順で既存のファイル (`tnsname.ora`、`listener.ora`) の名前が変更され、Oracle リファレンス実装用に新しいファイルが作成されます。

リファレンス環境の構築

buildenv スクリプトを実行すると、Replication Server、プライマリとレプリケートのデータ・サーバおよびデータベースが自動的に作成されます。

次のように入力します。

```
buildenv -f config_file
```

ユーザがファイルに指定できるパラメータを含むビルド設定ファイルの名前とロケーションを指定するには、*config_file* を使用します。

buildenv が正常に実行されると、次のような出力が表示されます。

```
Environment setup successfully completed.
```

リファレンス実装の設定ファイル

Sybase には、サポートされている UNIX および Microsoft Windows プラットフォームでの、Adaptive Server から Adaptive Server へ、および Oracle から Oracle への複写

用の設定ファイルのテンプレートが用意されており、ご使用の環境用の設定ファイルを作成できます。

ファイルは、 `$SYBASE/REP-15_5/REFIMP-01_0` にあります。

表 37 : リファレンス実装の設定ファイル

プライマリとレプリケートのデータ・サーバ、プラットフォーム	設定ファイル
ASE から ASE、UNIX	`ase_unix_refimp.cfg`
ASE から ASE、Windows	`ase_win_refimp.cfg`
Oracle から Oracle、UNIX	`ora_unix_refimp.cfg`
Oracle から Oracle、Windows	`ora_win_refimp.cfg`

ase_unix_refimp.cfg テンプレート・ファイルの例

ご使用の環境に合わせて、ディレクトリのロケーションやホスト名などの値を指定します。

```
#########################################################################
#######
# --- Part 1. release directory of repserver/ase/oracle/refimp ----#
#########################################################################
#######
#
# --- PLATFORM('unix': UNIX/Linux platform, 'win': Windows) ---#
#
os_platform=unix
# --- DATABASE ('ase': Adaptive Server Enterprise, 'ora': ORACLE) ---
#
#
db_type=ase
#
# --- RS RELEASE DIRECTORY --- #
#
rs release directory=/remote/repeng4/users/xiel/repserver
#
# --- RS RELEASE SUBDIRECTORY --- #
#
rs_sub_directory=REP-15_2
#
# --- ASE RELEASE DIRECTORY --- #
#
ase_release=/remote/repeng4/users/xiel/ase
#
# --- ASE/ORACLE RELEASE SUBDIRECTORY --- #
#
ase_subdir=ASE-15_0
#
# --- REFERENCE IMPLEMENTATION RELEASE DIRECTORY --- #
#
```

```
refimp_release_dir=/calm/repl/svr/refimp
#
#
#
# --- REFERENCE IMPLEMENTATION WORK DIRECTORY ---
#
refimp_work_dir=/remote/repeng4/users/xiel/test
#
# --- OCS RELEASE DIRECTORY --- #
#
ocs_release_directory=OCS-15_0
#
# --- PDS DEVICE NAME WITH FULL PATH --- #
#
pds_device_file=/remote/repeng4/users/xiel/pds
#
# --- RDS DEVICE NAME WITH FULL PATH --- #
#
rds_device_file=/remote/repeng4/users/xiel/rds
#
# --- rs_init RELEASE DIRECTORY --- #
#
rsinit_release=/remote/repeng4/users/xiel/repserver
#
#
# --- interface FILE NAME ---
#
ini_filename=interfaces
#
# --- HOST NAME ---
#
host_name=newgarlic
########################################################################
#######
# --- Part 2. login information of replication server and data server
---#
########################################################################
#######
#
# --- RS NAME --- #
#
rs_name=SAMPLE_RS
#
# --- RS USER NAME --- #
#
rs_username=sa
#
# --- RS PASSWORD --- #
#
rs_password=
#
#
#
# --- ERSSD NAME --- #
#
rssd_name=SAMPLE_RS_ERSSD
```

```
#
# --- ERSSD USER NAME --- #
#
rssd_username=rssd
#
# --- ERSSD PASSWORD --- #
#
rssd_password=rssd_pwd
#
# --- PDS NAME --- #
#
primary_ds=PDS
#
# --- PDB NAME ---
#primary_db=pdb
#
# --- PDB USER NAME ---
#
pdb_username=sa
#
# --- PDB PASSWORD ---
#
pdb_password=
#
# --- RDS NAME ---
#
replicate_ds=RDS
#
# --- RDB NAME ---
#
replicate_db=rdb
#
# --- RDB USER NAME ---
#
rdb_username=sa
#
# --- RDB PASSWORD ---
#
rdb_password=
#
# --- PORT FOR RS ---
#
rs_port=11754
#
# --- PORT FOR RSSD ---
#
rssd_port=11755
#
# --- PORT FOR PDS ---
#
pds_port=20000
#
# --- PORT FOR RDS ---
#
rds_port=20001
#
```

```
#########################################################################
#######
# --- Part 3. transaction profile configuration parameters  --- #
#########################################################################
#######
#
# --- number of transactions to be executed --- #
#
tran_number=100
#
# --- what kind of transction will be executed --- #
#    1."Tran_Profile_1(insert--48% delete--4% update 48%)"
#    2."Tran_Profile_2(insert--30% delete--5% update 65%)"
#    3."Tran_Profile_3(insert--61% delete--2% update 37%)"
#    4."Tran_Profile_LargeTran"
#
tran_option=1
#
#########################################################################
#######
# --- Part 4. system settings --- #
#########################################################################
#######
#
# --- WAIT TIME FOR CONNECTING SERVERS, SPECIFIED BY SECOND(S) ---
#
wait_time=10
```

リファレンス環境の設定

リファレンス複写環境を構築した後、**config** パラメータと設定ファイルを指定して **refimp** スクリプトを実行し、リファレンス・プライマリ・データベースとレプリケート・データベースにテーブルとストアド・プロシージャを作成し、リファレンス Replication Server にデータベース複写定義とサブスクリプションを作成します。

次のように入力します。

```
refimp config -f config_file
```

ユーザがファイルに指定できるパラメータを含む設定ファイルの名前とロケーションを指定するには、*config_file* を使用します。

構築プロセスの **buildenv** で指定したものと同じ設定ファイル情報を使用する必要があります。

refimp config が正常に実行されると、次のような出力が表示されます。

```
Task succeeded: configuring database replication environment
completed.
```

参照：
- リファレンス環境に作成されるオブジェクト (483 ページ)

リファレンス環境でのパフォーマンス・テストの実行

run パラメータを指定して **refimp** スクリプトを実行すると、データベース・レベルの複写を使用して、プライマリ・データ・サーバ上のデータが自動的に挿入、更新、または削除されます。

次のように入力します。

```
refimp run -f config_file
```

refimp config と同じ設定ファイルを指定するには、*config_file* を使用します。

refimp run が正常に実行されると、次のような出力が表示されます。

```
Task succeeded: insert data into primary database completed.
```

リファレンス環境からのテスト結果の取得

統計情報とパフォーマンス情報を収集するには、**analyze** パラメータを指定して **refimp** スクリプトを実行します。

次のように入力します。

```
refimp analyze -f config_file
```

refimp config と同じ設定ファイルを指定するには、*config_file* を使用します。

refimp analyze が正常に実行されると、次のような出力が表示されます。

```
Task succeeded: fetch performance data completed.
```

`$refimp_work_dir/report` から rs_ticket_history レポートと、モニタおよびカウンタのレポートを取得します。ここで、*refimp_work_dir* は設定ファイルで指定したロケーションです。

rs_ticket_history レポート

rs_ticket_history レポートは、各 Replication Server モジュールでチケットがレポートするタイム・スタンプを基にして、チケット・データが各モジュールを通過するのに要した時間を示します。

レポートは **rs_ticket** ストアド・プロシージャによって生成されます。『Replication Server リファレンス・マニュアル』の「RSSD ストアド・プロシージャ」の「**rs_ticket**」を参照してください。

プライマリ・データベースとレプリケート・データベースでチケットがレポートする時刻から、合計複写時間を計算できます。レポートには以下のカラムがあります。

- cnt – チケット・シーケンス番号。
- pdb_t – **rs_ticket** ストアド・プロシージャがプライマリ・データベースで実行された時刻。
- rdb_t – チケットがレプリケート・データベースに到着した時刻。
- ticket – チケットが各モジュールを通過した時刻を含む、チケットに関する情報。

rs_ticket_history レポートのサンプル

```
cnt       pdb_t                          rdb_t
---       -------------------            -------------------
  1       Jan 19 2010  2:17AM            Jan 19 2010  2:17AM

ticket
-------------------------------------------------------------
V=2;H1=profile1;H2=start;PDB(pdb)=01/19/10 02:17:19.406;
EXEC(40)=01/19/10 02:17:19.423;B(40)=1332;
DIST(26)=01/19/10 02:17:19.669;
DSI(35)=01/19/10 02:17:19.916;
DSI_T=1;DSI_C=3;RRS=SAMPLE_RS_XIEL


cnt       pdb_t                          rdb_t
---       -------------------            -------------------
  2       Jan 19 2010  2:20AM            Jan 19 2010  2:20AM

ticket
-------------------------------------------------------------

V=2;H1=profile1;H2=end;PDB(pdb)=01/19/10 02:20:32.206;
EXEC(40)=01/19/10 02:20:32.211;B(40)=5044893;
DIST(26)=01/19/10 02:20:32.249;DSI(35)=01/19/10 02:20:32.524;
DSI_T=5410;DSI_C=18297;RRS=SAMPLE_RS_XIEL
```

