日本データベース学会和文論文誌
Vol. 18-J, Article No. 1, 2020年3月

更新意図の外形的推測に基づいたビューの更新可能性とそのPostgreSQL上での実現可能性検証

増永 良文▾　長田 悠吾◆　石井 達夫◆

ビューはリレーショナルデータベースに格納された実リレーションではなく，その定義だけが格納されている仮想的なリレーションなので，ビューを更新しようとすると異状が発生する場合がある．では，どのようなビューで更新が許されるのか，それを問う「ビュー更新問題」はこれまで多くの研究がなされてきたが，更新可能とされるビューは極めて限定的であった．本論文では，その原因が，これまでのビュー更新問題への取り組みがデータベースの“スキーマレベル”でなされてきたことにあることを指摘し，そうではなく“インスタンスレベル”でそれを論じることにより，従来のアプローチでは更新不可とされてきた直積ビューや結合ビューが場合により更新可能となることを明らかにしている．この新規アプローチは「更新意図の外形的推測アプローチ」，略して「意図に基づくアプローチ」と名付けられている．意図に基づくアプローチの下での一般的に定義されたビューの更新可能性判定アルゴリズムを示すと共に，この新規アプローチをPostgreSQL 10の拡張機能の一つとしてプロトタイプして，その実現可能性検証を行った結果を報告する．

1. はじめに

ビュー(view)は，リレーショナルデータベースに対する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便表記法（short-hand notation)として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有用であると認識されている．

しかしながら，ビューは実リレーションではなく，その定義だけが格納されている仮想的なリレーションであるため，ビューを問合せの対象とする限り質問変形(query modification)という手法[1]により問題は生じないが，ビューを更新，つまり，ビューに対してタップル集合の削除，挿入，書換をしようとすると，異状が発生する場合があり，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．

ビュー更新問題はこれまで多くの研究がなされてきた．それらを要約すると，合成的，意味的，そして対話型アプローチに分類することができるが，そこで示された更新可能なビューは極めて限定的であった．本論文では，その原因がビュー更新問題がこれまでデータベースの“スキーマレベル”で論じられてきたことにあることを指摘し，そうではなく“インスタンスレベル”でそれを論じると，従来のアプローチでは更新不可とされてきた直積ビューや結合ビューが場合により更新可能となることを明らかにしている．この新しいアプローチは「更新意図の外形的推測アプローチ」，略して「意図に基づくアプローチ」と名付けられている．本論文では，この新規アプローチの下で直積ビューや結合ビューがどのような場合に更新可能となるのかを詳細に議論すると共に，この新規アプローチの下での一般的に定義されたビューの更新可能性判定アルゴリズムを示す．また，この新規アプローチをオープンソースソフトウエア(OSS)のリレーショナルDBMSとして多くの稼働実績を有するPostgreSQLの拡張(extension)機能の一つとしてプロトタイピングし，その実現可能性検証を行った結果を報告する．

以下，ビューとその更新可能性(第2章)，従来のアプローチとビューの更新可能性(第3章)，更新意図の外形的推測アプローチ(第4章)，意図に基づくアプローチの実現可能性検証(第5章)，おわりに(第6章)，と続く．

2. ビューとその更新可能性

2.1 ビューとは

ビューはリレーショナルデータモデルの提案者であるE. F. Coddにより導入され，リレーショナル代数を使ったその定義は次に示す通りである[2]．なお，リレーショナルデータモデルでは，リレーションは有限個のドメインの直積の有限部分集合として定義されるので，ビューも集合となり，本論文で展開する議論は集合意味論(set semantics)に従っている[1]．

【定義1】(ビュー定義)

(1) 実リレーションRはビューである．
(2) V_1，V_2をビューとするとき，V_1とV_2が和両立ならば，それらの和，差，共通，すなわち，$V_1 \cup V_2$，$V_1 - V_2$，$V_1 \cap V_2$はビューである．
(3) V_1，V_2をビューとするとき，それらの直積$V_1 \times V_2$はビューである．
(4) Vをビューとするとき，射影V[C]はビューである．ここに，CはVの属性集合である．
(5) Vをビューとするとき，θ-選択$V[A_i \theta A_j]$はビューである．ここに，Vの属性A_iとA_jはθ-比較可能とする．
(6) V_1，V_2をビューとするとき，θ-結合$V_1[A_i \theta B_j]V_2$はビューである．ここに，V_1の属性A_iとV_2の属性B_jはθ-比較可能とする．
(7) (1)から(6)で定義されるもののみがビューである．

2.2 ビューの更新可能性

ビューの更新可能性は次のように定義される[3]．ここに更新(update)とはタップル集合の削除(delete)，挿入(insert)，そして書換(rewrite)を意味する．本論文では，ビューVへの更新操作$\boldsymbol{u}$は更新されるべきビューのタップル集合を明示した形式で指定することとする[2]．つまり，削除操作は$\boldsymbol{d}$ = **delete** X **from** V;，挿入操作は$\boldsymbol{i}$ = **insert** X **into** V;，書換操作は$\boldsymbol{r}$ = **rewrite** X **of** V **to** Y;，ここにXやYはタップル集合を表す，という具合である．なお，書換操作は書換の対象となったタップル集合をまず削除し，その結果に書換えられたタップル集合を挿入するという操作の系列と定義する．

【定義2】(ビューの更新可能性)

ビューは実リレーション群がなすデータベース状態からビュー状態への関数である．s_τをある時刻τにおけるデータベース状態，Vをビュー定義，$V(s_\tau)$は(その時刻τにおける)ビュー状態，$\boldsymbol{u}$を$V(s_\tau)$に対して発行された更新操作とする．このとき，$\boldsymbol{u}$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

▾ 正会員　お茶の水女子大学名誉教授 masunaga.yoshifumi@ocha.ac.jp
◆ 正会員　SRA OSS, Inc.日本支社 nagata@sraoss.co.jp
▴ 正会員　SRA OSS, Inc.日本支社 ishii@sraoss.co.jp

[1] 国際標準リレーショナルデータベース言語SQLはテーブルに行の重複出現を許すバッグ(bag)意味論に従っている．

[2] 集合意味論に従っているのでビューも(バッグではなく)集合であり，従って更新操作の対象を集合で記述できる．

(i) 一意で, (ii) 副作用がない変換Tが存在するとき及びそのときのみをいう．すなわち，図1の可換図式が成立するときをいう．

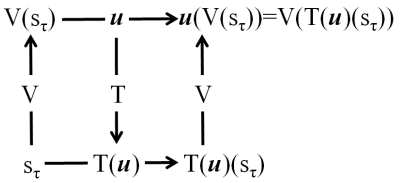

図1　時刻 τ においてビュー更新操作 $\boldsymbol{u}$ が変換可能であることを示す可換図式

ここで，(i)の変換Tに一意性を課した理由は，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である．(ii)の変換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．なお，(i)と(ii)に加えて，(iii) 変換がデータベースに対して余計な更新をかけないという条件を課すことができるが，本論文で議論している変換法の下では，ビューに対する更新操作の変換先は常にビューを定義している実リレーション(群)に限定され，かつそこでの更新操作は所望のビュー更新を実現させるために必要にして十分な操作という意味で，この条件は常に満たされている．

