情報処理学会論文誌　プログラミング　Vol.14　No.5　15–33 (Nov. 2021)

プログラム可能なスキーマの共存戦略の実現手法

田中 順平[1,2,a)]　バン ダン トラン[1,2,b)]　加藤 弘之[2,c)]　胡 振江[2,3,d)]

受付日 2021年4月9日, 採録日 2021年7月27日

概要：変化し続ける現実世界に対し，情報システムには柔軟かつ継続的に発展していくことが求められる．アプリケーションでは継続的な改修，複数バージョンの管理，運用などの技術の進展が著しい．一方データベースでは，スキーマを継続的に発展させ，複数のアプリケーションとともに，複数のスキーマバージョンをデータを共有しつつ同時並行的に運用するスキーマ共存の技術は，難度が高い．先行研究ではビューによるスキーマの共存手法が提案されているものの，更新データをスキーマ間で共有し合うルールや条件があらかじめ定められ，スキーマの共存戦略が限定されている．課題として共存戦略の記述力が不足している．これに対し本論文では，より柔軟にスキーマの共存戦略を記述するドメイン特化言語と，記述された戦略を 2 つの双方向変換の組により実装する手法とを提案する．提案した言語と実装手法を双方向変換エンジン BIRDS を利用して実装し，評価した．柔軟に多様なスキーマの共存戦略を記述でき，それを実装できることから，提案手法の有用性が確認された．

キーワード：スキーマの共存，双方向変換，データの共有

Programmable Strategy for Co-existence of Schemas

Jumpei Tanaka[1,2,a)]　Van-Dang Tran[1,2,b)]　Hiroyuki Kato[2,c)]　Zhenjiang Hu[2,3,d)]

Received: April 9, 2021, Accepted: July 27, 2021

Abstract: Information systems continuously evolve to meet the ever-changing real world, which requires iterative development and technology to maintain and operate multiple versions. It is, however, a challenging task in databases to make one schema version evolve to a new version and keep multiple schema versions co-existing by sharing data for multiple running applications. The existing work gives a view-based approach to the co-existence of schemas in a database, but the strategy to share data across schemas is limited to the predefined one, which lacks flexibility. In this paper, we propose a domain-specific language that can be used to specify various kinds of strategies for the co-existence of schemas and present a systematic method for implementing the strategies in our language by decomposing them into a pair of bidirectional transformations. The language and the system have been implemented over BIRDS, a bidirectional transformation engine, and the experimental results show its flexibility and usefulness in practice.

Keywords: co-existence of schemas, bidirectional transformation, data sharing

1. はじめに

スキーマの共存は，データベースに求められる重要な特徴の 1 つである．多くの情報システムはアプリケーションと関係データベースから構成され [6]，多様に変化し続ける現実世界に対し継続的に進化していくことが求められている．そのためアプリケーションでは，様々なバージョンへの進化と複数バージョンの共存による生産性向上の技術が大きく進展している [13], [18]．関係データベースにおいては，データ構造を定めるスキーマを進化させ，複数のス

1 総合研究大学院大学
The Graduate University for Advanced Studies, SOKENDAI, Hayama, Kanagawa 240–0193, Japan
2 国立情報学研究所
National Institute of Informatics, Chiyoda, Tokyo 101–8430, Japan
3 北京大学
Peking University, Beijing, 100871, P.R. China
a) jumpeitanaka@nii.ac.jp
b) dangtv@nii.ac.jp
c) kato@nii.ac.jp
d) zhenjianghu@pku.edu.cn

キーマバージョンとそれぞれのデータとを共存させる重要性が認識されている [2], [22], [24].

データベースのスキーマの共存には，進化前と進化後のスキーマバージョンがそれぞれ 1 つのデータベースのように任意の更新を取り扱うことができ，互いに更新データを共有できることが求められる．本論文では，スキーマの共存とは以下の要件を実現するものと定める．

- スキーマ進化により，元となる進化前のスキーマバージョンから新しい進化後のスキーマバージョンを定めることができる．
- 進化前，進化後のスキーマバージョンで任意のデータ更新が可能であり，その結果を参照できる．
- 進化前，進化後のスキーマバージョンにおける更新データを，互いに両方向に共有できる．

ここで更新データを互いに共有するとは，更新データを共有する場合，しない場合を含んでいる．スキーマの共存の戦略は，スキーマ進化によりどのように進化前のスキーマバージョンから進化後のスキーマバージョンへとデータを変換し共有するのか，あるいは結果として共有しないのか，また逆方向に進化後から進化前のスキーマバージョンへと更新データをどのように共有するのか，あるいはしないのかを定めるものとなる．

一方，スキーマの共存の研究自体は，世界的に見てもまだ緒についたばかりとなっている．既存の関係データベース管理システムではスキーマの共存をサポートする機能は用意されておらず，実務的には，進化前後のスキーマごとに物理データを保持するデータベースを用意し，データベース間で必要なデータを一方向，あるいは両方向に共有させている [1]．しかし重複するデータをデータベース間で同期させる手法が必要となり，両方向での更新の共有は課題となっている．これに対し Herrmann らは，1 つのデータベース内でスキーマの共存を実現する MSVDB（Multi-Schema-Version Database）を提案している [16], [17]．共有されるデータベースの物理データからビューを定義し，各スキーマの表をビューとする，つまりビュー定義の集まりを 1 つ 1 つのスキーマバージョンとすることで[*1]，スキーマバージョン間のデータの共有を実現している（**図 1**）．

MSVDB は，SMO（Schema Modification Operator，スキーマ変更演算子）により，スキーマの進化と共存の容易な設計可能にしている．SMO は関係完備な記述力を有し，演算子ごとに，スキーマ進化と逆方向の更新伝播のルールからなるスキーマの共存戦略があらかじめ用意されている．MSVDB のユーザが SMO の組合せにより進化後のスキーマを設計すると，進化前から進化後のスキーマバージョンへのスキーマ進化の関係を維持した更新データの共有と，逆方向の更新伝播に従う逆方向の更新データの共有とが実現される．物理データとビューの間の変換は，SMO ごとのスキーマの共存戦略に応じてあらかじめ用意されており，また共有される物理データだけではビューの更新結果を保てない場合に対して，追加的な物理データのための補助リレーションが設計されている．

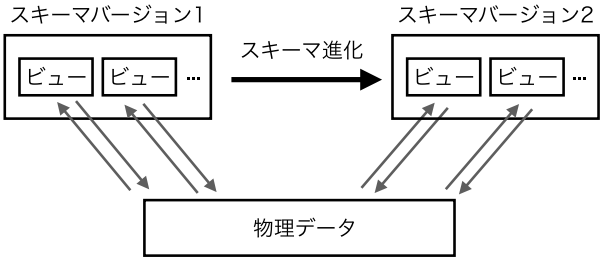

図 1　ビューを利用したスキーマの共存
Fig. 1　Co-existence of schemas based on views.

しかしながら，実用的な要件に対して次の課題が存在している．

- スキーマの共存戦略があらかじめ SMO ごとに定義されており，スキーマバージョン間の関連が 1 つに限定されてしまう．これにより，スキーマバージョン間での共存戦略を柔軟に設定できない．たとえば，スキーマの共存を実現したいユーザは，進化後のスキーマバージョン上で更新された新しいデータをすべて共有したいかもしれないし，極端な別の例では共有せず独立に扱いたいかもしれない．このようなユーザの意図を反映したスキーマの共存戦略を柔軟に記述することができない．
- スキーマの共存戦略の実現には補助リレーションを必要としているものの，体系的な設計手法が存在しておらず，MSVDB の設計者により毎度設計されている．ユーザが共存戦略を変更したり任意の共存戦略を導入したりしたい際に，ユーザを煩わせることなく，補助リレーションを自動的に導出することが難しい．さらには補助リレーションが必要とされる条件を定め，共有される物理データ以外の追加的な物理データを一定のものに限定することが難しい．

本論文の目的は，どのようにスキーマを進化させ共存させるかの共存戦略をプログラム可能にし，実現する手法を確立することで，スキーマの共存を実用化に近づけることである．上述の課題に対し，スキーマの共存戦略をプログラム可能にするためのドメイン特化言語と，記述された戦略を双方向変換 [15] により実現する手法とを提案する．進化前のスキーマの複数の表と，進化後のスキーマの 1 つの表との間のスキーマの共存戦略を対象に，これをプログラム可能にする．進化後の別の表に対しては，また別の 1 つのスキーマの共存戦略をプログラムすることにより対応する．

