双方向化技術によるデータベーススキーマの共存

田中　順平[†,††]　Van-Dang Tran[†,††]　加藤　弘之[††,†]　胡　振江[†††,††]

† 総合研究大学院大学　〒 240-0193 神奈川県三浦郡葉山町（湘南国際村）
†† 国立情報学研究所　〒 101-8430 東京都千代田区一ツ橋 2-1-2
††† Peking University　No.5 Yiheyuan Road Haidian District, Beijing, P.R.China 100871
E-mail: †{jumpeitanaka, dangtv, kato}@nii.ac.jp, †††zhenjianghu@pku.edu.cn

あらまし　ビジネス変化へ対応すべく，複数のアプリケーションバージョンを管理する技術が進展している．データベースにおいても，アプリケーションバージョンに対応した複数のデータベーススキーマの共存による貢献が期待される．一方，共存しつつそれぞれが 1 つのデータベースのように利用できるには，任意のデータ更新に各スキーマが対応し，必要に応じて破綻なく互いに更新を反映させることが求められ設計が難しい．既存研究ではそのデータ更新のルール設計が複雑であり，ユーザーの意図を反映できない場合もあり，課題を有している．本論文では双方向化技術を応用し，これらの課題を解決する手法を提案する．本手法によるデータ更新の整合性を検証し，代表的なケースに対してその有効性を確認した．

キーワード　データベーススキーマの共存, 双方向変換, 全域関数

1　は じ め に

継続的かつ急速に変化するビジネスを背景に，ソフトウェアの漸進的，進化的な開発手法への期待が高まっている．アジャイルや DevOps に代表されるこれらの手法においては，その根幹として複数のアプリケーションバージョンを管理する技術が進展している [1]．GIT や SVN は，アプリケーションのバージョン管理を実現する強力なツールである．他方データについては，データベースにアプリケーションに対応したデータベーススキーマ（以下，スキーマとも記す）を用意し，管理するのが一般的である．しかしながら，既存のデータベース管理システムでは，スキーマが進化した場合に，進化前の古いスキーマと進化後の新しいスキーマを共存させ，あたかもそれぞれが独立した一つの DBMS として稼働するような機能はサポートされておらず，ソフトウェアの漸進的な開発の制約となっている．ここでいうデータベーススキーマの共存とは，アプリケーションの進化に伴い元となるスキーマを進化させ新しいスキーマを定め，個々のアプリケーションで発生するデータをそれぞれのスキーマで管理し，また互いのデータの更新を必要に応じて破綻なく反映させることである（図 1）．しかし破綻のない双方向でのデータ更新の反映の設計の難しさと，データの変換，移行の作業は多くが手作業を伴い誤りを起こしやすいことから [1]，スキーマの共存を困難なものにしている．

これに対しリレーショナルデータベースにおけるスキーマの共存を目的した MSVDBs（Multi-schema-version data management systems）が，K. Herrmann らによって提案されている [2, 3]．スキーマ進化を古いスキーマから新しいスキーマへのビューとして Datalog を用いてを定義し，また新しいスキーマのデータ更新を古いスキーマのデータに破綻のなく反映させることを，「ビュー更新」と捉えることで対応している．この時，古いスキーマから構成されるデータベースと新しいスキーマから構成されるデータベースそれぞれが，あたかも独立した DBMS として機能するためには，古いスキーマのデータだけでなく新しいスキーマのデータに対しても任意の更新ができるようにする必要がある．つまり，新しいスキーマであるビューに対する更新のうち，古いスキーマのデータに反映できない更新を考慮する必要がある．このような更新に対して，K. Herrmann らは，補助リレーションを導入し古いスキーマのデータに反映できないタプルを管理することで，スキーマ共存を実現している．スキーマ進化を基本的な操作（SMO, Schema Modification Operation）に分類し，それぞれのビュー定義，ビュー更新の反映ルール，補助リレーションを設計している．しかしながら，MSVDBs には以下の二つの課題がある．

(A)　**（ビュー更新の曖昧さ）** 1 つのビュー定義に対して整合性あるビュー更新の反映ルールが複数存在する曖昧さがあり [4, 5]，予め設計された 1 つのビュー更新の反映ルールでは，ユーザの意図を反映できない場合がある．

(B)　**（補助リレーションの個別作り込み）** 補助リレーションを設計する体系的な手法が存在していない．SMO 毎に個別に作り込まれ，最大 5 つの補助リレーションが用意されているものの，その必然性や最小性が判じられていない．

本研究では，これらの課題を双方向化技術を用いて解決する．双方向化技術とは双方向変換を構築するための技術であり，双方向変換は，ソースデータをターゲットデータに変換した後，ターゲットデータ上の更新をソースデータに反映させることが可能な計算の枠組みのことである [6]．プログラミング言語の分野で研究されている技術であり，データベースのビュー更新問題を解決する手法としても注目されている．双方向変換は，順方向変換 *get* と逆方向変換 *put* のペアから構成される．順方向変換 *get* はビュー定義とみなすことができ，SMO におけるスキーマ定義が *get* に対応する．これに対して逆方向変換 *put* は

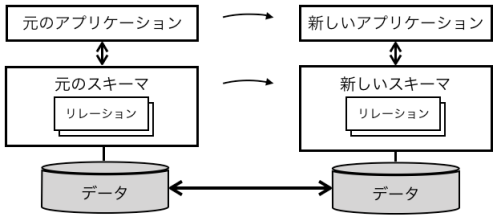

図 1　データベーススキーマの共存

ビュー更新の反映ルールとみなすことができ，SMO における新しいスキーマのデータ更新を古いスキーマと補助リレーションのデータへの反映するルールが *put* に対応する．双方向変換の枠組みを用い，スキーマの共存に対して本研究では以下の点について貢献する．

- （スキーマの埋め込みによるスキーマ進化）*get* によりスキーマ進化を定める SMO に対し，本研究では先に *put* を記述し，後述する putback-based の双方向化技術を用いた手法により一意に *get* を出力する．これにより課題 (A) を解決する．
- （*put* の全域関数化）本研究では， 元のスキーマのデータに反映できない新しいスキーマでのデータ更新は，*get* の値域を越える更新であることを明らかにする．これを管理することを目的に，高々 1 つの補助リレーションを導入する．そのビュー更新の反映ルールを機械的に導出し *put* に追加することで *put* を全域関数にし，課題 (B) を解決する．