モニタとカウンタのレポート

モニタとカウンタのレポートは、レポート期間中に実行するコマンドを Replication Server カウンタがモニタした結果のパフォーマンス・データを示します。

モニタとカウンタのレポートのサンプル
このレポートは長いため、1 つのカウンタのみ示します。

注意： 出力の右側にあるコメントは、例を説明するために含まれています。これらは、出力の一部ではありません。

```
Comment: refimp
Jan 19 2010 02:17:39:606AM                          *Start time stamp*
Jan 19 2010 02:20:22:576AM                          *End time stamp*
9                                                 *No of obs intervals*
0                                                *No of min between obs*
16384                                              *SQM bytes per block*
64                                               *SQM blocks per segment*
AOBJ                                                 *Module name*
10305                                                *Instance ID*
11                                                   *Instance value*
AOBJ dbo.district                                    *Module name*
AOBJ: Insert command                              *Counter external name*
AOBJInsertCommand                                  *Counter display name*
65000, , 10305, 11                                 *Counter ID, instance
ID,
                                                           instance value*
ENDOFDATA                                           *EOD for counter*

AOBJ: Update command                              *Counter external name*
AOBJUpdateCommand                                  *Counter display name*
65000, , 10305, 11                                 *Counter ID, instance
ID,
                                                           instance value*
Jan 19 2010 02:17:39:606AM, 50, 50,  1,  1       *Dump ts, obs, total,
                                                          last, max*
....
ENDOFDATA                                           *EOD for counter*
```

参照：

- カウンタを使ったパフォーマンスのモニタリング (339 ページ)

リファレンス実装サーバの停止

環境をクリーンアップした後に Replication Server とデータ・サーバを停止するには、**cleanenv** スクリプトを実行します。

次のように入力します。

```
cleanenv -f config_file
```

refimp config と同じ設定ファイルを指定するには、*config_file* を使用します。

cleanenv が正常に実行されると、次のような出力が表示されます。

```
Task succeeded: shut down all the servers.
```

リファレンス環境のクリーンアップ

次のテストの準備のためにテスト・データ、複写定義、サブスクリプション、テーブル、およびストアド・プロシージャを削除するには、**cleanup** パラメータを指定して **refimp** スクリプトを実行します。

次のように入力します。

```
refimp  cleanup -f config_file
```

refimp config と同じ設定ファイルを指定するには、*config_file* を使用します。

refimp cleanup が正常に実行されると、次のような出力が表示されます。

```
Task succeeded: clean up database replication environment completed.
```

リファレンス環境に作成されるオブジェクト

リファレンス実装ツールセットは、リファレンス複写環境内にストアド・プロシージャ、複写定義、サブスクリプション、およびテーブルの各オブジェクトを作成します。

表 38 : リファレンス実装用に作成されるストアド・プロシージャ

ストアド・プロシージャ	ロケーション
sp_load_warehouse_data	プライマリ・データベースおよびレプリケート・データベース
sp_load_district_data	プライマリ・データベースおよびレプリケート・データベース
sp_load_customer_data	プライマリ・データベースおよびレプリケート・データベース
sp_load_history_data	プライマリ・データベースおよびレプリケート・データベース
sp_load_item_data	プライマリ・データベースおよびレプリケート・データベース
sp_load_stock_data	プライマリ・データベースおよびレプリケート・データベース
sp_load_order_orderline_data	プライマリ・データベースおよびレプリケート・データベース
sp_load_neworder_data	プライマリ・データベースおよびレプリケート・データベース

ストアド・プロシージャ	ロケーション
sp_load_data_multi_tran	プライマリ・データベースおよびレプリケート・データベース
sp_gen_neworder_data	プライマリ・データベース
sp_gen_payment_data	プライマリ・データベース
sp_gen_delivery_data	プライマリ・データベース
sp_gen_neworder_data_large_tran	プライマリ・データベース
sp_gen_payment_data_large_tran	プライマリ・データベース
sp_gen_delivery_data_large_tran	プライマリ・データベース
sp_generator_data_1	プライマリ・データベース
sp_generator_data_2	プライマリ・データベース
sp_generator_data_3	プライマリ・データベース
sp_generator_data_4	プライマリ・データベース

表 39 : リファレンス実装用に作成される複写定義とサブスクリプション

ストアド・プロシージャ	サブスクリプション
複写定義：**pdbrepdefforrdb**	
サブスクリプション：**rdbsubforpdb**	**pdbrepdefforrdb**

表 40 : リファレンス実装用に作成されるテーブル

テーブル	ロケーション
WAREHOUSE	プライマリ・データベースおよびレプリケート・データベース
DISTRICT	プライマリ・データベースおよびレプリケート・データベース
CUSTOMER	プライマリ・データベースおよびレプリケート・データベース
HISTORY	プライマリ・データベースおよびレプリケート・データベース
NEW_ORDER	プライマリ・データベースおよびレプリケート・データベース
ORDER	プライマリ・データベースおよびレプリケート・データベース
ORDER_LINE	プライマリ・データベースおよびレプリケート・データベース
ITEM	プライマリ・データベースおよびレプリケート・データベース
Stock	プライマリ・データベースおよびレプリケート・データベース

テーブル・スキーマ

リファレンス実装用に作成されるテーブルのテーブル・スキーマ

表 41 : WAREHOUSE

フィールド名	フィールド定義	コメント
W_ID	2*W 個のユニークな ID	W はウェアハウス番号
W_NAME	可変テキスト、サイズ 10	
W_STREET1	可変テキスト、サイズ 20	
W_STREET2	可変テキスト、サイズ 20	
W_CITY	可変テキスト、サイズ 20	
W_STATE	固定テキスト、サイズ 2	
W_ZIP	固定テキスト、サイズ 9	
W_TAX	数値、4 桁	消費税
W_YTD	数値、12 桁	年度累計残高

キー：

- プライマリ・キー：(W_ID)

表 42 : DISTRICT

フィールド名	フィールド定義	コメント
D_ID	20 個のユニークな ID	ウェアハウスごとに 10 個が入力される
D_W_ID	2*W 個のユニークな ID	
D_NAME	可変テキスト、サイズ 10	
D_STREET1	可変テキスト、サイズ 20	
D_STREET2	可変テキスト、サイズ 20	
D_CITY	可変テキスト、サイズ 20	
D_STATE	固定テキスト、サイズ 2	
D_ZIP	固定テキスト、サイズ 9	

フィールド名	フィールド定義	コメント
D_TAX	数値、4 桁	消費税
D_YTD	数値、12 桁	年度累計残高
D_NEXT_O_ID	10, 000 個のユニークな ID	次に使用可能な発注番号のユニークな ID

キー：

- プライマリ・キー (D_W_ID、D_ID)
- 外部キー (D_W_ID) が (W_ID) を参照する

表 43 : CUSTOMER

フィールド名	フィールド定義	コメント
C_ID	96, 000 個のユニークな ID	ウェアハウスごとに 3, 000 個が入力される
C_D_ID	20 個のユニークな ID	
C_W_ID	2*W 個のユニークな ID	
C_FIRST	可変テキスト、サイズ 16	
C_MIDDLE	固定テキスト、サイズ 2	
C_LAST	可変テキスト、サイズ 16	
C_STREET1	可変テキスト、サイズ 20	
C_STREET2	可変テキスト、サイズ 20	
C_CITY	可変テキスト、サイズ 20	
C_STATE	固定テキスト、サイズ 2	
C_ZIP	固定テキスト、サイズ 9	
C_PHONE	固定テキスト、サイズ 16	
C_SINCE	日付および時刻	登録日
C_CREDIT	固定テキスト、サイズ 2	クレジット："GC"= 支払能力がある、"BC"= 支払能力がない
C_CREDIT_LIM	数値、12 桁	
C_DISCOUNT	数値、4 桁	

フィールド名	フィールド定義	コメント
C_BALANCE	符号付き数値、12 桁	
C_YTD_PAYMENT	数値、12 桁	
C_PAYMENT_CNT	数値、4 桁	
C_DELIVERY_CNT	数値、4 桁	
C_DATA	可変テキスト、サイズ 500	注釈用

