第4章

ビュー、ストアドプロシージャ、トリガ

ここでは、データベース内の重要なオブジェクトであるビュー、ストアドプロシージャ、トリガについて記述します。

ビュー

Oracleには、実データが格納されないビューとマテリアライズドビューの2種類があります。マテリアライズドビューは、実データのコピーを格納したビューで、スナップショットと似た性質を持ちます。

SQL Serverのビューには以下の2種類があります。

- **インデックスなしのビュー**
  Oracleの実データが格納されないビューと同様に、ビューに実データは格納されません。
- **インデックス付きビュー**
  ビューに実データが格納されます。これは、ビューのパフォーマンスを向上させるためのもので、複雑な集計を行うビューなどに使います。ただし、実際のテーブルが変更されると自動的にビューのデータも更新されるため、更新のパフォーマンスが悪くなることに注意する必要があります。

ストアドプロシージャ

SQL Serverのストアドプロシージャには以下の3種類があります。

- **システムストアドプロシージャ**
  SQL Serverインストール時に作成され、システム管理に必要な情報を表示したり、変更を行ったりするプロシージャです。masterデータベースに格納されていて、どのデータベースからでも実行可能です。
- **ローカルストアドプロシージャ**
  各ユーザーデータベースに格納され、ユーザーのアプリケーションロジックを実装します。通常、ストアドプロシージャという場合は、ローカルストアドプロシージャを指します。
- **拡張ストアドプロシージャ**
  拡張ストアドプロシージャ以外のストアドプロシージャはTransact-SQLステートメントの集まりですが、拡張ストアドプロシージャはSQL Server

環境下で呼び出すことのできるDLLです。

ストアドプロシージャの実行プロセスは、以下のとおりです。

① 作成　：SQLステートメントの解析が行われ、システムテーブルに登録されます。
② 初回実行　：最初に実行されるときに、解決、最適化、コンパイルおよび実行され、プロシージャキャッシュに残ります。
③ 2回目以降の実行：実行プランがプロシージャキャッシュにあればそのまま再利用されます。

以下の状況が発生すると、SQL Serverはストアドプロシージャを自動的に再コンパイルします。

- 参照しているテーブルおよびビューの構造が変更された。
- 参照しているテーブルに対して大量の挿入（INSERT）、削除（DELETE）が行われた。
- 実行プランが使用しているインデックスが削除された。
- 新しい統計情報が明示的にあるいは自動的に作成された。

ユーザーが明示的に再コンパイルするには以下の方法があります。

- **作成時に指定する方法**
  CREATE PROCEDURE WITH RECOMPILEを使用して作成したストアドプロシージャは、プロシージャキャッシュに残らず、毎回コンパイルされます。
- **実行時に指定する方法**
  EXECUTE WITH RECOMPILEを使用してストアドプロシージャを実行すると、その実行時に再コンパイルします。
- **あるテーブルを参照しているすべてのストアドプロシージャを再コンパイルする方法**
  sp_recompileシステムストアドプロシージャを使用すると、指定したテーブルを参照しているストアドプロシージャはすべて再コンパイルされます。
- **現在のプロシージャキャッシュをすべてクリアにする方法**
  DBCC FREEPROCCACHEを実行すると、プロシージャキャッシュからすべての要素が削除されます。プロシージャキャッシュ内に存在したストアドプロシージャは、次の実行時に再コンパイルされます。

ストアドプロシージャへのパラメータの受け渡し

ストアドプロシージャにパラメータを渡すには、ストアドプロシージャ内にパラメータの宣言が必要です。宣言には変数名とデータ型を指定する必要があり、初期値を設定することも可能です。以下に、2つのパラメータを使うストアドプロシージャの例を示します。@xには初期値0が設定されています。

```
CREATE PROCEDURE dbo.zangyou
@x INT = 0,
@y CHAR(5)
AS
SELECT * FROM kinmu
WHERE zangyou > @x , syozoku = @y
```

実行時のパラメータの渡し方には2つの方法があります。

- **パラメータ名による渡し方**
  以下の例のようにプロシージャ内に定義されたパラメータ名を指定して値を受け渡します。この場合、パラメータの順番は関係ありません。

```
EXECUTE zangyou @x = 45 , @y = 'AAAAA'
```

- **位置によるパラメータの渡し方**
  ストアドプロシージャ内に定義した順番どおりに値を渡します。

```
EXECUTE zangyou 45, 'AAAAA'
```

