自己参照テーブルを含むデータベースの Mobile Link 同期を確実に成功させる

このドキュメントでは、自己参照テーブルのデータを正しい順序で同期する必要がある状況で、問題を回避する方法について説明します。

概要

SQL Anywhere の同期において、Mobile Link ではデータの厳しい整合性は保証されません。つまり、すべてのデータはシステム全体にわたり一貫して同期されますが、同時または正しい順序での同期は実行されません。さらに、Mobile Link は複数のトランザクションを単一のトランザクションにまとめるため、同期するデータ量が少なく時間も短縮されますが、トランザクションの順序は失われます。したがって、データが実際に転送される順序は不明です。通常、これは問題とはなりませんが、自己参照テーブルがある場合には同期が困難となることがあります。

同期での主な懸念事項は、その実行順序ではなくデータが確実に同期されているかどうかです。しかし、外部キーが同一テーブル内の別のローを参照している場合のテーブルの同期時は、同期の順序が重要となることがあります。これは、外部キーが属するローがアップロードされてから、その外部キーが整合性をチェックする参照先ローがアップロードされると、チェックに失敗して同期がロールバックされるためです。このような状況でも問題を回避できる方法がいくつかあります。

セットアップ例

このドキュメントでは、統合データベースおよびリモート・データベースとして Adaptive Server Anywhere 9.0.2.3456 を使用します。自己参照するテーブル名は Employee であり、以下の SQL 文で定義されます。

```
CREATE TABLE Employee_and_Manager (
employeeID         INTEGER NOT NULL PRIMARY KEY,
employeeName       VARCHAR (40) NULL,
managerID          INTEGER NOT NULL,
FOREIGN KEY       "fk_EmptoMan" ("managerID")
REFERENCES   "DBA"."Employee_and_Manager" ("employeeID")
);
```

以下に、サンプル・データを入力した Employee テーブルを示します。

	employeeID	employeeName	managerID
1	100	Boss	100
2	101	one	100
3	102	two	100
4	103	three	101

ここでは、従業員エントリに対してリストにないマネージャを対応させないようにデータが設定されています。つまり、マネージャ自身もこの会社の従業員であり、社長は自身のマネージャであると設定されています。

症状

以下の症状は、自己参照テーブルの同期で発生する問題を示しています。

- 参照整合性のチェックの失敗による同期のロールバック
- dbmlsync より出力される以下のメッセージ

```
03/15 10:42:16. Cycle of foreign key references found.
(03/15 10:42:16. 外部キー参照の循環が見つかりました。)
Cannot guarantee referential integrity during upload.
(アップロード中の参照整合性を保証できません。)
I. 03/15 10:42:16. Tables involved in cycle:
I. 03/15 10:42:16.   Employee_and_Manager
```

これらの症状は、統合データベースへアップロードを適用する場合にのみ発生します。Mobile Link では、リモートにダウンロードを適用する場合、wait_for_commit というデータベース・オプションを使用してこの問題を回避します。

