

データ・レプリケーション

第二話

レプリケーションのソース

- 第一話では、DPropRを構成する要素についてとりあげました。今回は、DPropRでデータをレプリケーションするために重要な概念をいくつかご紹介します。
- レプリケーション対象のソースには、ユーザー表あるいはビューが指定できます。データをレプリケーションするには、どの表をソースとするか、また、どんな情報をどんなタイミングで収集し反映するのかを事前に定義しておきます。その際には、特に次の点を考慮にいます。
 - 列の変更前イメージが必要かどうか
 - 差分更新を収集し反映するのか、全件データをリフレッシュするのか
 - レプリカ・タイプで双方向のレプリケーションをするのであれば、同一データに更新が発生した場合、どちらの情報を優先するのか
- レプリケーションのソースを定義する際には、変更後のイメージだけ、または、変更前後のイメージ両方を収集するように指定ができます。どちらを選択するかは、アプリケーションの仕様によっても変わってきますが、以下に、変更前イメージ値のキャプチャーが必要となる可能性があるケースを挙げます。
 - ソース・データの履歴を保持する場合： 監査目的でデータを保持する必要がある場合
 - 双方向レプリケーション (update-anywhere) 構成で競合検出を行う場合： レプリカ表の間で競合が起こり得る構成では、CD 表に変更後イメージ列と変更前イメージ列の両方を登録して、競合が発生した場合には変更をロールバックできるようにする必要があります。
 - ターゲットでキー列が更新の対象となっている場合
- DPropRは差分更新の反映だけでなく、ソース表の全件データをターゲット表に一括反映することもできます。この場合、Captureにより収集された更新情報をもとにするのではなく、次のようなしくみで行われます。
 1. ターゲット表のすべての行を削除する
 2. ソース表からすべての行を読む
 3. ターゲット表に行をコピーするこの場合、直接ソース表をアクセスするので、全件リフレッシュを行う場合には実行する時間帯を考慮する必要があります。
- 双方向レプリケーション構成をくむ場合、レプリケーションの同一サイクル内で、ソース表とターゲット表の同じ行に更新があった場合の処置を考慮に入れる必要があります。つまり、更新が対立をおこした場合、どちらの情報を優先する(どちらの表が親表になる)のかを事前に定義しておきます。DPropRはソース表とターゲット表で同じ行に更新があったかどうかを検出するしくみをもっており、この定義情報をもとに更新情報の破棄あるいは反映を行います。

