ORACLE

Oracle GoldenGateの ベスト・プラクティス：

DDLとDML CDRを使用したアクティブ/アクティブ構成

2020年2月

公開

本書の目的

このドキュメントは、もっとも効率的な方法で内容を紹介することを目的としているため、多数の重要かつ複雑な概念について簡単に触れ、それぞれについての詳細な説明は省きます。読者が製品をよく理解し、可用性の高いOracle GoldenGate環境を適切に設計し、実装できるように支援することがこのドキュメントの目的です。したがって、実装を適切に行う上で非常に重要な設計、ユニット・テスト、統合テストのアクティビティは、このガイドでは意図的に省かれている点に留意してください。すべての例は現状のまま提供されます。カスタマイズされた実装には、オラクルのコンサルティング・サービスが強く推奨されます。また、具体的な本番環境のドキュメントを確認する必要があります。

免責事項

本文書には、ソフトウェアや印刷物など、いかなる形式のものも含め、オラクルの独占的な所有物である占有情報が含まれます。この機密文書へのアクセスと使用は、締結および遵守に同意したOracle Software License and Service Agreementの諸条件に従うものとします。本文書と本文書に含まれる情報は、オラクルの事前の書面による同意なしに、公開、複製、再作成、またはオラクルの外部に配布することはできません。本文書は、ライセンス契約の一部ではありません。また、オラクル、オラクルの子会社または関連会社との契約に組み込むことはできません。

本書は情報提供のみを目的としており、記載した製品機能の実装およびアップグレードの計画を支援することのみを意図しています。マテリアルやコード、機能の提供をコミットメント（確約）するものではなく、購買を決定する際の判断材料になさらないでください。本書に記載されている機能の開発、リリース、および時期については、弊社の裁量により決定されます。製品アーキテクチャの性質上、コードが大幅に不安定化するリスクなしに、本書に記載されているすべての機能を安全に含めることができない場合があります。

目次

本書の目的 **2**
免責事項 **2**
概要 **4**
はじめに **4**
前提条件： **5**
アクティブ/アクティブ構成でOGGを実装する手順 **5**
OGG管理者とOGGデータベースのユーザー・アカウントの作成 6
OGGソフトウェアのインストール 6
DDLレプリケーションを有効化するためのデータベースの構成 6
サプリメンタル・ロギングの有効化 6
フィルタリングおよび含めるDDLオブジェクトとスキーマの決定 7
追加のDDLパラメータ 9
Extract（プライマリおよびセカンダリ） 9
Replicat（プライマリおよびセカンダリ） 9
アクティブ/アクティブDMLに必要なパラメータ 9
CDRのパラメータ 10
順序の変更 11
構成例 11
CDRとDDLレプリケーションの例で使用される表 11
ソースおよびターゲットのManager 13
ソースのExtract 13
ソースのポンプ 14
ソースのReplicat 14
ターゲットのExtract 15
ターゲットのポンプ 15
ターゲットのReplicat 15
ソースのOGGプロセスの作成 16
ターゲット・データベースのインスタンス化 17
ターゲットのOGGプロセスの作成 17
構成例のテスト・スクリプト 18
挿入スクリプト 18
更新スクリプト 18
更新の競合 18
挿入の競合 19
削除の競合 19
結論 **20**

概要

このドキュメントでは、DDLレプリケーションが有効化され、組込み機能であるDMLの競合検出および解決（CDR）が使用されるアクティブ/アクティブ環境で、Oracle GoldenGate（OGG）を構成するためのOracle GoldenGateのベスト・プラクティスとガイドラインについて説明します。このドキュメントは、Oracle Database管理者（DBA）、Oracle GoldenGateソフトウェア製品の基礎知識をある程度持つOracle開発者、およびOracle GoldenGate管理者を対象としています。このドキュメントの目的は、オラクルが提供している一連の既存ドキュメントの基礎部分を補完することです。

- 読者はOracle GoldenGate製品およびコンセプトについての基礎知識がある
- Oracle GoldenGateバージョン11.2.1以降を使用している
- Oracleバージョン10.2.0.5以降を使用している
- 使用OS：Oracle GoldenGateがサポートされるすべてのOracle向けプラットフォーム

はじめに

Oracle GoldenGateは、トランザクション・ログからデータを取得し、同種および異種のデータベースやメッセージ・システムに配信します。Oracle GoldenGateのアーキテクチャは柔軟で分離されているため、実質的にどのようなデータベース・レプリケーション・シナリオの実装にも使用できます。

レプリケーション中に競合が発生する可能性があるアクティブ/アクティブのレプリケーション環境では、標準としてではなく、例外として実施される競合管理スキームを確立することが強く推奨されます。OGGバージョン11.2以前では、検出の場合は異なるものの、解決は、カスタムのSQLコードを使用して処理される必要がありました。レプリケーション中に競合が発生する可能性は通常は低いため、レコードをターゲットに適用する前に常に競合の有無を確認するのは効率的ではありません。しかし一方で、競合に遭遇した場合は必ず適切な方法で処理することも極めて重要です。