本論文の貢献と，先行研究における課題の解決手法を以

[*1] データベース管理システムの 3 層スキーマ構造における外部スキーマを，1 つ 1 つのスキーマバージョンとしている

下にまとめる．

- 1つ目の課題に対して，Datalogを基に，柔軟にスキーマの共存戦略を記述できるドメイン特化言語を提案する．Datalogは関係完備な記述力を持つため，共存戦略におけるスキーマ進化の記述に対して先行研究のSMOと同等な記述力を保証する．さらに安全な設計を可能とするために，記述した戦略がスキーマの共存として一貫性を保つための条件を明らかし，その検証手法を提案する．これらにより先行研究のSMOで提供されている共存戦略だけでなく，それ以外の戦略も設計，記述可能にし，より柔軟にデータ共有の要件に対応できるようにする．
- 2つ目の課題に対して，記述したスキーマの共存戦略から補助リレーションを自動的に導出し，共存戦略を物理データとビューの間の双方向変換として自動的に実現する手法を提案する．補助リレーションを必要とする場合を体系化し，限られた数のものでユーザ任意の共存戦略を実現できるようにする．

さらに提案した手法を双方向変換エンジンBIRDS [23]を利用して実装し，その有効性を検証した．先行研究のSMOで提供されているスキーマの共存戦略とそれ以外の共存戦略とを，極端に大きなプログラムとはならない数行から20行程度の記述量で設計でき，共存戦略を実現する双方向変換，補助リレーション，実装のためのSQLプログラムが自動的に導出されることを確認した．また補助リレーションについては，進化後のスキーマバージョンの1つの表に対して，たかだか2つに限定してユーザ任意のスキーマの共存戦略を実現できることを確認した．

従来のデータを二重に保持する手法では，更新データを一方向へのみ共有する場合であれば，重複データの管理が単純なためにスキーマの共存の実現が容易であった．またMSVDBでは，あらかじめ定められた共存戦略に限ればスキーマの共存の実現が容易であった．これに対して本論文が提案する手法では，既存手法で対応できる共存戦略だけでなく，ユーザ任意の共存戦略を記述し，それを自動的に実現することを可能にしている．

以降，2章で提案する手法の概要を説明し，3章でスキーマの共存戦略を記述する言語の詳細を説明する．4章でスキーマの共存戦略の一貫性の検証を扱い，5章で共存戦略を実現する手法について説明する．6章で提案手法の評価を行い，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. 提案手法の概要

本章では，単純化した注文管理システムを例に，本論文が提案するスキーマの共存戦略の記述手法，記述された戦略の一貫性の検証，そして戦略の実現手法について，概要を説明する．

2.1 スキーマの共存戦略の例

単純化した注文管理システムを例に，スキーマの共存を考える．注文管理システムの変更が行われ，ユーザを段階的に移行していくために，変更前と変更後のものが同時に利用されるものとする．

変更前の注文管理システムのデータベースのスキーマバージョンver1は，表ORD1とORD2から構成される（**図 2**）．表ORD2は，表ORD1の属性MEMOの利用目的が曖昧であったため，これを廃止するために暫定的に作られ，新しい注文は表ORD2で管理をしている．

次に注文管理システムとしてITEM Aの扱いを廃止し，新商品のITEM Cを扱う変更が行われるものとする．データベースでは，スキーマバージョンver2へのスキーマ進化を行う（図2(a)）．ITEMの変更へ対応し，かつ2つに別れていた表ORD1とORD2を統合するものとする．スキーマ進化により，属性MEMOを削除した表ORD1のタプルの集合と表ORD2のタプルの集合から，それぞれITEMの値がAではないタプルを抽出し，それらを統合したタプルの集合による表ORD3を得る．

一方，変更前と後の注文管理システムを同時に利用するためには，スキーマバージョンver1とver2を共存させる必要がある．それぞれのスキーマバージョンで更新が発生するため，それを互いに共有する必要がある．スキーマバージョンver1からver2へはスキーマ進化の関係を維持し，表ORD1やORD2の更新された結果のタプルの集合から表ORD3のタプルの集合を得ることで更新を共有する（図2(b)）．ver2からver1への逆方向へは，表ORD3への挿入に対して，ITEMの値がAでもCでもなければ表

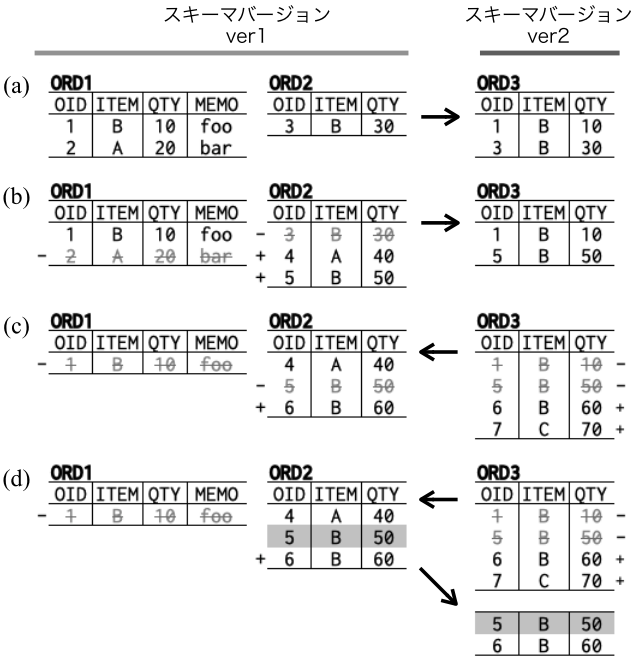

図 2　スキーマの共存戦略の例
Fig. 2　An example of co-existence of schemas.

ORD2 と共有し，表 ORD1 とは挿入を共有しないものとする（図 2 (c)）．削除については，ITEM の値が A でなければ表 ORD1 と共有するものとする．

2.2　スキーマの共存戦略の記述

スキーマの共存戦略を，スキーマバージョンの定義，スキーマ進化のルール，逆方向の更新伝播のルールにより定め，Datalog を用いて記述する．

2.2.1　スキーマバージョンの定義

進化前のスキーマバージョン ver1 をソーススキーマ，進化後の ver2 をターゲットスキーマとし，それぞれのスキーマバージョンの表を次のように定義する．Datalog の記述の慣習に従い，表の名前を小文字で定義している．

$$\text{source}: \quad \text{ver1}\#\texttt{ord1}(\texttt{OID}, \texttt{ITEM}, \texttt{QTY}, \texttt{MEMO}). \tag{1}$$

$$\text{source}: \quad \text{ver1}\#\texttt{ord2}(\texttt{OID}, \texttt{ITEM}, \texttt{QTY}). \tag{2}$$

$$\text{target}: \quad \text{ver2}\#\texttt{ord3}(\texttt{OID}, \texttt{ITEM}, \texttt{QTY}). \tag{3}$$

2.2.2　スキーマ進化

スキーマ進化は，ソーススキーマからターゲットスキーマへのデータの変換を定めるルールである．ソーススキーマの表 ORD1 と ORD2 からターゲットスキーマの表 ORD3 への変換を，Datalog により次のように記述する．

$$\texttt{ord3}(O, I, Q) :- \texttt{ord1}(O, I, Q, M), I \neq \text{`A'}. \tag{4}$$

$$\texttt{ord3}(O, I, Q) :- \texttt{ord2}(O, I, Q), I \neq \text{`A'}. \tag{5}$$

表名 ord1，ord2，ord3 による各述語は，対応する表のタプルの集合を表している．各表名の属性の順番に応じて変数を割り当てており，変数 (O, I, Q, M) であれば属性 (OID, ITEM, QTY, MEMO) に対応している．ルール (4) は，表 ORD1 のタプルの集合から projection により属性 MEMO を削除し，ITEM の値が A 以外のタプルを selection により抽出する．ルール (5) は，表 ORD2 のタプルの集合から ITEM の値が A 以外のタプルを selection として抽出する．2 つのルールの結果の union により，表 ORD3 のタプルの集合を定めている．

2.2.3　逆方向の更新伝播

逆方向の更新伝播は，ターゲットスキーマでの更新をどのようにソーススキーマへ伝播し共有するかを定めるルールである．本論文では更新を挿入，削除，修正の 3 つによるものとし，修正は削除と挿入からなる 1 つのシーケンスとする [19]．そのため逆方向の更新伝播を，ターゲットスキーマでの挿入/削除をソーススキーマへの挿入/削除に変換するルールとして定める．更新データの逆方向へ伝播，共有を，表 ORD3 の挿入/削除から表 ORD1 と表 ORD2 の挿入/削除への変換とし，これを Datalog により次のように記述する．

$$+\texttt{ord2}(O, I, Q) :- +\texttt{ord3}(O, I, Q), \neg\texttt{ord1}(O, I, Q, \_), \neg\texttt{ord2}(O, I, Q), I \neq \text{`A'}, I \neq \text{`C'}. \tag{6}$$

$$-\texttt{ord1}(O, I, Q, M) :- -\texttt{ord3}(O, I, Q), \texttt{ord1}(O, I, Q, M), I \neq \text{`A'}. \tag{7}$$

$$-\texttt{ord2}(O, I, Q) :- -\texttt{ord3}(O, I, Q), \texttt{ord2}(O, I, Q), I \neq \text{`A'}. \tag{8}$$

