Vol.2017-DBS-165 No.14
Vol.2017-IFAT-128 No.14
2017/9/18

意図に基づくアプローチによる更新可能ビューの拡張 — PostgreSQL におけるプロトタイピング —

長田悠吾[†1]　石井達夫[†1]　増永良文[†2]

概要：リレーショナルデータベースにおいてビューは問合せの定義のみを持つ仮想的なリレーションであり，それ自体はデータを持たない．そのため，ビューを更新しようとするときには，その更新可能性が問題となってくる．SQL 標準やいくつかのリレーショナルデータベース管理システム（RDBMS）で更新可能なビューがサポートされているが，その条件は限定的である．オープンソースの RDBMS である PostgreSQL では，PostgreSQL 9.3 以降，自動的に更新可能なビューをサポートしており，単純なビューならばそのままで更新可能である．しかし，たとえば JOIN 操作を含むような複雑なビューはそのままでは更新できず, 個々のビューに対して特別な INSTEAD OF トリガを定義するなどの準備が必要である.複雑なビューの更新は難解な問題であり長らく課題となっていた．これに対し，近年著者の一人が「意図に基づくアプローチ」と呼ばれる新規のアプローチを提案している．このアプローチでは,ビューの更新可能性は「更新意図の外形的推測」と名付けられたアルゴリズムにより，ユーザのビューの更新意図を推測することで判断する.これにより一般的に定義されたビューの更新可能性を拡張することが可能となる.このアプローチの実現可能性を検証するために,筆者らは PostgreSQL を拡張するプロトタイプを実装中である.本報告はこのプロトタイピングの現状を報告するものである．

キーワード：ビュー更新可能性，PostgreSQL，意図に基づくアプローチ，更新意図の外形的推測，プロトタイピング

Extending Updatable Views based on the Intention-based Approach: Prototyping on PostgreSQL

YUGO NAGATA[†1]　TATSUO ISHII[†1]
YOSHIFUMI MASUNAGA[†2]

1. はじめに

リレーショナルデータベースにおいてビュー(view)は，問合せの結果をあたかも実リレーションのように扱える簡略化表現(short-hand)でユーザの利便性に供し，ANSI/X3/SPARC のいう論理的データ独立性をリレーショナルデータモデルで達成する手段であり，またデータベースのセキュリティ実現のための手段として，理論的にも実践的にも多大の関心を集め続けてきた．

しかしながら，ビューはデータベースに格納されている実リレーションではなく，実リレーション群に対して発行された問合せの結果リレーション（SQL の用語でいえば導出表）を導く「定義」にしか過ぎない．そのため，ビューを問合せの対象とするときには問題は生じないが，物理的には存在していない仮想的なリレーションであるビューを「更新」しようとするとビューの更新可能性が問題となってくる．これをビュー更新問題（view update problem）という．

ビュー更新問題は形式的にも意味的にも難解な問題を孕んでいることが知られており，問題解決は積年の課題となっていたが，近年筆者らの一人増永により「意図に基づくアプローチ」[1][2]を提案され，これによりビューの更新可能性が飛躍的に向上することが期待されている. 本報告は，このアプローチを具体的にオープンソースのリレーショナル DBMS である PostgreSQL に実装してそのビューサポート能力を高めようとする試みであり[3]，その際どのようなアプローチが考えられ, またどのような課題が生じるのか，PostgreSQL におけるプロトタイピングの現状を報告するものである．

2. 意図に基づくアプローチ

本章では，意図に基づくアプローチの概要を述べる．詳細は文献[1][2]に譲る．

2.1 ビューの更新可能性とは

ビューは実リレーション群がなすデータベース状態(database state)からビュー状態(view state)への「関数」(function)である．s_τをある時刻τにおけるデータベースの状態，V をビュー定義，$V(s_\tau)$は（その時刻τにおける)ビューの状態, $\boldsymbol{u}$ を $V(s_\tau)$に対して発行された更新操作とするとき，$\boldsymbol{u}$ が**変換可能**(translatable)であるとは，$\boldsymbol{u}$ を s_τへの更新操作に変換する，(i) 一意(unique)で，(ii) 副作用がなく(no side effects)，かつ，(iii) 変換がデータベースに対して余計な更新をかけない(no extraneous update)という条件を満たす変換(translation) T が存在するときと定義される[4]. すなわち, 図 1 の可換図式(commutative diagram)が成立する