キー：

- プライマリ・キー (C_W_ID、C_D_ID、C_ID)
- 外部キー (C_W_ID、C_D_ID) が (D_W_ID、D_ID) を参照する

表 44 : HISTORY

フィールド名	フィールド定義	コメント
H_C_ID	96, 000 個のユニークな ID	
H_C_D_ID	20 個のユニークな ID	
H_C_W_ID	2*W 個のユニークな ID	
H_D_ID	20 個のユニークな ID	
H_W_ID	2*W 個のユニークな ID	
H_DATE	日付および時刻	
H_AMOUNT	数値、6 桁	
H_DATA	可変テキスト、サイズ 24	

キー：

- プライマリ・キー：なし
- 外部キー (H_C_W_ID、H_C_D_ID、H_C_ID) が (C_W_ID、C_D_ID、C_ID) を参照する
- 外部キー (H_W_ID、H_D_ID) が (D_W_ID、D_ID) を参照する

表 45 : NEW_ORDER

フィールド名	フィールド定義	コメント
N_O_ID	10, 000, 000 個のユニークな ID	
N_D_ID	20 個のユニークな ID	
NO_W_ID	2*W 個のユニークな ID	

キー：

- プライマリ・キー (NO_W_ID、NO_D_ID、NO_O_ID)
- 外部キー (NO_W_ID、NO_D_ID、NO_O_ID) が (O_W_ID、O_D_ID、O_ID) を参照する

表 46 : ORDER

フィールド名	フィールド定義	コメント
O_ID	10, 000, 000 個のユニークな ID	
O_D_ID	20 個のユニークな ID	
O_W_ID	2*W 個のユニークな ID	
O_C_ID	96, 000 個のユニークな ID	
O_ENTRY_D	日付および時刻	
O_CARRIER_ID	10 個のユニークな ID、または null	
O_OL_CNT	5 〜 15	
O_ALL_LOCAL	数値、1 桁	

キー：

- プライマリ・キー (O_W_ID、O_D_ID、O_ID)
- 外部キー (O_W_ID、O_D_ID、O_C_ID) が (C_W_ID、C_D_ID、C_ID) を参照する

表 47 : ORDER_LINE

フィールド名	フィールド定義	コメント
OL_O_ID	10, 000, 000 個のユニークな ID	
OL_D_ID	20 個のユニークな ID	

フィールド名	フィールド定義	コメント
OL_W_ID	2*W 個のユニークな ID	
OL_NUMBER	15 個のユニークな ID	
OL_I_ID	200,000 個のユニークな ID	
OL_SUPPLY_W_ID	2*W 個のユニークな ID	
OL_DELIVERY_D	日付と時刻、または null	
OL_QUANTITY	数値、2 桁	
OL_AMOUNT	数値、6 桁	
OL_DIST_INFO	固定テキスト、サイズ 24	

キー：

- プライマリ・キー (OL_W_ID、OL_D_ID、OL_O_ID、OL_NUMBER)
- 外部キー (OL_W_ID、OL_D_ID、OL_O_ID) が (C_W_ID、C_D_ID、C_ID) を参照する
- 外部キー (OL_SUPPLY_W_ID、OL_I_ID) が (S_W_ID、S_I_ID) を参照する

表 48 : ITEM

フィールド名	フィールド定義	コメント
I_ID	200, 000 個のユニークな ID	
I_IM_ID	200, 000 個のユニークな ID	
I_NAME	可変テキスト、サイズ 50	
I_PRICE	数値、5 桁	
I_DATA	可変テキスト、サイズ 50	

キー：

- プライマリ・キー (I_ID)

表 49 : STOCK

フィールド名	フィールド定義	コメント
S_I_ID	200, 000 個のユニークな ID	

フィールド名	フィールド定義	コメント
S_W_ID	2*W 個のユニークな ID	
S_QUANTITY	数値、4 桁	
S_DIST_01	固定テキスト、サイズ 24	
S_DIST_02	固定テキスト、サイズ 24	
S_DIST_03	固定テキスト、サイズ 24	
S_DIST_04	固定テキスト、サイズ 24	
S_DIST_05	固定テキスト、サイズ 24	
S_DIST_06	固定テキスト、サイズ 24	
S_DIST_07	固定テキスト、サイズ 24	
S_DIST_08	固定テキスト、サイズ 24	
S_DIST_09	固定テキスト、サイズ 24	
S_DIST_10	固定テキスト、サイズ 24	
S_YTD	数値、8 桁	
S_ORDER_CNT	数値、4 桁	
S_REMOTE_CNT	数値、4 桁	
S_DATA	可変テキスト、サイズ 50	

キー：

- プライマリ・キー (S_W_ID、S_I_ID)
- 外部キー (S_W_ID) が (W_ID) を参照する
- 外部キー (S_I_ID) が (I_ID) を参照する

用語解説

複写システムで使用される用語を解説します。

- **アクティブ・データベース** – ウォーム・スタンバイ・アプリケーションにおいて、スタンバイ・データベースに複写されるデータベースです。「*ウォーム・スタンバイ・アプリケーション*」も参照してください。
- **Adaptive Server** – Sybase バージョン 11.5 およびそれ以降のリレーショナル・データベース・サーバです。Replication Server の設定中に RSSD オプションを選択すると、Adaptive Server は RSSD データベースの Replication Server システム・テーブルを管理します。
- **アプリケーション・プログラミング・インタフェース (API)** – ユーザまたはプログラムが相互に通信するために使用する、事前に定義されたインタフェースです。Sybase Open Client および Sybase Open Server は、クライアント／サーバ・アーキテクチャで通信を行う API の 1 つです。RCL (複写コマンド言語) は、Replication Server の API です。
- **適用ファンクション** – ファンクション複写定義に対応する複写ファンクションです。Replication Server によってプライマリ・データベースからファンクション定義のサブスクリプションを持つレプリケート・データベースに配信されます。適用ファンクションがストアド・プロシージャにパラメータ値を渡し、そのストアド・プロシージャがレプリケート・データベースで実行されます。レプリケート・データベースにあるストアド・プロシージャはメンテナンス・ユーザによって実行されます。「*複写ファンクションの配信*」、「*要求ファンクション*」、「*ファンクション複写定義*」も参照してください。
- **アーティクル** – テーブルまたはストアド・プロシージャの複写定義を拡張したもので、パブリケーションの要素となります。アーティクルには、レプリケート・データベースが受信するローのサブセットを指定した **where** 句が含まれている場合もあれば、含まれていない場合もあります。
- **非同期プロシージャ配信** – プライマリ・データベースまたはレプリケート・データベースで複写するように指定されたストアド・プロシージャを実行できる Replication Server システムの一部です。
- **非同期コマンド** – クライアントが送信するコマンドです。クライアントは、完了ステータスの受信を待たずに、他のオペレーションを継続できます。Replication Server のコマンドの多くは、複写システム内で非同期コマンドとして動作します。
- **アトミック・マテリアライゼーション** – マテリアライゼーション・メソッドの 1 つです。**select** オペレーションを holdlock を指定して使用し、1 つのアトミック・オペレーションでネットワークを介して、プライマリ・データベースからレプリケート・データベースへサブスクリプション・データをコピーします。データの転送が完了するまで、プライマリ・データへの変更は行えません。レ

プリケート・データは、1 つのトランザクションとして、またはレプリケート・データベースのトランザクション・ログが一杯にならないようにトランザクションごとに 10 ローずつ挿入する方法のいずれかを使用して適用できます。アトミック・マテリアライゼーションは、**create subscription** コマンドのデフォルトのメソッドです。「*ノンアトミック・マテリアライゼーション*」、「*バルク・マテリアライゼーション*」、「*非マテリアライゼーション*」も参照してください。

- **autocorrection** – オートコレクションは、複写定義に適用する機能で、レプリケート・テーブルに、消失ローや重複したローが発生して障害が起こることを防ぎます。**set autocorrection** コマンドを使用して設定します。オートコレクションを有効にすると、Replication Server は各更新オペレーションまたは挿入オペレーションを削除と挿入の連続オペレーションに変換します。オートコレクションは、サブスクリプションがノンアトミック・マテリアライゼーションを使用している複写定義の場合だけ使用できます。
- **基本クラス** – 親クラスからファンクション文字列を継承しないファンクション文字列クラスです。「*ファンクション文字列クラス*」も参照してください。
- **ビットマップ・サブスクリプション** – ビットマップの比較に基づいてローを複写するサブスクリプションです。`int` データ型のカラムを作成し、複写定義を作成するときには、カラムを `rs_address` データ型として指定します。サブスクリプションを作成するときには、**where** 句でビットマップ比較演算子 (&) を使用し、各 `rs_address` カラムとビットマスクを比較します。サブスクリプションのビットマップと一致するローが複写されます。
- **バルク・コピー・イン** – Adaptive Server® Enterprise 12.0 以降で、大量の **insert** 文を同じテーブルで複写するときに Replication Server のパフォーマンスを向上させる機能です。Replication Server は、Open Client™ Open Server™ Bulk-Library を使用して、レプリケート・データベースにトランザクションを送信する Replication Server モジュールであるデータ・サーバ・インタフェース (DSI) にバルク・コピー・インを実装します。