+/− 記号のついた述語は対応する表へ挿入/削除するタプルの集合を表している．+ord2 であれば表 ORD2 に挿入するタプルの集合となる．ルール (6) は，表 ORD3 に挿入されたタプルの集合で表 ORD1 にも表 ORD2 にも存在せず，ITEM の値が A でも C でもないものを，表 ORD2 へ挿入するタプルの集合へと変換している．ルール (7)，(8) は，表 ORD3 から削除されたタプルの集合で，それぞれ表 ORD1，ORD2 に存在し ITEM の値が A 以外であれば，それぞれから削除するタプルの集合へと変換している．表 ORD1 へは挿入を共有しないため，そのルールを記述していない．先行研究 [16], [17] の union によるスキーマ進化の SMO では表 ORD1 とも挿入を共有する共存戦略を定めているが，この例では挿入を共有せず，1 つの別の戦略として記述している．

スキーマの共存戦略の記述については，3 章でさらに詳細を説明する．

2.3　一貫性の検証

本論文では，記述されたスキーマの共存戦略が一貫性を保つことの検証手法を提案する．一貫性を，ターゲットスキーマ上の更新を逆方向の更新伝播によりソーススキーマと共有し，その結果をスキーマ進化のルールにより変換した場合，ターゲットスキーマに対して余計なタプルを生成しないこと，と定める．次の例 1 は，一貫性を満たさない場合のスキーマの共存戦略を示している．

例 1．2.1 節の注文管理システムを例に，ターゲットスキーマの表 ORD3 での削除を，ソーススキーマの表 ORD2 とは共有しない場合の共存戦略を考える．逆方向の更新伝播はルールは (6) と (7) となり，その結果に対してスキーマ進化のルール (4) と (5) を適用した場合の結果を図 2 (d) に示す．表 ORD3 から削除したタプル (5, B, 50) が表 ORD2 とは共有されず削除されないため，スキーマ進化の結果，(5, B, 50) が余計なタプルとして生成される．そのため一貫性を満たさず，ターゲットスキーマは削除した結果を維持できなくなっている．□

提案する一貫性の検証では，ターゲットスキーマ上の更新に対して，逆方向の更新伝播に次いでスキーマ進化を適用した結果が，ターゲットスキーマに対して余計なタプルを生成しないことは保証するが，更新の結果と一致するために必要なすべてのタプルが存在していることまでは保証しない．この一貫性の定義は，双方向変換 [15] が定める整

合性を保つための条件を緩和したものとなっている．双方向変換は，ソーススキーマとターゲットスキーマの間であれば，ターゲットスキーマの更新データのすべてをソーススキーマと同期させ整合性を保つ枠組みである．整合性を保つ条件として，逆方向の更新伝播に次いでスキーマ進化を適用した結果に，更新の結果と一致するために必要なタプルがすべて存在すること，かつ余計なタプルを生成していないことが求められる．一方スキーマの共存では，ターゲットスキーマ上の更新に対して，ソーススキーマと共有する場合，しない場合が存在する．すべての更新を同期させ共有させる双方向変換とは，この点が異なっている．共有されないターゲットスキーマ上の更新が存在するため，逆方向の更新伝播に次いでスキーマ進化を適用した結果に，更新の結果と一致するすべてのタプルが存在しているとは限らず，そのため双方向変換が求める整合性のための条件を緩和している．

なお提案する一貫性の検証では，ソーススキーマ上の更新は検証の対象としていない．スキーマ進化のルールは，ソーススキーマ上の更新の結果からターゲットスキーマ上での挿入/削除のタプルの集合へとは変換せず，逆方向の更新伝播のルールを適用しても，再びソーススキーマを更新するような挿入/削除のタプルの集合へとは変換されないためである．

一貫性の具体的な検証手法については，4 章で詳細を説明する．

2.4　スキーマの共存戦略の実現

本論文では，スキーマの共存戦略を，ビューと物理データの間の 2 つの双方向変換により実現する．双方向変換の詳細は 5 章で説明する．

注文管理システムを例に，その実現手法の概略を**図 3** に示す．図 3 (a) と (b) は，スキーマの共存戦略の例の図 2 (b) と (c) に対応している．点線の上段がビュー，下段が物理データを保持するための実リレーションと補助リレーションを示している．構成として，ソーススキーマ ver1 の表 ORD1 と ORD2 をビューとし，これらに対し同じ属性を持ち物理データを保持するための実リレーション B_ORD1 と B_ORD2 を用意する．ターゲットスキーマ ver2 の表 ORD3 もビューとし，補足的な物理データを保持するための補助リレーションを用意する．

図 3 (a) は，ソーススキーマ ver1 上での更新の様子を表している．ソーススキーマに対しては，ビュー ORD1 と ORD2 と実リレーション B_ORD1 と B_ORD2 との間の恒等写像となる双方向変換を与えることで，参照，更新とを可能にする．ターゲットスキーマへは，スキーマ進化のルールを実現するように，実リレーション B_ORD1 と B_ORD2 からビュー ORD3 への変換を与える．

図 3 (b) は，ターゲットスキーマ ver2 上での更新の様子

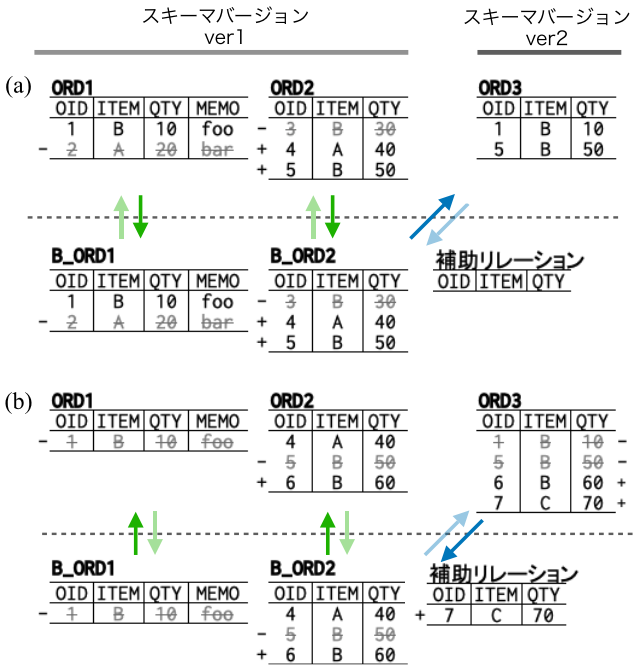

図 3　スキーマの共存戦略の実現

Fig. 3　Realization of a strategy for co-existence of schemas.

を表している．挿入/削除されたビュー ORD3 のタプルの集合から実リレーション B_ORD1 と B_ORD2 への挿入/削除のタプルの集合への変換を，逆方向の更新伝播を実現するような変換として与える．これにより，実リレーションを介した更新の共有を実現する．ただしタプル (7, C, 70) の挿入のように，更新がソーススキーマと共有されず実リレーションだけではターゲットスキーマ上の更新を維持できない場合に対しては，補助リレーションへ挿入を反映し，ターゲットスキーマの更新結果をビュー ORD3 で参照可能となるようにする．ターゲットスキーマに対してこのようなビューと物理データ間の双方向変換を与えることで，スキーマの共存戦略を実現する．

双方向変換と補助リレーションの導出手法の詳細については，5 章で説明する．

3.　スキーマの共存戦略を記述する言語

本章では，始めにスキーマの共存について定義し，Datalog [8] を基本としたドメイン特化言語の文法を与え，スキーマの共存戦略の記述手法について説明する．

3.1　スキーマの共存の定義

スキーマの共存を次に定める．

定義 1.（スキーマの共存） 次を満たす場合，これをスキーマの共存と定義する．

(1) スキーマ進化として，元となるスキーマバージョン（ソーススキーマ）から新しいスキーマバージョン（ターゲットスキーマ）を定めることができる．

(2) ソーススキーマ，ターゲットスキーマでの任意のデー

タ更新が可能であり，その結果を参照できる．
(3) ソーススキーマ，ターゲットスキーマにおける更新データを，両方向に共有できる．　□

本論文では，ソーススキーマとターゲットスキーマの間ではスキーマ進化の関係が維持され，ソーススキーマからターゲットスキーマへの更新データの共有は，スキーマ進化のルールに従うものとする．そのためスキーマの共存戦略を，スキーマの定義，スキーマ進化のルール，ターゲットスキーマからソーススキーマへの逆方向の更新伝播のルールから記述する．