†1 SRA OSS, Inc. 日本支社
SRA OSS, Inc. Japan
†2 お茶の水女子大学（名誉教授）
Ochanomizu University (Emeritus)

Vol.2017-DBS-165 No.14
Vol.2017-IFAT-128 No.14
2017/9/18

ときをいう．なお，変換 T に副作用がないとは，$\boldsymbol{u}(V(s_\tau)) = V(T(\boldsymbol{u})(s_\tau))$ が成立することを意味する．変換 T に一意性を課したことは議論の余地があるところだとしながらも，その理由は T に代替案があった時，その選択基準を設けられないためとしている．なお，(iii)の条件は図 1 の可換図式に影響を与えるものではないが，$\boldsymbol{u}$ がビューを更新するために必要最低限の変換であることを要求しているということである．

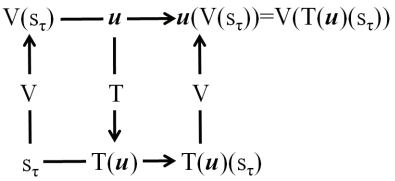

図 1　時刻 τ においてビュー更新要求 $\boldsymbol{u}$ が変換可能であることを示す可換図式

Figure 1　Commutative diagram of representing translatability of view update $\boldsymbol{u}$ at time τ.

2.2 意図に基づくアプローチ

意図に基づくアプローチが従来のアプローチとどこが本質的に異なるのか，端的に述べる．

まず，従来のアプローチは大別すると，歴史的に見て，次の 3 つに分類される．

(a) 構文的（syntactic）アプローチ
(b) 意味的（semantic）アプローチ
(c) 対話式（interactive）アプローチ

これらの説明はすでに文献[1][2]で行っているところであるが，若干補足する．(a)の構文的アプローチは更新操作 $\boldsymbol{u}$ が，ビューへの(ア)削除要求，(イ)挿入要求，そして(ウ)書換要求の個々の場合について，$\boldsymbol{u}$ が変換可能であるための必要かつ十分条件を関数従属性とキーの概念を用いて明らかにする[4]. (b)の意味的アプローチは変換可能性の定義の(i)一意性の条件を緩めると，この問題は意味的曖昧性を解決する問題となることを示している[5]. (3)の対話式アプローチは(b)が指摘した意味的曖昧性の解消を（ビュー更新者との）対話で解消しようとする取り組みである[6]．近年，曖昧性を解消する基準に対称性に依ろうとする提案がみられるが[7]，対称性はユーザのビュー更新意図の解明には何も寄与しないので，意味のないアプローチである．

さて，筆者らが提案し実装を試みている「意図に基づくアプローチ」は図 1 の可換図式が成立するか否かを，（構文的アプローチのように）関数従属性やキーを用いるのではなく，必要に応じて（中間）ビューをマテリアライズしてビュー更新変換の一意性を検証しようとするアプローチである．マテリアライゼーションに要するコストは嵩むが，従来更新不可とされてきたビューが更新可能となるという意味で，ビューの更新可能性を大いに高めることができる．

2.3 意図に基づくアプローチのもとでのビュー更新変換

意図に基づくアプローチとはどのような発想に基づくのか，簡単な例をあげて説明する．

ここでは，従来のアプローチでは，更新（タップルの挿入，削除，書換）が不可とされている自然結合ビューが意図に基づくアプローチのもとでは，更新可能となる場合があることを例題で示す[1].

【例題 1】自然結合ビューは外形的推測に基づき削除可能

図 2 に示した自然結合ビューR*S に対して，タップルの集合 D 2= {(a, b, c), (a, b, c')}の削除要求 $\boldsymbol{d}$ = delete D2 from R*S が発行されてきた場合を考えてみる．この要求に対して，意味的アプローチに基づき変換候補を求めると，次に示す 3 つの代替案があることがわかる：

T1($\boldsymbol{d}$) = delete (a, b) from R
T2($\boldsymbol{d}$) = delete {(b, c), (b, c')} from S
T3($\boldsymbol{d}$) = do both T1($\boldsymbol{d}$) and T2($\boldsymbol{d}$)