また，注意点として，この定義はデータベースに課せられた意味的制約を形式化していないので，たとへ変換Tが一意で副作用を引き起こすことなく $\boldsymbol{u}$ をT($\boldsymbol{u}$)に変換できたとしても，s_τ にT($\boldsymbol{u}$)を施すとデータベースに課せられていた意味的制約に抵触するような場合があり得る(脚注6参照)．

更にもう一つ注意するべき点として，ビューの更新可能性は，ある時刻 τ におけるデータベースの状態 s_τ でビューに対する更新操作 $\boldsymbol{u}$ が変換可能かどうかを問うている，つまりデータベースの“インスタンスレベル”で定義されている．しかしながら，これまでビューの更新可能性は，例えば“自然結合ビューは削除可能か?”というように“スキーマレベル”でその更新可能性を問うてきた．つまり，もし図1の可換性を満たさないような自然結合ビュー，それへの削除操作，そしてデータベース状態の組合せが一つでも存在すれば，“自然結合ビューは削除不可である”と結論付けてきた[3]．逆に，図1の可換性を損なうような組合せが一つも生起しなければ，例えば“和ビューは削除可能である”という具合に結論される．この立場を採ると，リレーショナルDBMSがビューサポート機能を備えるにあたり，システムカタログを参照することでそれを実現することが可能となり，実装が容易になる．これが従来のアプローチがとってきた立場であるが，更新可能なビューは極めて限定的であった．一方，本論文で導入した新規アプローチはスキーマレベルでは更新不可とされてしまうビューをインスタンスレベルで捉え直す，つまり定義2に忠実にその変換可能性を追求している．その結果，これまで更新不可とされてきた直積ビューや結合ビューが場合により更新可能となる．以下でそれを詳しく論じる．

なお，定義2はインスタンスレベルの定義なので，マテリアライズドビュー(materialized view)の更新可能性にも適用可能であるが，本論文の主題はあくまで定義1で与えられた仮想的なビューの更新可能性である．

[3] スキーマとインスタンスの関係は次の通りである：リレーションスキーマ $\boldsymbol{R}$ で性質Pが成り立つとは，$\boldsymbol{R}$ のすべてのインスタンスRでPが成り立つとき，及びそのときのみをいう．従って，Pが成り立たないようなインスタンスRが一つでも存在すれば，その性質はスキーマ $\boldsymbol{R}$ では成り立たない．

3. 従来のアプローチとビューの更新可能性

3.1 従来のアプローチ

関連研究としてビュー更新問題に対する従来のアプローチを総括すると，それらは次の3つに分類できる．

(a) 合成的(synthetic)アプローチ
(b) 意味的(semantic)アプローチ
(c) 対話型(interactive)アプローチ

まず，(a)は，更新操作 $\boldsymbol{u}$ がビューへの削除操作，挿入操作，そして書換操作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やキーなどの概念を用いて明らかにしようとするアプローチである[3, 4, 5]．(b)は変換可能性に課した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[6]．(c)は(b)が指摘した意味的曖昧性の解消を，ビュー更新操作を発行した者との対話で行おうとする取組みである[7]．近年，曖昧性解消の基準を対称性に求める提案が見られるが[8]，対称性では意味的曖昧性を解消できず問題解決とはならない．

3.2 従来のアプローチの下でのビューの更新可能性

2.2節で述べたように，従来のアプローチはビューの更新可能性をデータベースのスキーマレベルで捉えていたので，更新可能であることを示すには図1の可換図式が例外なく成立すること，逆に更新不可であることを示すには一つでも可換性が成り立たないような例(＝反例)を示せばよかった(脚注3)．以下，この観点から，定義1で導入した7つの基本ビューの更新可能性がどのように論じられてきたか要約する．なお，ビューはリレーショナル代数で定義されているので(定義1)，以下，すべての議論は集合意味論の下で行われる．

3.2.1 和・差・共通ビューの更新可能性

RとSを和両立な2つのリレーションとする．

(1) 和ビューの更新可能性

和ビューV = R∪Sを考える．Vの定義はV = R∪S = {t | t∈R∨t∈S}なので，その意味(meaning)は次の通りである[6]（iffはif and only ifを，∨は論理和を表す)．

t∈V iff t∈R∨t∈S

(a) 和ビューは削除可能

Vに対して削除操作 $\boldsymbol{d}$ が発行されたとする．

$\boldsymbol{d}$ = **delete** X **from** V;

ここに，X = {t_1, t_2, ..., t_n}はタップル集合とする．VからXを削除するということは，Vの意味によりXの各元 t_i に対して t_i∈Vではなくなった，つまり，(∀t_i∈X)(¬(t_i∈R)∧¬(t_i∈S))が成立しなければならない(¬は論理否定を，∧は論理積を表す)．このとき，$\boldsymbol{d}$ の変換Tは次の通りである．

T($\boldsymbol{d}$) = **delete** X **from** R; **and delete** X **from** S;

この変換Tは一意で，かつ副作用を発生させない．従って，和ビューは削除可能である．

(b) 和ビューは挿入不可

Vに対して挿入操作 $\boldsymbol{i}$ が発行されたとする．

$\boldsymbol{i}$ = **insert** X **into** V;

ここに，X = {t_1, t_2, ..., t_n}とする．そこで，$\boldsymbol{i}$ のデータベース操作への変換Tが存在するとすれば，Xが単集合(singleton)，X = {t}とする，に対してもそれが適用されねばならないが，このときはVの意味により t∈R∨t∈S が成立しなければならない．従って，$\boldsymbol{i}$ の変換として，$T_1(\boldsymbol{i})$ = **insert** t **into** R;, $T_2(\boldsymbol{i})$ = **insert**

t **into** S;, $T_3(\boldsymbol{i})$ = **execute both** $T_1(\boldsymbol{i})$ **and** $T_2(\boldsymbol{i})$ と3つの代替案があることになる．しかし，この曖昧性を $\boldsymbol{i}$ から解消することはできない[4]．従って，和ビューは挿入不可である．

(c)　和ビューは書換不可

定義2で述べたごとく書換操作は削除操作とそれに続く挿入操作の系列と定義されるので，和ビューは書換不可である．

(2)　差ビューの更新可能性

差ビューV = R−S の意味は，t∈V iff t∈R∧¬(t∈S)なので，差ビューは削除不可，挿入可能，従って，書換不可である．

(3)　共通ビューの更新可能性

共通ビューV = R∩S の意味は，t∈V iff t∈R∧t∈S なので，共通ビューは削除不可，挿入可能，従って，書換不可である．

3.2.2 射影ビューの更新可能性

リレーション R の射影ビュー V = R[C], ここに C は R の属性集合とするとき，その意味は t∈V iff (∃u∈R)(t = u[C])である．