3.2　記述言語の文法

図 4 に，スキーマの共存戦略プログラムを記述する言語の文法を示す．

<schema> によりスキーマの定義を記述する．ソーススキーマ source とターゲットスキーマ target に対して，バージョン名 <version>，それぞれのスキーマバージョンを構成する表名 <relname>，その属性 <varname> を記述する．主キー制約は，pk の表記に対して表名 <relname> と主キーとなる属性 <varname> により記述する．スキーマの進化と逆方向の更新伝播は，Datalog のルールとして <rule> により記述する．挿入/削除のタプルの集合に対応する述語 <predeicate> は，表名 <relname> に +/− の記号を付けたものとする．主キー制約以外の制約については，<constraint> により，ヘッドを真偽値の偽を表す ⊥() とする Datalog のルールとして記述する．記述の仕方の詳細を，次の 3.3 節と 3.4 節で説明する．

3.3　スキーマ進化の記述

スキーマ進化を記述する手法を説明する．また記述されたスキーマ進化と逆方向の更新伝播のルールとが一貫性を満たすことを検証できるようにするために，その記述の仕方の制約について説明する．

<program> ::= <statement>*
<statement> ::= <schema> | <rule> | <constraint>
<schema> ::=
  source: <version> # <relname> (<varname> {, <varname>}*).
| target: <version> # <relname> (<varname> {, <varname>}*).
| pk(<relname>, [<varname> {, <varname>}*]).
<constraint> ::= ⊥() :− <literal> {, <literal>}*.
<rule> ::= <predeicate> :− <literal> {, <literal>}*.
<literal> ::=
  <predeicate> |not <predeicate> | <builtin> |not <builtin>
<predeicate> ::= [+|−] <relname> (<term> {, <term>}*)
<builtin> ::= <varname> (= | <> | < | > | <= | >=) <const>
<term> ::= <varname> | <annonvar> | <const>

図 4　スキーマの共存戦略プログラムを記述する言語の文法
Fig. 4　Syntax of a language to describe a program of strategy for co-existence of schemas.

ソーススキーマを表名 s_i（$i \in [1,n]$）の集合，ターゲットスキーマを表名 t_j（$j \in [1,m]$）の集合とする．ソーススキーマの各表のタプルの集合を S_i（$i \in [1,n]$），その全体を S，ターゲットスキーマの各表のタプルの集合を T_j（$j \in [1,m]$），その全体を T とするとき，スキーマ進化は S から T への変換である．その変換を定める Datalog プログラムを F とし，次のように表す．

$$T = F(S) \tag{9}$$

F をソーススキーマの複数の表のタプルの集合から，ターゲットスキーマの 1 つの表のタプルの集合へ変換する Datalog のルールにより記述する．個々のルールを次の形式により記述する．

$$H_j \text{ :− } L_{j.1}, \ldots, L_{j.k}, \ldots, L_{j.kn}. \tag{10}$$

$\vec{x_i}$ をソーススキーマの表名 s_i の属性の集まり，$\vec{y_j}$ をターゲットスキーマの表名 t_j の属性の集まり，ソーススキーマの各表のタプルの集合に対応する述語を $s_i(\vec{x_i})$，ターゲットスキーマの各表のタプルの集合に対応する述語を $t_j(\vec{y_j})$ とする．各 $L_{j.k}$ はリテラルであり，述語 $s_i(\vec{x_i})$ か，他のルールのヘッドに現れる述語か，$=$，$<$ などの算術比較による組み込み述語とする．ヘッドの H_j は述語 $t_j(\vec{y_j})$ か，$s_i(\vec{x_i})$ 以外の述語とする．

一貫性の検証をできるようにするための制約として，ルール (10) は非再帰的な GN-Datalog [4], [23] により記述する．非再帰的とするために，リテラル $L_{j.k}$ にはそのルールのヘッドに現れる述語が含まれないように記述する．GN-Datalog とするために，ヘッドに現れる述語とボディに現れる否定をともなう述語は，その変数のすべてが，ボディに現れ否定をともなわない述語の表名の変数か，$=$ による組み込み述語の変数に含まれているように記述する．

例 2. 次の Datalog ルールは GN-Datalog とはならない．

$$t1(X_1, X_2) \text{ :− } s_1(X_1, X_2), \neg s_2(X_1, X_2, X_3). \tag{11}$$

ボティに現れ否定をともなう述語 $\neg s_2(X_1, X_2, X_3)$ に対して，その変数 X_3 がボディの否定をともなわない述語に現れないためである．　□

GN-Datalog による Datalog プログラムでは，ヘッドの述語に対応するタプルの集合が空とはならないような，ボディに現れる述語に対応するタプルの集合が存在すること（充足可能性）が決定可能であることが知られている [4], [23]．一貫性の検証において，この性質を利用する．

3.4　逆方向の更新伝播の記述

本節では逆方向の更新伝播を記述する手法を説明する．

またその記述の仕方の制約について説明する．

逆方向の更新伝播を，ターゲットスキーマでの挿入/削除をソーススキーマへの挿入/削除に変換するルールとして定める．ここでスキーマ進化は単射とは限らず，ターゲットスキーマはソーススキーマの情報を失っているかもしれない．そのためソーススキーマのタプルの集合 S も参照し，ターゲットスキーマでの挿入/削除をソーススキーマへの挿入/削除に変換する．ターゲットスキーマへ挿入/削除するタプルの集合を Δ_T^+/Δ_T^-，それらの全体を ΔT，ソーススキーマへ挿入/削除するタプルの集合を Δ_S^+/Δ_S^-，それらの全体を ΔS とするとき，逆方向の更新伝播の変換を定める Datalog プログラム F' を，次のように表す．

$$\Delta S = F'(S, \Delta T) \tag{12}$$

F' はターゲットスキーマの 1 つの表 T_j への挿入/削除のタプルの集合から，ソーススキーマの複数の表 $S_i\,(i \in [1,n])$ への挿入/削除のタプルの集合へ変換する Datalog のルールとして記述する．個々のルールを次の形式により記述する．

$$H_i^+ \text{ :- } L_{i.1}^+, \ldots, L_{i.k}^+, \ldots, L_{i.kn+}^+. \tag{13}$$

$$H_i^- \text{ :- } L_{i.1}^-, \ldots, L_{i.k}^-, \ldots, L_{i.kn-}^-. \tag{14}$$

ここで $\Delta_{T_j}^+$，$\Delta_{T_j}^-$，$\Delta_{S_i}^+$，$\Delta_{S_i}^-$ に対応する述語を，$+t_j(\vec{y_j})$，$-t_j(\vec{y_j})$，$+s_i(\vec{x_i})$，$-s_i(\vec{x_i})$ とする．$L_{i.k}^+$ と $L_{i.k}^-$ はリテラルであり，$s_i(\vec{x_i})$，$+t_j(\vec{y_j})$，$-t_j(\vec{y_j})$ による述語か，他のルールのヘッドに現れる述語か，$=$，$<$ などの算術比較による組み込み述語を含む．ヘッドの H_i^+ は $+s_i(\vec{x_i})$ か $-s_i(\vec{x_i})$ 以外の述語，H_i^- は $-s_i(\vec{x_i})$ か $+s_i(\vec{x_i})$ 以外の述語とする．

逆方向の更新伝播のルールは，単調と一次であることを制約として加えた GN-Datalog により記述する．本論文では，挿入から挿入，削除から削除への変換を単調であるとする．逆方向の更新伝播は単調である必要があり，ルール (13)/(14) におけるリテラル $L_{i.k}^+/L_{i.k}^-$ には否定をともなわない $+t_j(\vec{y_j})/-t_j(\vec{y_j})$ が現れ，否定をともなわない $-t_j(\vec{y_j})/+t_j(\vec{y_j})$ は現れないように記述する．またターゲットスキーマの 1 つの表とソーススキーマの各表との関係として逆方向の更新伝播のルールを記述するために，一次の制約として，否定をともなわない $+t_j(\vec{y_j})$ や $-t_j(\vec{y_j})$ は，ルールのボディにたかだか 1 回しか現れないものとする．同じ $+t_j(\vec{y_j})$ や $-t_j(\vec{y_j})$ が，2 回以上現れることも禁止する．この場合，挿入/削除のタプルの集合に対する self-join となり，実際に挿入/削除されたタプルの集合以上のものの変換となってしまうためである．

例 3．次の GN-Datalog のルールは，単調と一次の制約を満たしていない．

$$+s_1(X_1) \text{ :- } -t1(X_1), \neg s_1(X_1). \tag{15}$$

$$+s_2(X_1, X_2, X_3) \text{ :- } \neg s_2(X_1, X_2, X_3), +t1(X_1, X_2), \\ +t2(X_1, X_3). \tag{16}$$