この曖昧性から従来型アプローチでは削除不可能と結論されている．しかし，ここで，R*S をマテリアライズし $\boldsymbol{d}$ を実行すると共に，T1 を仮に実行し，その結果リレーションと S を自然結合して，再びビューをマテリアライズして変換候補 T1 の「外形」を求めてみると，それは R*S－D2 となり(－は差集合演算)，マテリアライズされた R*S に対する $\boldsymbol{d}$ の実行結果と同じとなることを知る．一方，T2 や T3 を仮に実行すると外形は共に R*S－{D2∪{(a', b, c), (a', b, c')}}となり，副作用が発生することが分かる．つまり，唯一 T1 を実行した場合に限り図 1 の可換図式が成立することがこの試算で分かり，我々はこの状況を「ユーザがなぜ自然結合ビューR*S に対して D2 の削除要求を発行してきたのか，その意図は(本人に聞く以外には)知る由もないが，ユーザが要求しているタップル群の削除を丁度実現できる変換候補がただ T1 ひとつだけあるので，きっと実世界で T1 に該当する事象，つまり R からタップル(a, b)を削除することに相当する事象が発生して D2 の削除要求になったのであろうと「推測」(guessing)して，このようにビュー更新を行う」こととした．このとき，R*S への削除要求 $\boldsymbol{d}$ は外形的推測に基づき変換可能である，一般的に自然結合ビューは外形的推測に基づき削除可能であるという．

R

A	B
a	b
a'	b
a	b'

S

B	C
b	c
b	c'
b'	c

R*S

A	B	C
a	b	c
a	b	c'
a'	b	c
a'	b	c'
a	b'	c

図 2　自然結合ビューR*S のインスタンス

Figure 2　An instance of natural join view R*S.

なお，外形的推測を行ったからといって，必ず変換の曖昧性が解消されるわけではない．また，和集合ビューに対するタップル(群)の挿入要求，差集合ビューからのタップ

ル(群)の削除要求，共通集合ビューからの削除要求は変換の曖昧性が生じるが，これらは外形を計算しても解消できないので，本質的に更新不可能である．表 1 に，意図に基づくアプローチのもとでのリレーショナル代数の 7 つの基本ビューの更新可能性を従来型と比較して示す[1].

表 1　7 つの基本ビューの更新可能性の比較一覧
Table 1　List of updatability of 7 basic view definitions.

基本ビューの種類	更新操作	更新意図の外形的推測に基づく更新可能性	従来型アプローチによる更新可能性
和集合ビュー	削除	○	○
	挿入	×*	×
	書換	□	×
差集合ビュー	削除	×*	×
	挿入	○	○
	書換	×*	×
共通集合ビュー	削除	×*	×
	挿入	○	○
	書換	×*	×
直積ビュー	削除	◎	×
	挿入	◎	×
	書換	◎(書換要求両立)	×
射影 f	削除	○	○
	挿入	○(主キーを含む)	○(主キーを含む)
	書換	□	○(主キーを含む)
選択ビュー	削除	○	○
	挿入	○	○
	書換	○	○
結合ビュー	削除	◎	×
	挿入	◎	×
	書換	◎(書換要求両立)	×

○，×：従来型アプローチで更新可能，不可能．◎：更新意図の外形的推測に基づくアプローチで更新可能．□：直接書換法で書換可能．×*：本質的に不可能，を表す．

また，意図に基づくアプローチのもとでのビュー更新可能性の判定アルゴリズムは文献[1]で与えられている．

3. PostgreSQL における更新可能ビュー

3.1 SQL における更新可能ビュー

国際標準リレーショナルデータベース言語 SQL でビューの更新可能性が初めて規格化されたのは SQL-92 であり，その後 SQL:1999 で大幅な改正が試みられている[8][9].

(1) SQL-92 での更新可能ビュー

まず，SQL-92 のこの問題に関する基本的認識は，ビュー表は次のように定義されており，「問合せ指定が更新可能ならばビュー表は更新可能とする」というものである．

CREATE　VIEW　<表名>[(<ビュー列リスト>)]
　　AS <問合せ指定>

ここで，規格によれば，問合せ指定(query specification)は次の条件が全て成立するとき及びそのときのみ更新可能である：

- DISTINCT 指定を用いていないこと
- SELECT リスト中の値式は全て列参照であること．また，同じ列参照は 2 度以上現れないこと
- FROM 句は唯一のテーブルを参照し，そのテーブルは実表か更新可能な導出表であること
- そのテーブル参照は WHERE 句中の副問合せの FROM 句で参照されていないこと (つまり相関を有しないこと)
- GROUP BY 句や HAVING 句を使用していないこと