ソース・データベースとターゲット・データベースの両方でDDLがアクティブに発行されている環境にDDLレプリケーションを設定することは、通常は推奨されません。DDLに使用できるCDR機能は現在はありません。そのため、DDLは一度にシステムの1つのみから発行される必要があります。もしくは、異なる複数のシステムから発行された場合は、DDL操作は同じオブジェクトで発生しないことが必要です。そうでない場合は、DDLの競合が発生し、DDLの問題とDMLの潜在的なレプリケーションの問題が原因となり、レプリケーションは失敗に終わります。

このドキュメントの目的は、アクティブ/アクティブ環境でDDLを構成し、DML操作の競合を検出および解決する組込み機能のCDRを有効化するための一般的なアプローチを説明することです。組込み機能のCDRを活用することで、カスタム・プログラムの必要性が、完全になくならないまでも、低減されます。

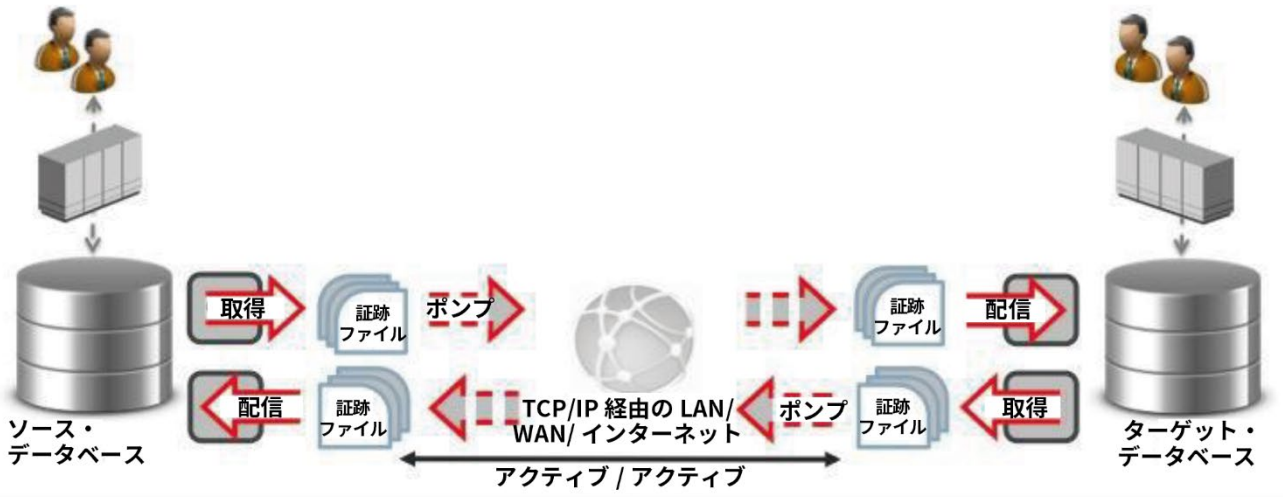
説明するアプローチは、このドキュメントの最重要事項です。OGGを使用して例外として競合を処理するための構成とプロシージャを示しています。また、このアプローチを説明するために、このドキュメントに記載される要件とOracleデータベースに固有のサンプル実装が提供されます。ただし、実際の要件とユースケースに応じて、異なる実装を推進することが必要になる可能性は非常に高いでしょう。そのため、このドキュメントで提供される実装を機能、パフォーマンス、メンテナンスの観点から評価した上で、実際のユースケースで採用することが強く推奨されます。

前提条件：

- OGGバージョン11.2以降が両方のノードにインストールされている。
- DDLレプリケーションが両方のシステムで有効化されている。この手順については、インストール・ガイドに記載されています。
- OGG管理者アカウントがインストール・ガイドの説明どおりに作成されている。

アクティブ/アクティブ構成でOGGを実装する手順

これらの手順は、OGGを実装する、DDLとCDRが有効化されたアクティブ/アクティブ環境に必要です。それぞれの手順では、具体的なコマンドを例として示します。例は、以下のダイアグラムで概略するリソースとプロセスを参照します。このダイアグラムは、OGGの典型的なアクティブ/クティブ構成を描写しています。



OGG管理者とOGGデータベースのユーザー・アカウントの作成

OGGインスタンスには、OSのユーザー・アカウントとデータベースのユーザー・アカウントが必要です。詳細な手順については、Oracle GoldenGateのインストール・ガイドの指示に従ってください。この手順は両方のシステムで実施する必要があります。

OGGソフトウェアのインストール

ソフトウェアの具体的なインストール手順については、Oracle GoldenGateのインストール・ガイドに記載される手順に従ってください。この手順は両方のシステムで実施する必要があります。

DDLレプリケーションを有効化するためのデータベースの構成

DDLレプリケーションのためにデータベースを構成する具体的な手順については、Oracle GoldenGateのインストール・ガイドに記載される手順に従ってください。OGGバージョン11.2以前では、DDLレプリケーションをサポートするために必要なオブジェクトをインストールするには、OGGを停止する必要があります。この手順は両方のシステムで実施する必要があります。ただし、ソース・システムがDDLをサポートするように構成されており、ターゲット・システムがOracle Recovery Manager（Oracle RMAN）を使用してインスタンス化されている場合、ソースのバックアップからターゲット・データベースが作成されたときにこの構成が行われます。

サプリメンタル・ロギングの有効化

最小サプリメンタル・ロギングをデータベース・レベルで有効にする必要があります。また、表レベルでも有効にする必要があります。最小サプリメンタル・ロギングをデータベース・レベルで有効にするには、以下を実行します。

```
SQL> alter database add supplemental log data ;
```