(a)　射影ビューは削除可能

削除操作 $\boldsymbol{d}$ = **delete** X **from** V;，ここに X = {t_1, t_2, ..., t_n}，とすれば，T($\boldsymbol{d}$) = **delete** u **from** R **such that** (∃t∈X)(t = u[C]); に一意に，そして副作用なく変換できるので削除可能である．

(b)　射影ビューはキー保存の下で挿入可能

挿入操作についてはCがRのキーを含む[5], 即ち射影がキー保存であれば一意に副作用なく変換可能であるから，この条件の下で挿入可能である．

(c)　射影ビューはキー保存の下で書換可能

射影ビューはキー保存の下で書換可能である．

3.2.3 θ-選択ビューの更新可能性

リレーション R の属性 A_i と A_j 上の θ-選択ビュー V = R[$A_i \theta A_j$]については，その意味は次の通りである．

t∈V iff t∈R∧t[A_i] θ t[A_j]

(a)　θ-選択ビューは削除不可

V に対して削除操作 $\boldsymbol{d}$ が発行されたとする．

$\boldsymbol{d}$ = **delete** X **from** V;

ここに, X = {t_1, t_2, ..., t_n}とする．すると $\boldsymbol{d}$ は(∀t_k∈X)(¬(t_k∈R)∨¬(t_k[A_i] θ t_k[A_j]))の成立を意味する．従って，$\boldsymbol{d}$ の変換は，X = {t}と単集合(singleton)の場合でも成り立たないといけないから，この場合には次のようになる．

T($\boldsymbol{d}$) = **delete** t **from** R; **or else rewrite** t **of** R **to** t' **such that** ¬(t'[A_i] θ t'[A_j]);

しかし，$\boldsymbol{d}$ から変換 T が有するこの二者択一の曖昧性を解消することはできない．従って，θ-選択ビューは削除不可である．

なお，関連する議論を4.3節「同種変換の原理」で与える．

(b)　θ-選択ビューは挿入可能

V に対する挿入操作を $\boldsymbol{i}$ = **insert** X **into** V;，ここに，X = {t_1, t_2, ..., t_n}で，かつ(∀t_k∈X)(t_k[A_i] θ t_k[A_j])が成立しているとき，$\boldsymbol{i}$ は T($\boldsymbol{i}$) = **insert** X **into** R;に一意に副作用なく変換できる．従って，θ-選択ビューは挿入可能である[6]．

(c)　θ-選択ビューは書換不可

θ-選択ビューは書換不可である．

3.2.4 直積ビューの更新可能性

リレーション R(A)と S(B)，ここに A と B は一般に属性の集合で，A∩B = ϕ(空)とする，の直積ビューを V = R(A)×S(B) = {(r, s) | r∈R∧s∈S}とするとき，その意味は次の通りである．

t∈V iff t[A]∈R∧t[B]∈S

(a)　直積ビューは削除不可

V の意味から直接導かれるように，V に対する削除操作 $\boldsymbol{d}$ = **delete** X **from** V;，ここに X = {t_1, t_2, ..., t_n}，が変換可能であるとすれば，(∨t_i∈X)(¬(t_i[A]∈R)∨¬(t_i[B]∈S))が成立しなければならない．つまり，$\boldsymbol{d}$ の変換には，X = {t}とした場合には次に示す3つの代替案：$T_1(\boldsymbol{d})$ = **delete** t[A] **from** R;, $T_2(\boldsymbol{d})$ = **delete** t[B] **from** S;, $T_3(\boldsymbol{d})$ = **execute both** $T_1(\boldsymbol{d})$ **and** $T_2(\boldsymbol{d})$ があることになるが, この変換の曖昧性を $\boldsymbol{d}$ から解消することはできない[7]．従って，直積ビューは削除不可である．

(b)　直積ビューは挿入不可

V に対する挿入操作 $\boldsymbol{i}$ = **insert** X **into** V, ここに X = {t_1, t_2, ..., t_n}，が変換可能かどうか検証する．そこで，t_i=(t_i[A], t_i[B])として, V の意味から X の V への挿入により (∀t_i∈X)(t_i[A]∈R∧t_i[B]∈S)が成立しなければならない．従って，$\boldsymbol{i}$ の変換は T($\boldsymbol{i}$) = e**xecute both insert** X[A] **into** R; **and insert** X[B] **into** S;となる．この変換は一意であるが，V が直積ビューであることから，この変換により一般に副作用が発生する場合があり[8]，従って，直積ビューは挿入不可である．

(c)　直積ビューは書換不可

直積ビューは書換不可である．

以上の結果から，直積ビューは更新不可である．

3.2.5 θ-結合ビューの更新可能性

V = R[$A_i \theta B_j$]S をリレーション R と S の属性 A_i と属性 B_j 上の θ-結合ビューとする．ここに A_i と B_j は θ-比較可能とする．θ-結合演算の定義から，V = R[$A_i \theta B_j$]S = (R×S)[R.$A_i \theta$ S.B_j] であるので, V の更新可能性はまず θ-選択ビューに対する更新可能性が問われ，続いて直積ビューの更新可能性が問われることになる．上述の通り，θ-選択ビューは挿入可能ではあるが，一方直積ビューは更新不可なので，θ-結合ビューは更新不可である．

以上議論してきた従来のアプローチの下での7つの基本ビューの更新可能性の結果をまとめて表1の第3列に示す．

4. 更新意図の外形的推測アプローチ

4.1 更新意図の外形的推測とは

リレーショナルデータベース応用にとって，直積演算や結合演算は異なるリレーションに分散して格納されているデータを関連付けるために欠かせない演算である．しかしながら，前章で見た通り従来のアプローチの下では，これらの演算を使って定義され

[4] 真の意図はユーザに聞くしか分からない．

[5] 空(null)はいかなるドメインの元でもないので，リレーションのタップルの属性は空をとりえず，キーは主キー，候補キーを問わない．もし，空を許せば，キーは主キーとなる．

[6] R へのXの挿入で，R に課せられていた意味的制約に抵触することがあるかもしれない．データベースの一貫性を保つ立場からはそのような挿入操作を許すべきではないが，先述のごとく，定義2で与えられたビューへの更新操作の変換可能性はそこまでは形式化していない．

[7] ごく単純な例としては，V = R(A)×S(B),ここに R(A) = {1}, S(B) = {1}としたとき，直積ビューV に対する削除操作 $\boldsymbol{d}$ = **delete** (1, 1) **from** V;は(a) $T_1(\boldsymbol{d})$ = **delete** 1 **from** R;, (b) $T_2(\boldsymbol{d})$ = **delete** 1 **from** S;, (c) (a)と(b)を共に実行する，のいずれでも実現できるが，この曖昧性を $\boldsymbol{d}$ から解消することはできない．