もし上記の条件にひとつでも適合しないならば，問合せ指定は読出し専用 (read-only)ビューとなる．つまり，SQL-92 では極めて単純なビュー表，つまり 1 枚の実表 (か更新可能な導出表)の (リレーショナル代数演算でいえば)射影か選択演算で切り出された結果のみが更新可能といっている．

(2) SQL:1999 でのビューの更新可能

SQL:1999 では，SQL-92 の更新可能(updatable)という用語に対して，それを細分化して 3 種の更新可能性と 1 つの挿入可能性を定義した ．それらは問合せ式の，潜在的更新可能(potentially updatable)，更新可能，単純更新可能(simply updatable)，そして挿入可能(insertable-into)である[9]．この考え方のもとでは，問合せ式の列が更新可能であるための十分条件を満たす UNION ALL ビューと結合ビューは更新可能となる．

3.2 PostgreSQL における自動更新可能ビュー

PostgreSQL 9.3 以来，PostgreSQL は自動的に更新可能なビューをサポートしている．これにより，一定条件を満たすビューに関しては，ユーザが手動でトリガやルールを定義せずとも自動的に更新可能となる．しかしながら，ビューが更新可能となる条件は基本的に SQL-92 に準拠しており，極めて限定的といえる．

4. PostgreSQL における意図に基づくアプローチのプロトタイピング

4.1 プロトタイプの実装方法

(1) PostgreSQL の問合せ処理の概要

プロトタイプの実装について説明する前に，PostgreSQL の問合せ処理の流れ[10]を概説する（図 3 左部分参照）．

クライアントから SQL 文字列として受信された問合せは，まず構文解析過程により構文のチェックが行われ，パース木に変換された後，最終的には問合せツリーと呼ばれる木構造の表現へと変換される．

次に，問合せツリーは書換システム（またはルールシステム）と呼ばれるモジュールで処理される．この時に適用する書換ルールが存在する場合には，そのルールに従って問合せ内容の書換が行われる．そのため，書換システムは

ルールシステムとも呼ばれる．PostgreSQL のビュー機能は，この書換システムにより実現されており，ビュー（すなわち仮想テーブル）に対する問合せは，書換システムにより実テーブルに対する問合せへと書き換えられる．

続いて，書換後の問合せツリーはプランナ（すなわちオプティマイザ）に渡され，実行計画が生成される．

最後にエクゼキュータがこの実行計画に基づき行を抽出し，結果がクライアントに返される．

(2) 2 つのアプローチ

PostgreSQL に提案手法を実装する方法として，ルールベースとトリガベースの 2 通りのアプローチが考えられる．この 2 つの違いは，上述した問合せ処理のどの段階でビュー更新可能性の判断を行うかによる．また，この違いによって，ビューに対する更新要求がどのような表現で得られるかが変わってくる．

- ルールベースアプローチ

ルールベースアプローチは PostgreSQL のルールシステムの中に本手法を実装する方法である．前述の通り，PostgreSQL のビュー機能はルールシステムにより実現されており，PostgreSQL 本体で実装されている自動更新可能ビューもこの段階で行われている．すなわち，ビューに対する更新要求が実行可能かどうかを判断し，可能な場合には実テーブルへの更新要求へと書き換えられる．これと同様にして，提案手法の実装もルールシステムに組み込むのが適しているように思われる．

しかしながら，このアプローチにはいくつかの困難がある．この方法では，ビューに対する更新要求は問合せツリーの表現で得られる．この表現には問合せを実行するのに必要な情報が全て含まれているが，その構造は複雑であり解析が難しい．また，本理論ではビューに対する更新要求は「どのタップルが更新されるか」といった具体的なタップルの集合として表現されることを前提としているが，このタップル集合を問合せツリーから直接得ることはできず，これを得るにはデータベースに対して検索を行う必要がある．さらに，ユーザが独自のルールを作成することが可能ではあるが，複雑な書換処理を指定することはできない．そのため，このアプローチを採用する場合には PostgreSQL の本体のコードを改造する必要ある．

- トリガベースアプローチ

もう 1 つのトリガベースアプローチは SQL のトリガ機能を使用して本手法を実装する方法である．ビューに対してトリガを作成し，更新が発生したときに実行されるトリガ関数の中で更新可能性の判断と実テーブルへの更新を行う．