表レベルのサプリメンタル・ロギングは、レプリケーションに不可欠です。ただし、競合の検出および解決（CDR）が有効化されている場合は、キー列のほかに他のフィールドも補足的にログに記録しなければならない場合があります。これは、アクティブ/アクティブでDDLが有効化されている場合に問題となります。表の構造や索引は変更される可能性があるためです。通常はDDLレプリケーション環境では、“ADD SCHEMATRANDATA <schema>”が利用されます。これにより、表の構造や索引が変更された場合に、下層のサプリメンタル・ロギングも自動的に更新されます。CDRで使用される列が常にログに記録される（つまり、あらゆる種類のDMLと常に関係する）場合は、“ADD SCHEMATRANDATA”を使用してサプリメンタル・ロギングを有効化する必要があります。このようなシナリオの良い例として、挿入や更新が行われるたびに入力または変更されるCDRのLAST_UPDATE_DATEなどの列の使用が挙げられます。

スキーマ・レベルでサプリメンタル・ロギングを有効にするには、以下を実行します。

```
GGSCI> ADD SCHEMATRANDATA <schema>
```

キー列のほかにさらなる列がCDRに必要であり、それらの列が通常のDMLの一環として常に更新されるわけではない場合、“ADD TRANDATA <schema>.<table>, COLS (<CDR col1>, <CDR col2>….)”を使用します。“ADD TRANDATA”を使用してサプリメンタル・ロギングを有効化する場合に問題になるのは、表構造または主キー索引が変更されると、サプリメンタル・ロギングがExtractによって変更されることです。Extractによって、追加されたCDR列がサプリメンタル・ロギングから削除されます。そのため、DDLは、OGGの停止中、またはDMLアクティビティがデータベースで停止されているときに、発行される必要があります。また、確実にCDR列も補足的にログに記録されるようにするために、DDLアクティビティが完了したら、サプリメンタル・ロギングにCDR列を再び追加する必要があります。

これが行われない場合、CDRに必要な列がサプリメンタル・ロギングにないために、表の構造や索引を変更するDDLがレプリケーション・エラーを引き起こします。

表レベルでサプリメンタル・ロギングを有効にするには、以下を実行します。

```
GGSCI> ADD TRANDATA <schema>.<table>, COLS ( CDR_COL1, CDR_COL2, … )
```

CDR列が主キーの一部ではなく、DDLの変更中にDMLの発行が必要な場合、アプリケーションは、DDL変更の一環としてサプリメンタル・ロギングを変更する必要があります。このアプローチを利用する場合は、Extractパラメータの“DDLOPTION ADDTRANDATA”を使用しないでください。このパラメータは、表または索引の構造が変更されたときに、キー列のみを含むようにサプリメンタル・ロギングを自動的に変更します。

フィルタリングおよび含めるDDLオブジェクトとスキーマの決定

一部の環境では、すべてのオブジェクトがレプリケートされるわけではありません。DDLレプリケーションのオーバーヘッドを軽減するために、スキーマまたは特定のオブジェクト・タイプを、DDLトリガー自体によって取得対象から除外できます。デフォルトでは、データベースで発行されるすべてのDDLは、DDLトリガーによって取得されます。たとえば、グローバル一時表はOGGでサポートされていません。そのため、この種のオブジェクトはDDLトリガーから除外するのが適切です。また、多くのアプリケーションには複数のスキーマがあります。複数のスキーマのうちの1つのみをレプリケートしている場合、それ以外のスキーマはDDLトリガーから除外するのが適切です。トリガー・レベルでDDLをフィルタリングするには、ddlaux_addrule()関数を使用する必要があります。この関数の詳細については、OGGのインストール・ガイドを参照してください。この関数を使用するために、OGGを停止する必要はありません。

特定のスキーマ“BEE_MDS”とグローバル一時表の取得を除外するには、以下を実行します。

```
SQL> @ddlaux_addrule.sql
Contents of ddlaux_addrule.sql:
```

```
BEGIN
DECLARE
sno NUMBER;
BEGIN
/* スキーマBEE_MDSの除外  */
sno := oragg.ddlaux.addrule(owner_name => 'BEE_MDS');
/* 一時表の除外  */
sno := oragg.ddlaux.addrule(base_obj_property => ddlaux.TB_TEMP, obj_type => ddlaux.TYPE_TABLE);
END;
END;
/
```

おそらくパラメータ・レベルで、よりきめ細かいフィルタリングが必要です。一部のオブジェクトは、Replicatによる実行を許可するsysdba権限が必要です。“ALTER SYNONYM”、“GRANT ON DIRECTORY”は、OGGユーザーにsysdba権限がないためにReplicatで失敗する一般的なコマンドです。パブリック・シノニムでのDDLにもsysdba権限が必要です。DDL EXCLUDEパラメータを使用することで、これらのオブジェクトをExtractから除外できます。

Extractレベルで除外するには、以下をパラメータ・ファイルに追加します。

```
DDL
EXCLUDE MAPPED OBJTYPE 'SYNONYM' OPTYPE ALTER &
EXCLUDE OPTYPE GRANT INSTR 'on directory' &
EXCLUDE OPTYPE GRANT INSTR 'ON DIRECTORY'
```