  バルク・コピー・インにより、サブスクリプション・マテリアライゼーションのパフォーマンスも向上します。**dsi_bulk_copy** を on にすると、各トランザクションの **insert** コマンドの数が **dsi_bulk_threshold** を超えた場合に、Replication Server は、バルク・コピー・インを使用してサブスクリプションをマテリアライズします。
- **バルク・マテリアライゼーション** – マテリアライゼーションのメソッドの 1 つです。これは、複写システム以外でレプリケート・データベースのサブスクリプションのデータを初期化します。たとえば、磁気テープ、フロッピー・ディスク、CD-ROM、または光磁気ディスクなどのメディアを使用して、プライマリ・データベースからデータを転送できます。バルク・マテリアライゼーションでは、**define subscription** から始まる一連のコマンドを使用します。バルク・マテリアライゼーションは、テーブル複写定義とファンクション複写定義のどちらのサブスクリプションにも使用できます。「*アトミック・マテリアライ*

ゼーション」、「*ノンアトミック・マテリアライゼーション*」、「*非マテリアライゼーション*」も参照してください。

- **集中型データベース・システム** – 中央サイトに設置された 1 つのデータベース管理システムでデータを一元管理するデータベース・システムです。
- **クラス** –「*エラー・クラス*」と「*ファンクション文字列クラス*」を参照してください。
- **クラス・ツリー** – 派生クラスと親クラスの複数のレベルから構成されるファンクション文字列クラスのセットです。これは、同じ基本クラスから派生します。「*ファンクション文字列クラス*」も参照してください。
- **クライアント** – クライアント／サーバ・アーキテクチャにおいて、サーバに接続されたプログラムです。ユーザが実行するフロントエンド・アプリケーション・プログラムの場合もあれば、システムの拡張機能として実行されるユーティリティ・プログラムの場合もあります。
- **Client/Server Interfaces (C/SI)** – クライアント／サーバ・アーキテクチャで実行するプログラムのための Sybase のインタフェース標準です。
- **同時実行性** – 複数のクライアントが、データまたはリソースを共有できることを示します。データベース管理システムにおける同時実行性は、あるクライアントが使用中のデータを別のクライアントが変更しようとするときに発生する競合からクライアントを保護するシステムに依存します。
- **接続** – Replication Server からデータベースへ、または LTM から Replication Server への接続です。「*データ・サーバ・インタフェース (DSI)*」と「*論理コネクション*」も参照してください。
- **接続プロファイル** – 接続プロファイルを使用すると、事前に定義されたプロパティのセットでコネクションを設定できます。
- **コーディネート・ダンプ** – 複写システムでファンクション **rs_dumpdb** または **rs_dumptran** を実行することによって、複数のサイト間で同期がとられているデータベース・ダンプ・セット、またはトランザクション・ダンプ・セットです。
- **データベース** – 相互に関連するデータ・テーブルとその他のオブジェクトが、特定の目的に合わせて編成、表現されたものです。
- **データベース世代番号** – データベースおよびデータベースを管理する Replication Server の RSSD に格納されます。データベース世代番号は、各ログ・レコードのオリジン・キュー ID (*qid*) の最初の部分です。オリジン・キュー ID は、Replication Server が重複したレコードを処理していないことを示します。リカバリ・オペレーション中に、データベース世代番号を増やさなければならない場合があります。このようにすることで、Replication Server は、データベースが再ロードされた後に送信されたレコードを無視しません。
- **データベース複写定義** – サブスクリプションを作成できる対象のデータベース・オブジェクト (テーブル、トランザクション、ファンクション、システム・ストアド・プロシージャ、DDL) を集めて記述したものです。

テーブル複写定義とファンクション複写定義も作成できます。「*テーブル複写定義*」と「*ファンクション複写定義*」も参照してください。

- **データベース・サーバ** – Sybase Adaptive Server などのサーバ・アプリケーションです。クライアントにデータベース管理サービスを提供します。
- **データ定義言語 (DDL)** – Transact-SQL などのクエリ言語のコマンド・セットであり、データとデータベース内でのデータの関係を記述します。Transact-SQL の DDL コマンドには、**create**、**drop**、**alter** キーワードを使用するものなどがあります。
- **データ操作言語 (DML)** – Transact-SQL などのクエリ言語のコマンド・セットであり、データの操作を行います。Transact-SQL の DML コマンドには、**select**、**insert**、**update**、**delete** があります。
- **データ・サーバ** – Sybase Client/Server Interfaces に準拠のクライアント・インタフェースによって、データベースの複写テーブルの物理表現を管理するのに必要な機能を提供するサーバです。データ・サーバは、通常、データベース・サーバと同じものですが、Replication Server に必要なインタフェースと機能を備えたデータ・リポジトリの場合もあります。
- **データ・サーバ・インタフェース (DSI)** – Replication Server とデータベース間のコネクションに対応している Replication Server スレッドです。DSI スレッドは、DSI アウトバウンド・キューからレプリケート・データ・サーバへトランザクションを送信します。DSI スレッドは 1 つのスケジューラ・スレッドと 1 つまたは複数のエグゼキュータ・スレッドで構成されます。スケジューラ・スレッドは、トランザクションをコミット順にグループ分けしてから、それらをエグゼキュータ・スレッドにディスパッチします。エグゼキュータ・スレッドは、ファンクションをファンクション文字列にマッピングし、レプリケート・データベースのトランザクションを実行します。DSI スレッドは、データベースへの Open Client コネクションを使用します。「*アウトバウンド・キュー*」と「*コネクション*」も参照してください。
- **データ・ソース** – リレーショナル・データ・サーバまたはノンリレーショナル・データ・サーバなどのデータベース管理システム (DBMS) 製品、その DBMS にあるデータベース、および複写システムの他のコンポーネントから DBMS にアクセスするための通信方法の組み合わせからなる概念をデータ・ソースといいます。「*データベース*」と「*データ・サーバ*」も参照してください。
- **意思決定支援アプリケーション** – アドホック・クエリ、レポート、計算などの処理が行え、少数データの更新トランザクションを特徴とするデータベース・クライアント・アプリケーションです。
- **宣言したデータ型** – Replication Agent から Replication Server に配信された値のデータ型です。
  - Replication Agent が `datetime` などの基本 Replication Server データ型を Replication Server に配信する場合、宣言したデータ型は基本データ型です。

  - 上記以外の場合、宣言したデータ型は、プライマリ・データベースの元のデータ型に対する UDD でなければなりません。
- **デフォルトのファンクション文字列** – システム提供クラス `rs_sqlserver_function_class` と `rs_default_function_class`、およびこれらのクラスからファンクション文字列を直接的または間接的に継承するクラスに対してデフォルトで提供されるファンクション文字列です。*「ファンクション文字列」*参照。
- **マテリアライゼーション解除** – サブスクリプションが削除されたときに、適宜実行される処理です。これによって、他のサブスクリプションに使用されていない特定のローが、レプリケート・データベースから削除されます。
- **派生クラス** – 親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です。「*ファンクション文字列クラス*」と「*親クラス*」も参照してください。
- **直接ルート** – 中間 Replication Server を使用しないで、送信元 Replication Server から送信先 Replication Server へ直接メッセージを送信するために使用するルートです。「*間接ルート*」と「*ルート*」も参照してください。
- **ディスク・パーティション** –「*パーティション*」を参照してください。
- **分散データベース・システム** – データをネットワーク上にある複数のデータベースに格納するデータベース・システムです。これら複数のデータベースは、同じタイプのデータ・サーバ (たとえば、Adaptive Server) で管理される場合と、異機種のデータ・サーバで管理される場合があります。
- **ディストリビュータ** – インバウンド・キューにある各トランザクションの送信先を決定するために使用する Replication Server スレッド (DIST) です。
- **ダンプ・マーカ** – ダンプの実行時に Adaptive Server がデータベース・トランザクション・ログに書き込むメッセージです。ウォーム・スタンバイ・アプリケーションでは、アクティブ・データベースのデータでスタンバイ・データベースを初期化するときに、Replication Server がダンプ・マーカを使用して、トランザクション・ストリームのどこからトランザクションをスタンバイ・データベースに適用するかを決定するように指定できます。「*ウォーム・スタンバイ・アプリケーション*」も参照してください。
- **Embedded Replication Server システム・データベース (ERSSD)** – Replication Server システム・テーブルを格納する SQL Anywhere (SA) データベースです。Replication Server システム・テーブルを ERSSD と Adaptive Server RSSD のどちらに格納するかを選択できます。「*Replication Server システム・データベース (RSSD)*」も参照してください。
- **Enterprise Connect Data Access (ECDA)** – LAN ベースの ASE 以外のデータ・ソースやメインフレームのデータ・ソースなど、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。