トリガは問合せを実行するエゼキュータの段階で処理されるため，このアプローチではビューに対する更新要求をタップル集合の表現で得ることができる．このことは，本理論との親和性が高い．また，この方法では PostgreSQL 本体のコードに手を加える必要がなく，C 言語などの手続き言語を使用した複雑なトリガ関数も定義可能である．

以上の理由から，本プロトタイプの実装にはトリガベースアプローチを採用する．INSTEAD OF トリガは実際に要求された更新に代わって別の処理を行うためのトリガであり，このトリガを用いてビューを更新可能にする手法は幅広く使用されている．しかしその際には，ユーザは個々のビューに対し固有のトリガを定義するのが普通である．これに対し，本手法ではビュー全般に適用可能な一般的なトリガを定義する．

本実装でも当初は一般的な手法である INSTEAD OF トリガを使ってビューの更新処理を行うことを考えたが，PostgreSQL では文レベルの INSTEAD OF トリガがサポートされていないことが問題となった. 行レベルの INSTEAD OF トリガは定義可能であるが，この方法ではタップルを 1 行ずつしか処理ができないため，複数のタップルを同時に処理する必要がある本手法では利用できない．そこで，本実装ではビューの更新処理を INSTEAD OF トリガではなく AFTER トリガにて行う．AFTER トリガは問合せ文の実行後のタイミングで駆動されるトリガであり，文レベルでの定義が可能である．

(3) 遷移表（Transition table）の利用

AFTER トリガでは遷移表（transition table）が使用可能である．遷移表は SQL 標準で規定されており，PostgreSQL では PostgreSQL 10 からサポートされている**[11]**．これは，トリガを駆動した問合せ文の実行中に削除，挿入，および変更されたタップルの集合を，トリガ関数の中から表として参照できる機能である．本実装ではこれを利用して，ビューに対する更新要求をタップル集合として取得する．

4.2 プロトタイプの概要

本実装の概要を図 3 に示す．主要な処理はビュー定義された AFTER トリガにて行うが．前処理のため BEFORE トリガも併用する．

BEFORE トリガは問合せ文の実行前に駆動されるトリガである．このトリガ関数ではビュー定義を解析し，ビュー定義木を構築する．ビュー定義木とはビューを一般的に表現する木構造であり，ビューの更新可能性をチェックするのに使用される．

AFTER トリガでは, まず遷移表からビューに対する更新要求を抽出する．そして，ビュー定義木と更新要求に基づきビューの更新可能性をチェックし，更新可能な場合には実テーブルへの更新を行う．

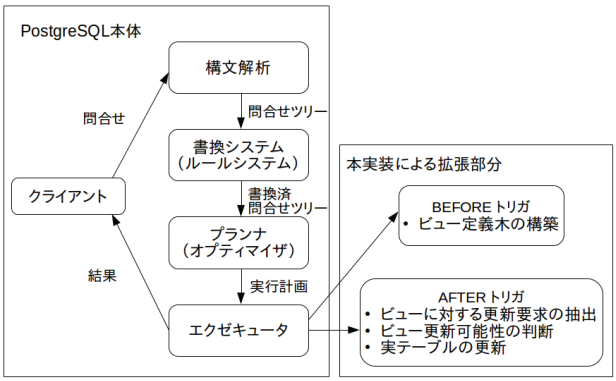

図 3　プロトタイプ実装の概要

Figure 3　Overview of our prototype implementation

4.3 プロトタイピングの詳細

以下では実装を構成する各要素の詳細について述べる．

(1) ビュー定義木の構築

ビュー定義木とはビューを一般的に表現する木構造である．各ノードは表 1 に示されているリレーショナル代数の基本ビューのいずれか，または実リレーションを表す．一般的なビューはこれら表 1 に示した 7 つのリレーショナル代数演算を再帰的に適用することで定義されるので，ビュー定義木は一般的なビューを表現することが可能である．例えば実リレーションとして，base1(a,b)と base2(b,c)があるときに，これらをカラム b で自然結合するようなビューは図 4 のようなビュー定義木で表現することができる．図中の結合ビューなど，根ノードではない中間のノードで表現されるビューは中間ビューと呼ばれる．

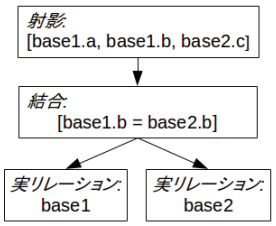

図 4　ビュー定義木の例

Figure 4　An example of view definition tree

本実装ではビューに対する更新が発生すると，そのビューのビュー定義木を BEFORE トリガの内部で構築する．ビュー定義の情報は PostgreSQL システムカタログに問合せツリーの形式で格納されているので，これを解析する．