ほとんどの場合、レプリケートしているスキーマでマッピングされたDDLとマッピングされていないDDLの両方を抽出する必要があります。MAPPEDパラメータとUNMAPPEDパラメータの詳細な説明については、OGGのインストール・ガイドとリファレンス・ガイドを参照してください。

DEMOスキーマでマッピングされたDDLとマッピングされていないDDLをExtractレベルで含めるには、以下を指定します。

```
DDL
INCLUDE MAPPED &
INCLUDE UNMAPPED OBJNAME DEMO.* &
```

追加のDDLパラメータ

DDL問題のトラブルシューティング、DDLデータの管理、メタデータの更新、およびサプリメンタル・ロギングの更新を容易にするには、いくつかの追加パラメータが必要です。

ExtractとReplicatの両方のトラブルシューティングを容易にするには、以下を指定します。

```
---このパラメータはレプリケートされているDDLを報告します。
DDLOPTIONS REPORT
```

表またはキー索引が変更されると、通常は所定の表のサプリメンタル・ロギングも更新する必要があります。サプリメンタル・ロギングの有効化にADD SCHEMATRANDATAが使用された場合、この更新はロックなしで自動的に行われます。サプリメンタル・ロギングの有効化にADD TRANDATAが使用された場合、Extractでサプリメンタル・ロギングを更新する必要があります。この更新は、DDLOPTIONS ADDTRANDATAパラメータを使用して行います。ADD TRANDATAを使用する場合は、タイミングやロックに関する問題を考慮して、DDL変更と同時にDMLを発行しないことが推奨されます。DDLとDMLを同時に発行する必要がある場合は、レプリケーション問題を避けるために、ADD SCHEMATRANDATAを使用してサプリメンタル・ロギングを有効化する必要があります。また、CDRによって使用される列はいずれも、キー列の一部であるか、常にアプリケーションによって変更される列である必要があります。

ExtractでADD TRANDATAが使用される場合にサプリメンタル・ロギングを更新するには、以下を指定します。

```
--- ADDSCHEMATRANDATA SCOTTを使用する場合は必要ありません
--- DDLOPTIONS ADDTRANDATA
```

アクティブ/アクティブ構成では、各Replicatは、取得されたReplicat DDL文をリモートExtractが送信するたびに、オブジェクト・メタデータ・キャッシュを更新するように構成される必要があります。この要件を満たすには、両システムのReplicatパラメータ・ファイルのDDLOPTIONS文でUPDATEMETADATAオプションを使用します。両システムのExtractパラメータ・ファイルでも、DDLOPTIONS文にGETREPLICATESオプションを含める必要があります。

作成されるDDLOPTIONS文は次のようになります。

Extract（プライマリおよびセカンダリ）

```
DDLOPTIONS GETREPLICATES
```

Replicat（プライマリおよびセカンダリ）

```
DDLOPTIONS UPDATEMETADATA
```

DDLが抽出されている間、そのデータを含むDDLオブジェクトでは、オブジェクト・ストレージの要件を適切に維持するために、古いデータが定期的に消去される必要があります。Managerは、古い証跡データを消去する役割を担うのと同様に、古いDDLデータを消去する必要もあります。

Managerパラメータ・ファイルに含まれる古いDDLデータを消去するには、以下を指定します。

```
--- DDLオブジェクトのメンテナンス
USERID oragg, PASSWORD <encrypted_password>, ENCRYPTKEY <keyname>
PURGEDDLHISTORY MINKEEPDAYS 30 MAXKEEPDAYS 60, FREQUENCYHOURS 24
PURGEMARKERHISTORY MINKEEPDAYS 30 MAXKEEPDAYS 60, FREQUENCYHOURS 24
```

アクティブ/アクティブDMLに必要なパラメータ

アクティブ/アクティブDMLレプリケーションは、OGGを使用して非常に簡単に実行できます。Extractは、Replicatデータベース・ユーザーを認識してそのアクティビティを除外する必要があるだけです。

Replicatアクティビティを除外するには、このパラメータをExtractに追加します。

```
TRANLOGOPTIONS EXCLUDEUSER <replicat username>
```

CDRのパラメータ

ExtractとReplicatはどちらも、CDRのパラメータが必要です。Extractでは、抽出される変更前イメージ情報を定義する必要があります。Replicatでは、競合ルールと解決ルールを定義する必要があります。これらのパラメータの多数のオプションについて詳しくは、OGGの管理ガイドを参照してください。

Extractでは、競合検出の変更前イメージを収集するには、LOGALLSUPPCOLSオプションまたはGETUPDATEBEFORESが必要です。LOGALLSUPPCOLSを使用する場合、GETBEFORECOLSを使用して必要な列を定義します。GETBEFORECOLSで定義した列が何であれ、これらの列は補足的にログに記録される必要があります。GETBEFORECOLSでALLオプションを使用すると、補足的にログに記録される列のみが取得されます。ALLオプションを使用しても、実際にすべての列が補足的にログに記録されていない限り、すべての列の変更前イメージが抽出されるわけではありません。NOCOMPRESSDELETESは、完全な削除行を取得します。これを使用しない場合は、削除対象のキー列

```
Extract：
LOGALLSUPPCOLS
--- NOCOMPRESSDELETESを有効にして完全な削除行を取得します。有効にしない場合は、
--- 削除対象のキー列のみが取得されます。
NOCOMPRESSDELETES
--- CDROPTIONSを使用する場合はGETBEFORECOLSが必要です。その場合、
--- 補足的にログに記録されるすべての列が抽出されます。TABLE  demo.demo_users,
GETBEFORECOLS (
ON UPDATE ALL,
ON DELETE ALL);
```