- **ExpressConnect for Oracle** – Replication Server と Oracle データベース間の直接通信に使用できるライブラリのセットです。
- **エラー・アクション** – データ・サーバのエラーに対する Replication Server の応答処理です。Replication Server のエラー・アクションの種類としては、**ignore**、**warn**、**retry_log**、**log**、**retry_stop**、**stop_replication** があります。エラー・アクションは、特定のデータ・サーバ・エラーに割り当てられます。
- **エラー・クラス** – 指定したデータベースで使用するデータ・サーバのエラー・アクションの集まりに名前を付けたものです。
- **例外ログ** – データ・サーバ上で失敗したトランザクションの情報を保存する Replication Server の 3 つのシステム・テーブルがセットになったものです。例外ログに記録されたトランザクションは、ユーザまたはインテリジェント・アプリケーションが処理しなければなりません。例外ログに問い合わせるには、**rs_helpexception** ストアド・プロシージャを使用します。
- **フェールオーバ** – Sybase フェールオーバを使用すると、バージョン 12.0 以降の 2 つの Adaptive Server をコンパニオンとして設定できます。プライマリ・コンパニオンに障害が発生した場合、そのサーバのデバイス、データベース、コネクションをセカンダリ・コンパニオンが引き継ぐことができます。

  Adaptive Server における Sybase フェールオーバの動作の詳細については、『高可用性システムにおける Sybase フェールオーバの使用』を参照してください。これは、Adaptive Server Enterprise のマニュアル・セットの一部です。
- **フォールト・トレランス** – 1 つまたは複数のコンポーネントで障害が発生した場合でも、システムが正常に処理を継続できるシステムの機能です。
- **ファンクション** – insert、delete、select、begin transaction などのデータ・サーバのオペレーションを表す Replication Server オブジェクトです。Replication Server は、これらのオペレーションをファンクションとして他の Replication Server に配信します。各ファンクションは、ファンクション名とデータ・パラメータのセットから構成されます。ファンクションを送信先データベースで実行するために、Replication Server はファンクション文字列を使用して、そのファンクションを特定タイプのデータベース用のコマンドまたはコマンド・セットに変換します。「*ユーザ定義のファンクション*」と「*複写ファンクションの配信*」も参照してください。
- **ファンクション複写定義** – 複写ファンクションの配信で使用される、複写ファンクションの記述です。ファンクション複写定義は Replication Server によって管理され、複写されるパラメータと影響を受けるデータのプライマリ・バージョンがあるロケーションを示す情報などからなります。ファンクション複写定義には、適用ファンクション複写定義と要求ファンクション複写定義の 2 つのタイプがあります。「*複写ファンクションの配信*」も参照してください。
- **ファンクションのスコープ** – ファンクションの適用範囲です。ファンクションは、複写定義スコープまたはファンクション文字列クラス・スコープを持ちます。複写定義スコープを持つファンクションは、特定の複写定義に指定され、他の複写定義には適用できません。ファンクション文字列クラス・スコープを

持つファンクションは、ファンクション文字列クラスに対して 1 回だけ定義され、そのクラスでのみ使用できます。

- **ファンクション文字列** – Replication Server が、データベース・コマンドをデータ・サーバの API にマップするために使用する文字列です。文字列の最初の部分は入力テンプレートで、**rs_select** および **rs_select_with_lock** ファンクションのファンクション文字列を検索するためにのみ使用します。後半部分は出力テンプレートで、データベース・コマンドを対象のデータ・サーバ用にフォーマットするために使用します。
- **ファンクション文字列クラス** – 指定したデータベース・コネクションで使用される、名前付きのファンクション文字列のコレクションです。ファンクション文字列クラスには、Replication Server によって提供されるものとユーザが作成したものがあります。ファンクション文字列クラスは、ファンクション文字列の継承によってファンクション文字列定義を共有できます。システムで提供されるファンクション文字列クラスには、`rs_sqlserver_function_class`、`rs_default_function_class`、`rs_db2_function_class` の 3 つがあります。「*基本クラス*」、「*クラス・ツリー*」、「*派生クラス*」、「*ファンクション文字列の継承*」、「*親クラス*」も参照してください。
- **ファンクション文字列の継承** – クラス間でファンクション文字列定義を共有する機能です。この機能によって、派生クラスは親クラスからファンクション文字列を継承します。「*派生クラス*」、「*ファンクション文字列クラス*」、「*親クラス*」も参照してください。
- **ファンクション文字列変数** – 実行時に代入される値を表すために、ファンクション文字列内で使用する識別子です。ファンクション文字列内の変数は、疑問符 (?) で囲まれています。この変数は、カラムの値、ファンクションのパラメータ、システム定義の変数、ユーザ定義の変数を表します。
- **ファンクション・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です (適用ファンクションおよび要求ファンクションの配信で使用されます)。
- **ゲートウェイ** – 異なるネットワーク・アーキテクチャを持つ複数のコンピュータ・システム間での通信を可能にする接続ソフトウェアです。
- **世代番号** – 「*データベース生成番号*」を参照してください。
- **異機種データ・サーバ** – 同じ分散データベース・システム内で使用される複数ベンダのデータ・サーバです。
- **ハイバネーション・モード** – Replication Server の状態です。この状態では、**admin** と **sysadmin** コマンドを除くすべての DDL コマンドは拒否され、すべてのルートとコネクション、および DSI、RSI などのほとんどのサービス・スレッドがサスペンドされます。また、RSI と RepAgent ユーザはログオフされログオンできません。ルートのアップグレード中に使用され、Replication Server が問題をデバッグするためにオン状態になることがあります。

- **高可用性 (HA)** – ダウン時間が非常に少ないことです。HA を提供するコンピュータ・システムは、通常、99.999% の可用性 (予定外のダウン時間が、年間約 5 分) を実現しています。
- **High Volume Adaptive Replication (HVAR)** – 最終的な結果とそれ以降のレプリケート・データベースへの最終的な結果のバルク適用を生成する、**insert**、**delete**、**update** の各オペレーションのグループのコンパイルです。
- **ホット・スタンバイ・アプリケーション** – クライアント・アプリケーションを中断したり、トランザクションを失ったりすることなく、スタンバイ・データベースをアクティブに切り替えられるデータベース・アプリケーションです。「*ウォーム・スタンバイ・アプリケーション*」も参照してください。
- **ID サーバ** – 複写システムのいずれか 1 つの Replication Server が、ID サーバとなります。ID サーバは、Replication Server の通常の作業の他に、複写システムにあるすべての Replication Server とデータベースにユニークな ID 番号を割り当て、複写システムのバージョン情報を管理します。
- **インバウンド・キュー** – Replication Agent から Replication Server へのメッセージをスプールするために使用されるステーブル・キューです。
- **間接ルート** – 送信元 Replication Server から送信先 Replication Server へ、1 つ以上の中間 Replication Server を経由してメッセージを送るために使用するルートです。「*直接ルート*」と「*ルート*」も参照してください。
- **interfaces ファイル** – Sybase クライアント／サーバ・アーキテクチャ上のサーバ・プログラムが使用する、ネットワークのアクセス情報を定義するエントリのあるファイルです。サーバ・プログラムには、Adaptive Server、ゲートウェイ、Replication Server、Replication Agent があります。クライアントとサーバは、interfaces ファイルにあるエントリを使用して、ネットワーク上で相互に接続できます。
- **遅延時間** – プライマリ・データベースで最初に適用されたデータ修正オペレーションが、レプリケート・データベースに分配されるまでに要する時間の単位です。この時間には、Replication Agent での処理時間、Replication Server での処理時間、ネットワークのオーバヘッドなどが含まれます。
- **ローカル・エリア・ネットワーク (LAN)** – コンピュータとプリンタや端末などのデバイスを、データやデバイスの共有のためにケーブルで接続したシステムです。
- **ロケータ値** – Replication Server の RSSD の `rs_locater` テーブルに格納されている値です。この値によって、複写中に Replication Server によって受信および確認された、直前の各サイトからの最新のログ・トランザクション・レコードが特定されます。
- **論理コネクション** – Replication Server が、ウォーム・スタンバイ・アプリケーションのアクティブ・データベースとスタンバイ・データベースとのコネクションにマップするデータベース・コネクションです。「*コネクション*」と「*ウォーム・スタンバイ・アプリケーション*」も参照してください。