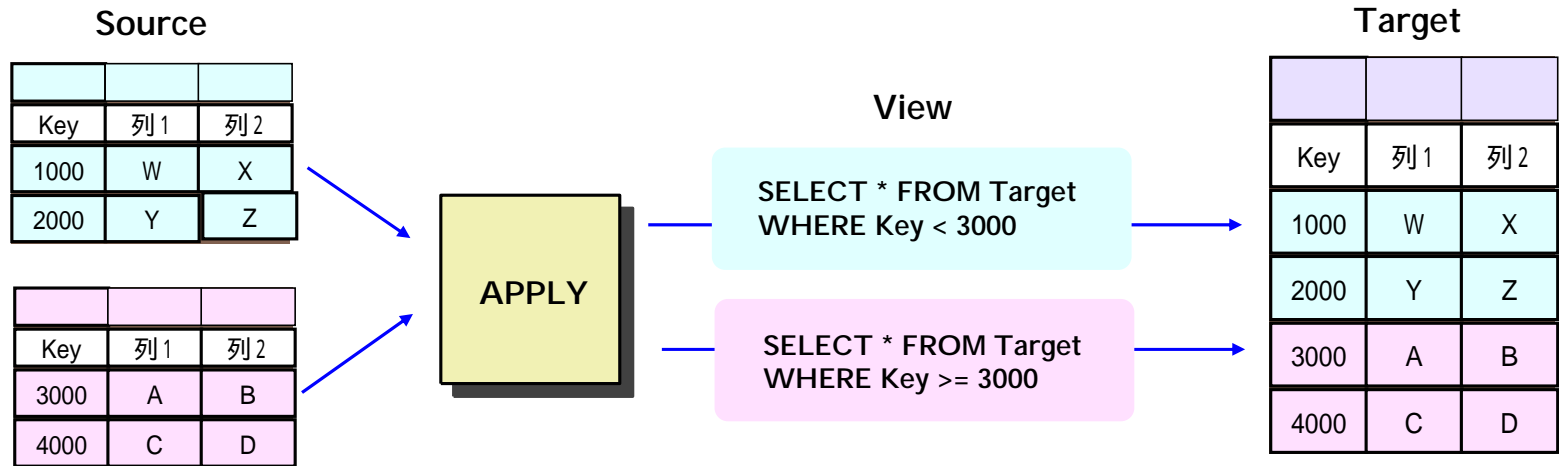
レプリケーション対象のデータの選択

- ソース表のサブセットだけをレプリケーションしたり、ビューやJOIN、UNIONを使ってソース表のデータ構造とは異なるイメージでターゲット表にレプリケーションすることもできます。
- レプリケーションのシナリオによっては、全ての列をターゲット表にレプリケーションする必要がない場合もあります。ソース表にLOBのような非常に大きなデータの列があり、ターゲット表には不要な場合は、パフォーマンス上、これを対象からはずしたほうがよいでしょう。また、ソース表に定義したすべてのデータタイプをターゲット表でサポートできない場合もあります。そこで、ターゲット表をソース表より列数の少ない列サブセットで定義することができます。(レプリカ・タイプの表では、列のサブセット化はできません。)
- また、全ての行をターゲット表にレプリケーションする必要がない場合もあるでしょう。一定の条件(WHERE文節)と一致する行を行サブセットとして定義し、この条件に合致するソース表の行のみをレプリケーションすることができます。たとえば、センターと各拠点にある事業所間でレプリケーションを行う場合、センターのマスター・データから、各事業所に関連するデータに対する変更だけを取り出して適用することができます。
- ソース表にビューを作成し、そのビューをレプリケーション対象とすることで、列と行のサブセット化を行うこともできます。また、複数の表にまたがるビューから1つのターゲット表に対してレプリケーションを実行することもできます。たとえば、社員番号をキーとして、住所などの個人データの格納された表と写真の格納された表に対してビューを作成し、これをソースとして、ターゲット表に新しい構造の表を作成してレプリケーションすることができます。
- レプリケーションにJOINやUNIONしたビューを利用すると、単一および複数のデータソース・サーバーの表を一つのソース表として定義することができます。

ビューをレプリケーションに使用した例

- レプリケーションにビューを使用した効果的な例を次にご紹介しましょう。
- 各地に散在する複数データ・サーバーの各テーブルと、センター・サーバーの1テーブルとの間でレプリカ・タイプの双方向レプリケーションを行うケースです。
 - 各拠点のデータ・サーバーでは売上データなどの情報が逐次更新されていきます
 - センター・サーバーには各拠点のマスター・データ、サマリー・データが格納されています
 - 各拠点のデータ・サーバーでも他の拠点のデータの参照が必要なため、センター・サーバーにある各拠点のサマリー・データが必要です。
 - センター・サーバーには各拠点のデータ・サーバーに蓄積されていくデータを一定間隔で反映します
- こういうケースでは、差分更新の反映を一定間隔で行うのが一般的ですが、H/W障害、N/W障害などの発生で、ある拠点のデータ・サーバーとセンター・サーバーとの間のレプリケーションが長時間停止してしまった場合、全件リフレッシュを行ったほうが、効率よくデータのリカバリーができる場合があります。
- 差分更新はCaptureが収集した変更情報分のみを反映しますが、全件リフレッシュの場合は、ターゲット表の全削除が行われます。この例で、各拠点から収集された売上データなどが物理的に1表に統合されていると、あるデータ・サーバーから全件リフレッシュ要求が発生した場合、センター・サーバーのデータが全件削除され、最新ではない他の拠点のデータを含むデータ・サーバーのデータに置き換わってしまいます。
- これを回避する方法として以下の2つが挙げられます。
 - センター・サーバーのターゲット・テーブルに対して、各拠点对応のためのビューを作成する。
各拠点のデータ・サーバーからは、このビューをターゲットにレプリケーションを行う。
その際に、キー・レンジを考慮して重複するデータがないように、ビューを作成しておく
 - センター・サーバーにターゲット・テーブルとして各拠点ごとのテンポラリー・テーブルを作成する。
各拠点のデータ・サーバーからは、このテンポラリー・テーブルをターゲットにレプリケーションを行う。
センター・サーバーでは、各拠点からレプリケーションされたテンポラリー・テーブルをUNIONで一つにまとめて使用する

