ストアードモジュール / トリガを使ったスキーマ情報管理

4R−7

鴨井 功[1], 麦谷 尊雄[1], 北澤 敦[1], 高橋 真幸[2], 栗林 憲司[3]
[1] 日本電気(株)　[2] 北海道日本電気ソフトウェア(株)　[3] 日本電気ソフトウェア(株)

1 はじめに

近年、国際標準としてのSQLも機能強化が進み、これに伴ってデータベースを構成する要素や構造を表現するためのスキーマ情報も複雑になってきている。我々は、リレーショナルデータベース管理システムRIQSII-V2を開発するにあたり、スキーマ情報の発展性を考慮し、SQL自身のスキーマ定義言語によるスキーマ情報の定義と、ストアードモジュールおよびトリガによるスキーマ情報の参照、保守方式を採用した柔軟性のあるスキーマ情報管理方式を開発・実装した。ストアードモジュールおよびトリガは手続きであるが、データベース構成要素の一部を構成し、スキーマ情報として管理される。本稿では、我々の実装したスキーマ情報と、その管理方式を報告する。また、RI(リファレンシャルインテグリティ)に代表される表制約定義のトリガを用いた実装方式を示す。

2 スキーマ情報の管理

2.1 スキーマ情報

データベース構成資源の管理方法は特に規定されておらず、任意であるが、データベース管理システム自身の記述力、管理力が強力であれば、データベースを構成する資源の管理自身もそのデータベースのメタ資源として、自己記述的に定義できる。SQLをベースとしたリレーショナルデータベースも、情報管理能力、拡張性に富む強力なデータベース管理システムである。データベースの構成資源情報を整理してみると、以下が含まれる:

1) リレーションの構成を示す情報 (表, 列)
2) 各リレーションが満足すべき制約情報
3) 権限情報
4) VIEW表とVIEW表の構成情報

この他に、実装上の資源情報として、以下が存在する:

5) リレーションと格納構造とのマッピング情報

さらに、制約の強制や情報の参照 / 更新等に関する以下が存在する:

6) 手続き情報 (トリガ, モジュール)

これらの情報は全てシステム表と呼ばれるリレーションとして管理される。これらは互いに関連を持ち、定められた制約の下で一括して管理する必要がある。各々の情報を管理するために利用するSQLステートメントは、

- CREATE TABLE(表制約を含む)
- CREATE TRIGGER
- MODULE

及び、格納構造を指示する幾つかのステートメントである。

以降に、上記3種類のSQLステートメントによる情報管理方式を説明する。

2.2 スキーマ情報の参照 / 更新

SQL-DML(データ操作文)を処理するにあたり、指定された資源の存在の有無、アクセス権の妥当性、比較 / 代入における属性の合致性、等をチェックするため、システム表を参照して資源の定義情報を取得する必要がある。また、SQL-DDL(データ定義文)においては、指定された資源の定義情報をシステム表に反映する必要がある。すなわち、システム表を更新しなければならない。

RIQSII-V2では、システム表に対する参照権および更新権は、システム表の定義者であるデータベース管理者にのみ与えられている*。よって、一般利用者が直接システム表をアクセスすることはできないが、上記のようなシステム表に対する参照 / 更新処理は定形化されており、データベース環境作成時に、ストアードモジュールとして登録されている。このストアードモジュールの認可IDは、システム管理者となっており、代行アクセスとして処理することによって、システム表に対する参照 / 更新処理を実現している。

ストアードモジュールとして登録することによる利点としては、以下の2つが挙げられる:

スキーマ情報の保護: SQLで勝手にアクセスさせない

処理の高速化: 動的な問合せ / 更新の実行に比べ高速

2.3 モジュールの有効性

資源定義が追加・変更・削除された場合、その資源を参照しているストアードモジュールが無効となる場合がある。例えば、ある表に対してインデックスが新規に追加されたとすると、その表を参照しているだけのモジュールであればそのままでも実行可能であるが†、その表を更新している場合は新たにインデックスを更新する処理を追加する必要があり、モジュールを再生成しなければならない。

RIQSII-V2では、モジュールもスキーマ情報として、システム表であるモジュール管理表とモジュール関連表によって一括管理されている。モジュール関連表には、モジュールから参照されている各種データベース資源を識別するための情報が格納される。モジュール管理表には、各ストアードモジュールの実体とそれに関する情報が格納され、そのモジュールを生成する際に入力となったSQLステートメントも登録される。