のみが取得されます。

Replicatでは、マップ文で以下を定義する必要があります。

1. 競合を判断するために使用される列（COMPARECOLS）
2. 競合のタイプ（INSERTROWEXISTS、UPDATEROWEXISTS、UPDATEROWMISSING、DELETEROWEXISTS、DELETEROWMISSING）
3. 競合の解決方法。競合解消（RESOLVECONFLICT）には多くのオプションがあります。さらなる例とオプションについては、OGGの管理ガイドとリファレンス・ガイドを参照してください。

```
Replicat：
MAP demo.demo_users, TARGET demo.demo_users,
RESOLVECONFLICT (INSERTROWEXISTS,
(DEFAULT, USEMAX(CREATION_DATE))),
RESOLVECONFLICT (UPDATEROWEXISTS,
(DEFAULT, USEMAX(LAST_UPD_DATE))),
RESOLVECONFLICT (UPDATEROWMISSING,
(DEFAULT, DISCARD)),
RESOLVECONFLICT (DELETEROWEXISTS,
(DEFAULT, DISCARD)),
RESOLVECONFLICT (DELETEROWMISSING,
(DEFAULT, DISCARD)),
COMPARECOLS (
ON UPDATE, KEYINCLUDING(LAST_UPD_DATE),
ON DELETE, ALL));
MAP demo.demo_users, target demo.demo_users_exp,
colmap (USEDEFAULTS,
op_type=@GETENV('GGHEADER', 'OPTYPE'),
```

```
op_time=@GETENV('GGHEADER", 'COMMITTIMESTAMP')),
EXCEPTIONSONLY,
INSERTALLRECORDS;
```

順序の変更

順序は、アクティブ/アクティブ環境でレプリケートされるべきではありません。通常は行の一意の識別子を作成するために使用されます。アクティブ/アクティブ環境では、一方のシステムに奇数値の順序を設定し、もう一方のシステムに偶数値の順序を設定する必要があります。そうすることで、生成された値の競合が発生しません。OGGプロセスが作成されてターゲットのインスタンス化が開始される以前に、ソースの順序を、偶数値または奇数値のみになるように変更する必要があります。

ターゲットのインスタンス化が完了したら、ソースと反対の値になるように順序を変更する必要があります。たとえば、ソースで順序が偶数に変更されている場合は、ターゲットの順序を奇数に変更する必要があります。

マルチマスター環境では、順序は、マルチマスターを構成しているシステムの数だけ増加します。たとえば、あるマルチマスター環境が3つのシステムで構成されている場合、各システムの順序は3ずつ増加します。そのため、現在の順序番号が100であれば、システム1の順序は101から始まるように変更され、3ずつ増加します。同様に、システム2の順序は102から始まるように変更され、3ずつ増加し、システム3の順序は103から始まるように変更され、3ずつ増加します。

構成例

この例は、DDLおよびDMLのアクティブ/アクティブ構成です。使用されているCDR列は、挿入で入力されるCREATION_DATEと、更新操作で変更されるLAST_UPD_DATEです。CREATION_DATEは、挿入の競合を判断して解決するために使用され、LAST_UPD_DATEは、更新の競合を判断して解決するために使用されます。削除の競合では、競合の判断に完全なレコードが使用されます。

解決ロジックでは、競合が発生した場合、直近の挿入のCREATION_DATEと更新のLAST_UPD_DATEを保有するレコードが使用されます。削除では、レコードが一致しない場合、レコード全体が例外表に挿入され、そのレコードはデータベースから削除されません。

CDRとDDLレプリケーションの例で使用される表

```
CREATE TABLE "DEMO"."DEMO_USERS"
( "ID" NUMBER,
"FIRST_NAME" VARCHAR2(30),
"LAST_NAME" VARCHAR2(30),
"CREATION_DATE" TIMESTAMP (6),
"LAST_UPD_DATE" TIMESTAMP (6),
CONSTRAINT "PK_DEMO" PRIMARY KEY ("ID") )
TABLESPACE "USERS";
CREATE TABLE "DEMO"."DEMO_USERS_EXP"
( "ID" NUMBER,
"FIRST_NAME" VARCHAR2(30),
"LAST_NAME" VARCHAR2(30),
"CREATION_DATE" TIMESTAMP (6),
"LAST_UPD_DATE" TIMESTAMP (6),
"OP_TYPE" VARCHAR2(20),
"OP_TIME" TIMESTAMP (6)
)
TABLESPACE "USERS";
```

ソースおよびターゲットのManager

```
PORT 7801
--- OGGで使用されるポートのロックダウン
DYNAMICPORTLIST 7802-7809
--- DDLオブジェクトのメンテナンス
USERID gg_admin, PASSWORD <encrypted_password>, ENCRYPTKEY <keyname>
PURGEDDLHISTORY MINKEEPDAYS 30 MAXKEEPDAYS 60, FREQUENCYHOURS 24
PURGEMARKERHISTORY MINKEEPDAYS 30 MAXKEEPDAYS 60, FREQUENCYHOURS 24
--- 証跡ファイルのメンテナンス
PURGEOLDEXTRACTS ./* , USECHECKPOINTS
---パフォーマンスとアラートのパラメータ
AUTORESTART ER *, RETRIES 3, WAITMINUTES 10, RESETMINUTES 60
DOWNREPORTMINUTES 15
DOWNCRITICAL
LAGCRITICALSECONDS 10
LAGINFOMINUTES 0
LAGREPORTMINUTES 15
```