- **ログイン名** – ユーザまたは Replication Server などのシステム・コンポーネントがデータ・サーバ、Replication Server、または Replication Agent にログインするために使用する名前です。
- **ログ転送言語 (LTL)** – 複写コマンド言語 (RCL) のサブセットです。プライマリ・データベースのトランザクション・ログから取得した情報は、RepAgent などの Replication Agent によって、LTL コマンドを使用して、Replication Server に送信されます。
- **Log Transfer Manager (LTM)** – Sybase SQL Server 用の Replication Agent プログラムです。「*Replication Agent*」と「*RepAgent スレッド*」も参照してください。
- **メンテナンス・ユーザ** – Replication Server がレプリケート・データを管理するために使用するデータ・サーバのログイン名です。ほとんどのアプリケーションでは、メンテナンス・ユーザのトランザクションは複写されません。
- **マテリアライゼーション** – プライマリ・データベースからレプリケート・データベースへ、サブスクリプションによって指定されたデータをコピーする処理です。これによって、レプリケート・テーブルが初期化されます。レプリケート・データはネットワークを介して転送するか、またはサブスクリプションが大量のデータを扱う場合は、メディアからロードできます。「*アトミック・マテリアライゼーション*」、「*バルク・マテリアライゼーション*」、「*非マテリアライゼーション*」、「*ノンアトミック・マテリアライゼーション*」も参照してください。
- **マテリアライゼーション・キュー** – マテリアライゼーションまたはマテリアライゼーション解除されているサブスクリプションに関連したメッセージをスプールするために使用されるステーブル・キューです。
- **消失ロー** – プライマリ・テーブルには存在するが、そのテーブルの複写コピーには存在しないローです。
- **混合バージョン・システム** – ソフトウェア・バージョンとサイト・バージョンの違いによって異なる機能を持った、ソフトウェア・バージョンの異なる Replication Server がある複写システムです。混合バージョン・サポートは、システム・バージョンが 11.0.2 以降の場合のみ使用できます。

  たとえば、Replication Server バージョン 11.5 以降とバージョン 11.0.2 がある複写システムは、混合バージョン・システムです。バージョン 11.0.2 以降の Replication Server では、特定の新機能の使用がシステム・バージョンによって制限されていますが、それより前のバージョンではこの機能がサポートされていないため、バージョン 11.0.2 より前の Replication Server を持った複写システムは、混合バージョン・システムではありません。「*サイト・バージョン*」と「*システム・バージョン*」も参照してください。
- **カラム数の増加** – 複写定義には 251 ～ 1,024 のカラムを含めることができます。カラム数の増加は、Replication Server バージョン 12.5 以降でサポートされています。
- **Multi-Site Availability (MSA)** – テーブル、ファンクション、トランザクション、システム・ストアド・プロシージャ、DDL などのデータベース・オブジェク

トを、プライマリ・データベースからレプリケート・データベースへ複写する方法です。「*データベース複写定義*」も参照してください。

- **マルチパス・レプリケーション** – 送信元データベースからターゲット・データベースへのデータの並列パスを有効にすることによってパフォーマンスを向上させる Replication Server の機能。マルチパス・レプリケーションは、ウォーム・スタンバイ環境と Multi-Site Availability (MSA) 環境で設定できます。これらの複数のパスではデータが個別に処理され、それらのパス間のトランザクションの一貫性を必要とせずにデータ・セットを並列処理できる場合に適用されます。パス内でのデータ整合性を維持しますが、さまざまなパス間でのコミット順には従いません。
- **ネーム・スペース** – オブジェクト名がユニークでなければならない範囲 (スコープ) です。
- **ノンアトミック・マテリアライゼーション** – マテリアライゼーションのメソッドの 1 つです。これは、holdlock を使わずに 1 つのオペレーションで、ネットワークを介してプライマリ・データベースからレプリケート・データベースへサブスクリプション・データをコピーします。データの転送中もプライマリ・テーブルを変更できるので、レプリケート・データベースとプライマリ・データベース間で一時的に不一致が生じる可能性があります。データは、レプリケート・データベースのトランザクション・ログが満杯にならないように、トランザクションごとに 10 ローずつ挿入する方法を使用して適用されます。ノンアトミック・マテリアライゼーションは、**create subscription** コマンドのオプションのメソッドです。「*オートコレクション*」、「*アトミック・マテリアライゼーション*」、「*非マテリアライゼーション*」、「*バルク・マテリアライゼーション*」も参照してください。
- **ネットワークベース・セキュリティ** – ネットワーク上のデータを安全に転送することです。Replication Server は、ユーザの認証、統一化ログイン、Replication Server 間の安全なメッセージ転送などのサード・パーティのセキュリティ・メカニズムをサポートします。
- **非マテリアライゼーション** – マテリアライゼーションのメソッドの 1 つです。サブスクリプション・データがレプリケート・サイトにすでに存在する場合、サブスクリプションを作成できます。**without materialization** 句を指定して **create subscription** コマンドを使用してください。このメソッドを使用して、テーブル複写定義とファンクション複写定義へのサブスクリプションを作成できます。「*アトミック・マテリアライゼーション*」と「*バルク・マテリアライゼーション*」も参照してください。
- **オンライン・トランザクション処理 (OLTP) アプリケーション** – データ修正 (挿入、削除、更新) を伴うさまざまなトランザクションを頻繁に実行するデータベース・クライアント・アプリケーションです。
- **オリジン・キュー ID (qid)** – `qid` は、RepAgent によって形成され、Replication Server に渡された各ログ・レコードをユニークに識別します。`date`、

timestamp、およびデータベース世代番号が含まれます。「*データベース生成番号*」も参照してください。

- **孤立したロー** – テーブルの複写コピーにあるローの中で、アクティブなサブスクリプションと一致しないローのことをいいます。
- **アウトバウンド・キュー** – メッセージをスプールするのに使用するステーブル・キューです。DSI アウトバウンド・キューは、レプリケート・データベースへのメッセージを、RSI アウトバウンド・キューは、レプリケート Replication Server へのメッセージをスプールします。
- **並列 DSI** – 単一の DSI スレッドではなく、並列で機能する複数の DSI スレッドを使用して、レプリケート・データ・サーバにトランザクションが適用されるようにデータベース・コネクションを設定する方法です。「*コネクション*」と「*データ・サーバ・インタフェース (DSI)*」も参照してください。
- **パラメータ** – プロシージャの実行時に提供される値を表す識別子です。ファンクション文字列で使用するパラメータ名は @ 記号で始まります。プロシージャをファンクション文字列から呼び出すと、Replication Server はパラメータ値をそのまま変更しないでデータ・サーバへ渡します。「*サーチャブル・パラメータ*」も参照してください。
- **親クラス** – 派生クラスが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です。「*ファンクション文字列クラス*」と「*派生クラス*」も参照してください。
- **パーティション** – Replication Server が、ステーブル・キューを格納するために使用するロー・ディスク・パーティションまたはオペレーティング・システムのファイルです(オペレーティング・システムのファイルはテスト環境でのみ使用してください)。
- **物理コネクション** – 「*コネクション*」を参照してください。
- **プライマリ・データ** – 複写システムの中で、複写の対象となるデータ・セットのことです。プライマリ・データは、データ・サーバによって管理されます。このデータ・サーバは、データのサブスクリプションがあるすべての Replication Server で認識されています。
- **プライマリ・データベース** – 複写システムによって別のデータベースに複写されるデータが格納されたデータベースです。
- **プライマリ・フラグメント** – 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。
- **プライマリ・キー** – 各ローをユニークに識別するテーブル・カラムのセットです。
- **プライマリ・サイト** – ファンクション文字列クラスまたはエラー・クラスが定義されている Replication Server です。「*エラー・クラス*」と「*ファンクション文字列クラス*」を参照してください。

- **プリンシパル・ユーザ** – アプリケーションを開始するユーザです。ネットワークベース・セキュリティを使用する場合、Replication Server はプリンシパル・ユーザとしてリモート・サーバにログインします。
- **プロファイル** – プロファイルを使用すると、事前に定義されたプロパティのセットでコネクションを設定できます。
- **射影** – テーブルの垂直方向のスライスです。テーブル・カラムのサブセットを表します。
- **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グループです。パブリケーションを使用すると、関連するテーブルかストアド・プロシージャまたはその両方の複写定義を収集して、グループとしてそれらのサブスクリプションを作成できます。送信元 Replication Server のパブリケーション内で、アーティクルとして複写定義を収集し、送信先 Replication Server でパブリケーション・サブスクリプションを使用してそれらにサブスクリプションを作成できます。「*アーティクル*」と「*パブリケーション・サブスクリプション*」も参照してください。
- **パブリケーション・サブスクリプション** – パブリケーションへのサブスクリプションです。「*アーティクル*」と「*パブリケーション*」も参照してください。
- **パブリッシュ・データ型** – レプリケート・データ・サーバにおけるカラム・レベル変換後 (続いてクラス・レベル変換をする場合はその前) のカラムのデータ型です。パブリッシュ・データ型は、Replication Server 基本データ型か、ターゲット・データ・サーバのデータ型に対する UDD のどちらかでなければなりません。パブリッシュ・データ型が複写定義から省略された場合、デフォルトで宣言したデータ型になります。
- **クエリ** – データベース管理システムで、指定した基準を満たすデータを取得するための要求です。SQL データベース言語では、クエリを指定するときに **select** コマンドを使用します。
- **クワイス状態** – すべての更新がその送信先に送信された状態の複写システムのことをクワイス状態の複写システム (静止している複写システム) といいます。Replication Server コマンドやプロシージャの中には、最初に複写システムをクワイスすることを必要とするものがあります。
- **引用符付き識別子** – スペースや非英数字などの特殊文字が含まれる、アルファベット以外の文字で始まる、または予約語に相当するオブジェクト名は、正しく解析されるように二重引用符文字で囲む必要があります。
- **Real-Time Loading (RTL) –** Sybase IQ データベースへの High-Volume Adaptive Replication (HVAR)。HVAR の変更を Sybase IQ レプリケート・データベースに適用するには、関連するコマンドとプロセスを使用します。「*High Volume Adaptive Replication*」を参照してください。
- **リモート・プロシージャ・コール (RPC) –** リモート・サーバに常駐しているプロシージャを実行するための要求です。プロシージャを実行するサーバには、Adaptive Server、Replication Server、または Open Server を使用して構築されたサーバなどがあります。プロシージャの実行要求は、これらのサーバやクライ