解決方法

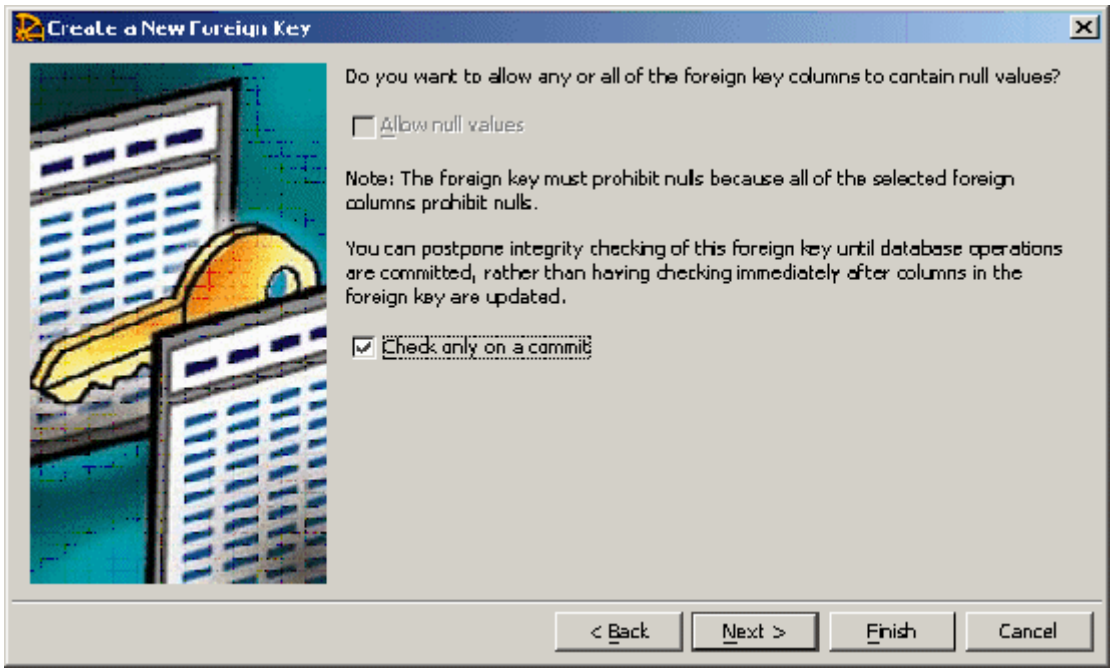
wait_for_commit オプションおよび CHECK ON COMMIT 句の使用
この問題に対する最も簡単かつ最善の解決策とは、wait_for_commit オプションと CHECK ON COMMIT 句を使用することです。ただし、これらのオプションは、Adaptive Server Anywhere および SQL Anywhere 統合データベース専用のオプションです。これらのオプションを使用することで、データベースが外部キーの参照整合性をどのタイミングでチェックするのかを指定できます。デフォルトでは Off に設定されており、データの挿入、更新、または削除の操作ごとに外部キーがチェックされます。自己参照テーブルで、ローが先にアップロードされてからその参照先ローがアップロードされ、その直後に外部キー参照がチェックされる場合に問題となることがあります。これらのオプションを On に設定すると、統合データベースは、アップロードがすべて正しく適用された後に Mobile Link がコミットを実行するまで、外部キーの整合性チェックを遅延させます。その時点で、すべての外部キーが有効となっています。

wait_for_commit データベース・オプションは、一般的なデータベース・オプションと同じように設定できます。

- SQL 文 `SET OPTION wait_for_commit = On;` を実行します。
  Public、Temporary などのその他の値も指定できます。
- wait_for_commit オプションは、Sybase Central からも On に設定できます。データベースを右クリックして [Options] を選択し、[Database Options] ダイアログの wait_for_commit に値を設定します。

CHECK ON COMMIT 句も同様に、コミットが実行されるまで外部キーの整合性チェックを遅延させます。ただし、CHECK ON COMMIT 句は、データベース全体に対する設定ではなく、特定の外部キーを設定する点で異なります。CHECK ON COMMIT 句は、いくつかの方法で On に設定できます。

- Sybase Central で外部キーを作成する場合は、ウィザード内で [Check Only on Commit] オプションを選択します。



- CHECK ON COMMIT 句で外部キーを追加する CREATE TABLE または ALTER TABLE SQL 文を実行します。*たとえば、このテーブルでまだ外部キーが作成されていない場合は、以下を実行できます。*

```
ALTER TABLE Employee_and_Manager
ADD FOREIGN KEY "fk_EmptoMan" ("managerID")
REFERENCES "DBA"."Employee_and_Manager" ("employeeID")
CHECK ON COMMIT;
```

これらの 2 つのオプションを使用すると参照整合性のチェックを遅延できるため、Mobile Link によるローの同期順序が制御できないという問題を回避できます。

これらのオプションは Adaptive Server Anywhere および SQL Anywhere 統合データベース専用のオプションですが、各 RDBMS で類似機能を実行できる独自のオプションが提供されていることがあります。詳細については、各 RDBMS のマニュアルを参照してください。

トランザクション・レベル・アップロード

この代替策は、リモート・データベースが Adaptive Server Anywhere または SQL Anywhere の場合にのみ使用できます。Mobile Link 同期ユーティリティの -tu オプションを使用して、トランザクション・レベル・アップロードを制御します。このオプションを On に設定すると、Mobile Link はリモート・データベースのすべてのトランザクションをマージせずに、各トランザクションを別個のトランザクションとして同期します。これにより、トランザクションがアップロードされる順序を指定できます。したがって、リモート・データベースと統合データベースで同じ制約が設定されていると想定した場合、リモート・データベースの参照整合性を保てるトランザクション順序が、常に統合データベースでも保持されます。ただし、単一トランザクションで複数の操作が実行される場合は、エラーが発生する可能性はあります。そのため、確実に正しい順序を保つには、自己参照テーブルでの各操作の後にコミットしてください。