以降，2 章で本論文で用いる基本的な事項を説明し，3 章でスキーマの共存の例を示し，4 章で提案手法を説明する．5 章で実装について説明し，6 章で提案手法の適用を示す．7 章で関連研究との関係を説明し，8 章で結論と今後の課題を述べる．

2 準　　備

本論文で用いる表記法と双方向化技術について記す．

2.1 表　記　法

リレーションを $R_i(x_{i.1}, ..., x_{i.n})$ と表す．その属性の集合を X_i とし，$R_i(X_i)$ あるいは単に R_i とも表す．ビューも同様に表す．有限個のリレーションを $R_1, ..., R_m$ とするとき，データベーススキーマを $\mathrm{S}\langle R_1(X_1), ..., R_m(X_m)\rangle$ と表す．

2.2 双方向変換

双方向変換は，順方向変換 *get* と逆方向変換 *put* のペアから構成される [6]．順方向変換 $T = get(S)$ は，変換元のソースデータ S を変換先のターゲットデータ T へ変換する．逆方向変換 $S' = put(S, T')$ は，一般的に *get* は単射とは限らず全てのソースデータがターゲットデータに反映されるとは限らないため，元のソースデータ S と更新されたターゲットデータ T' から，更新されたソースデータ S' へ変換する．双方向での変換の整合性を保証するために，*get* と *put* のペアは次の round-tripping law を満たすことが求められる．

$$put(S, get(S)) = S \quad \text{(GetPut)}$$
$$get(put(S, T')) = T' \quad \text{(PutGet)}$$

GetPut は「変換後のターゲットデータが更新されないときは，元のソースデータは更新されないこと」を表しており，PutGet は「更新されたターゲットデータは，更新されたソースデータから得られること」を表している．

スキーマの共存におけるビュー更新との対応では，元のスキーマのデータがソースデータに，新しいスキーマのデータがターゲットデータに相当する．ビュー定義が *get* に，ビュー更新の元のスキーマのデータへの反映が *put* に相当する．

2.3 putback-based の双方向化技術

双方向化技術とは，*get* と *put* ペアである双方向変換において，一方から他方を導出する技術である．その際 *get* からは *put* が一意に定まらない曖昧さがあり，これを解決する手法として *put* から *get* を一意に導出する putback-based の双方向化技術が提案されている [7,8]．特に V. D. Tran らはリレーショナルデータベースにおいて更新可能なビューを実現する putback-based の双方向化技術（BIRDS）を提案している [9,10]．本研究ではこの手法を利用する．BIRDS は *putdelta* の Datalog プログラムを入力とし，*get* の Datalog プログラムを出力する．*putdelta* を次に定める．

定義 2.1.（***putdelta***）ソースデータを S，更新されたターゲットデータを T' とする．ソースデータの更新分を ΔS とする．*putdelta* は，S と T' より S の更新分 ΔS を変換する．

$$\Delta S = putdelta(S, T')$$

ここで更新分 ΔS は，ソースデータ S に挿入するタプルの集合 $+S$ と，S から削除するタプルの集合 $-S$ からなる．更新されたソースデータ S' との関係を，演算子 $\oplus$ を用いて集合意味論により次のように表す．

$$S' = S \oplus \Delta S = (S \setminus -S) \cup +S \tag{2.1}$$

put は *putdelta* を定めることにより定められる．

定理 2.2.（***putdelta* と *put***）

$$S' = S \oplus putdelta(S, T') = put(S, T')$$

putdelta の例とその Datalog プログラムを示す．

例 2.3. ソースデータをリレーション $R_1(x)$ のインスタンス，ターゲットデータをビュー $T(x)$ のインスタンスとする．*putdelta* を，更新された $T(x)$ のインスタンスから，更新が $x < 4$ を満たす挿入，削除による場合に，それを $R_1(x)$ への挿入分 $(+R_1(x))$，削除分 $(-R_1(x))$ のインスタンスに変換する関数とする．Datalog により次のように表す．

$$\begin{aligned} +R_1(x) \quad &:- T(x), \neg R_1(x), x < 4. \\ -R_1(x) \quad &:- \neg T(x), R_1(x), x < 4. \end{aligned} \tag{2.2}$$

タプルの集合として $R_1(x)$ のインスタンスを $\{(1),(2)\}$，$T(x)$ のインスタンスを $\{(1),(2)\}$ とする．$T(x)$ にタプル (3) が挿入され $\{(1),(2),(3)\}$ となる場合には，$T(x)$ にあり $R_1(x)$ になく $x < 4$ を満たすタプルの集合として，挿入分 $+R_1(x) = \{(3)\}$ に変換される．また $T(x)$ からタプル (2) が削除され $\{(1)\}$ となる場合には，$T(x)$ になく $R_1(x)$ にあり $x < 4$ を満たすタプルの集合として，削除分 $-R_1(x) = \{(2)\}$ に変換される．

3　データベーススキーマの共存

3.1　データベーススキーマの共存の定義

データベーススキーマの共存 [2,3] とは，1 つのデータベースにおいて，1) 元のスキーマとデータから新しいスキーマとそのデータへ進化させることができ，2) それらが同時に利用可能であり，3) それぞれのスキーマは 1 つのデータベースのようにアプリケーションが扱うデータの read/write に対応でき，4) 必要に応じて互いのスキーマでのデータ更新を整合性を保って反映できること，である．

データ更新の整合性は，一方のスキーマのデータ更新を他方のスキーマのデータに反映した際，その更新された他方のスキーマのデータから元のスキーマのデータへ変換しても，互いに元の更新が保持され，更新分以外の既存のデータには影響がないこと，とする．

3.2　データベーススキーマの共存の例

社員データシステムを例にスキーマの共存を示す．はじめに全社員データを管理するアプリケーションが用意され，次にこれを進化させ，リサーチ部門の社員データを管理する新しいアプリケーションが作られたものとする．2 つのアプリケーションは共存し，それぞれのアプリケーションに対応したスキーマも共存するものとする．