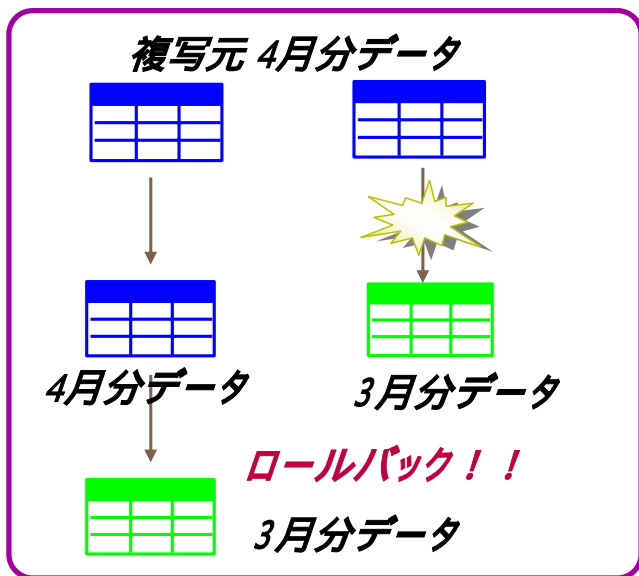
ビューをレプリケーションに使用した例



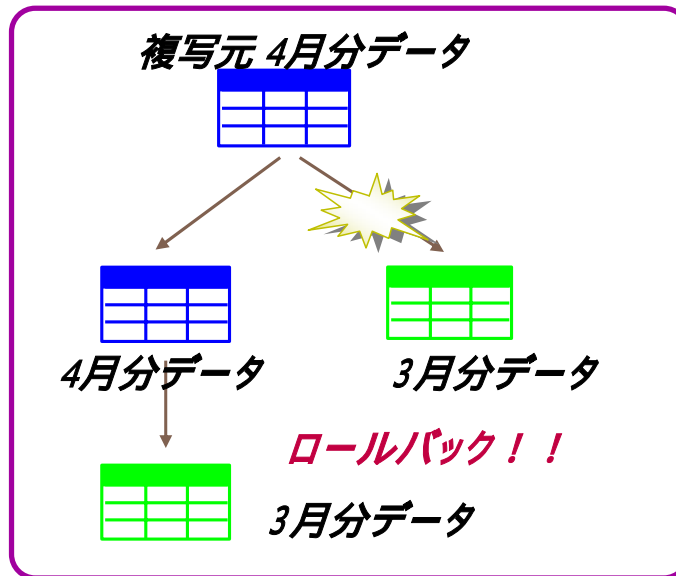
サブスクリプション

- サブスクリプションとは、レプリケーション対象のソース表と、ターゲット表の間の一連の関係を設定した定義体です。
- サブスクリプションは、ターゲットとなる表あるいはビューごとに一つ作成する必要があります。これをサブスクリプション・セット・メンバーといいます。いくつかのサブスクリプション・セット・メンバーを集めたものを、サブスクリプション・セットといいます。
- レプリケーションはサブスクリプション・セットを一つのコミット効力範囲内で処理します。つまり、複数の表・ビューに対するレプリケーションを同一作業単位内で実行するので、複数の表の間の安全性が保たれるのです。
- 下図は、同一サブスクリプション・セット内のある一つのサブスクリプション・セット・メンバーにおいて、4月分のデータの複製が失敗した場合、他の表でも同期をとって3月分のデータにロールバックされるという例を示しています。1サブスクリプション・セット内のいずれかで更新が失敗した場合、以下の2つのパターンのように、どのような場合でも、すべての表(サブスクリプション・セット・メンバー)は複製前の状態にロールバックされ、安全性が保たれるようになっています。