[8] 例えば，V = R(A)×S(B),ここに R(A) = {1}, S(B) = {1}としたとき，直積ビューV に対する挿入操作 $\boldsymbol{i}$ = **insert** (2, 2) **into** V;は T($\boldsymbol{i}$) = e**xecute both insert** 2 **into** R; **and insert** 2 **into** S;と一意だが，V には(2, 2)以外に，(1, 2)と(2, 1)も挿入され副作用が生じる．

たビューは更新不可である．

しかしながら，これらのビューが更新不可とされてきた状況を検証してみると，ビューへの更新操作の変換可能性がこれまでデータベースの“スキーマレベル”で規定されてきたが故にこのような結果となっており，(そうではなく)もしそれを“インスタンスレベル”で考察し直すと更新可能と結論付けてよい場合が存在することを示し得る．それが「更新意図の外形的推測アプローチ」，略して「意図に基づくアプローチ」である[9, 10]．「更新意図の外形的推測」とは，更新操作の変換可能性をインスタンスレベルで検証していくために，ビューや中間ビューを一時的にマテリアライズ(materialize)し，それに対してユーザの発行した更新操作を適用してユーザの更新意図を推測し，その意図を満たす一意で副作用のない更新操作の変換が存在するのか否かを検証することを意味している．以下で，なぜこの新規アプローチによって従来のアプローチでは更新不可とされた直積ビューや結合ビューが場合により更新可能となるのか示す．まず，それをデータベース応用で多用される自然結合ビューを例にとり説明する．

【例題1】リレーション R(A, B)と S(B, C)の自然結合ビューV = R*S を考える[9]．ある時刻 τ における R と S のインスタンス，及びその時点での V のインスタンスを図2に示す[10]．

R

A	B
1	1
2	1
1	2

S

B	C
1	1
1	2
2	1

V (=R*S)

A	B	C
1	1	1
1	1	2
2	1	1
2	1	2
1	2	1

図2　自然結合ビューV (= R*S)のインスタンス

このとき，V に対して削除操作 d_1 が発行されたとする．

d_1 = **delete** {(1, 1, 1), (1, 1, 2)} **from** V;

V の意味は次の通りである．

t∈V iff t[A, B]∈R∧t[B, C]∈S

従って，V から X = {t_1, t_2}，ここに t_1 = (1, 1, 1)，t_2 = (1, 1, 2) とする，を削除するということは，各 t_i (i = 1, 2)に対して t_i∈V が否定されるということであるから，次が成立しなければならない(≡は同値を表す)．

(∀t_i∈{t_1, t_2})(¬(t_i[A, B]∈R)∨¬(t_i[B, C]∈S))
≡(¬(t_1[A, B]∈R)∨¬(t_1[B, C]∈S))
　∧(¬(t_2[A, B]∈R)∨¬(t_2[B, C]∈S))
≡(¬(t_1[A, B]∈R)∧¬(t_2[A, B]∈R))
　∨(¬(t_1[A, B]∈R)∧¬(t_2[B, C]∈S))
　∨(¬(t_1[B, C]∈S)∧¬(t_2[A, B]∈R))
　∨(¬(t_1[B, C]∈S)∧¬(t_2[B, C]∈S))

即ち，d_1 の変換候補を暴力法(brute-force)で列挙してみると次の4つとその組合せを含む計15通りあることが分かる[11]．

$T_1(d_1)$ = **delete** (1, 1) **from** R;
$T_2(d_1)$ = **delete** (1, 1) **from** R; **and delete** (1, 2) **from** S;
$T_3(d_1)$ = **delete** (1, 1) **from** S; **and delete** (1, 1) **from** R;
$T_4(d_1)$ = **delete** {(1, 1), (1, 2)} **from** S;

さて，意図に基づくアプローチでは，ビューV を一時的にマテリアライズして変換可能性を検証する[12]．マテリアライズされたビュー V^m = {(1, 1, 1), (1, 1, 2), (2, 1, 1), (2, 1, 2), (1, 2, 1)} は図2に示されたインスタンスの通りである．V^m に d_1 を施すと，その結果は{(2, 1, 1), (2, 1, 2), (1, 2, 1)}である．これが所望の更新結果で，ユーザの“更新意図”と推測される．

次に d_1 の4つの変換候補 $T_1(d_1)$～$T_4(d_1)$ に対して一つひとつその変換によって所望のビュー更新結果が実現できるか否かを「外形」(extension)を計算し比較して検証する．ここに外形とは一時的にマテリアライズされたビューに対して，それに更新操作を適用して得られた結果リレーションのことをいう．まず，$T_1(d_1)$ をデータベースに施すと更新結果は{(2, 1, 1), (2, 1, 2), (1, 2, 1)}と所望の結果となる．一方，$T_2(d_1)$ を施すと{(2, 1, 1), (1, 2, 1)}，$T_3(d_1)$ を施すと{(2, 1, 2), (1, 2, 1)}，$T_4(d_1)$ を施すと{(1, 2, 1)}となり，$T_2(d_1)$～$T_4(d_1)$ では副作用が発生することが分かる．従って，残る11通りの変換でも副作用が発生することになるから，結果として $T_1(d_1)$ が一意で副作用を起こすことなく図1の可換図式を成立させることが分かる．依って，この場合には自然結合ビューV への削除操作 d_1 は変換可能と判定し，$T_1(d_1)$ でデータベースを更新する．

勿論，意図に基づくアプローチにより自然結合ビューV に対するすべての削除操作が変換可能となるわけではない．次のような場合には削除操作は変換不可と判定される．

【例題2】ある時刻 τ における実リレーション R(A, B)，S(B, C)，R と S の自然結合ビューV = R*S は例題1の通りとする．このとき，V に対して削除操作 d_2 が発行されたとする．

d_2 = **delete** (1, 2, 1) **from** V;

V の意味記述から，(¬(1, 2)∈R)∨(¬(2, 1)∈S)が成立しないといけない．例題1と同様に暴力法の下で d_2 の変換には次の3つの代替案があることが分かる．

$T_1(d_2)$ = **delete** (1, 2) **from** R;
$T_2(d_2)$ = **delete** (2, 1) **from** S;
$T_3(d_2)$ = **execute both** $T_1(d_2)$ **and** $T_2(d_2)$