(a) スキーマの定義　図 2(a) にスキーマの定義を示す．元のアプリケーションに対する元のスキーマはリレーション `EMP` を持つ．`EMP` は属性に名前 (`name`), 雇用タイプ (`type`)，部門番号 (`dno`) を持ち，全社員データを持つ．新しいアプリケーションに対する新しいスキーマはリレーション `RD` を持つ．`RD` は `EMP` と同じ属性を持ち，リサーチ部門に所属するフルタイムの社員データを持つものとする．

(b) スキーマ進化　元のスキーマとデータから新しいスキーマとそのデータへの進化を，図 2(b) に示す．新しいスキーマに対し，元のスキーマの `EMP` のデータをリサーチ部門 (`dno=1`) とフルタイムの社員 (`type=F`) により selection することにより，新しいスキーマの `RD` のデータに変換する．

(c) 新しいスキーマのデータ更新と元のスキーマのデータへの反映　更新として，新しいスキーマへタプルを挿入する例を，図 2(c) に示す．リレーション `RD` へ挿入したタプル `(Ken,F,1)` は，(b) で定めたスキーマ進化のデータ変換のルール（リサーチ部門 (`dno=1`) とフルタイムの社員 (`type=F`)）を満たすため，元のスキーマのリレーション `EMP` に反映される．更新が反映された `EMP` のデータを再び `RD` のデータへ変換した際に，この更新が保持され整合性を保つためである．スキーマ進化におけるデータ変換を関数とみなした場合，新しいスキーマでのデータ更新がその関数の値域内となっている．

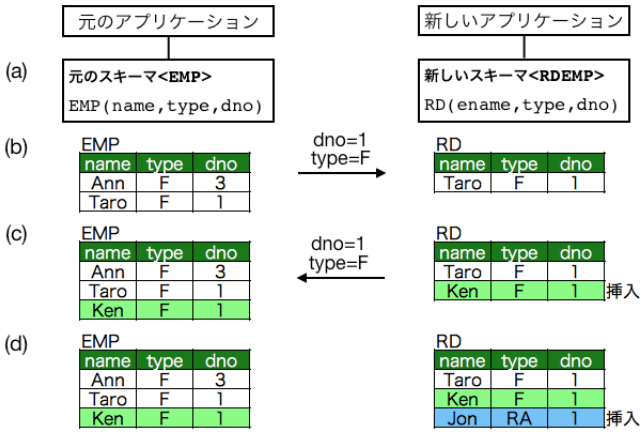

図 2　社員データシステムの例

(d) 新しいスキーマのデータ更新を元のスキーマのデータへ反映できない場合　図 2(d) に示す．リサーチ部門に独自な雇用タイプとしてリサーチアシスタント (`type=RA`) が発生したとして，それを新しいスキーマのリレーション `RD` に挿入した例である．挿入したタプル `(Jon,RA,1)` は (b) で定めたスキーマ進化のデータ変換のルールを満たさず，`EMP` に反映されない．仮に `EMP` に反映させたとしても，この挿入したタプルへは変換できず整合性を保てないためである．スキーマ進化におけるデータ変換を関数とみなした場合に，新しいスキーマでのデータ更新がその関数の値域を超えている．

スキーマの共存には，各スキーマ独自のデータを保持しつつ，スキーマ間でのデータの双方向な変換が求められる．双方を実現する手法を，次章以降で説明する．

4　双方向変換を用いたキーマの共存手法

概要を述べたのち，新しいスキーマのデータ更新が元のスキーマに反映される場合と，反映できない場合とに分け，手法の詳細を説明する．

4.1　概　　要

スキーマの共存を双方向変換の枠組みを用いて実現するための構成を，図 3 に示す．1 つのデータベースにおいて，各スキーマはビューにより構成される．ネイティブスキーマは物理データを持ち，各スキーマの補助リレーションと，特定のスキーマのビューに 1 対 1 に対応するベースリレーションとから構成される．ベースリレーションと 1 対 1 に対応する場合を除き，各スキーマのビューのデータは，*get* による元となるスキーマと補助リレーションのデータの変換として得られ，その更新は *put* により元となるスキーマか補助リレーションのデータに反映される．

次にスキーマの共存を実現する双方向変換である get_{total} と put_{total} を導出するための手法を，図 4 に示す．はじめに，元

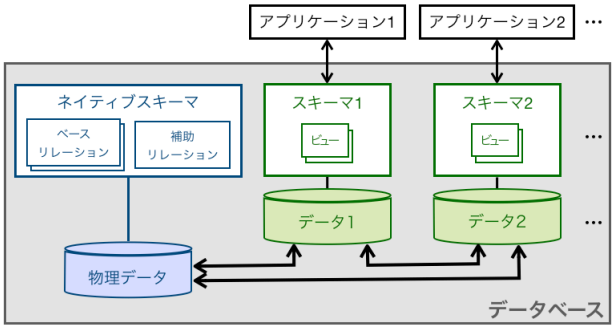

図 3　スキーマの共存の構成

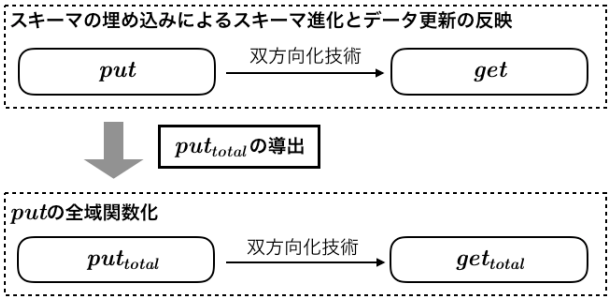

図 4　双方向変換を用いたスキーマの共存の概要

のスキーマとデータから新しいスキーマとそのデータへの進化を，逆方向に，スキーマの埋め込みとして定める．新しいスキーマ上でのデータ更新を元のスキーマのデータ更新に反映させる put として記述する．双方向化技術により，put に対応するビュー定義となる get を出力する．次に，新しいスキーマ上でのデータ更新を元のスキーマのデータへ反映できない場合に，その更新を補助リレーションに反映する put_{total} を全域関数として導出する．双方向化技術により，put_{total} に対応したビュー定義となる get_{total} を出力する．