またビュー定義木の全ての中間ビューは一時的にマテリアライズされるために，ビューを定義する SQL 文字列に変換できる必要があるので，ビュー定義木を構築するときには，そのために必要な情報（例えば，射影ビューでは取得するカラム名のリスト，選択ビューでは WHERE 句の文字列など）を各ノードに含める必要がある．

(2) ビューに対する更新要求の抽出

AFTER トリガでは，最初にビューに対する更新要求が遷移表から抽出される．更新の種類は削除（delete），挿入（insert），書換（rewrite）のいずれかとする．遷移表には更新要求の対象となったすべてのタップルの変更前の値と変更後の値が含まれている．更新が挿入の場合には変更後の値のみ，削除の場合には変更前の値のみ，書換の場合にはその両方が含まれる．

実際に発行される更新要求は，例えば DELETE FROM v WHERE c=1 OR c=3; など，WHERE 句に様々な条件を取り得る．しかし，遷移表から抽出される更新要求は WHERE 句の表現にかかわらず，タップルの集合を用いた表現をとる．例えば削除なら delete {(1,2,1), (1,2,3)}，書換なら rewrite {(1,1,2), (2,1,2)} to {(1,1,20), (2,1,20)}，挿入なら insert {(1,2,3), (1,2,4)}といった表現となる．

(3) ビューの更新可能性の判断

ビュー定義木，およびビューへの更新要求が与えられると，その更新要求のもとにビューが更新可能かどうかを判定できる．更新可能性の判断はビュー定義木を根ノードから再帰的に幅優先探索していくことで行う．

各ノードで表現されるビューの更新可能性は表 1 の 3 列目に従って判定する．もし現在のノードで表現される中間ビューが更新可能ならば，そのビューへの更新要求を子ノードで表現される中間ビューへの更新要求に変換し，探索を続ける．更新要求の変換は射影ビューの場合に必要となる．これはカラムの名前や順番の変更を伴う射影演算では，親ノードのビューと子ノードのビューでカラム間の対応付けが必要となるからである．また，直積または結合ビューの場合には，更新要求のタップルを子ノードのビューに対応するように 2 つに分割するという変換を行った上で，さらに外形的推測に基づく更新可能性の判定が必要となる．これについては次節で説明する．

探索の途中で 1 つでも更新不可能の中間ビューがあった場合には，そのビューは更新できない．逆に，すべての中間ビューが更新可能である場合が唯一存在するならばそのビューは更新可能であり，同時にそれを可能にする実テーブルへの更新内容も定まる．

(4) 外形的推測による判定

直積または結合ビューの場合には，外形的推測に基づく更新可能性の判定を行う．

まず，現在のノードの中間ビューへの更新要求に基づき，「子ノードの中間ビューに対する更新の候補」を生成する．更新の候補は論理的に求めることができ，その詳細は文献[1], [12]で与えられている．

次に，現在のノードの中間ビュー，およびその子ノードである 2 つの中間ビューに対し，一時的なマテリアライズを行う．ここでマテリアライズとは，現時点のビューの内容を実データに変換する操作であり，ビュー定義に基づい

て一時テーブルを作成することで実現している．その後，マテリアライズされた現在のノードの中間ビューに対して，更新要求を適用する．これにより，ユーザがビュー更新によって得たい「望ましい結果」の外形が求まる．

次に，マテリアライズされた子ノードの中間ビューに対してそれぞれ更新の候補を適用し，さらにそれらの直積または結合の結果を求める．これで得られた「更新候補を適用した結果」を，上で求めた「望ましい結果」と比較する．これが一致すれば，今回適用した更新候補によりユーザの意図するビュー変更が実現可能であることがわかる．

この手順を全ての更新候補について行い，最終的にそのような候補がただひとつ見つかった場合には，その更新候補の元に，現在のノードの中間ビューは更新可能であると判断される．もしそのような候補が見つからない，または複数見つかるような場合は，更新不可能と判断される．

4.4 プロトタイプの動作例

以下に本プロトタイプの動作例を示す．ここで v はテーブル base1 と base2 の自然結合として定義されたビューである. 通常の PostgreSQL ではこのようなビューを更新しようとすると以下のようにエラーとなる．

```
=# INSERT INTO v VALUES (1,2,3),(1,2,4);
ERROR: cannot insert into view "v"
DETAIL: Views that do not select from a single table or view are not automatically updatable.
HINT: To enable inserting into the view, provide an INSTEAD OF INSERT trigger or an unconditional ON INSERT DO INSTEAD rule.
```