この場合，例題1とは異なり，何れの代替案を採っても d_2 を実現できる．しからば，3つの代替案のどれを採っても d_2 が実現できるのであれば，どれか1つを任意に選択すればよいのではないかと考えるかもしれないが，それは間違っている．何故ならば，これら3つの代替案が持つ意味が全く異なるからである．つまり，$T_1(d_2)$ は R のタップル(1, 2)が意味を失ったから V に対して d_2 が発行された場合に，$T_2(d_2)$ は S のタップル(2, 1)が意味を失ったから V に対して d_2 が発行された場合に，$T_3(d_2)$ はそれら2つが同時に発生したので V に対して d_2 が発行された場合に採られるべきであるが，削除操作 d_2 のみからではその曖昧性を解消することはできない．従って，この場合 d_2 は変換不可と結論される．

上記の議論をまとめると，“自然結合ビューは意図に基づくアプローチの下で場合により削除可能である”と結論することができる．同様な議論で，自然結合ビューは意図に基づくアプローチの下で場合により挿入可能であり，書換可能であることを示せる．

[9] V = R*S = ((R×S)[R.B=S.B])[A, R.B, C]と直積，等結合，射影演算を使って定義されるので，従来のアプローチでは更新不可である．

[10] ビューは定義だけが存在するが，ビューV のインスタンスは V をマテリアライズすると得られる．

[11] ${}_4C_1+{}_4C_2+{}_4C_3+{}_4C_4$ = 4+6+4+1 = 15．ここに，${}_nC_m$ = n!/(m!×(n-m)!)である．

[12] マテリアライズする代わりに，一時テーブル（temporary table）を生成しても同じことを行える．

続けて，直積ビューと結合ビューの更新可能性を意図に基づくアプローチの下で議論する．

4.2 意図に基づくアプローチの下での直積ビューの更新可能性

4.2.1 直積ビューの削除可能性

V = R(A)×S(B)をリレーションR(A)とS(B)，ここにA∩B = ϕ(空)，の直積ビューとする．直積ビューは3.2.4節で議論した通り，従来のアプローチでは削除不可である．しかしながら，意図に基づくアプローチでは場合により削除可能となる．例えば，R(A) = {1, 2, 3}，S(B) = {1}である場合，VをマテリアライズするとV^m = {(1, 1), (2, 1), (3, 1)}となり，このとき，意図に基づくアプローチの下では，例えば，削除操作 d = **delete** (1, 1) **from** V; にはVの意味から3つの変換候補：$T_1(d)$ = **delete** 1 **from** R;, $T_2(d)$ = **delete** 1 **from** S;, $T_3(d)$ = **execute both** $T_1(d)$ **and** $T_2(d)$ があるが，$T_1(d)$がdの一意で副作用のない変換であることが検証できるので，削除可能である．

なお，一般に，直積ビューV = R(A)×S(B)への削除操作 d = **delete** X **from** V;，ここにX = $\{t_1, t_2, ..., t_n\}$，はVの意味から，$(\forall t_i \in X)(\neg(t_i[A] \in R) \vee \neg(t_i[B] \in S))$[13]が成立しないといけないが，$d$の変換候補は暴力法の下では$T_1(d)$ = **delete** X[A] **from** R; $T_2(d)$ = **execute both delete** (X−$\{t_n\}$)[A] **from** R; **and delete** t_n[B] **from** S; ..., $T_2^n(d)$ = **delete** X[B] **from** S;で示される2^n個の変換候補とその組合せからなることが分かるが，$T_1(d)$～$T_2^n(d)$の中から一意で副作用のない変換が見つかれば，直積ビューは意図に基づくアプローチの下で削除可能である．

4.2.2 直積ビューの挿入可能性

V = R(A)×S(B)をリレーションR(A)とS(B)，ここにA∩B = ϕ(空)，の直積ビューとする．直積ビューは従来のアプローチでは挿入不可である．しかしながら，意図に基づくアプローチでは場合により挿入可能となる．

そこで，i = **insert** X **into** V;ここにX = $\{t_1, t_2, ..., t_n\}$，が発行されたとする．Vの意味から$(\forall t_i \in X)(t_i[A] \in R \wedge t_i[B] \in S)$[14]が成立しなければならない．従って，$i$は$T(i)$ = **execute both insert** X[A] **into** R; **and insert** X[B] **into** S;に変換されねばならない．明らかに，この変換は一意である．しかしながら，Vが直積ビューであることから，この変換により副作用が生じることは脚注8で示した通りである．従って，従来のアプローチでは直積ビューは挿入不可と結論されたが，意図に基づくアプローチでは，V∪X = (R∪X[A])×(S∪X[B])が成立するか否かを外形を計算することで判定できるので，等号が成立すれば変換可能，そうでなければ挿入操作は変換不可と結論できる[15]．

4.2.3 直積ビューの書換可能性と書換操作の両立性

従来のアプローチでは直積ビューは削除不可と挿入不可であるので，書換不可である(3.2.4節)．しかしながら，この状況は意図に基づくアプローチの下では変ってくる．まず，このことを例で示す．

【例題3】 実リレーションR(A, B)とS(C, D)の直積ビューV = R×Sのある時刻 τ におけるインスタンスを図3に示す通りとする．

R

A	B
1	1
2	2

S

C	D
1	1
2	2

V (=R×S)

A	B	C	D
1	1	1	1
1	1	2	2
2	2	1	1
2	2	2	2

図3 直積ビューV (= R×S)のインスタンス

いま，ユーザがVのタップル群{(1, 1, 1, 1), (1, 1, 2, 2)}をタップル群{(3, 3, 1, 1), (3, 3, 2, 2)} に書換えたいと書換操作 r_1 を発行したとする．

r_1 = **rewrite** {(1, 1, 1, 1), (1, 1, 2, 2)} **of** V **to** {(3, 3, 1, 1), (3, 3, 2, 2)};

前述の通り，書換操作は削除操作と挿入操作の系列であるので，r_1はまず削除操作 d_1を行い，続いて挿入操作 i_1を行うという系列と見なされる．

d_1 = **delete** {(1, 1, 1, 1), (1, 1, 2, 2)} **from** V;

i_1 = **insert** {(3, 3, 1, 1), (3, 3, 2, 2)} **into** V;

まず，d_1について意図に基づくアプローチの下でその変換可能性を検証すると，直積ビューVはd_1で削除可能であり，この時の一意で副作用のない変換は次の通りである．

$T_1(d_1)$ = **delete** (1, 1) **from** R;

続いて，r_1を実現させるためには，挿入操作i_1が変換可能でなければならないが，これは変換$T(i_1)$により実現できることが分かる[16]:

$T(i_1)$ = **insert** (3, 3) **into** R;

このとき，「変換$T_1(d_1)$と変換$T(i_1)$は書換操作r_1に関して両立(compatible)している」，略して「書換操作r_1が両立している」ということにする．