1サブスクリプション・セット

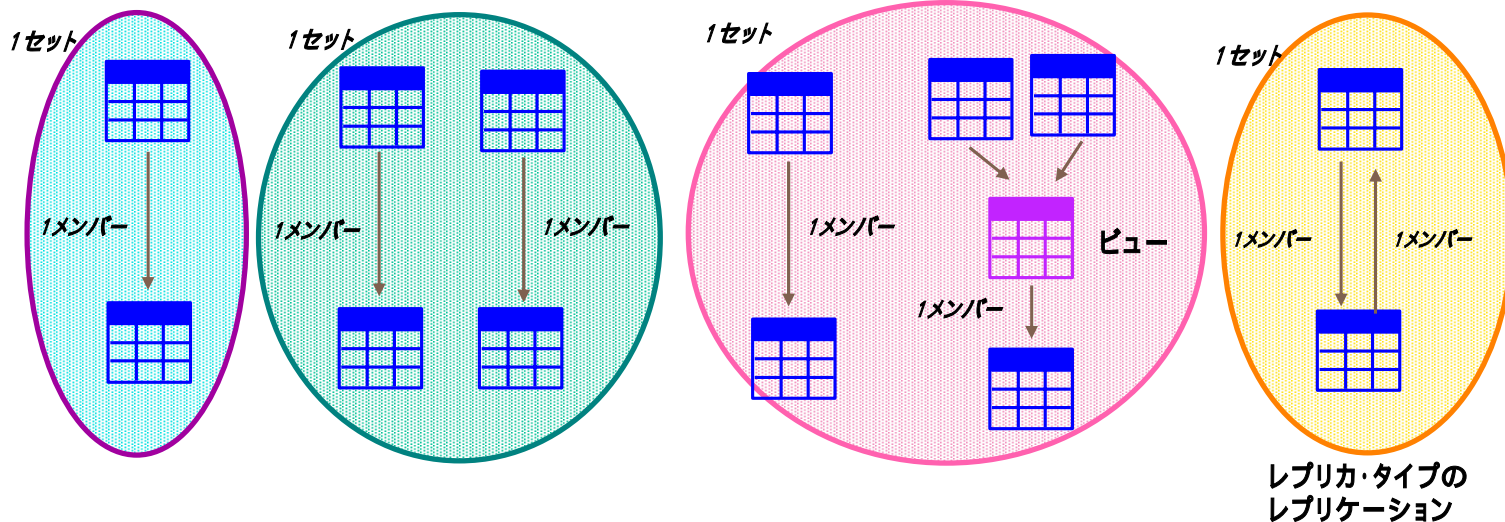


1サブスクリプション・セット



サブスクリプション

- サブスクリプション・セット・メンバーには次のような項目を定義します。
 - どの表またはビューをソースにするか、どの表またはビューをターゲットにするか
 - ターゲット表またはビューの構造 - 新規に作成するか、既存の表を利用するか
 - 複製する列 (レプリケーション対象の列をある条件で選択する)
 - 複製する行 (レプリケーション対象の行をある条件で選択する)
- サブスクリプション・セットには次のような項目を定義します。
 - サブスクリプション・セットの名前
 - ソース表のあるサーバー名、ターゲット表のあるサーバー名
 - レプリケーションの開始時間とそのインターバル、またはレプリケーションのトリガーとなるイベント
これら2種類のレプリケーション起動方法は併用することもできる
- 下図はサブスクリプション・セットとサブスクリプション・セット・メンバーの関係を模式化したものです。



アプライ修飾子 (Apply Qualifier)

- アプライ修飾子(Apply Qualifier)は、Applyと1つ以上のサブスクリプション・セットを関連付けたものです。アプライ修飾子の単位でApplyは開始できます。アプライ修飾子を分けることにより、複数のApplyを同時に稼働させ、並列的に処理することによりレプリケーションのスループットを高めたり、運用を分けることができます。