ストアドプロシージャからパラメータによって値を返す場合は、パラメータにOUTPUTキーワードを指定します。以下に、ストアドプロシージャを作成するときの例を示します。

```
CREATE PROCEDURE search_cust
@cust_id int,
@cust_name VARCHAR(50) OUTPUT
AS
SET @cust_name = (SELECT cust_name FROM customer
                  WHERE cust_id = @cust_id)
```

次に、ストアドプロシージャ実行時の例を示します。

```
DECLARE @name VARCHAR(50)
EXECUTE search_cust 11111 , @name OUTPUT
```

トリガの種類

トリガとは、表に対して何らかの修正（INSERT、UPDATE、DELETE）が加えられたときに自動的に実行される特殊なストアドプロシージャです。OracleとSQL Serverのトリガを比較すると、次のようになります。

項目	Oracle	SQL Server
トリガを起動するDML	INSERT、UPDATE、DELETE	同左
トリガを起動するタイミング	BEFORE、AFTER、INSTEAD OF※1	AFTER、INSTEAD OF
トリガのレベル	文レベルと行レベル	文レベルのみ

※1　OracleのINSTEAD OFは、ビュー表に対してだけ定義可能で、必ず行レベルトリガとして定義する必要があります。

トリガを起動するDML

OracleとSQL Serverのどちらも、トリガを起動するSQLステートメントは同じです。

トリガを起動するタイミング

Oracleでは、BEFOREを指定して、トリガを呼び出した修正（INSERT、UPDATE、DELETE）を実行する前にトリガ処理を実行できます。

SQL Serverでは、AFTERとINSTEAD OFを指定できます。

- AFTERトリガ：　テーブルに作成するトリガです。実際にトリガを呼び出した修正（INSERT、UPDATE、DELETE）を実行したあとに実行されます。
- INSTEAD OFトリガ：　テーブルおよびビューに作成できるトリガです。実際にトリガを呼び出した修正（INSERT、UPDATE、DELETE）の代わりに実行されます。

トリガのレベル

Oracleでは、トリガのレベルとして行と文を指定できます。文レベルトリガは、トリガを実行するSQLステートメントに対して1回実行されるトリガです。一方、行レベルトリガは、トリガを実行するSQLステートメントが対象とする行それぞれに対応して実行されるトリガです。なお、SQL Serverには、行レベルトリガはありません。

トリガ内の処理

Oracleでは、トリガ内で修正（INSERT、UPDATE、DELETE）されたデータを参照するときは、相関名（デフォルトは：NEW、：OLD）を使用します。相関名は、行レベルのトリガで使用されます。

SQL Serverでは、次のようにしてトリガ内で修正されたデータを参照します。

- **INSERTトリガの場合**
  INSERTEDテーブルを使用して、追加された行を参照します。INSERTEDテーブルは、追加した行のコピーを保持している論理的なテーブルです。メモリ内にあり、呼び出されたトリガ以外からは操作できません。同一のINSERTステートメントで複数行を追加した場合は、INSERTEDテーブルには複数行が入っています。
- **DELETEトリガの場合**
  DELETEDテーブルを使用して削除された行を参照します。DELETEDテーブルは、削除した行のコピーを保持している論理的なテーブルです。メモリ内にあり、呼び出されたトリガ以外からは操作できません。同一のDELETEステートメントで複数行を削除した場合は、DELETEDテーブルに複数行が入っています。
- **UPDATEトリガの場合**
  INSERTEDテーブルに更新後のデータが入っており、DELETEDテーブルに更新前のデータが入っています。

ネストしたトリガ

SQL Serverでは、トリガがトリガを呼び出すことをトリガのネストと呼んでいます。トリガのネストは32レベルまで可能です。ネストのレベルを調べるには、@@NESTLEVEL関数を使用します。また、ネストさせないようにシステムを構成することも可能です。

これには、以下のシステムストアドプロシージャを使うか、SQL Server Enterprise Managerで設定します（図4-1）。

```
sp_configure 'nested triggers', 0
```