勿論，両立していない書換操作も存在する．例えば，図3に示される実リレーションR, Sと直積ビューV = R×Sに対して，書換操作r_2が発行されたとする．

r_2 = **rewrite** {(1, 1, 1, 1), (1, 1, 2, 2)} **of** V **to** {(1, 1, 3, 3), (1, 1, 4, 4)};

r_2はVからタップル群{(1, 1, 1, 1), (1, 1, 2, 2)}を削除し(この要求をd_2としよう)，続いてタップル群{(1, 1, 3, 3), (1, 1, 4, 4)}を挿入する(この要求をi_2としよう)操作の系列である．d_2はd_1と同じ操作なので，$T_1(d_1)$と同様に意図に基づくアプローチの下で削除可能である．しかしながら，i_2を実現しようとすると，直積の意味から，次の2つの挿入操作を共に実行しなければならないが，これらはSへの挿入操作を含んでおり，d_2とi_2は両立しえないことが分かる．

insert (1, 1) **into** R;

insert {(3, 3), (4, 4)} **into** S;

実際，r_2を実現しようとして，d_2と上記の2つの挿入操作を実行すれば，その結果は，次に示されるように，副作用が発生するので変換不可である．

V∪{(1, 1, 3, 3), (1, 1, 4, 4), (2, 2, 3, 3), (2, 2, 4, 4)}

以上，直積ビューは，書換操作が両立していれば意図に基づく

[13] $(\forall t_i \in X)(\neg(t_i[A] \in R) \vee \neg(t_i[B] \in S)) \equiv (\neg(t_1[A] \in R) \vee \neg(t_1[B] \in S)) \wedge ... \wedge (\neg(t_n[A] \in R) \vee \neg(t_n[B] \in S)) \equiv (\neg(t_1[A] \in R) \wedge \neg(t_2[A] \in R) \wedge ... \wedge \neg(t_n[A] \in R)) \vee (\neg(t_1[A] \in R) \wedge \neg(t_2[A] \in R) \wedge ... \wedge \neg(t_n[B] \in S)) \vee ... \vee (\neg(t_1[B] \in S) \wedge \neg(t_2[B] \in S) \wedge ... \wedge \neg(t_n[B] \in S))$

[14] $(\forall t_i \in X)(t_i[A] \in R \wedge t_i[B] \in S) \equiv (t_1[A] \in R \wedge t_1[B] \in S) \wedge ... \wedge (t_n[A] \in R \wedge t_n[B] \in S) \equiv (X[A] \in R \wedge X[B] \in S)$

[15] 例えば，V = R(A)×S(B)，ここにR(A) = {1}, S(B) = {1}としたとき，直積ビューVに対する挿入操作 i = **insert** (2, 1) **into** V;は$T(i)$ = **insert** 2 **into** R;に一意に副作用なく変換可能である．

[16] この検証を行う時点では，$T_1(d_1)$が実行されているので，マテリアライズされたビューはV^m = {(2, 2, 1, 1), (2, 2, 2, 2)}である．

アプローチの下で場合により書換可能であるが，両立していなければ書換不可である．

4.3 同種変換の原理

θ-選択ビューへの削除操作は，従来のアプローチでは変換の曖昧性により変換不可であった(3.2.3 節)．これは θ-選択ビューの意味記述から生じる本質的な変換の曖昧性によるものである．しかしながら，以下に示すように「同種変換の原理」を導入するとこの曖昧性を解消できて変換可能となる．本来，この原理は意図に基づくアプローチの考え方とは無縁であるが，θ-選択ビューと θ-結合ビューの更新可能性を高める効果があるので，便宜的に意図に基づくアプローチに組み込むこととする．

まず，例を挙げて同種変換の原理を説明する．リレーション社員(社員番号, 給与) の＜-選択ビュー(＝小なり選択ビュー) 貧乏社員 = 社員[給与<20] を定義し，それに対して削除操作 ***d*** = **delete** (007, 15) **from** 貧乏社員;が発行されたとする．3.2.3 節で述べた通り，この削除操作の変換候補は，(a) T_1(***d***) = **delete** (007, 15) **from** 社員;，そうでなければ，(b) (007, 15)を例えば(007, 30)に書換える操作：T_2(***d***) = **rewrite** (007, 15) **of** 社員 **to** (007, 30); の 2 通りが考えられる．従って，従来のアプローチではこの曖昧性から＜-選択ビューは削除不可と結論されている．

さて，θ-選択ビューV = R[$A_i \theta A_j$]の削除可能性を論じるとき，V に対して発行された削除操作は R に対する挿入操作や書換操作ではなくまた削除操作に変換されなければならないという規則を導入すると，***d***の変換候補は T_1(***d***)のみとなり，この下で副作用を生じさせることなく ***d*** を実現できる．これを同種変換の原理という．

この原理の正当性であるが，もしユーザが社員番号 007 の社員の給与を 15 から 30 にアップしたいのであれば，ユーザはビュー貧乏社員に対して ***d*** を発行するのではなく，その要求をリレーション社員に対して直接発行するべきであると考えられることに依る．

4.4 意図に基づくアプローチの下での θ-結合ビューの更新可能性

V = R[$A_i \theta B_j$]S をリレーション R と S の属性 A_i と属性 B_j 上の θ-結合ビュー，ここに A_i と B_j は θ-比較可能，とすると，θ-結合演算の定義から，V = R[$A_i \theta B_j$]S = (R×S)[R.$A_i \theta$S.B_j]であるので，3.2.5 節で述べた通り，V の更新可能性はまず θ-選択ビューに対する更新可能性が問われ，続いて直積ビューの更新可能性が問われることになる．

さて，θ-結合ビューV に対して発行された削除操作は同種変換の原理の下で一意に副作用を生じさせることなく直積ビューR×S への削除操作に変換でき，従って，4.2.1 節の結果により θ-結合ビューは意図に基づくアプローチの下で場合により削除可能となる．

一方，3.2.3 節で示した通り，θ-選択ビューは挿入可能であるので，θ-結合ビューV への挿入操作の変換可能性は直積ビューR×S への挿入操作の変換可能性にかかってくる．いま，V に対して挿入操作 ***i*** = **insert** X **into** V;が発行されたとすると，4.2.2 節で示した通り，意図に基づくアプローチでは，V∪X = (R∪X[A])×(S∪X[B])が成立するか否かを外形を計算することで判定できるので，等号が成立すれば変換可能，そうでなければ挿入操作は変換不可と結論できる．従って，θ-結合ビューは意図に基づくアプローチの下で場合により挿入可能となる．

以上の議論から，θ-結合ビューへの書換操作は同種変換の原理の下で書換操作が両立していれば意図に基づくアプローチの下で場合により書換可能である．

表 1 の第 4 列に意図に基づくアプローチの下での 7 つの基本ビューの更新可能性の結果をまとめて示す．

表 1　7 つの基本ビューの更新可能性一覧

基本ビューの種類	更新操作	従来のアプローチ	意図に基づくアプローチ
和ビュー (∪)	削除	○	○
	挿入	×	×
	書換	×	×
差ビュー (−)	削除	×	×
	挿入	○	○
	書換	×	×
共通ビュー (∩)	削除	×	×
	挿入	○	○
	書換	×	×
直積ビュー (×)	削除	×	◎
	挿入	×	◎
	書換	×	◎†2
射影ビュー	削除	○	○
	挿入	○†1	○†1
	書換	○†1	○†1
θ-選択ビュー	削除	×	○†3
	挿入	○	○
	書換	×	○†3
θ-結合ビュー	削除	×	◎†3
	挿入	×	◎
	書換	×	◎†3,†2