4.2　スキーマの埋め込みによるスキーマ進化と元のスキーマへのデータ更新の反映

スキーマ進化を，新しいスキーマのデータの更新を元のスキーマのデータへ反映させる**スキーマの埋め込み**により定める．新しいスキーマの 1 つのビューのデータの更新を，逆方向に元のスキーマを構成する各ビューのデータに反映するルールとして，双方向変換の put として記述する．3.2 節の例 (c) にあるような，逆方向のデータ変換に相当する．ここで元のスキーマを構成するビューのデータは，別のスキーマのデータを変換して得られるか，ベーステーブルが与えられその物理データと 1 対 1 に対応しているものとする．先に put を定めることで，2.3 節で述べた putback-based の双方向化技術を用いて，GetPut, PutGet を満たす整合性のある get を一意に出力する．得られた get は，元のスキーマのデータを新しいスキーマのデータへ変換するビュー定義であり，3.2 節の例 (b) の順方向なデータ変換に相当する．以下，元のスキーマを有限個のビューからなる $\mathrm{S_O}\langle R_1(X_1), ..., R_m(X_m)\rangle$ とし，新しいスキーマを 1 つのビューからなる $\mathrm{S_N}\langle T(Y)\rangle$ とする．

定義 4.1.（元のスキーマへデータ更新を反映する双方向変換）元のスキーマのデータをソースデータ S とする．新しいスキーマのデータを，ターゲットデータ T_{in} とする．更新されたそれぞれを S', $T_{in}^{'}$ とする．ソースデータからターゲットデータへの順方向変換を get により $T_{in} = get(S)$ とし，逆方向変換を put により $S' = put(S, T_{in}^{'})$ とする．

また定理 (2.2) により，put は次の $putdelta$ により定められる．

定義 4.2.（元のスキーマへデータ更新を反映する ***putdelta***）ソースデータを S，更新されたターゲットデータを $T_{in}^{'}$ とする．$putdelta$ は，S と $T_{in}^{'}$ より S の更新分 ΔS に変換する．

$$\Delta S = putdelta(S, T_{in}^{'})$$

4.3　*put* の全域関数化

4.2 節で定める整合性ある get, put において，put により元のスキーマのデータに反映される新しいスキーマのデータ更新は，get の値域内に限られる．get の値域を越えるデータは，元のスキーマのデータを get により変換しても得られないためである．3.2 節の例 (d) のように，get の値域を越える更新は元のスキーマのデータを更新しない．$putdelta$（定義 4.2）であれば，get の値域を越える更新に対しては $putdelta$ が定義されず，更新分 ΔS を計算しない場合となる．一方スキーマの共存においては，新しいスキーマは任意のデータ更新に対して read/write できる必要がある．そのため，1 つの補助リレーションを設け，get の値域を越える更新であり $putdelta$ が定義されない場合には，そこに反映する．このルールを put に加え，全域関数として put_{total} を定める．putback-based の双方化技術を用いて，put_{total} より一意に get_{total} を出力し，スキーマの共存を実現する双方向変換である get_{total} と put_{total} を得る．

$putdelta$ が定義されない場合の例を示す．

例 4.3. 例 2.3 で定めた $putdelta$ を用いる．$T(x)$ にタプル (5) が挿入され $\{(1), (2), (5)\}$ となる場合，$x < 4$ を満たさないため (5) は $R_1(x)$ へは挿入されない．$x \geqq 4$ に対しては $putdelta$ が定義されない．

$putdelta$ が定義されない場合を，$putdelta(S, T^{'}) = \perp$ と表す．以下，補助リレーションから定める．

定義 4.4.（補助リレーション）補助リレーションは新しいスキーマのビュー $T(Y)$ と同じ属性をもち，$T_{out}^{B}(Y)$ とする．

定義 4.5.（$\boldsymbol{get_{total}}$ と $\boldsymbol{put_{total}}$）$get, put, putdelta$ のソースデータ，ターゲットデータをそれぞれ S と T_{in} とする．S と補助リレーションの物理データ T_{out} の組み $\hat{S} = (S, T_{out})$ を，get_{total} と put_{total} のソースデータとする．また新しいスキーマのデータ T を，ターゲットデータとする．それぞれの更新分を $\Delta\hat{S}$, ΔT，更新された結果を $\hat{S}'$, T' とする．get_{total} は，$T = get_{total}(\hat{S})$ を，$T_{in} = get(S)$ と T_{out} の union により T を得る関数と定める．

$$\begin{aligned} T &= get_{total}(\hat{S}) \\ &= T_{in} \cup T_{out} \end{aligned}$$

put_{total} は，$\hat{S'} = put_{total}(\hat{S}, T')$ を，更新されたターゲットデータ $T' = T \oplus \Delta T$ に対して $putdelta$ が定義されない場合は更新分 ΔT を T_{out} に反映し，それ以外は put により S を更新するものと定める．

$$\begin{aligned}\hat{S'} &= put_{total}((S, T_{out}), T \oplus \Delta T) \\ &= \begin{cases}(S, T_{out} \oplus \Delta T) & \text{if } putdelta(S, T \oplus \Delta T) = \bot, \\ (put(S, T_{in} \oplus \Delta T), T_{out}) & \text{otherwise.}\end{cases}\end{aligned}$$

get_{total}, put_{total} は GetPut, PutGet を満たし，双方向変換の整合性を保証する．

定理 4.6.（get_{total}, put_{total} に対する GetPut, PutGet） get_{total}, put_{total} は GetPut, PutGet を満たす．

$$\begin{aligned}&put_{total}(\hat{S}, get_{total}(\hat{S})) = \hat{S} \quad \text{(GetPut)} \\ &get_{total}(put_{total}(\hat{S}, T')) = T' \quad \text{(PutGet)}\end{aligned}$$

5 BIRDS による実装

5.1 実装の概要

双方向変換を用いたスキーマの共存を，putback-based の双方向化技術（BIRDS）[9,10] により実装する．BIRDS はビューの更新をベースリレーションへ反映する $putdelta$ の Datalog プログラムを入力とし，get の Datalog プログラムを出力する．それらが GetPut, PutGet を満たすかを検証し，満たす場合は get の Datalog プログラムからは SQL のビュー定義を，$putdelta$ の Datalog プログラムからは SQL のトリガを更に出力する．実装は次のステップから構成される．

ステップ 1. 元のスキーマ，新しいスキーマを定める．put により定めるスキーマの埋め込みを，$putdelta$ の Datalog プログラムにより記述する．これを入力に，BIRDS により get の Datalog プログラムを出力する．