ソースのExtract

```
EXTRACT e_a_conf
USERID gg_admin, PASSWORD gg_admin
exttrail ./dirdat/ea
--- DDLパラメータ  DDL
INCLUDE MAPPED &
INCLUDE UNMAPPED OBJNAME DEMO.* &
EXCLUDE MAPPED OBJTYPE 'SYNONYM' OPTYPE ALTER &
EXCLUDE OPTYPE GRANT INSTR 'on directory' &
EXCLUDE OPTYPE GRANT INSTR 'ON DIRECTORY'
---このパラメータはレプリケートされているDDLを報告します。
---これはトラブルシューティングで非常に有用です。
DDLOPTIONS REPORT
--- ターゲットのReplicatがメタデータを更新できるようにするために、DDLの変更を取得します
DDLOPTIONS GETREPLICATES
--- ADDSCHEMATRANDATAデモを使用する場合は必要ありません
--- DDLOPTIONS ADDTRANDATA
--- ASMログ・ファイルを読み取るクラシックExtract
TRANLOGOPTIONS DBLOGREADER
--- Replicatアクティビティを除外します
TRANLOGOPTIONS EXCLUDEUSER gg_admin
--- 補足的にログに記録される列の変更前イメージを取得します
LOGALLSUPPCOLS
--- 削除対象のすべての列を取得します
NOCOMPRESSDELETES
--- レプリケートしている表と必要な変更前イメージを定義します。
--- この場合、更新と削除で記録されるすべての列の変更前イメージが
--- 必要です。
TABLE demo.demo_users,
GETBEFORECOLS (
ON UPDATE ALL,
ON DELETE ALL );
```

ソースのポンプ

```
EXTRACT p_a_conf
PASSTHRU
RMTHOST <Target Hostname>, MGRPORT 7801
RMTTRAIL ./dirdat/ra
TABLE demo.*;
```

ソースのReplicat

```
REPLICAT r_a_conf
USERID gg_admin, PASSWORD gg_admin
ASSUMETARGETDEFS
DISCARDFILE ./dirrpt/r_a_conf.dsc APPEND
--- 削除カスケード制約とトリガーによって作成されるデータに関する
--- 問題を回避するために、トリガーと制約を遅延させます
DBOPTIONS DEFERREFCONST
DBOPTIONS SUPPRESSTRIGGERS
--- DDLパラメータ
DDL INCLUDE ALL
DDLOPTIONS REPORT
--- DDL変更がソースに適用された後にメタデータを更新します
DDLOPTIONS UPDATEMETADATA
--- 競合ルールを設定します
--- USEMAX解決により、挿入または更新の競合で使用される直近の行が
--- 強制的に使用されます。
--- 更新または削除で行がない場合は、
--- レコードが例外表に書き込まれます。
--- 削除で行が存在するものの、その行がすべての列と一致しない場合は、
--- レコードが例外表に書き込まれます。
MAP demo.demo_users, TARGET demo.demo_users,
RESOLVECONFLICT (INSERTROWEXISTS,
(DEFAULT, USEMAX(CREATION_DATE))),
RESOLVECONFLICT (UPDATEROWEXISTS,
(DEFAULT, USEMAX(LAST_UPD_DATE))),
RESOLVECONFLICT (UPDATEROWMISSING,
(DEFAULT, DISCARD)),
RESOLVECONFLICT (DELETEROWEXISTS,
(DEFAULT, DISCARD)),
RESOLVECONFLICT (DELETEROWMISSING,
(DEFAULT, DISCARD)),
COMPARECOLS (
ON UPDATE, KEYINCLUDING(LAST_UPD_DATE),
ON DELETE, ALL);
--- OPTYPEとCOMMITTIMESTAMPを使用して、競合が例外表に書き込まれます
MAP demo.demo_users, target demo.demo_users_exp,
colmap (USEDEFAULTS,
op_type=@GETENV('GGHEADER','OPTYPE'),
op_time=@GETENV('GGHEADER', 'COMMITTIMESTAMP')), E
XCEPTIONSONLY,
INSERTALLRECORDS;
```