ルール (15) はボディに現れる $-t1(X_1)$ に対してヘッドに $+s_1(X_1)$ が現れるため，削除に対する挿入への変換となり単調ではない．ルール (16) は，ボディに $+t_1(X_1, X_2)$ と $+t_2(X_1, X_3)$ の 2 つが現れるため，一次の制約を満たしていない．□

3.5　スキーマの共存戦略プログラムの例

2 章では union，selection，projection によるスキーマ進化に対する逆方向の更新伝播のルールを例示した．本節では，projection と join によるスキーマ進化と，それに対する逆方向の更新伝播のルールの例を示す．

3.5.1　projection によるスキーマ進化

ソーススキーマ ver1 の表 `S1` からターゲットスキーマ ver2 の表 `T1` へと projection によりスキーマ進化させ，逆方向へは表 `T1` への挿入/削除を `S1` への挿入/削除に変換し共有するものとする．このスキーマの共存戦略を次のように記述する．

```
% スキーマの定義
source : ver1#s1(X,Y,Z).
target : ver2#t1(X,Y).
% スキーマ進化
t1(X,Y) :- s1(X,Y,Z).
% 逆方向の更新伝播
+s1(X,Y,Z) :- +t1(X,Y), ¬s1(X,Y,_), Z='w'.
-s1(X,Y,Z) :- -t1(X,Y), s1(X,Y,Z)
```

スキーマ進化では，ルールのボディに現れる述語 $\texttt{s1}(X,Y,Z)$ の属性 Z がヘッドには現れず projection となっている．逆方向の更新伝播では，表 `T1` への挿入に対して属性 Z にデフォルト値 `w` を与えて表 `S1` へ挿入するタプルに変換する．`T1` から削除するタプルに対しては，属性 Z の値に関係なく，表 `T1` からの削除を表 `S1` への削除へと変換している．

3.5.2　join によるスキーマ進化

ソーススキーマ ver1 の表 `S1` と `S2` からターゲットスキーマ ver2 の表 `T1` へと，主キーによる join によりスキーマ進化させ，逆方向へは表 `T1` への挿入/削除を `S1` と `S2` への挿入/削除に変換し共有するものとする．このスキーマの共存戦略を次のように記述する．属性 X を主キーとする．

```
% スキーマの定義
source : ver1#s1(X,Y).
source : ver1#s2(X,Z).
target : ver2#t(X,Y,Z).
pk(s1,['X']).
pk(s2,['X']).
pk(t1,['X']).
```

```
% スキーマ進化
t(X, Y, Z) :− s1(X, Y), s2(X, Z).
% 逆方向の更新伝播
+s1(X, Y) :− +t1(X, Y, Z), ¬s1(X, Y), s2(X, Z).
+s2(X, Z) :− +t1(X, Y, Z), ¬s2(X, Z), s1(X, Y).
−s1(X, Y) :− −t1(X, Y, _), ¬+t1(X, Y, _), s1(X, Y),
      s2(X, _).
−s2(X, Z) :− −t1(X, _, Z), ¬+t1(X, _, Z), s1(X, _),
      s2(X, Z).
```

スキーマ進化では，表 S1 と S2 のタプルで主キーとなる変数 X の値が同じタプルが存在すれば，join により表 T1 のタプルを得る．逆方向の更新伝播で $-$s1(X, Y) を定めるルールのボディに否定付きの $+$t1 による述語が現れるのは，削除と挿入のシーケンスによるタプルの値の修正に対応するためである．たとえば表 T1 に存在するタプル (1, 1, 1) を (1, 1, 2) へ修正する場合，(1, 1, 1) の削除と (1, 1, 2) の挿入とによる．しかし表 S1 にはない属性 Z の値の変更であり，表 S1 がタプル (1, 1) を持つ場合，それを削除する必要はない．挿入するタプル (1, 1, 2) と S1 のタプル (1, 1) で変数 Z 以外の値が同じである場合であり，これを否定付きの $+$t1 による述語を用いてルールに表している．$-$s2(X, Z) を定めるルールも同様である．

3.5.3　join と projection によるスキーマ進化

主キーによる join と projection を組み合わせたスキーマ進化とその逆方向の更新伝播によるスキーマの共存戦略を，次に記述する．

```
% スキーマの定義
source : ver1#s1(X, Y).
source : ver1#s2(X, Z, W).
target : ver2#t1(X, Y, Z).
pk(s1, [‘X’]).
pk(s2, [‘X’]).
pk(t1, [‘X’]).
% スキーマ進化
t(X, Y, Z) :− s1(X, Y), s2(X, Z, W).
% 逆方向の更新伝播
+s1(X, Y) :− +t1(X, Y, Z), ¬s1(X, Y).
+s2(X, Z, W) :− +t1(X, Y, Z), ¬s2(X, Z, _), W = ‘w’.
−s1(X, Y) :− −t1(X, Y, _), ¬+t1(X, Y, _), s1(X, Y),
      s2(X, _, _).
−s2(X, Z, W) :− −t1(X, _, Z), ¬+t1(X, _, Z), s1(X, _),
      s2(X, Z, W).
```

前項の主キーによる join の例と比較して，$+$s2(X, Z, W) をヘッドとするルールに，projection される変数 W にデフォルト値 w を与える述語が加えられている．

4.　一貫性の検証

本章では，スキーマの共存戦略の一貫性を検証する手法について説明する．

4.1　一貫性の定義

一貫性とは，ターゲットスキーマ上の更新を逆方向の更新伝播 F' によりソーススキーマと共有し，その結果をスキーマ進化 F により変換した場合にターゲットスキーマに対して余計なタプルを生成しないこと，である．ターゲットスキーマのタプルの集合を T，挿入するタプルの集合を Δ_T^+，削除するタプルの集合を Δ_T^-，その全体を ΔT，演算子 $\oplus$ により ΔT を T に適用した結果のターゲットスキーマの更新後のタプルの集合を T' とした場合，これらの関係は集合の意味論により次のように表される．

$$T' = T \oplus \Delta T = (T \cap \neg \Delta_T^-) \cup \Delta_T^+ \tag{17}$$

ターゲットスキーマのタプルの集合 T と削除分のタプルの集合 Δ_T^- の差集合 $T \setminus \Delta_T^- = T \cap \neg \Delta_T^-$ と，挿入分のタプルの集合 Δ_T^+ との和集合の結果が，更新後のターゲットスキーマのタプルの集合 T' となる．このとき ΔT を逆方向の更新伝播によりソーススキーマと共有し，その結果の $S \oplus F'(S, \Delta T)$ をスキーマ進化 F により変換した結果が T' の部分集合となっていれば，余計なタプルを生成せず一貫性を満たすといえる．ターゲットスキーマのタプルの集合 T は，更新前のソーススキーマのタプルの集合 S をスキーマ進化 F により変換した結果であるため，一貫性は次のように定義される．

定義 2.（スキーマの共存における一貫性）　次を満たすとき，スキーマの共存戦略は一貫性を満たすという．

$$F(S \oplus F'(S, \Delta T)) \subseteq F(S) \oplus \Delta T \tag{18}$$

□

4.2　検証手法

記述されたスキーマの共存戦略が一貫性を満たすかの検証は，Datalog プログラムの決定可能な充足可能性問題として取り扱えることを示す．$A \subseteq B \Leftrightarrow A \cap \neg B = \emptyset$ であることから，定義 2 の一貫性を定める式 (18) は次のように変形される．

$$F(S \oplus F'(S, \Delta T)) \cap \neg (F(S) \oplus \Delta T) = \emptyset \tag{19}$$

左辺を T_{diff} とし，$F(S \oplus F'(S, \Delta T)) = T'$, $F(S) \oplus \Delta T = T''$ とすると，T_{diff} は次のように表される．

$$T_{diff} = T'' \cap \neg T' \tag{20}$$

これにより，T_{diff} が空であれば，スキーマ進化 F と逆方向の更新伝播 F' によるスキーマの共存戦略は一貫性を満たす，といえる．Datalog プログラムである F と F' から，式 (20) に基づき T_{diff} のタプルの集合を定める Datalog プログラムを導出できれば，この Datalog プログラムが充足不可能であることの証明が，T_{diff} が空であることの証明となる．2 章のスキーマの共存戦略を例に，T_{diff} を GN-Datalog によるプログラムとして導出し，決定可能な充足可能性問題として扱えることを，次の例により示す．

例 4. 2.1 節の例に対して，一貫性を検証するための T_{diff} を定める Datalog プログラムを導出し，その充足可能性問題が決定可能であることを示す．ここでスキーマ進化と逆方向の更新伝播のルール (4)–(8) は，GN-Datalog となっている．