データベースの資源が変更されると、モジュール関連表から無効とすべきモジュールを検出し、モジュール管理表に無効である旨マークアップされる。無効であるとマークアップされたモジュールに対する実行が要求された場合、SQLステートメントからモジュールが動的に再生成される。もちろん、特定のモジュールを明に再生成することも可能である。

3 スキーマ情報に対する制約

3.1 制約の分類

スキーマ情報としてシステム表に登録する情報は複雑化してきており、これらの情報に矛盾が生じないように、システム表に対して幾つかの制約が定義されている。これらの制約は、大別すると以下の3つに分類することができる。

check: 制約の対象となる表に閉じて処理可能なもの。

RI (リファレンシャルインテグリティ): 2つの表間にまたがる関連および動作を示す。RIの定義はネストすることが可能であり、定義がループすることも有り得る。

assertion: 上記以外の制約で動作を伴わないもの。checkとRIの一部(動作以外)を包含する。

3.2 制約の例

RIQSII-V2ではシステム表に対して、以下のような制約が定義されている。

1) 表定義が存在しない列定義は存在しない

$$\forall\, columns \left\{ \begin{array}{l} \text{columns.table\_name in} \\ \text{(select table\_name from tables)} \end{array} \right\}$$

Schema Information Maintenance using Stored Modules and Triggers
Isao Kamoi [1], Takao Bakuya [1], Atsushi Kitazawa [1],
Masaki Takahashi [2], Kenji Kuribayashi [3]
[1] NEC Corporation, [2] NEC Software Hokkaido, Ltd.,
[3] NEC Software, Ltd.

*データベース管理者が他の利用者に権限を譲渡することは可能。

†ただし、検索処理では新規に追加されたインデックスを利用しないので、そのままでは実行性能は上がらない。

2) 列定義が存在しない表定義は存在しない

$$\forall\ tables \left\{ \begin{array}{lll} \text{exists (} & \text{select * from columns} & \\ & \text{where} & \text{columns.table\_name =} \\ & & \text{tables.table\_name)} \end{array} \right\}$$

3) モジュールの生成日付は、モジュールの構成要素の生成日付より大きい

$$\forall\ modules \left\{ \begin{array}{l} \text{modules.version} \geq \text{all} \\ \quad \text{(select version from tables where ...) and} \\ \text{modules.version} \geq \text{all} \\ \quad \text{(select version from indexes where ...) and} \\ \text{modules.version} \geq \text{all} \\ \quad \text{(select version from triggers where ...)} \end{array} \right\}$$

ここで、1) はリファレンシャルインテグリティ、2) および 3) は assersion として定義することができる。

3.3 制約の強制

制約を違反するケースの一般的な抽出方法は [2] に述べられており、この方法を適用すると、次のイベントが発生した時点でなんらかの対処を行う必要があることがわかる。

- 制約の対象となる表に対する insert, update, delete
- 制約で参照されている表に対する insert, update, delete

最も単純な方式では、制約で参照されている表すべてに対する更新で制約をチェックすれば良いが、それでは効率が悪く、実用的ではない。そこで、制約を解析することにより、上記のイベントのうち、本当に制約チェックを必要とするものだけを抽出するオプティマイズ処理が有効となる。この処理については、[2] に詳しい。

上記のイベントが発生した際、制約が満たされない場合のアクションとして、次の 4 つのケースが存在する。

1) 例外の発生 (制約違反とする):
check 制約および RI における RESTRICT 指定に対応する。
2) 更新の波及 (CASCADE する):
RI における ON DELETE CASCADE および ON UPDATE CASCADE 指定に対応する。
3) 実行状態に応じて、例外の発生または更新の波及を行う:
資源削除 (DROP/ALTER) あるいは権限の剥奪 (REVOKE) の処理に用いられ、CASCADE 指定の有無によって、実行状態を変更し、どちらか一方を無効とする。
4) 無効ビットの設定:
資源定義の変更 (DROP/ALTER) あるいは権限の剥奪 (REVOKE) によって、無効となったストアードモジュールをマークアップする処理で用いられる。

3.3.1 トリガ定義

上記に示すようなスキーマ情報に対する様々な制約は、全て RIQSII-V2 の提供するトリガ定義機能で記述することが可能である。実際には、トリガ定義中に以下を指定可能とすることにより、トリガでの記述を可能としている:

- 明示的な例外の発生
- システム変数の値の参照

以下にトリガでの記述例を示す:

1) 例外の発生:
表を定義する CREATE TABLE の実行に伴い、表情報を格納する TABLES に対する追加が発生した際に、表定義のスキーマが、スキーマ情報を格納する SCHEMAS に存在することをチェックするトリガを定義する。

```
CREATE TRIGGER CHECK_SCHEMA
  BEFORE INSERT ON TABLES
  REFERENCING NEW NNN
  WHEN   (NOT EXISTS
            (SELECT * FROM SCHEMAS
              WHERE SCHEMA_NAME=NNN.SCHEMA_NAME ) )
         (RAISE EXCEPTION 'CONSTRAINT VIOLATION')
```

ここで、*RAISE EXCEPTION* の指定は、RIQSII-V2 が独自に提供している機能であり、例外を発生してトリガの実行を終了することを意味している。

2) 更新の波及:
表を削除する DROP TABLE の実行に伴い、表情報を格納する TABLES からの削除が発生した際に、カラム情報を格納する COLUMNS から、その表に含まれるカラムに関する情報を全て削除するトリガを定義する。

```
CREATE TRIGGER CASCADE_COLUMNS
  BEFORE DELETE ON TABLES
  REFERENCING OLD OOO
  WHENEVER   ( DELETE FROM COLUMNS
                 WHERE TABLE_NAME=OOO.TABLE_NAME AND
                 SCHEMA_NAME=OOO.SCHEMA_NAME )
```

3) 例外の発生または更新の波及:
スキーマを削除する DROP SCHEMA の実行に伴い、スキーマ情報を格納する SCHEMAS からの削除が発生した際に、DROP SCHEMA における CASCADE 指定の有無によって、そのスキーマに表が存在しないことをチェックするか、または、そのスキーマに含まれる表を波及的に削除するトリガを定義する。

```
CREATE TRIGGER CASCADE_OR_RESTRICT
  BEFORE DELETE ON SCHEMAS
  REFERENCING OLD OOO
  WHEN   ( $CASCADE='YES' )
         ( DELETE FROM COLUMNS
             WHERE TABLE_NAME=OOO.TABLE_NAME AND
             SCHEMA_NAME=OOO.SCHEMA_NAME )
  WHEN   ( $CASCADE='NO' AND
           EXISTS (SELECT * FROM TABLES
                     WHERE SCHEMA_NAME=OOO.SCHEMA_NAME ) )
         ( RAISE EXCEPTION 'CONSTRAINT VIOLATION' )
```

上の例では、システム変数として "$CASCADE" を参照することにより、実行状態を判定している。

4) 無効ビットの設定:
表を削除する DROP TABLE の実行に伴い、表情報を格納する TABLES からの削除が発生した際に、その表を参照しているストアードモジュールの無効ビット (DELETED) をセットする (値を 'Y' に更新する) トリガを定義する。

```
CREATE TRIGGER MARK_DELETION
  BEFORE DELETE ON TABLES
  REFERENCING OLD OOO
  WHENEVER
  (   UPDATE MODULES SET DELETED='Y'
      WHERE MODULE_NAME IN
      (   SELECT   MODULE_NAME FROM MODULE_RESOURCE_USAGE
          WHERE    RESOURCE_TYPE='TABLE' AND
                   RESOURCE_SCHEMA=OOO.SCHEMA_NAME AND
                   RESOURCE_NAME=OOO.TABLE_NAME ) )
```

4 おわりに

本稿では、SQL 自身のスキーマ定義言語によるスキーマ情報の定義と、ストアードモジュールおよびトリガによるスキーマ情報の参照、保守方式を採用した柔軟性のあるスキーマ情報管理方式について説明した。スキーマ情報に対する制約を全てトリガとして表現しているため、新たな制約の追加は、それに対応するトリガ定義の追加だけで実現でき、システムプログラムを変更することなく、制約を強制することが可能となる。したがって、本方式を採用することにより、拡張性が高く、柔軟性のあるデータベースシステムの構築が可能となる。

尚、本方式は当社のデータベース管理システム RIQSII-V2 に実装し、その動作は確認済みである。

参考文献

[1] ISO/IEC 9075 (E), Information processing systems – Database Language SQL with integrity enhancement, Second edition, 1989-04-01.

[2] Deriving Production Rules for Constraint Maintenance, Stefano Ceri and Jennifer Widom, Proceedings of the 16th VLDB Conference, Brisbane, Australia 1990.