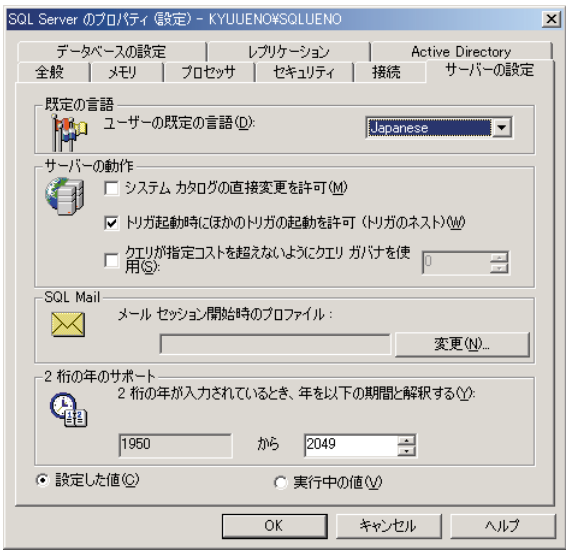
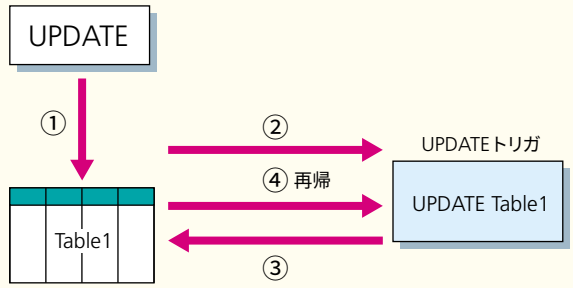


図4-1
ネストしたトリガ

ネストしたトリガの特殊なものとして再帰トリガがあります。再帰トリガは、自分自身を呼び出すトリガのことです。再帰トリガには、直接再帰と間接再帰があります。図4-2のパターン1ではテーブルにUPDATEトリガが作成してあり、そのトリガの中にTable1へのUPDATEステートメントがあります。パターン2ではINSERTトリガが作成してあり、直接Table1にINSERTはしていませんが、トリガによって実行された別のトリガがTable1にINSERTを行っています。

パターン1（直接再帰）



パターン2（間接再帰）

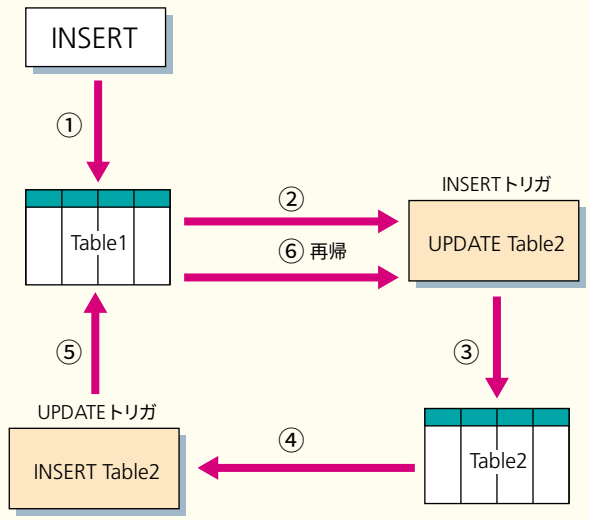


図4-2
再帰トリガ

再帰トリガはデフォルトで実行されません。つまり、上記の図のパターン1の④とパターン2の⑥は実行されません。実行するためには、以下のシステムストアドプロシージャを使うか、SQL Server Enterprise Managerで設定します（図4-3）。

```
sp_dboption databasename, 'recursive triggers', 'TRUE'
```

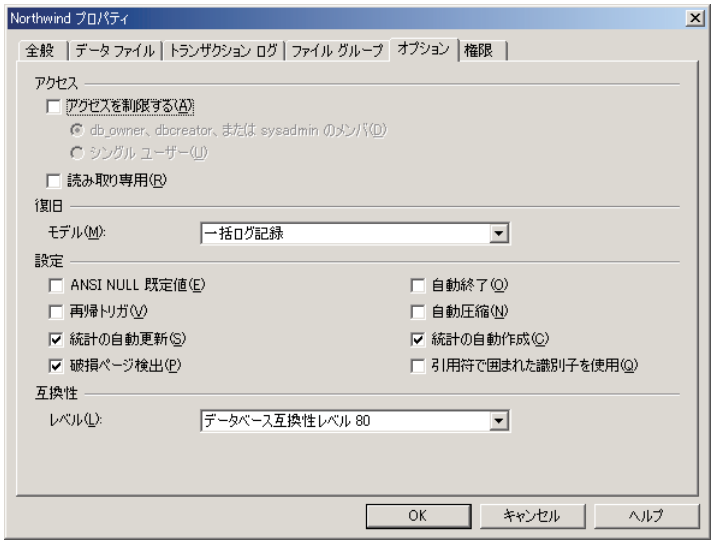


図4-3
再帰トリガ
(SQL Server Enterprise Manager)