式 (20) の T' のタプルの集合を表す述語を $\texttt{ord3'}(O,I,Q)$ とする．T' を定める Datalog のルールは，$T' = F(S) \oplus \Delta T = (F(S) \cap \neg \Delta_T^-) \cup \Delta_T^+$ であることから，$\texttt{ord3'}(O,I,Q)$ を定めるルールは，$\texttt{ord3}(O,I,Q)$ を定めるスキーマ進化 F のルール (4) と (5)，および次のルールから表される．

$$\texttt{ord3'}(O,I,Q) :- \texttt{ord3}(O,I,Q), \neg -\texttt{ord3}(O,I,Q). \tag{21}$$

$$\texttt{ord3'}(O,I,Q) :- +\texttt{ord3}(O,I,Q). \tag{22}$$

式 (20) の T'' のタプルの集合を表す述語を $\texttt{ord3''}(O,I,Q)$ とする．$T'' = F(S \oplus F'(S, \Delta T))$ を定める Datalog のルールは，逆方向の更新伝播 F' を定めるルール (6)–(8)，および次のルールから表される．

$$\texttt{ord3''}(O,I,Q) :- \texttt{ord1'}(O,I,Q,M), I \neq \text{`A'}. \tag{23}$$

$$\texttt{ord3''}(O,I,Q) :- \texttt{ord2'}(O,I,Q), I \neq \text{`A'}. \tag{24}$$

$$\texttt{ord1'}(O,I,Q,M) :- \texttt{ord1}(O,I,Q,M), \neg -\texttt{ord1}(O,I,Q,M). \tag{25}$$

$$\texttt{ord2'}(O,I,Q) :- \texttt{ord2}(O,I,Q), \neg -\texttt{ord2}(O,I,Q). \tag{26}$$

$$\texttt{ord2'}(O,I,Q) :- +\texttt{ord2}(O,I,Q). \tag{27}$$

ルール (23) と (24) は，GN-Datalog によるスキーマ進化 F のルール (4) と (5) のボディに現れる述語の表名を，ord1', ord2', ord3'' に置き換えたものである．

T_{diff} のタプルの集合を表す述語を $\texttt{ord3}_{\texttt{diff}}(O,I,Q)$ とすると，式 (20) の T_{diff} を定めるルールは次に表される．

$$\texttt{ord3}_{\texttt{diff}}(O,I,Q) :- \texttt{ord3''}(O,I,Q), \neg -\texttt{ord3'}(O,I,Q). \tag{28}$$

ルール (21)，(22)，(25)–(27) は，挿入/削除を適用する対象の述語に対して，属性として同じ変数を持つ述語により挿入/削除を適用している．そのため GN-Datalog となっている．ルール (28) も同じ変数を持つ述語からなるルールとなるため，GN-Datalog である．そのため $\texttt{ord3}_{\texttt{diff}}(O,I,Q)$ を定めるルール (4)–(8)，(21)–(28) からなる Datalog プログラムは GN-Datalog によるものであり，その充足可能性は決定可能である．□

例 4 に現れる (21)，(22)，(25)–(28) は，スキーマの共存戦略に依存せず同じ属性の変数だけを扱うルールであるため，任意のスキーマの共存戦略において，T_{diff} を定める Datalog プログラムは GN-Datalog によるものとなり，その充足可能性問題が決定可能である．

5. スキーマの共存戦略の実現

既存の関係データベース管理システム上でスキーマの共存戦略を実現するには，記述されたスキーマ進化と逆方向の更新伝播のルールを，各スキーマバージョンのビューと物理データとの間の論理的な変換に置き換える必要がある．本章では，2 つの双方向変換を導出することにより，これを実現する手法を説明する．始めに双方向変換について説明し，全体的な概要を説明したのち，補助リレーションが必要とされる場合を整理し，導出の詳細を説明する．

5.1 双方向変換

双方向変換 [15] はソースデータ $\mathcal{S}$ とビューデータ $\mathcal{V}$ の 2 つの集合の間の変換であり，順方向変換 *get* と逆方向変換 *put* とからなる．順方向変換 *get* は，$\mathcal{V} = get(\mathcal{S})$ によりソースデータ $\mathcal{S}$ をビューデータ $\mathcal{V}$ へと変換する．逆方向変換 *put* は，一般的に *get* は単射とは限らずすべてのソースデータがビューデータに反映されるとは限らないため，$\mathcal{S}' = put(\mathcal{S}, \mathcal{V}')$ により，元のソースデータ $\mathcal{S}$ と更新されたビューデータ $\mathcal{V}'$ から，更新されたソースデータ $\mathcal{S}'$ へと変換する．双方向での変換の整合性を保証するために，*get* と *put* のペアは次の性質を満たすことが求められる．

$$put(\mathcal{S}, get(\mathcal{S})) = \mathcal{S} \quad (\textsc{GetPut})$$

$$get(put(\mathcal{S}, \mathcal{V}')) = \mathcal{V}' \quad (\textsc{PutGet})$$

GetPut は「変換後のビュー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「更新されたビューデータは，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．

5.2 スキーマの共存戦略を実現する双方向変換

記述されたスキーマの共存戦略を実現するにあたり，ソーススキーマ，ターゲットスキーマのそれぞれが，スキーマの共存の定義 1 を満たすための要件を明らかにする．その要件を実現する双方向変換が，どのようなものであるかを説明する．

5.2.1　ソーススキーマ，ターゲットスキーマに求められる要件

ソーススキーマには，次の要件が求められる．

- 要件 1：定義 1 の (2) を満たすために，ソーススキーマに対する任意の更新を受け付け，その結果を保持する．
- 要件 2：定義 1 の (3) を満たすために，ターゲットスキーマから逆方向の更新伝播のルールにより更新が共有される場合は，その更新を受け付け結果を保持する．

ターゲットスキーマには，次の要件が求められる．

- 要件 3：定義 1 の (2) を満たすために，ターゲットスキーマに対する任意の更新を受け付け，その結果を保持する．
- 要件 4：定義 1 の (1) を満たすために，ソーススキーマからのスキーマ進化により共有されるデータと更新を受け付け，その結果を保持する．

これら 4 つの要件に対し，要件 1 をソーススキーマに対する双方向変換により，要件 2 から 4 をターゲットスキーマに対する双方向変換により，実現する．次にそれぞれの双方向変換がどのようなものであるのか，その概要を説明する．

5.2.2　ソーススキーマに対する双方向変換

ビューデータをソーススキーマのビュー S とし，ソースデータをそれと同じデータ構造で物理データを保持する実リレーション S_b とする．S_b と S の間で GetPut と PutGet を満たす恒等写像となる双方向変換を導出することで，要件 1 を満たすようにする．このような双方向変換を get_S，put_S として導出する．

5.2.3　ターゲットスキーマに対する双方向変換

ビューデータをターゲットキーマのビュー T とし，ソースデータをソーススキーマに対応する実リレーション S_b と補助リレーション A からなる (S_b, A) とする．ターゲットスキーマに対する双方向変換を，これらの間の双方向変換とする．ターゲットスキーマの更新が逆方向に共有される場合と，されない場合とから，補助リレーションが必要とされる場合を整理し，それに基づいて双方向変換を導出していく．

逆方向に更新を共有する場合，S と S_b の間が恒等写像であるため，要件 2 を，ターゲットスキーマのビュー T に対する更新を，逆方向の更新伝播のルールに従い実リレーション S_b に反映できること，と置き換える．これを実現する T から S_b への put を導出することで，要件 2 を満たす．同様に要件 4 は，実リレーション S_b から，スキーマ進化のルールに従いターゲットスキーマのビュー T を得ることと置き換える．これを実現する S_b から T への get を導出することで，要件 4 を満たす．要件 3 は，このような get と put が GetPut と PutGet を満たせば，ビュー T の更新が保たれ実現される．しかしながら，任意のスキーマの共存戦略に対して GetPut と PutGet を満たすとは限らない．そのため，満たされない場合に補助リレーションを利用することで GetPut と PutGet を満たすようにし，要件 2，3，4 を満たす get と put を導出していく．

前提としてスキーマの共存戦略が一貫性を満たしている場合，ビュー T の更新に対して逆方向の更新伝播に次いでスキーマ進化を適用した際に，更新されたビュー T に対し余計なタプルは生成されないものの，必要なすべてのタプルが生成されることは保証されていない．この場合必要なタプルが不足し PutGet が満たされない．そのため put を逆方向の更新伝播を実現し，かつこの不足するタプルを補助リレーション A に反映するものとして導出する．そして get を，実リレーション S_b からはスキーマ進化を実現し，その結果と補助リレーション A との union により T を得るものとして導出する．このような get と put を，get_T，put_T として GetPut と PutGet を満たすように導出し，要件 2，3，4 を満たすようにする．

逆方向に更新を共有しない場合は，要件 2 は求められず，実リレーション S_b を更新しない put が求められる．そのため要件 3 に対しては，ターゲットスキーマのビュー T の更新のすべてを補助リレーション A に反映する put を導出する．要件 4 に対しては，実リレーション S_b からはスキーマ進化の関係を実現し，かつ補助リレーション A を利用してビュー T の更新された結果のタプルを生成できるような get を導出する．このような get と put を，get_T^{uns}，put_T^{uns} として GetPut と PutGet を満たすように導出し，要件 3，4 を満たすようにする．