ステップ 2. 補助リレーション T^B_{out} を用意し，その更新ルールを Datalog ルールとして導出する．これをステップ 1 の put のプログラムに加え，全域関数 $putdelta_{total}$ の Datalog プログラムを得る．

ステップ 3. $putdelta_{total}$ の Datalog プログラムを入力とし，BIRDS により get_{total} の Datalog プログラムを出力する．またそれぞれに対応した SQL のトリガ，ビュー定義を出力する．

5.2 $putdelta$ の記述

ステップ 1 の $putdelta$ の Datalog プログラムは，次の形式で記述する．

$$\begin{aligned}+R_i(X_i) &:- L^{p+}_{i.1}, ..., L^{p+}_{i.j}, ..., L^{p+}_{i.mp}. \\ -R_i(X_i) &:- L^{p-}_{i.1}, ..., L^{p-}_{i.k}, ..., L^{p-}_{i.np}.\end{aligned} \tag{5.1}$$

個々の subgoal $L^{p+}_{i.j}, L^{p-}_{i.k}$ は，元のスキーマのビューか新しいスキーマのビューか算術式であり，否定（¬）を伴うことができる．これを入力とし，BIRDS は次の形式で表される get を出力する．

$$T_{in}(Y) \quad :- L^g_1, ..., L^g_h, ..., L^g_{lg}. \tag{5.2}$$

$T_{in}(Y)$ は，get の値域内の更新を扱う新しいスキーマのビューのインスタンスを持つこと表し，個々の subgoal L^g_h は，元のスキーマのビューか算術式であり，否定（¬）を伴うことができる．

5.3 $putdelta_{total}$ とその導出

定義 4.5 の put_{total} に対し実装のために $putdelta_{total}$ を定め，その Datalog プログラムを導出する手順を説明する．

定義 5.1.（$putldeta_{total}$） $putdelta_{total}$ は，$\Delta\hat{S'} = putdelta_{total}(\hat{S}, T')$ を，更新されたターゲットデータ $T' = T \oplus \Delta T$ に対して $putdelta$ が定義されない場合は更新分 ΔT を T_{out} に反映し，それ以外は $putdelta$ により ΔS に変換するものと定める．$\Delta S^{\emptyset}$, $\Delta V^{\emptyset}$ は更新がなく，それぞれが空であることを示す．

$$\begin{aligned}\Delta\hat{S} &= putdelta_{total}((S, T_{out}), T \oplus \Delta T) \\ &= \begin{cases}(\Delta S^{\emptyset}, \Delta T) & \text{if } putdelta(S, T \oplus \Delta T) = \bot, \\ (putdelta(S, T_{in} \oplus \Delta T), \Delta V^{\emptyset}) & \text{otherwise.}\end{cases}\end{aligned}$$

$putdelta_{total}$ の Datalog プログラムを導出する手順を示す．$putdelta(S, T \oplus \Delta T) = \bot$ となる場合に T_{out} のデータを更新するルールを，次の Datalog のルールとして記述する．ここで $L^{p+}_{i.1}, ..., L^{p+}_{i.mp}$ は，ルール (5.1) の $+R_i(X_i)$ の subgoal であり $T(Y)$ を除いたものとする．

$$\begin{cases}+T^B_{out}(Y) :- +T(Y), \neg+R_1(X_1), ..., \neg+R_m(X_m), \\ \qquad\qquad \neg-R_1(X_1), ..., \neg-R_m(X_m). \\ -T^B_{out}(Y) :- -T(Y), \neg L^{p+}_{i.1}. \\ \qquad\qquad ... \\ -T^B_{out}(Y) :- -T(Y), \neg L^{p+}_{i.mp}.\end{cases} \tag{5.3}$$

補助リレーションへの挿入分（$+T^B_{out}(Y)$）のルールは，挿入 $+T(Y)$ に対して元のスキーマのデータ更新（$+R_i(X_i), -R_i(X_i)$）が発生しない場合に，$+T(Y)$ を $+T^B_{out}(Y)$ に反映させることを表している．削除分（$-T^B_{out}(Y)$）のルールは，元々に元のスキーマへの挿入とはならなかった挿入分の削除であることを表している．例 4.3 においてタプル (5) を削除する場合に該当し，削除分 $-T(Y)$ は元々に $+R_i(X_i)$ とはならなかったタプルに対する削除であることを，$+R_i(X_i)$ の $T(Y)$ 以外の subgoal の否定を $-T(Y)$ に連結することにより表わしている．

$putdelta_{total}$ の $putdelta(S, T \oplus \Delta T) = \bot$ 以外の場合のルールは，ステップ 1 のルール (5.1) である．ここでルール (5.3) の形式をルール (5.1) の形式に合わせるために，ルールを追加する．$+T(Y), -T(Y)$ は，更新されたタプルを含む $T(Y)$ と更新前タプルからなる $T_{tmp}(Y)$ との差集合として，次のルールによる．

$$\begin{array}{lll} +T(Y) & :- & T(Y), \neg T_{tmp}(Y). \\ -T(Y) & :- & \neg T(Y), T_{tmp}(Y). \end{array} \tag{5.4}$$

$T_{tmp}(Y)$ は get_{total} による変換の結果であるため，定義 4.5 により次のルールとする．

$$\begin{array}{lll} T_{tmp}(Y) & :- & T_{in}(Y). \\ T_{tmp}(Y) & :- & T^B_{out}(Y). \end{array} \tag{5.5}$$

$T_{in}(Y)$ はステップ 1 の get のルール (5.2) による変換の結果である．またルール (5.3) の body に現れる $+R_i(x_j), -R_i(x_j)$ は，ルール (5.1) による変換の結果である．以上により $putdelta_{total}$ を，ルール (5.1), (5.2), (5.3), (5.4), (5.5) として表す．

5.4 $putdelta_{total}, get_{total}$ の例（selection）

例 (2.3) の selection を表す $putdelta$ より，$putdelta_{total}$ と get_{total} を出力する例を示す．ステップ 2 の手法で 1 つの補助リレーションとその更新ルールを導出することにより，課題 (B) を解決していることを示す．