○や◎が更新可能[17]，×が更新不可を表す．†は成立条件を表す：†1 キー保存，†2書換操作の両立，†3 同種変換の原理

5. 意図に基づくアプローチの実現可能性検証

5.1 ビューの更新可能性判定アルゴリズム

一般的に定義されたビューは表 1 に示されている 7 つのビュー定義演算を再帰的に適用して得られるので，そのビュー定義は木構造で表現することができる．これを「ビュー定義木」という．7 つのビュー定義演算は単項あるいは 2 項の演算なので，一般にビュー定義木はビューを根とする 2 分木となる．葉は実リレーションを，根や葉以外のノードは中間ビューを表す．集合演算の存在から[18]，この木は順序木となり，幅優先探索が可能となる．従って，根に 0，それ以降は 1，2, . . . , N_{max} と幅優先探索順にノードを番号付けすることができる．

ビューへの更新操作の変換可能性は 7 つの基本ビューの更新可能性を表した表 1 の第 4 列を表参照(table lookup)しながら検証するが，そのためにビュー定義木のノードは VNAME フィール(ノードの識別子を格納)，VDEF フィールド(ノードを定義するために使われたリレーショナル代数演算を具体的に格納．ノードが葉の

[17] ◎は意図に基づくアプローチの下での更新可能性を表すが，○と◎を使い分けたのは，5.2 節で示すビュー更新操作の変換可能性判定アルゴリズムのためである．

[18] R−S≠S−R である．ここに，−は差集合演算を表す．

場合 Base とする)，LLINK フィールドと RLINK フィールド(そのビューを定義するために使われた中間ビューや実リレーションを表すノードへのポインタを格納．それらがない場合は NULL とする)からなる．このとき，意図に基づくアプローチの下での一般的に定義されたビューへの更新操作の変換可能性を判定する「アルゴリズム D」はステップ D1 から始まり，順次 D2，D3, . . . , D7 と続くが，条件により前後にスキップする．なお，同種変換の原理は常に成立しているとする．

【アルゴリズム D】 V をビューとし，u を V に対して発行された更新操作とする．P (0≦P≦N_{max})をノード番号を値としてとるポインタ変数，UREQ(P) を NODE(P)への更新操作，UREL(P) を NODE(P)の更新操作への関連度，とする．

D1. [初期化] V のビュー定義木を作る．P ← 0 (零), UREQ(0) = u, 全ての P (0≦P≦N_{max})に対して UREL(P) = 1 (1 は“関連有”，0 は“関連無”)と設定する．

D2. [終了検出] もし P = N_{max} なら，D7 に行く．そうでなければ，もし UREL(P) = 1 なら, D3 に行く． もし UREL(P) = 0 なら, P ← P+1 として D2 に戻る．

D3. [表参照] 表 1 の第 4 列を参照し，対 VDEF(P) - UREQ(P) の更新可能性を見る．もし“◯”，“◎”，“×”なら，それぞれ D4，D5，D6 に行く．

D4. ① VDEF(P) = ∪，−，∩の場合，UREQ(P)を LLINK(P)と RLINK(P)でリンクされた部分木の根に対する更新操作 UREQ(LLINK(P)) と UREQ(RLINK(P))にそれぞれ然るべく変換し，P ← P+1 として D2 に戻る．

② VDEF(P) = θ-選択演算の場合，同種変換の原理の下で，UREQ(P)を LLINK(P)でリンクされた部分木の根に対する更新操作 UREQ(LLINK(P))に然るべく変換し，P ← P+1 として D2 に戻る．

③ VDEF(P) = 射影演算の場合，UREQ(P) = 削除操作なら，UREQ(LLINK(P))を然るべく設定し，P ← P+1 として D2 に戻る．もし UREQ(P) = 挿入演算か書換操作なら，射影がキー保存ならば UREQ(LLINK(P))を然るべく設定し，P ← P+1 として D2 に戻る．キー保存でなければ，D6 に行く．

D5. ① VDEF(P) =直積演算の場合，UREQ(P) = 削除操作か挿入操作なら，NODE(P) を一時的にマテリアライズし，UREQ(P)を実行する．その結果に基づき UREQ(P)を補完し，それを UREQ(LLINK(P))と UREQ(RLINK(P))に変換する候補を求める．次に，NODE(LLINK(P))と NODE(RLINK(P))を一時的にマテリアライズし，それらを使って各候補について外形を求める．もし, 副作用のない変換がただ一つ見つかれば，UREQ(P)を LLINK(P)と RLINK(P)でリンクされた部分木の根に対する更新操作 UREQ(LLINK(P)) と UREQ(RLINK(P))に然るべく変換し，P ← P+1 として D2 に戻る．このとき，UREQ(RLINK(P))が空，つまり UREQ(P)が RLINK(P)への更新操作に変換されることがなかったならば，RLINK(P)でリンクされた部分木の全てのノードに対して UREL(P) = 0 と設定する．もし UREQ(LLINK(P))が空なら，LLINK(P)でリンクされた部分木の全てのノードに対して UREL(P) = 0 と設定する．それら以外では UREL(P)の操作はしない．もし，一意で副作用のない変換が見つからなければ，D6 に行く．UREQ(P) = 書換操作の場合，上記と同様に処理をするが，もし書換操作が両立していなければ，D6 に行く．

② VDEF(P) = θ-結合演算の場合，UREQ(P)の処理は θ-選択演算とそれに続く直積演算に掛かる処理の合成となるが，同種変換の原理の下では UREQ(P)はそのまま θ-結合ビューを定義している直積ビューへの更新操作に変換されるので，以後①と同様に処理する．

D6. [更新不可] V に対して発行された更新操作 u は変換不可である．

D7. [更新可能] V に対して発行された更新操作 u は変換可能である．

5.2 PostgreSQL 上での意図に基づくアプローチの実現可能性検証

意図に基づくアプローチを PostgreSQL 10 上にプロトタイピングして実現可能性検証を行った結果をまとめて報告する[11, 12]．プロトタイプシステムの概要を図 4 に示す．