5.3　ソーススキーマに対する双方向変換の導出

5.3.1　導出手法

恒等写像となる双方向変換 get_S と put_S を導出する．get_S は恒等写像であるため，ソースデータ S_b をそのままにビューデータ S へと変換する get_S を次のように定める．

$$\begin{aligned} S &= get_S(S_b) \\ &= S_b \end{aligned} \tag{29}$$

put_S は，ビューデータ S へ挿入したタプルの集合をソースデータ S_b にそのままに挿入し，S から削除したタプルの集合をそのままに S_b から削除する恒等写像として，次に定める．

$$\begin{aligned} S'_b &= put_S(S_b, S') \\ &= (S_b \cap \neg(\neg S' \cap S_b)) \cup (S' \cap \neg S_b) \end{aligned} \tag{30}$$

更新後のソースデータ $S'_b = (S_b \cap \neg\Delta^-_{S_b}) \cup \Delta^+_{S_b}$ に対し，ソースデータ S_b への挿入/削除のタプルの集合 $\Delta^+_{S_b}/\Delta^-_{S_b}$ を，更新後のビューデータ S' からの恒等写像となるように $\Delta^+_{S_b} = S' \cap \neg S_b$，$\Delta^-_{S_b} = \neg S' \cap S_b$ と定めている．

5.3.2　性質

get_S と put_S は次の性質を満たす．

命題 3.（ソーススキーマの双方向変換） get_S と put_S は GetPut, PutGet を満たす．□

5.3.3　Datalog プログラムの導出

スキーマの共存戦略を実現するために，双方向変換 get_S，put_S の Datalog プログラムを導出し，さらに BIRDS [23] を利用し SQL プログラムへと変換する．

ソーススキーマの各表 S_i（$i \in [1,n]$）をビューとし，それぞれの双方向変換 $get_{S.i}$，$put_{S.i}$ を導出する．$\vec{x_i}$ を属性，ソーススキーマのビューに対応する述語を $s_i(\vec{x_i})$，そのビューに対応する実リレーションの述語を $s_{b.i}(\vec{x_i})$ とするとき，式 (29) を表す $get_{S.i}$ の Datalog プログラムは次のようになる．

$$s_i(\vec{x_i}) :- \ s_{b.i}(\vec{x_i}). \tag{31}$$

$s'_i(\vec{x_i})$ を更新後のソーススキーマのビューに対応する述語とするとき，式 (30) を表す $put_{S.i}$ の Datalog プログラムは次のようになる．

$$\begin{aligned}
&+ s_{b.i}(\vec{x_i}) :- s'_i(\vec{x_i}), \neg s_{b.i}(\vec{x_i}). \\
&- s_{b.i}(\vec{x_i}) :- \neg s'_i(\vec{x_i}), s_{b.i}(\vec{x_i}). \\
&s'_{b.i}(\vec{x_i}) :- s_{b.i}(\vec{x_i}), \neg - s_{b.i}(\vec{x_i}). \\
&s'_{b.i}(\vec{x_i}) :- + s_{b.i}(\vec{x_i}).
\end{aligned} \tag{32}$$

5.4　逆方向に更新を共有する場合のターゲットスキーマに対する双方向変換の導出

5.4.1　導出手法

双方向変換 get_T と put_T をスキーマの共存戦略から導出する．

get_T を次に定める．

$$\begin{aligned}
T &= get_T(S_b, A) \\
&= get_{T.S_b}(S_b) \cup A
\end{aligned} \tag{33}$$

get_T は，スキーマ進化のルールを実現する $get_{T.S_b}$ の結果と補助リレーション A との union によりビュー T を得る．スキーマ進化のルール F は式 (9) の $T = F(S)$ よりソーススキーマのビュー S を変換するものであり，S は実リレーション S_b との間で恒等写像であることから，$get_{T.S_b}$ を S_b を F により変換するものとして次のように定める．

$$get_{T.S_b}(S_b) = F(S_b) \tag{34}$$

次に put_T を以下に定める．

$$\begin{aligned}
(S'_b, A') &= put_T((S_b, A), T') \\
&= (put_{T.S_b}(S_b, T'), put_{T.A}((S_b, A), T'))
\end{aligned} \tag{35}$$

ここで $put_{T.S_b}$ を $S'_b = put_{T.S_b}(S_b, T')$ とし，逆方向の更新伝播のルールを実現するものとする．$put_{T.A}$ を $A' = put_{T.A}((S_b, A), T')$ とし，ビュー T の任意の更新を保つようにするために補助リレーション A を更新するものとする．$put_{T.S_b}(S_b, T')$ は，付録 A.2 に示す導出手法により，スキーマ進化を定める F と逆方向の更新伝播を定める F' から次のように定められる．

$$\begin{aligned}
S'_b &= put_{T.S_b}(S_b, T') \\
&= (S_b \cap \neg(F'(S_b, (\neg T' \cap (F(S_b) \cup A))))) \\
&\quad \cup F'(S_b, T' \cap \neg(F(S_b) \cup A))
\end{aligned} \tag{36}$$

$put_{T.A}((S_b, A), T')$ は，付録 A.3 に示す導出手法により，$get_{T.S_b}$ と $put_{T.S_b}$ から次のように定められる．

$$\begin{aligned}
A' &= put_{T.A}((S_b, A), T') \\
&= (A \cap \neg(\neg T' \cap A)) \\
&\quad \cup (T' \cap \neg(get_{T.S_b}(put_{T.S_b}(S_b, T'))) \cap \neg A)
\end{aligned} \tag{37}$$

5.4.2　性質

get_T と put_T は，次の性質を満たす．

命題 4.（逆方向に更新を共有する場合のターゲットスキーマの双方向変換） get_T と put_T は GetPut, PutGet を満たす．□

5.4.3　Datalog プログラムの導出

スキーマの共存戦略を実現するために，get_T と put_T を表す Datalog プログラムを導出し，BIRDS [23] を利用し SQL プログラムへと変換する．ターゲットスキーマの表 T_j をビューとし，式 (33) と (34) を表す $get_{T.j}$ の Datalog プログラムを次に定める．

$$t_j(\vec{y_j}) :- \ L'_{j.1}, \ldots, L'_{j.k}, \ldots, L'_{j.kn}. \tag{38}$$

$$t_j(\vec{y_j}) :- \ a_j(\vec{y_j}). \tag{39}$$

式 (38) のリテラル $L'_{j.k}$ は，スキーマ進化 F を定める (10) のリテラル $L_{j.k}$ に現れる述語 $s_i(\vec{x_i})$ を実リレーションに対応する $s_{b.i}(\vec{x_i})$ に置き換えたものである．$a_j(\vec{y_j})$ はビュー T_j に対する補助リレーション A_j に対応する述語である．ターゲットスキーマのビュー T_j に挿入されたタプルのうち，一定の条件を満たすものを補助リレーション A_j へと挿入するため（付録 A.3），$a_j(\vec{y_j})$ と $t_j(\vec{y_j})$ とは同じ属性 $\vec{y_j}$ を持っている．

次に $put_{T.j}$ の Datalog プログラムを以下に定める．

$$s'_{b.i}(\vec{x_i}) :- \ s_i(\vec{x_i}), \neg - s_i(\vec{x_i}). \tag{40}$$

$$s'_{b.i}(\vec{x_i}) :- \ + s_i(\vec{x_i}). \tag{41}$$

$$+ s_{b.i}(\vec{x_i}) :- \ L'^{+}_{i.1}, \ldots, L'^{+}_{i.k}, \ldots, L'^{+}_{i.kn+}. \tag{42}$$

$$- s_{b.i}(\vec{x_i}) :- \ L'^{-}_{i.1}, \ldots, L'^{-}_{i.k}, \ldots, L'^{-}_{i.kn-}. \tag{43}$$

$$+ t_j(\vec{y_j}) :- \ t_j(\vec{y_j}), \neg t_{tmp.j}(\vec{y_j}). \tag{44}$$

$$- t_j(\vec{y_j}) :- \ \neg t_j(\vec{y_j}), t_{tmp.j}(\vec{y_j}). \tag{45}$$

$$t_{tmp.j}(\vec{y_j}) :- L'_{j.1}, \ldots, L'_{j.k}, \ldots, L'_{j.kn}. \quad (46)$$
$$t_{tmp.j}(\vec{y_j}) :- a_j(\vec{y_j}). \quad (47)$$
$$a'_j(\vec{y_j}) :- a_j(\vec{y_j}), \neg -a_j(\vec{y_j}). \quad (48)$$
$$a'_j(\vec{y_j}) :- +a_j(\vec{y_j}). \quad (49)$$
$$t''_j(\vec{y_j}) :- L''_{j.1}, \ldots, L''_{j.k}, \ldots, L''_{j.kn}. \quad (50)$$
$$+a_j(\vec{y_j}) :- t_j(\vec{y_j}), \neg t''_j(\vec{y_j}), \neg a_j(\vec{y_j}). \quad (51)$$
$$-a_j(\vec{y_j}) :- \neg t_j(\vec{y_j}), a_j(\vec{y_j}). \quad (52)$$