このビューに対し本実装による拡張を施すことで，以下のように更新が可能になることが確認された．

```
=# INSERT INTO v VALUES (1,2,3),(1,2,4);
INSERT 0 2
=# DELETE FROM v WHERE c in (1,3);
DELETE 2
=# UPDATE v SET c = 20 WHERE c = 2;
UPDATE 2
```

しかし全ての更新が可能なわけではなく，外形的推測の結果，更新不可能と判断された場合には，以下のエラーとなることも確認された. これは意図された振る舞いである．

```
=# DELETE FROM v WHERE c = 4;
ERROR: not updatable based on pro forma guessing
```

5. おわりに

本報告では，リレーショナルデータベースにおけるビュー更新問題を概観すると共に，意図に基づくアプローチについて，更新意図の外形的推測とは何かを具体的に従来型アプローチでは更新不可とされてきた結合ビューの更新可能性を取り上げて説明した．さらに，具体的にオープンソースのリレーショナルデータベース管理システムである PostgreSQL におけるプロトタイピングの現状を報告した．

今後の課題として，本アプローチの限界やパフォーマンスの問題の考察が挙げられる．より現実的なビューの定義とそれに対する更新操作など，様々なケースにおけるプロトタイプの動作を検証することで，本アプローチの適用可能範囲とその限界を探る必要があろう．実際の動作検証の結果を理論にフィードバックすることが，このプロトタイピングの目的の１つである．

実装に関しては，トリガベースアプローチの限界についても考察が必要であり，今回は採用しなかったルールベースアプローチを用いた実装法の検討も必要となるだろう．

また，本手法ではビューの更新可能性を判断するために中間ビューのマテリアライズを，多数の更新候補について繰り返す必要があり, これがパフォーマンスに与える影響, および計算量を削減する工夫の検討が必要となる．

謝辞 本研究は JSPS 科研費 16K00152 の助成を受けている．

参考文献

[1] 増永良文．更新意図の外形的推測に基づくリレーショナルデータベースビューの更新可能性. 第 8 回 Web とデータベースに関するフォーラム(WebDB Forum 2015)，2015 年 11 月．
[2] Masunaga, Y.. An Intention-based Approach to the Updatability of Views in Relational Databases. Proceedings of the 11th International Conference on Ubiquitous Information Management and Communication (ACM IMCOM '17), January 05-07, 2017, Beppu, Japan.
[3] Nagata, Y. and Masunaga, Y.. Extending View Updatability by a Novel Theory -Prototype Implementation on PostgreSQL. Talk at PGCon 2017-The PostgreSQL Conference, May 25-26, 2017, Ottawa, Canada.
[4] Dayal, U. and Bernstein, P.. On the updatability of relational views. Proceedings of the 4th VLDB, pp.368-377, 1978.
[5] Masunaga, Y.. A relational database view update translation mechanism. Proceedings of the 10th VLDB, pp.309-320, 1984.
[6] Sheth, A., Larson, J. and Watkins, E..TAILOR, A Tool for Updating View. LNCS, Vol. 303, pp.190-213, Springer, 1988.
[7] Date, C. J. View Updating and Relational Theory: Solving the View Update Problem. O'Reilly, 2013.
[8] 増永良文．リレーショナルデータベース入門[第 3 版]．サイエンス社，2017，415p.
[9] Melton, J. and Simon, A.. SQL:1999 Understanding Relational Language Components. Morgan Kaufmann, 2002, 893p.
[10] "PostgreSQL 10 Documentation, Chapter 50. Overview of PostgreSQL Internals, 50.1. The Path of a Query". https://www.postgresql.org/docs/10/static/query-path.html, (参照 2017-08-10).
[11] "PostgreSQL 10 Documentation, Appendix E. Release Notes, E.1. Release 10". https://www.postgresql.org/docs/10/static/release-10.html, (参照 2017-08-10).
[12] 増永良文，長田悠吾，石井達夫．バッグ意味論のもとでのビュー更新問題の検討 -更新意図の外形的推測に基づくアプローチの適用可能性-. 第 9 回データ工学と情報マネジメントに関するフォーラム(DEIM2017)，2017 年 3 月．