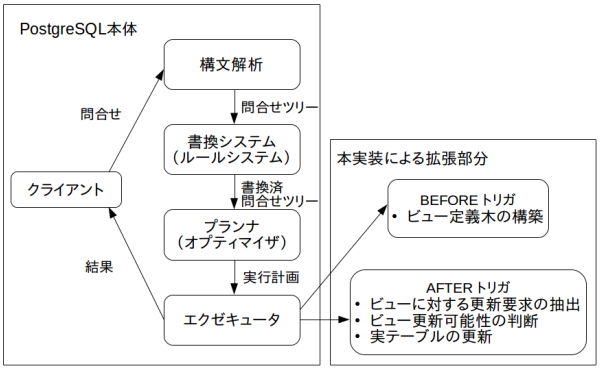

図 4 プロトタイプシステムの概要

プロトタイピングの要点を述べる．本来，PostgreSQL のビューサポート機能は書換システム(＝ルールシステム)で実装されているので[13]，本論文で提案している新規アプローチも書換システムに組込んで実装することを検討したが，新規アプローチはビューの更新可能性判定にビューや中間ビューを一時的にマテリアライズすることを必要としている．しかしながら，それを書換システム内で行うことには無理があった．そこで，その機能をエグゼキュータの拡張機能として実装することとした．より具体的には，それを文レベルの BEFORE トリガ(trigger)と AFTER トリガ機能を使用して実装した[19]．BEFORE トリガは専らビュー定義木を構築する．その結果を受けて，AFTER トリガは PostgreSQL 10 からサポートされ始めた遷移表（transition table）[20]を活用して[14]，ビューに対する更新要求をタップル集合として取得し，アルゴリズム D に従って，表 1 の第 4 列を表参照しながら，ビューに対して発行された更新操作の変換可能性を判定している．

図 5 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(a, b)と base2(b, c)の自然結合として定義されたビューである．現行の PostgreSQL では，ビューサポートの機能としては SQL-92 標準で規定されている機能しか実装されていないために結合ビューへの更新操作は変換不可であり，同図(a)のようにエラー

[19] PostgreSQL では文レベルの INSTEAD OF トリガをサポートしていないため，その機能を BEFORE トリガと AFTER トリガで実現している．

[20] 遷移表は SQL 標準で規定されており，トリガを駆動した更新操作の実行中に削除，挿入，および書換られたタップル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．

となる(この例はvへ2本のタップルを挿入しようとしている). しかしながら，意図に基づくアプローチをPostgreSQL 10に組み込んだ結果，この自然結合ビューに対して，同図(b)のように更新が可能になることが確認された．一方，ビュー更新が，更新意図の外形的推測の結果として更新不可と判断された場合には，同図(c)のようにエラーとなる．これは意図された振る舞いである．

```
=# 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.
```

(a) 従来のPostgreSQLで更新不可とされる場合

```
=# 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
```

(b) 意図に基づくアプローチの下で更新可能な場合

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

(c) 意図に基づくアプローチの下で更新不可とされる場合

図5　本プロトタイプの動作例

6. おわりに

リレーショナルデータベースにおいて結合演算は大変重要な演算であり，「結合ビューを更新可能としたい」という願望はリレーショナルデータベースの現場に根強くある．しかしながら，SQL-92で更新可能なビューは制約の強い1枚のテーブルでしかない．その後の改正でSQL:1999で一定の条件を満たす結合ビューや和ビューが更新可能となった[15, 16]．しかしながら，そこで更新可能とされる結合ビューは，例えばOracle Databaseでの実装に見るように[17]，一枚のキー保存テーブル(key-preserved table)に他のテーブルの列を付随させたテーブルであり，更新はそのテーブルの列にしか許されず制約が強い．一方，INSTEAD OFトリガを使えばどのようなビューも更新可能になるとその使用を勧める動向が見られる[18]．PostgreSQLも例外ではないが，INSTEAD OFトリガのコアコードをデータベースの一貫性を損なうことなく書き下す指針はどこにも示されてはいない．「意図に基づくアプローチ」はまさしく，このような状況を打破しようと考案された新しい取り組みである．

しかしながら，この新規アプローチをリレーショナルデータベースの現場に適用していくためには，少なくとも次の2つの問題について更に考察を重ねる必要がある．1つは性能問題である．提案された新規アプローチの下で，例えば直積ビューの削除可能性を判定しようとすると，削除したいタップルの数をnとしたとき，2^n個の変換候補を検証の対象としなければならない(4.2.1節)．これは変換候補の列挙のために暴力法を用いているからであり，今後これに代わるより洗練された手法の提案が必要である．もう一つはリレーショナルデータベースの現場ではビューはリレーショナル代数ではなく，SQLで定義される．脚注1で指摘したように，SQLはバッグ意味論に基づいている．従って，集合意味論の下で展開された意図に基づくアプローチをテーブルに行の重複出現を許すバッグ意味論の下での議論に拡張する必要がある．

謝辞

本研究は一部JSPS科研費16K00152の助成を受けて遂行された．末筆ながら，的確なコメントを下さった査読者に感謝する．

参考文献

[1] Stonebraker, M., Implementation of integrity constraints and views by query modification, Proc. 1975 ACM SIGMOD, pp. 65-78, 1975.
[2] Codd, E., Recent investigations in a relational database system, Information Processing 74, pp. 1017-1021, North-Holland, 1974.
[3] Dayal, U. and Bernstein, P., On the updatability of relational views, Proc. 4th VLDB, pp.368-377, 1978.
[4] Bancilhon, F. and Spyratos, N., Update semantics of relational views, ACM TODS, Vol.6, No.4, pp.557-575, 1981.
[5] Cosmadakis, S. and Papadimitriou, C., Updates of relational views, ACM PODS, pp.317-331, 1983.
[6] Masunaga, Y., A relational database view update translation mechanism, Proc. 10th VLDB, pp.309-320, 1984.
[7] Sheth, A., Larson, J., and Watkins, E., TAILOR, A Tool for Updating Views, LNCS, Vol. 303, pp.190-213, Springer, 1988.
[8] Date, C. J. View Updating and Relational Theory: Solving the View Update Problem. 240p., O'Reilly, 2013.
[9] 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), 8p., 2015.
[10] 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.
[11] 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.
[12] 長田悠吾，石井達夫，増永良文，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—，第10回Webとデータベースに関するフォーラム (WebDB Forum 2017), 6p., 2017.
[13] "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).
[14] "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).
[15] ISO/IEC 9075-2:1999 Information technology -- Database languages -- SQL -- Part 2: Foundation (SQL/Foundation), 1999.
[16] Melton, J. and Simon, A., SQL:1999 Understanding Relational Language Components, 893p., Morgan Kaufmann, 2002.
[17] "Database Administrator's Guide, 24 Managing Views, Sequences, and Synonyms, 24.1 Managing Views," Oracle Database Online Documentation Library, 12c Release 1 (12.1.0.2) https://docs.oracle.com/database/121/ADMIN/views.htm#ADMIN020 (参照 2018-12-22).
[18] Rielau, S., INSTEAD OF Triggers - All Views are Updatable! https://www.ibm.com/developerworks/data/library/techarticle/0210rielau/0210rielau.html (参照 2018-12-22).