ステップ 1. 元のスキーマを $\mathrm{S_O}\langle R_1(x)\rangle$ とする．新しいスキーマを $\mathrm{S_N}\langle T(x)\rangle$ とする．$putdelta$ は，$T(x)$ に $x < 4$ となる更新があった際に，それを $R_1(x)$ に反映させるものとし，Datalog により次のように記述する．

$$\begin{array}{lll} +R_1(x) & :- & T(x), \neg R_1(x), x < 4. \\ -R_1(x) & :- & \neg T(x), R_1(x), x < 4. \end{array} \tag{5.6}$$

これを入力に，BIRDS により次の get の Datalog プログラムが出力される．

$$T_{in}(x) \quad :- \quad R_1(x), x < 4. \tag{5.7}$$

更新を同期させる場合として $T_{in}(x)$ のルールは，$x < 4$ による selection として $R_1(x)$ のインスタンスを $T_{in}(x)$ のインスタンスに変換することを表している．

ステップ 2. T^B_{out} の更新ルールを，(5.3) の形式により表す．

$$\begin{array}{lll} +T^B_{out}(x) & :- & +T(x), \neg + R_1(x), \neg - R_1(x). \\ -T^B_{out}(x) & :- & -T(x), R_1(x). \\ -T^B_{out}(x) & :- & -T(x), \neg(4 < X). \end{array} \tag{5.8}$$

put_{total} の Datalog プログラムを，ルール (5.6), (5.7), (5.8) および 属性 $Y = (x)$ とするルール (5.4), (5.5) とする．

ステップ 3. $putdelta_{total}$ の Datalog プログラムを入力に，BIRDS により次の get_{total} の Datalog プログラムが出力される．

$$\begin{array}{lll} T(x) & :- & R_1(x), x < 4. \\ T(x) & :- & T^B_{out}(x), \neg(x < 4). \\ T(x) & :- & T^B_{out}(x), R_1(x). \end{array} \tag{5.9}$$

$T(x)$ のルールは，$R_1(x)$ のインスタンスの $x < 4$ による selection の結果と，$T^B_{out}(x)$ のインスタンスの $\neg(x < 4)$ による selection の結果と，$T^B_{out}(x)$ と $R_1(x)$ のインスタンスで共通のものとの union として，$T(x)$ のインスタンスに変換する．

6 提案手法の適用

スキーマの共存において新しいスキーマはビューにより構成される．そのビューのデータはリレーショナル代数による元のデータの変換として得られるため，前章の selection に加えて本手法を各リレーショナル代数（union, projction, join）によるケースに適用し，有効性を確認した．結果を次に示す．

6.1 union

union の例を示す．元のスキーマを $\mathrm{S_O}\langle R_1(x), R_2(x)\rangle$，新しいスキーマを $\mathrm{S_N}\langle T(x)\rangle$ とする．ビュー更新に曖昧さのある例であり，R_1, R_2 から T への union としてビュー定義を定めた場合，ビュー更新の反映として T への挿入を R_1, R_2 のどちらか，あるいは両方に反映する場合でも整合性がある．スキーマの埋め込みとして，先に $putdelta$ により更新のルールを記述することで，ユーザーの意図を反映したビュー更新から一意にビュー定義を得，課題 (A) を解決することを示す．

$putdelta$ を，$T(x)$ への挿入は $R_1(x)$ に反映し，削除は $R_1(x), R_2(x)$ の両方から削除するものとし，Datalog により次のように記述する．

$$\begin{array}{lll} +R_1(x) & :- & T(x), \neg R_1(x), \neg R_2(x). \\ -R_1(x) & :- & \neg T(x), R_1(x). \\ -R_2(x) & :- & \neg T(x), R_2(x). \end{array} \tag{6.1}$$

これを入力に BIRDS により get を得る．

$$\begin{array}{lll} T_{in}(x) & :- & R_1(x). \\ T_{in}(x) & :- & R_2(x). \end{array} \tag{6.2}$$

T^B_{out} の更新ルールを，(5.3) の形式により表す．

$$\begin{array}{lll} +T^B_{out}(x) & :- & +T(x), \neg + R_1(x), \neg - R_1(x), \neg - R_2(x). \\ -T^B_{out}(x) & :- & -T(x), R_1(x). \\ -T^B_{out}(x) & :- & -T(x), R_2(x). \end{array} \tag{6.3}$$

put_{total} の Datalog プログラムを，ルール (6.1), (6.2), (6.3) および属性 $Y = (x)$ とするルール (5.4), (5.5) とする．これを入力に BIRDS により次の get_{total} の Datalog プログラムを得る．

$$\begin{array}{lll} T(x) & :- & R_1(x). \\ T(x) & :- & R_2(x). \end{array} \tag{6.4}$$

6.2 projection

projection の例を示す．get の値域を越える更新が存在しないため，補助リレーションは不要である．提案した手法により最終的に得る get_{total} では補助リレーションが不要となり，課題 (B) を解決していることを示す．元のスキーマを $\mathrm{S_O}\langle R_1(x, y, z)\rangle$ とする．新しいスキーマを $\mathrm{S_N}\langle T(x, y)\rangle$ とする．$putdelta$ を，$T(x, y)$ への挿入は R_1 へは $z =' null'$ として反映し，削除は該当するタプルの R_1 からの削除とする．その Datalog プログラムを次に表す．

$$\begin{array}{lll} +R_1(x,y,'null') & :- & T(x,y), \neg R_1(x,y,\_). \\ -R_1(x,y,z) & :- & \neg T(x,y), R_1(x,y,z). \end{array} \tag{6.5}$$

これを入力に BIRDS により *get* を得る．

$$T_{in}(x,y) \quad :- \quad R_1(x,y,\_). \tag{6.6}$$

T^B_{out} の更新ルールを，(5.3) の形式により表す．

$$\begin{array}{lll} +T^B_{out}(x,y) & :- & +T(x), \neg + R_1(x,y,\_), \neg - R_1(x,y,\_). \\ -T^B_{out}(x,y) & :- & -T(x), R_1(x,y,\_). \end{array} \tag{6.7}$$

put_{total} の Datalog プログラムを，ルール (6.5), (6.6), (6.7) および属性 $Y = (x)$ とするルール (5.4), (5.5) とする．これを入力に BIRDS により get_{total} の Datalog プログラムを得る．