ターゲットのExtract

```
EXTRACT e_b_conf
USERID gg_admin, PASSWORD gg_admin
exttrail ./dirdat/eb
--- DDLパラメータ  DDL INCLUDE MAPPED &
INCLUDE UNMAPPED OBJNAME DEMO.* &
EXCLUDE MAPPED OBJTYPE 'SYNONYM' OPTYPE ALTER &
EXCLUDE OPTYPE GRANT INSTR 'on directory' &
EXCLUDE OPTYPE GRANT INSTR 'ON DIRECTORY'
---このパラメータはレプリケートされているDDLを報告します。
---これはトラブルシューティングで非常に有用です。
DDLOPTIONS REPORT
--- ターゲットのReplicatがメタデータを更新できるようにするために、DDLの変更を取得します
DDLOPTIONS GETREPLICATES
--- ADDSCHEMATRANDATAデモを使用する場合は必要ありません
--- DDLOPTIONS ADDTRANDATA
--- ASMログ・ファイルを読み取るクラシックExtract
TRANLOGOPTIONS DBLOGREADER
--- Replicatアクティビティを除外します
TRANLOGOPTIONS EXCLUDEUSER gg_admin
--- 補足的にログに記録される列の変更前イメージを取得します
LOGALLSUPPCOLS
--- 削除対象のすべての列を取得します
NOCOMPRESSDELETES
--- レプリケートしている表と必要な変更前イメージを定義します。
--- この場合、更新と削除で記録されるすべての列の変更前イメージが
--- 必要です。
TABLE demo.demo_users,
GETBEFORECOLS (
ON UPDATE ALL,
ON DELETE ALL );
```

ターゲットのポンプ

```
EXTRACT p_b_conf
PASSTHRU
RMTHOST <Source Hostname>, MGRPORT 7801
RMTTRAIL ./dirdat/ra
TABLE demo.*;
```

ターゲットのReplicat

```
REPLICAT r_b_conf
USERID gg_admin, PASSWORD gg_admin
ASSUMETARGETDEFS
DISCARDFILE ./dirrpt/r_b_conf.dsc APPEND
--- 削除カスケード制約とトリガーによって作成されるデータに関する
--- 問題を回避するために、トリガーと制約を遅延させます
DBOPTIONS DEFERREFCONST
DBOPTIONS SUPPRESSTRIGGERS
---DDLパラメータ
DDL INCLUDE ALL
```

```
DDLOPTIONS REPORT
--- DDL変更がソースに適用された後にメタデータを更新します
DDLOPTIONS UPDATEMETADATA
--- 競合ルールを設定します
--- USEMAX解決により、挿入または更新の競合で使用される直近の行が
--- 強制的に使用されます。
--- 更新または削除で行がない場合は、
--- レコードが例外表に書き込まれます。
--- 削除で行が存在するものの、その行がすべての列と一致しない場合は、
--- レコードが例外表に書き込まれます。
MAP demo.demo_users, TARGET demo.demo_users,
RESOLVECONFLICT (INSERTROWEXISTS,
(DEFAULT, USEMAX(CREATION_DATE))),
RESOLVECONFLICT (UPDATEROWEXISTS,
(DEFAULT, USEMAX(LAST_UPD_DATE))),
RESOLVECONFLICT (UPDATEROWMISSING,
(DEFAULT, DISCARD)),
RESOLVECONFLICT
(DELETEROWEXISTS, (DEFAULT, DISCARD)),
RESOLVECONFLICT (DELETEROWMISSING,
(DEFAULT, DISCARD)),
COMPARECOLS (
ON UPDATE, KEYINCLUDING(LAST_UPD_DATE),
ON DELETE, ALL);
--- OPTYPEとCOMMITTIMESTAMPを使用して、競合が例外表に書き込まれます
MAP demo.demo_users, target demo.demo_users_exp,
colmap (USEDEFAULTS,
op_type=@GETENV('GGHEADER', 'OPTYPE'),
op_time=@GETENV('GGHEADER', 'COMMITTIMESTAMP')),
EXCEPTIONSONLY,
INSERTALLRECORDS;
```

ソースのOGGプロセスの作成

1. ソース・データベース表でサプリメンタル・ロギングを有効にします。ORACLE_SIDが環境で設定され、最小サプリメンタル・ロギングがデータベースで有効化されていることを確認します。DDLレプリケーションを行う場合は、ADD SCHEMATRANDATAがベスト・プラクティスのアプローチです。

```
GGSCI> DBLOGIN USERID gg_adm PASSWORD gg_adm
GGSCI> ADD SCHEMATRANDATA DEMO
または
GGSCI> ADD TRANDATA DEMO.DEMO_USER
```

2. ソースおよびターゲットでパラメータ・ファイルを使用してManagerを作成します

3. ソースでExtractプロセスを作成します

a. 統合Extractのためにデータベースに Extractを登録します。環境にORACLE_SIDが設定されていることを確認します。

```
GGSCI> DBLOGIN USERID gg_adm PASSWORD gg_adm
GGSCI> REGISTER EXTRACT e_a_conf DATABASE
```

b. Extractを作成します

```
統合Extractの場合
GGSCI> ADD EXTRACT e_a_conf, INTEGRATED TRANLOG, BEGIN NOW
Classic Extractの場合
GGSCI> ADD EXTRACT e_a_conf, TRANLOG, BEGIN NOW
ローカル証跡ファイルの作成
GGSCI> ADD EXTTRAIL ./dirdat/ea, EXTRACT e_a_conf, MEGABYTES 100
```

4. ソースでExtractポンプを作成します

```
GGSCI> ADD EXTRACT p_a_conf, exttrailsource ./dirdat/ea
Create Remote Trail File
GGSCI> ADD RMTTRAIL ./dirdat/ra, EXTRACT p_a_conf, MEGABYTES 100
```

5. ソースでReplicatを作成します。フェイルオーバー・イベントでデータが再実行されないようにするために、チェックポイント表は不可欠です。

```
GGSCI> DBLOGIN USERID gg_adm, PASSWORD gg_adm
GGSCI> ADD REPLICAT r_b_conf, exttrail ./dirdat/rb, CHECKPOINTTABLE ggckpt
```