アント・アプリケーションから発行できます。RPC 要求のフォーマットは、Sybase Client/Server Interfaces の一部です。

- **RepAgent スレッド –** Adaptive Server データベース用の Replication Agent です。RepAgent は Adaptive Server のスレッドです。プライマリ・データベースから Replication Server にトランザクション・ログ情報を転送して、他のデータベースに分配します。
- **レプリケート・データベース –** 複写システムによって別のデータベースから複写されたデータが格納されたデータベースです。
- **複写ファンクションの配信 –** ファンクション複写定義に対応するストアド・プロシージャを送信元データベースから送信先データベースに複写する方法です。「*適用ファンクション*」、「*要求ファンクション*」、「*ファンクション複写定義*」も参照してください。
- **複写ストアド・プロシージャ – sp_setrepproc** または **sp_setreplicate** システム・プロシージャを使用して、複写するようにマーク付けされた Adaptive Server ストアド・プロシージャです。複写ストアド・プロシージャは、ファンクション複写定義またはテーブル複写定義に関連付けることができます。「*複写ファンクションの配信*」と「*非同期プロシージャ配信*」も参照してください。
- **複写テーブル –** 複数サイトのデータベースで、Replication Server が一部または全部を管理するテーブルです。これらのテーブルのうち、システム・プロシージャの **sp_setreptable** または **sp_setreplicate** を使用して複写するようにマーク付けされた 1 つのバージョンがプライマリ・バージョンで、それ以外のすべてのバージョンは複写コピーです。
- **Replication Agent –** プライマリ・データへの修正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、データベース・サーバから Replication Server に転送するプログラムまたはモジュールです。RepAgent は、Adaptive Server データベース用の Replication Agent です。
- **複写コマンド言語 (RCL) –** Replication Server の情報を管理するために使用するコマンドです。
- **複写定義 –** サブスクリプションを作成するためのテーブルの定義です。複写定義は Replication Server によって管理され、この中で複写されるカラムとテーブルのプライマリ・バージョンがあるロケーションが指定されています。

  ファンクションの複写定義も作成できます。複写定義がテーブルに関するものかファンクションに関するものかを区別するために、「テーブル複写定義」という用語を使用することもあります。「*ファンクション複写定義*」も参照してください。
- **Replication Server –** Sybase のサーバ・プログラムです。通常、LAN 上で複写データを管理し、同じ LAN または WAN 上にある別の Replication Server から受け取ったデータのトランザクションを処理します。
- **Replication Server インタフェース (RSI) –** 送信先 Replication Server へログインし、RSI アウトバウンド・ステーブル・キューから送信先 Replication Server へコマンドを転送するスレッドです。プライマリまたは中間 Replication Server か

らコマンドを受け取る送信先 Replication Server ごとに、1 つの RSI スレッドが存在します。「*アウトバウンド・キュー*」と「*ルート*」も参照してください。

- **Replication Monitoring Services (RMS) –** Sybase Unified Agent Framework (UAF) を使用して構築された小さな Java アプリケーションで、複写環境のモニタとトラブルシューティングを行います。
- **複写システム管理者 –** Replication Server の定型作業を管理するシステム管理者です。
- **Replication Server システム・データベース (RSSD) –** Replication Server のシステム・テーブルを格納する Adaptive Server データベースです。Replication Server システム・テーブルを、RSSD と SQL Anywhere (SA) ERSSD のどちらに格納するかを選択できます。「*Embedded Replication Server システム・データベース (ERSSD)*」も参照してください。
- **Replication Server システム Adaptive Server –** Replication Server のシステム・テーブル (RSSD) を格納するデータベースがある Adaptive Server です。
- **複写システム –** 複数のデータベースにデータを複写することで、リモート・ユーザがそれぞれのローカル・データにアクセスできるようにするデータ処理システムです。複写システムは Replication Server を基にして構成され、Replication Agent やデータ・サーバのような他のコンポーネントも含まれています。
- **複写システム・ドメイン –** 同じ ID サーバを使用する複写システムのすべてのコンポーネントです。
- **要求ファンクション –** ファンクション複写定義に対応する複写ファンクションであり、Replication Server によってプライマリ・データベースからレプリケート・データベースに配信されます。要求ファンクションがストアド・プロシージャにパラメータ値を渡し、そのストアド・プロシージャがレプリケート・データベースで実行されます。ストアド・プロシージャは、プライマリ・サイトと同じユーザによってレプリケート・サイトで実行されます。「*複写ファンクションの配信*」、「*要求ファンクション*」、「*ファンクション複写定義*」も参照してください。
- **resync marker –** Replication Agent を再同期モードで再開すると、Replication Agent は、再同期処理が進行中であることを示すデータベース再同期マーカを Replication Server に送信します。Replication Agent は最初のメッセージとして再同期マーカを送信してから、SQL データ定義言語 (DDL: data definition language) またはデータ操作言語 (DML: data manipulation language) のトランザクションを送信します。
- **ルート –** 送信元 Replication Server から送信先 Replication Server への一方向のメッセージ・ストリームです。ルートは、データ修正コマンド (RSSD に対するものを含む) と複写ファンクションまたはストアド・プロシージャを Replication Server 間でやりとりします。「*直接ルート*」と「*間接ルート*」も参照してください。

- **ルート・バージョン** – ルートの送信元と送信先の Replication Server のサイト・バージョン番号のうち、低い方の番号です。Replication Server バージョン 11.5 以降では、レプリケート・サイトに送信するデータを決定するのにルート・バージョン番号を使用します。「*サイト・バージョン*」も参照してください。
- **ロー・マイグレーション** – テーブルのプライマリ・バージョン内のローでカラム値が変更されたとき、テーブルのレプリケート・バージョン内の対応するローも、サブスクリプションの **where** 句内の値の比較に基づいて挿入または削除されるプロセスです。
- **SQL Server** – 11.5 より前の Sybase リレーショナル・データベース・サーバです。
- **SQL 文の複写** – SQL 文の複写では、Replication Server は、個々のローの変更ではなく、プライマリ・データを変更した SQL 文をトランザクション・ログから受け取ります。Replication Server は、SQL 文をレプリケート・サイトに適用します。RepAgent は、SQL データ操作言語 (DML) と個々のローの変更の両方を送信します。設定に応じて、Replication Server が、個々のローの変更によるログの複写または SQL 文の複写のどちらかを選択します。
- **スキーマ** – データベースの構造体です。DDL コマンドとシステム・プロシージャは、データベースに格納されているシステム・テーブルを変更します。Replication Server バージョン 11.5 以降と Adaptive Server バージョン 11.5 以降を使用している場合には、サポートされている DDL コマンドとシステム・プロシージャは、スタンバイ・データベースに複写できます。
- **サーチャブル・カラム** – 複写するローをサイトで制限するために、サブスクリプションまたはアーティクルの **where** 句で指定できる複写テーブル内のカラムです。
- **サーチャブル・パラメータ** – サブスクリプションの **where** 句で指定できる複写ストアド・プロシージャのパラメータです。このパラメータを使用して、ストアド・プロシージャを複写するかどうかを決定します。「*パラメータ*」も参照してください。
- **セカンダリ・トランケーション・ポイント** – 「*トランケーション・ポイント*」を参照してください。
- **サイト** – 最低でも Replication Server、データ・サーバ、データベースで構成され、場合によっては Replication Agent も含まれるインストレーション環境で、通常は地理的に離れた場所にあります。各サイトのコンポーネントは、WAN を介して複写システムにある他のサイトのコンポーネントに接続されます。「*プライマリ・サイト*」も参照してください。
- **サイト・バージョン** – 個々の Replication Server のバージョン番号です。サイト・バージョンが一度あるレベルに設定されると、Replication Server でそのレベル特有の機能が有効になり、レベルをダウングレードすることはできません。「*ソフトウェア・バージョン*」、「*ルート・バージョン*」、「*システム・バージョン*」も参照してください。