$$T(x,y) \quad :- \quad R_1(x,y,\_). \tag{6.8}$$

R_1 のインスタンスの projection として T のインスタンスを算出することと，補助リレーションへの反映が起こらないことを表している．

6.3 join

外部キーによる inner join を扱う．元のスキーマを $\mathrm{S_O}\langle R_1(p,x,fk), R_2(fk,y)\rangle$ とする．新しいスキーマを $\mathrm{S_N}\langle T(p,x,y)\rangle$ とする．ここで p は主キー，fk は外部キーを表す．*putdelta* を，$T(p,x,y)$ への挿入はその y の値が同じとなるタプルが $R_2(fk,y)$ にある場合に $R_1(p,x,y)$ に挿入を反映し，削除も同様に y の値が同じとなるタプルが $R_2(fk,y)$ にある場合に $R_1(p,x,y)$ から削除するものとし，Datalog により次のように記述する．

$$\begin{array}{lll} +R_1(p,x,fk) & :- & T(p,x,y), \neg R_1(p,\_,\_), R_2(fk,y). \\ -R_1(p,x,fk) & :- & \neg T(p,x,y), R_1(p,x,fk), R_2(fk,y). \end{array} \tag{6.9}$$

これを入力に BIRDS により *get* を得る．

$$T_{in}(x) \quad :- \; R_1(p,x,fk), R_2(fk,y). \tag{6.10}$$

T^B_{out} の更新ルールを，(5.3) の形式により表す．

$$\begin{array}{lll} +T^B_{out}(p,x,y) & :- & +T(p,x,y), \neg + R_1(p,x,y), \neg - R_1(p,x,y). \\ -T^B_{out}(p,x,y) & :- & -T(p,x,y), R_1(p,\_,\_). \\ -T^B_{out}(p,x,y) & :- & -T(p,x,y), \neg R_2(\_,y). \end{array} \tag{6.11}$$

put_{total} の Datalog プログラムを，ルール (6.9), (6.10), (6.11) および属性 $Y = (p,x,y)$ とするルール (5.4), (5.5) とする．これを入力に BIRDS により get_{total} の Datalog プログラムを得る．

$$\begin{array}{lll} T(p,x,y) & :- & R_1(p,x,fk), R_2(fk,y). \\ T(p,x,y) & :- & T^B_{out}(p,x,fk), R_1(p,\_,\_). \\ T(p,x,y) & :- & T^B_{out}(p,x,fk), \neg R_2(\_,y). \end{array} \tag{6.12}$$

$T(p,x,y)$ のルールは，外部キー fk による R_1, R_2 の inner join と，T^B_{out} と R_1 で主キー p が同じもの，T^B_{out} にあり R_2 には含まれないもの，これらの union として T のインスタンスに変換することを表している．

7 関連研究

スキーマ進化については，非常に多くの研究がなされている [11]．スキーマ進化の基本的な操作を SMO として定めた研究に，SMO 毎に新しいスキーマに対して問い合わせを順方向に変換する PRISM [12] や，逆方向に元のスキーマに対して変換する PRIMA [13] が挙げられる．

ビューとデータベースのデータ間の変換は，ビュー更新問題として古くから取り組まれている [4, 5]．プログラミング言語の分野では双方向変換 [6] として研究されている．4.3 節で述べた *get* の値域を超える更新に関連するものとして，*get* および *put* により捨てられてしまう情報を complement に保持しておく symmetric lenses が提案されている [14]．

スキーマの共存については，クラウド型のアプリケーションにおいて様々な利用者向けのスキーマに対応する必要から multitenacy [15] や独自データベースの構築 [16] が報告されているが，カラムの追加などに留まっている．これに対し K. Herrmann らは，complement に相当する補助リレーションを利用し，SMO 毎にスキーマ間の双方向のデータ変換のルールを設計した MSVDBs を提案している [2, 3]．1 章に挙げた課題を有しており，本研究では，曖昧さに対して先に *put* を定めることによりユーザーの意図を反映可能にし，また高々 1 つの補助リレーションとその更新ルールを機械的に導出することを実現している．更に MSVDBs では明示的に設計されていない，外部キーによる inner join に対応できることを確認している．一方，追加されたカラムや分割された複数リレーションを新しいスキーマが持つ場合に，MSVDBs ではデータを新しいスキーマに移行せずとも primary key と補助リレーションとにより対応しているのに対し，本研究ではデータ移行を前提に projection，union，join として対応している．

8 おわりに

本研究では，双方向化技術を応用したスキーマの埋め込みによる手法を提案した．スキーマの共存においては，補助リレーションを含めたスキーマ間での整合性ある双方向のデータ変換ルールを設計することが難しく，提案した手法により先行研究における課題を解決している．今後の課題としては，追加されたカラムや複数のリレーションをもつ新しいスキーマとの共存における，補助リレーションの体系的な設計などが挙げられる．

文　献

[1] J.Humble, D. Farley, "Continuous Delivery: Reliable Software Releases Through Build, Test, and Deployment Automation." Addison Wesley Professional, 1st edition, 2010.

[2] K. Herrmann, H. Voigt, A.Behrend, J. Rausch, W. Lehner, "Living in parallel realities: Co-existing schema versions with a bidirectional database evolution language," SIGMOD Conference, pp. 1101-1116, 2017.

[3] K. Herrmann, H. Voigt, T. B. Pedersen, W. Lehner, "Multi-schema-version data management: data independence in the twenty-first century," The VLDB Journal 27, 4, pp. 547-