このオプションを On に設定するには、dbmlsync スタート・ラインにオプションを追加します。以下は記述例です。

```
dbmlsync -c "uid=dba;pwd=sql;DSN=rem1" -tu
```

トランザクション・レベル・アップロードの難点は、大幅なパフォーマンス低下を招く可能性があることです。-tu オプションを使用すると、Mobile Link の主要機能である、複数トランザクションのマージ機能が無効になります。Adaptive Server Anywhere 9.0.2 で –tu オプションを使用する場合、トランザクション・レベル・アップロードによるパフォーマンスへの影響を最小限に抑えるために、Mobile Link サーバ・オプション –us の併用をおすすめします。-us オプションを使用すると、Mobile Link 同期ユーティリティは、アップロードするデータがないテーブルに対するスクリプトを実行しなくなります。以下は記述例です。

```
dbmlsrv9 -c "uid=dba;pwd=sql;dsn=cons" -us.
```

SQL Anywhere 10 ではデフォルトの動作となったため、-us オプションはありません。

パフォーマンス要件は状況により大きく異なるため、このオプションで満足できるパフォーマンス・レベルが達成できるのかを事前にテストしてから実装してください。

スキーマの修正

この解決方法は、どの統合データベースにも適用できるという利点があります。可能であれば、テーブルが自己参照しないようにスキーマを修正します。

たとえば、統合データベースが単にデータ格納用であり、すべてのデータをリモート・データベースから取得する場合、統合データベースの自己参照外部キーを取り除き、リモートで整合性のチェックを実行できることがあります。

テーブルの分割

スキーマを修正する別の方法としては、自己参照テーブルを複数のテーブルに分割する方法があります。セットアップ例を使用した場合、Employee テーブルから manager カラムを削除して、新しい Organization テーブルを作成します。Organization テーブルには、manager と employee という 2 つの整数カラムを作成し、employeeID を使用して各従業員をそれぞれのマネージャに対応させます。

これらのテーブルは以下の SQL 文で定義され、サンプル・データが入力されています。

```
CREATE TABLE Employee (
    employeeID      INTEGER NOT NULL PRIMARY KEY,
    employeeName    VARCHAR (40) NULL,
);
```

	employeeID	employeeName
1	100	boss
2	101	one
3	102	two
4	103	three

```
CREATE TABLE Organization (
   employee       INTEGER NOT NULL PRIMARY KEY,
   manager       INTEGER NOT NULL,
);
```

	employee	manager
1	100	100
2	101	100
3	102	100
4	103	101

2 つのテーブルの関係は、以下のように定義されます。

```
ALTER TABLE Organization
   ADD FOREIGN KEY "fk_Employee" ("employee")
   REFERENCES    "DBA"."Employee" ("employeeID")
   CHECK ON COMMIT;

ALTER TABLE Organization
   ADD FOREIGN KEY "fk_Manager" ("manager")
   REFERENCES   "DBA"."Employee" ("employeeID")
   CHECK ON COMMIT;
```

これらのテーブルでは、各従業員のマネージャを見つけることができ、すべてのマネージャは必ず従業員でもあるという制約も設定されるため、機能的には、最初のスキーマに非常に近いものです。しかし、この解決方法の難点は、従来の単一テーブルではなく 2 つのテーブルを管理しなければならず、データも 2 つのテ

ーブルに分割されているために多少見づらくなることです。ただし、ローのアップデート順序の場合とは異なり、Mobile Link は、分割したテーブルをアップロードするための最善の順序を判断できるので、問題は回避できます。Mobile Link は、このような状況で参照整合性を保つためのテーブルの同期順序を判断できるように設計されています。さらに、この方法で不十分な場合には、Mobile Link 拡張オプション –tor を使用してテーブルの同期順序を手動で指定することもできます。たとえば、 `dbmlsync -e "tor=Employee,Organization"` と指定すると、まず Employee テーブルが同期され、その後で Organization テーブルが同期されます。

結論

Mobile Link を使用して同期を実行する場合は、自己参照テーブルのデータの同期が困難となることがあります。特に、統合データベースが Adaptive Server Anywhere または SQL Anywhere ではない場合は、wait_for_commit オプションや CHECK ON COMMIT 句が提供されないので問題となります。しかし、そのような場合でも、上記に示したいくつかの方法により問題を回避できます。