- **ソフトウェア・バージョン** – 個々の Replication Server のソフトウェア・リリースのバージョン番号です。「*サイト・バージョン*」と「*システム・バージョン*」も参照してください。
- **ステーブル・キュー・マネージャ (SQM)** – ステーブル・キューを管理するスレッドです。インバウンド・キュー、アウトバウンド・キューのいずれの場合でも、Replication Server がアクセスするステーブル・キューに対して、それぞれ 1 つのステーブル・キュー・マネージャ (SQM) スレッドがあります。
- **ステーブル・キュー・トランザクション (SQT) インタフェース** – コミット順にトランザクション・コマンドを再構築するスレッドです。ステーブル・キュー・トランザクション (SQT) インタフェース・スレッドは、インバウンド・ステーブル・キューを読み取って、トランザクションをコミット順に配列し、それらをディストリビュータ (DIST) スレッドと DSI スレッドのうち、SQT によるトランザクションの並び替えを要求した方に送信します。
- **ステーブル・キュー** – Replication Server が、ルートまたはデータベース・コネクション用のメッセージを格納するための蓄積転送キューです。ステーブル・キューに書き込まれたメッセージは、送信先の Replication Server またはデータベースに配信されるまで、このキューに格納されます。Replication Server は、割り当てられたディスク・パーティションを使用してステーブル・キューを構築します。「*インバウンド・キュー*」、「*アウトバウンド・キュー*」、「*マテリアライゼーション・キュー*」も参照してください。
- **スタンドアロン・モード** – リカバリ処理を開始するために使用する Replication Server の特別な動作モードです。
- **スタンバイ・データベース** – ウォーム・スタンバイ・アプリケーションでは、アクティブ・データベースからデータ変更を受信し、そのデータベースのバックアップとして機能するデータベースのことです。「*ウォーム・スタンバイ・アプリケーション*」も参照してください。
- **ストアド・プロシージャ** – Adaptive Server データベースに名前付きで格納されている SQL 文とオプションのフロー制御文の集まりです。Adaptive Server が提供するストアド・プロシージャは、システム・プロシージャと呼ばれます。Replication Server ソフトウェアには、RSSD に問い合わせるストアド・プロシージャがいくつか組み込まれています。
- **サブスクリプション** – 指定したサイトのレプリケート・データベースにあるテーブルの複写コピー、またはテーブルからのローのセットを管理するために、Replication Server に対して行う要求のことです。適用ファンクション配信を行うために、ファンクション複写定義のサブスクリプションも作成できます。
- **サブスクリプション・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*」を参照してください。
- **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*」を参照してください。

- **サブスクリプション・マイグレーション** – 「*ロー・マイグレーション*」を参照してください。
- **Sybase Central** – Sybase および Powersoft 製品を管理する共通のインタフェースを提供するグラフィカルなツールです。Replication Server は、Sybase Central プラグインとして Replication Manager を使用します。「*Replication Monitoring Services (RMS)*」も参照してください。
- **対称型マルチプロセッシング (SMP)** – マルチプロセッサ・プラットフォームで、アプリケーションのスレッドを並列に実行できる機能です。Replication Server は、サーバのパフォーマンスと効率が高められる SMP をサポートしています。
- **同期コマンド** – クライアントが完了ステータスを受信後、初めて完了したとみなすコマンドです。
- **システム・ファンクション** – あらかじめ定義され、Replication Server 製品に組み込まれているファンクションです。**rs_begin** などの複写アクティビティを調整するシステム・ファンクション、または **rs_insert**、**rs_delete**、**rs_update** などのデータ操作のオペレーションを実行するシステム・ファンクションがあります。
- **システム提供クラス** – Replication Server が提供するエラー・クラス `rs_sqlserver_error_class` とファンクション文字列クラス `rs_sqlserver_function_class`、`rs_default_function_class`、`rs_db2_function_class` のことです。ファンクション文字列は、システムで提供されるファンクション文字列クラスとこれらのクラスから直接的または間接的に継承する派生クラス用に自動的に生成されます。「*エラー・クラス*」と「*ファンクション文字列クラス*」も参照してください。
- **システム・バージョン** – リリース 11.0.2 以前の Replication Server に対して、新しい機能が有効なバージョンを表す複写システムのバージョン番号です。このバージョン番号より低いバージョンには、Replication Server をダウングレードまたはインストールできません。Replication Server バージョン 11.5 では、特定の新機能を使用するために、サイト・バージョン 1150 と最低でもシステム・バージョン 1102 が必要です。「*混合バージョン・システム*」、「*サイト・バージョン*」、「*ソフトウェア・バージョン*」も参照してください。
- **テーブル複写定義** – 「*複写定義*」を参照してください。
- **テーブル・サブスクリプション** – テーブル複写定義に対応するサブスクリプションです。
- **スレッド** – Replication Server 内で実行されるプロセスです。Sybase Open Server で構築された Replication Server は、マルチスレッド・アーキテクチャに基づいています。各スレッドは、ユーザ・セッションを管理したり、Replication Agent または別の Replication Server からメッセージを受信したり、メッセージをデータベースに適用したりする特定のファンクションを実行します。「デー

タ・サーバ・インタフェース (DSI)」、「*ディストリビュータ*」、「*Replication Server インタフェース (RSI)*」も参照してください。

- **トランザクション** – 文をグループ化するためのメカニズムです。このメカニズムによって、文はグループ内の単なる構成単位として扱われ、グループ内のすべての文が実行されるか、グループ内の文がまったく実行されないことになります。
- **Transact-SQL** – Adaptive Server で使用するリレーショナル・データベース言語です。これは、標準 SQL (Structured Query Language) をベースとして、Sybase の拡張機能を付加したものです。
- **トランケーション・ポイント** – プライマリ・データを保存している Adaptive Server データベースには、トランザクション・ログ内で Adaptive Server がどこまで処理を完了したかを示すアクティブなトランケーション・ポイントがあります。これをプライマリ・トランケーション・ポイントといいます。

  Adaptive Server データベースの RepAgent が管理するのが、セカンダリ・トランケーション・ポイントで、Replication Server に正常に送信されたログの部分と、まだ送信されていない部分とを分けるために、トランザクション・ログ内の場所にマークを付けます。セカンダリ・トランケーション・ポイントにより、ログの一部がトランケートされる前に、各オペレーションが必ず複写システムへ入力されることが可能になります。
- **ユーザ定義ファンクション** – このファンクションを使用すると、Replication Server を使用して、複写システムのサイト間で複写ファンクションまたは非同期ストアド・プロシージャを配信するカスタム・アプリケーションを作成できます。複写ファンクションの配信では、ファンクション複写定義を作成すると、Replication Server によって自動的にユーザ定義ファンクションが作成されます。
- **変数** – 「*ファンクション文字列変数*」を参照してください。
- **バージョン** – *混合バージョン・システム*

  「*混合バージョン・システム*」、「*サイト・バージョン*」、「*ソフトウェア・バージョン*」、「*システム・バージョン*」を参照してください。
- **ウォーム・スタンバイ・アプリケーション** – Replication Server を使用して、アクティブ・データベースと呼ばれるデータベースに対するスタンバイ・データベースを管理するアプリケーションです。アクティブ・データベースで障害が発生した場合、Replication Server とクライアント・アプリケーションはデータベースをスタンバイ・データベースに切り替えられます。
- **広域ネットワーク (WAN)** – データ通信回線で接続されているローカル・エリア・ネットワーク (LAN) のシステムです。
- **ワイド・カラム** – `char`、`varchar`、`binary`、`varbinary`、`unichar`、`univarchar`、または Java `inrow` データで構成されている、255 バイトより大

きい複写定義のカラムです。ワイド・カラムは、Replication Server バージョン 12.5 以降でサポートされています。

- **ワイド・データ** – データ・サーバのデータ・ページのサイズを上限とする、幅の広いデータ・ローです。Adaptive Server は、2K、4K、8K、16K のページ・サイズをサポートしています。ワイド・データは、Replication Server バージョン 12.5 以降でサポートされています。
- **ワイド・メッセージ** – 複数のブロックにまたがる 16K より大きいメッセージです。ワイド・メッセージは、Replication Server バージョン 12.5 以降でサポートされています。

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

A

B

C

D

E

F

G

H

I

L

M

N

O

P

Q

R

S

T

U

W

X

あ

い

う

え

お

か

き

し

す

せ

そ

た

ち

つ

て

と

に

の

は

ひ

ふ

へ

ま

め

も

ゆ

ろ