571, August 2018.
[4] F. Bancilhon, N. Spyratos, "Update semantics of relational views," ACM Trans. Database Syst., Vol. 6, No.4, pp. 557-575, 1981.
[5] Y. Masunaga, "A relational database view update translation mechanism," Proceedings of the 10th International Conference on Very Large Data Bases, pp. 309-320, 1984.
[6] J. N. Foster, M. B. Greenwald, J. T. Moore, B. C. Pierce and A. Schmitt, "Combinators for bi-directional tree transformations: a linguistic approach to the view update problem," POPL, Palsberg, J. and Abadi, M. (eds.), ACM, pp. 233-246, 2005
[7] S. Fischer, Z. Hu, H. Pacheco, "The essence of bidirectional programming, Science China Information Sciences," 58(5), pp.1-21, May 2015.
[8] H.S. Ko, T. Zan, Z. Hu, "BiGUL: A formally verified core language for putback-based bidirectional programming," In Partial Evaluation and Program Manipulation, PEPM'16, pp. 61-72, ACM, 216.
[9] V. D. Tran, H. Kato, Z. Hu, "Programmable View Update Strategies on Relations," PVLDB, 13(5), 2020.
[10] https://dangtv.github.io/BIRDS/
[11] L. Caruccio, G. Polese, and G. Tortora, "Synchronization of Queries and Views Upon Schema Evolutions: A Survey," ACM TODS 41(2), pp. 9:1-9:41, 2016.
[12] C. A. Curino, H. J. Moon, C. Zaniolo, "Graceful database schema evolution: The PRISM workbench," VLDB Endowment, 1(1), pp. 761-772, 2008.
[13] H. J. Moon, C. A. Curino, A. D. C.-Y. Hou, and C. Zaniolo, "Managing and querying transaction-time databases under schema evolution", PVLDB 1(1), pp. 882-895, 2008.
[14] M. Hofmann, B. Pierce, and D. Wagner, "Symmetric lenses," Principles of Programming Languages (POPL), 46(1), pp. 371-384, 2011.
[15] S. Aulbach, M. Seibold, D. Jacobs, and A. Kemper, "Extensibility and Data Sharing in evolving multi-tenant databases," In International Conference on Data Engineering (ICDE), pp. 91-110, IEEE, 2011.
[16] J. Chen et al., "Data Management at Huawei: Recent Accomplishments and Future Challenges," In IEEE International Conference on Data Engineering (ICDE), pp. 13-24, 2019.

付　　録

1　定理 2.2 の証明

証明.

$$\begin{aligned} S' &= \{\ \text{式 (2.1)}\ \} \\ &\quad S \oplus \Delta S \\ &= \{\ \Delta S \text{ の定義}\ \} \\ &\quad S \oplus putdelta(S, T') \end{aligned}$$

一方 $S' = put(S, T')$ であり $S' = S \oplus putdelta(S, T') = put(S, T')$. □

2　定理 4.6 の証明

証明. GetPut を示す.

i) $putdelta(S, T \oplus \Delta T) = \bot$ の場合

$$\begin{aligned} put_{total}(\hat{S}, get_{total}(\hat{S})) &= \{\ \hat{S}\text{の定義と } get_{total}\text{の定義}\ \} \\ &\quad put_{total}((S, T_{out}), T) \\ &= \{\ put_{total}\text{の定義}\ \} \\ &\quad (S, T_{out}) \\ &= \{\ \hat{S}\text{の定義}\ \} \\ &\quad \hat{S} \end{aligned}$$

ii) i) 以外の場合

$$\begin{aligned} put_{total}(\hat{S}, get_{total}(\hat{S})) &= \{\ \hat{S}\text{の定義と } get_{total}\text{の定義}\ \} \\ &\quad put_{total}((S, T_{out}), T) \\ &= \{\ put_{total}\text{の定義}\ \} \\ &\quad put((S, T_{in}), T_{out}) \\ &= \{\ put\ \text{の定義}\ \} \\ &\quad (S, T_{out}) \\ &= \{\ \hat{S}\text{の定義}\ \} \\ &\quad \hat{S} \end{aligned}$$

次に PutGet を示す.

i) $putdelta(S, T \oplus \Delta T) = \bot$ の場合は，ΔT は T_{in} を更新しないため，$T_{in} \setminus \Delta_T^- = T_{in}$ である.

$$\begin{aligned} get_{total}(put_{total}(\hat{S}, T')) &= \{\ \hat{S}\text{の定義と } T'\text{に} \oplus \text{を適用}\ \} \\ &\quad get_{total}(put_{total}((S, T_{out}), T \oplus \Delta T)) \\ &= \{\ put_{total}\text{の定義}\ \} \\ &\quad get_{total}(S, T_{out} \oplus \Delta T) \\ &= \{\ get_{total}\text{の定義}\ \} \\ &\quad get(S) \cup (T_{out} \oplus \Delta T) \\ &= \{\ get(S) = T_{in}, T_{in} = T_{in} \setminus \Delta_T^-\ \} \\ &\quad (T_{in} \setminus \Delta_T^-) \cup (T_{out} \setminus \Delta_T^-) \cup \Delta_T^+ \\ &= \{\ \text{分配法則}\ \} \\ &\quad ((T_{in} \cup T_{out}) \setminus \Delta_T^-)) \cup \Delta_T^+ \\ &= \{\ get_{total}\text{の定義より } T_{in} \cup T_{out} = T\ \} \\ &\quad (T \setminus \Delta_T^-) \cup \Delta_T^+ \\ &= \{\ (T \setminus \Delta_T^-) \cup \Delta_T^+ = T \oplus \Delta T = T'\ \} \\ &\quad T' \end{aligned}$$

ii) i) 以外の場合は，ΔT は T_{out} を更新しないため，$T_{out} \setminus \Delta_T^- = T_{out}$ である.

$$\begin{aligned} get_{total}(put_{total}(\hat{S}, T')) &= \{\ \hat{S}\text{の定義と } T'\text{に} \oplus \text{を適用}\ \} \\ &\quad get_{total}(put_{total}((S, T_{out}), T \oplus \Delta T)) \\ &= \{\ put_{total}\text{の定義}\ \} \\ &\quad get_{total}(put(S, T_{in} \oplus \Delta T), T_{out}) \\ &= \{\ get_{total}\text{の定義}\ \} \\ &\quad get(put(S, T_{in} \oplus \Delta T)) \cup T_{out} \\ &= \{\ get, put\ \text{に対する PutGet}\ \} \\ &\quad (T_{in} \oplus \Delta T) \cup T_{out} \\ &= \{\ \oplus\text{の展開と } T_{out} = T_{out} \setminus \Delta_T^-\ \} \\ &\quad (T_{in} \setminus \Delta_T^-) \cup \Delta_T^+ \cup (T_{out} \setminus \Delta_T^-) \\ &= \{\ \text{分配法則}\ \} \\ &\quad ((T_{in} \cup T_{out}) \setminus \Delta_T^-) \cup \Delta_T^+ \\ &= \{\ get_{total}\text{の定義より } T_{in} \cup T_{out} = T\ \} \\ &\quad (T \setminus \Delta_T^-) \cup \Delta_T^+ \\ &= \{\ (T \setminus \Delta_T^-) \cup \Delta_T^+ = T \oplus \Delta T = T'\ \} \\ &\quad T' \end{aligned}$$

□