ターゲット・データベースのインスタンス化

このトピックは、それ自体が完全なホワイト・ペーパーです。この手順の詳細については、メタリンクのドキュメント1276058.1を参照してください。これは一例のため、表は空です。したがって、ターゲット・データベースでただ表を作成するだけで構いません。

ターゲットのOGGプロセスの作成

1. ターゲットでExtractプロセスを作成します

a. 統合Extractのためにデータベースにextractを登録します。環境にORACLE_SIDが設定されていることを確認します。

```
GGSCI> DBLOGIN USERID gg_adm PASSWORD gg_adm
GGSCI> REGISTER EXTRACT e_b_conf DATABASE
```

b. Extractを作成します

```
統合Extractの場合
GGSCI> ADD EXTRACT e_b_conf, INTEGRATED TRANLOG, BEGIN NOW
Classic Extractの場合
GGSCI> ADD EXTRACT e_b_conf, TRANLOG, BEGIN NOW
ローカル証跡ファイルの作成
GGSCI> ADD EXTTRAIL ./dirdat/eb, EXTRACT e_b_conf, MEGABYTES 100
```

2. ターゲットでExtractポンプを作成します

```
GGSCI> ADD EXTRACT p_b_conf, exttrailsource ./dirdat/eb
Create Remote Trail File
GGSCI> ADD RMTTRAIL ./dirdat/rb, EXTRACT p_b_conf, MEGABYTES 100
```

3. ターゲットでReplicatを作成します。フェイルオーバー・イベントでデータが再実行されないようにするために、チェックポイント表は不可欠です。

```
GGSCI> DBLOGIN USERID gg_adm, PASSWORD gg_admin
GGSCI> ADD REPLICAT r_a_conf, exttrail ./dirdat/ra, CHECKPOINTTABLE ggckpt
```

構成例のテスト・スクリプト

さまざまな競合の一部を実証するために、以下のスクリプトが含まれています。競合を作成するスクリプトは、ユーザーgg_adminを使用して、ソースで競合を生成します。このスクリプトではEXCLUDEUSERパラメータを指定しているため、このユーザーはターゲットにレプリケートされません。競合を生成するスクリプトは、レプリケートされるユーザーdemoを使用します。以下がすべてのシェル・スクリプトです。

```
#!/bin/sh
sqlplus -s "/ as sysdba"<<EOFF
insert into demo.demo_users values (1, 'FIRST', 'LAST', sysdate, sysdate); commit;
exit
EOFF
```

挿入スクリプト

更新スクリプト

更新の競合

この更新の競合は競合を発生させますが、last_upd_dateの方が最新のため、レコードがターゲットで更新されます。

```
#!/bin/sh
sqlplus -s "gg_admin/gg_admin"<<EOFF
update demo.demo_users
set last_upd_date=sysdate-2 WHERE id=1;
commit;
Generate update conflict
#!/bin/sh
sqlplus -s "demo/demo"<<EOFF
update demo_users
set first_name='FIRST_NAME_UPD_CON_H',
last_name='LAST_NAME_UPD_CON_H',
last_upd_date=sysdate+5 WHERE id=1;
commit;
exit
EOFF
```

更新の競合の作成

挿入の競合

ここでは、2つの挿入の競合を紹介します。ソースで行を削除してから、各タイプの挿入の競合を生成する必要があります。挿入の低競合では、ターゲットのCREATE_DATEが適用されている行よりも最新であるため、例外表に書き込まれます。挿入の高競合では、CREATE_DATEがターゲットのCREATE_DATEよりも最新であるため、ターゲットのレコードが上書きされます。

```
Create Insert conflict
#!/bin/sh
sqlplus -s "gg_admin/gg_admin"<<EOFF
delete from demo.demo_users;
commit;
exit
EOFF
Generate Low Insert conflict
#!/bin/sh
sqlplus "demo/demo"<<EOFF
insert into demo.demo_users values (1, 'FIRST_CON_L','LAST_CON_L',sysdate-1,sysdate-1);
commit;
exit
EOFF
Generate High Insert conflict
#!/bin.sh
sqlplus "demo/demo"<<EOFF
insert into demo.demo_users values (1, 'FIRST_CON_H','LAST_CON_H',sysdate+1,sysdate+1);
commit;
exit
EOFF
```

削除の競合

この削除の競合では、削除レコードがターゲットのレコードと一致しないため、ターゲットのレコードは削除されません。

```
Create delete conflict
#!/bin/sh
sqlplus -s "gg_admin/gg_admin"<<EOFF
update demo.demo_users
set last_name='LAST_NAME_DEL_EXISTS' WHERE id=1;
commit;
exit
EOFF
Generate delete conflict
#!/bin/sh
sqlplus -s "demo/demo"<<EOFF
delete from demo_users WHERE id=1;
commit;
exit
EOFF
```

結論

本書では、DDLが有効化された包括的なアクティブ/アクティブ・ソリューションを提供するために連携されるさまざまなOGGの機能を紹介しました。本書の目的は、DMLのCDRが有効化された、DDLの基本的なアクティブ/アクティブ実装に対処することです。お客様の実装には、より複雑な構成のCDRルールが必要な場合があります。CDRのより広範な視点については、OGGの管理ガイドを参照してください。