付録 A.2 の $put_{T.S_b}$ の導出における式 (A.1)–(A.6) のそれぞれを Datalog ルール (40)–(47) により表し，付録 A.3 の $put_{T.A}$ の導出における式 (A.8)–(A.11) のそれぞれを，Datalog ルール (48)–(52) により表す．ルール (42) と (43) に現れるリテラル $L'^{+}_{i.k}/L'^{-}_{i.k}$ は，ルール (13) と (14) のリテラル $L^{+}_{i.k}/L^{-}_{i.k}$ に現れる述語 $s_i(\vec{x_i})$ を，実リレーションに対応する述語 $s_{b.i}(\vec{x_i})$ に置き換えたものである．ルール (50) は更新された実リレーション S'_b から更新されたビュー T'' を得る式 (A.9) に対応しており，リテラル $L''_{j.k}$ は，ルール (10) のリテラル $L_{j.k}$ に現れる述語 $s_i(\vec{x_i})$ を $s'_{b.i}(\vec{x_i})$ に置き換えたものである．

5.5 逆方向に更新を共有しない場合のターゲットスキーマに対する双方向変換の導出

5.5.1 導出手法

双方向変換 get^{uns}_T と put^{uns}_T をスキーマの共存戦略から導出する．ターゲットスキーマ上の更新をソーススキーマと共有しないため，put^{uns}_T は実リレーション S_b は更新せずに補助リレーション A を更新するようにする．補助リレーションを $A = (A_{ins}, A_{del})$ とし，ターゲットスキーマの表 T に対して挿入されたタプルを A_{ins} に，削除されたタプルを A_{del} に物理データとして保持する．スキーマ進化によって得られるタプルの集合 $F(S_b)$ に対して A_{ins} と A_{del} を適用することで，更新を反映したターゲットスキーマのビュー T を得られるように，get^{uns}_T を次に定める．

$$\begin{aligned} T &= get^{uns}_T(S_b, (A_{ins}, A_{del})) \\ &= (F(S_b) \cap \neg A_{del}) \cup A_{ins} \end{aligned} \quad (53)$$

put^{uns}_T を次に定める．

$$\begin{aligned} (S_b, (A'_{ins}, A'_{del})) &= put^{uns}_T((S_b, (A_{ins}, A_{del})), T') \\ &= (S_b, ((A_{ins} \cap \neg \Delta^{-}_{A_{ins}}) \cup \Delta^{+}_{A_{ins}}, \\ &\quad (A_{del} \cap \neg \Delta^{-}_{A_{del}}) \cup \Delta^{+}_{A_{del}})) \end{aligned} \quad (54)$$

A_{ins}，A_{del} に対する挿入/削除の $\Delta^{+}_{A_{ins}}/\Delta^{-}_{A_{ins}}$，$\Delta^{+}_{A_{del}}/\Delta^{-}_{A_{del}}$ は，付録 A.5 に示す導出手法により，次のように定められる．

$$\Delta^{+}_{A_{ins}} = T' \cap \neg F(S_b) \cap \neg A_{ins} \quad (55)$$
$$\Delta^{-}_{A_{ins}} = \neg T' \cap A_{ins} \quad (56)$$
$$\Delta^{+}_{A_{del}} = \neg T' \cap F(S_b) \cap \neg A_{del} \quad (57)$$
$$\Delta^{-}_{A^{del}} = T' \cap A_{del} \quad (58)$$

5.5.2 性質

補助リレーション A_{ins} と A_{del} は，次の性質を満たす．

補題 5.（互いに素） A_{ins} と A_{del} は互いに素である． □

get^{uns}_T と put^{uns}_T は，次の性質を満たす．

命題 6.（逆方向に更新を共有しない場合のターゲットスキーマの双方向変換） get^{uns}_T と put^{uns}_T は GetPut，PutGet を満たす． □

5.5.3 Datalog プログラムの導出

スキーマの共存戦略を実現するために，get^{uns}_T と put^{uns}_T を表す Datalog プログラムを導出し，BIRDS [23] を利用し SQL プログラムへと変換する．ターゲットスキーマの表 T_j をビューとし，スキーマ進化の Datalog プログラム F から，双方向変換 $get^{uns}_{T.j}$ と $put^{uns}_{T.j}$ を導出する．

式 (53) を表す $get^{uns}_{T.j}$ の Datalog プログラムを，次に定める．

$$t_{tmp.j}(\vec{y_j}) :- L'_{j.1}, \ldots, L'_{j.k}, \ldots, L'_{j.kn}. \quad (59)$$
$$t_j(\vec{y_j}) :- t_{tmp.j}(\vec{y_j}), \neg a_{del.j}(\vec{y_j}). \quad (60)$$
$$t_j(\vec{y_j}) :- a_{ins.j}(\vec{y_j}). \quad (61)$$

ルール (59) のリテラル $L'_{j.k}$ は，スキーマ進化 F を定める (10) のリテラル $L_{j.k}$ に現れる述語 $s_i(\vec{x_i})$ を実リレーションに対応する $s_{b.i}(\vec{x_i})$ に置き換えたものである．ルール (61) の $a_{ins.j}(\vec{y_j})$，ルール (60) の $a_{del.j}(\vec{y_j})$ は，ビュー T_j に対する補助リレーション $A_{ins.j}$，$A_{del.j}$ に対応する述語であり，述語 $t_j(\vec{y_j})$ と同じ属性を持つ．

式 (54)–(58) を表す $put^{uns}_{T.j}$ の Datalog プログラムを，次に定める．

$$a'_{ins.j}(\vec{y_j}) :- a_{ins.j}(\vec{y_j}), \neg -a_{ins.j}(\vec{y_j}). \quad (62)$$
$$a'_{ins.j}(\vec{y_j}) :- +a_{ins}(\vec{y_j}). \quad (63)$$
$$a'_{del.j}(\vec{y_j}) :- a_{del.j}(\vec{y_j}), \neg -a_{del}(\vec{y_j}). \quad (64)$$
$$a'_{del.j}(\vec{y_j}) :- +a_{del}(\vec{y_j}). \quad (65)$$
$$+a_{ins.j}(\vec{y_j}) :- t_j(\vec{y_j}), \neg t_{tmp.j}(\vec{y_j}), \neg a_{ins.j}(\vec{y_j}). \quad (66)$$
$$-a_{ins.j}(\vec{y_j}) :- \neg t_j(\vec{y_j}), a_{ins.j}(\vec{y_j}). \quad (67)$$
$$+a_{del.j}(\vec{y_j}) :- \neg t_j(\vec{y_j}), t_{tmp.j}(\vec{y_j}), \neg a_{del.j}(\vec{y_j}). \quad (68)$$
$$-a_{del.j}(\vec{y_j}) :- t_j(\vec{y_j}), a_{del.j}(\vec{y_j}). \quad (69)$$
$$\bot :- a_{ins.j}(\vec{y_j}), a_{del.j}(\vec{y_j}). \quad (70)$$

ルール (62)–(65) により式 (54) を，ルール (66)–(69) により式 (55)–(58) を，Datalog ルールとして表している．ルール (70) は，補題 5 の補助リレーション A_{ins} と A_{del} は互いに素となることを，制約として加えている．

5.6　健全性

5.3 節，5.4 節，5.5 節に示した双方向変換の導出手法は，5.2 節のスキーマの共存戦略を実現するために求められる 4 つの要件を満たすための双方向変換を導出し，かつそれぞれの双方向変換が GetPut，PutGet を満たすため，健全である．

6.　評価

提案したスキーマの共存戦略の実現手法を OCaml で実装し，提案手法の有効性を確認した．一貫性の検証を行う Datalog プログラムの充足可能性の判定は，Datalog プログラムを同等な関係論理に変換し，その充足可能性を SMT ソルバ Z3 を利用し判定している．双方向変換エンジン BIRDS [23] を利用し，双方向変換を表す Datalog プログラムが GetPut と PutGet を満たすことを実際に確認し，ビュー定義とトリガーの SQL プログラムを出力する．出力された SQL プログラムは，PostgreSQL 10.16 上での動作を確認している．

先行研究 [16], [17] の SMO を文献 [16] の Figure 2 から抽出し，そのスキーマの共存戦略[*2]，およびそれ以外の新しい戦略とを提案した手法により記述し，実現のための双方向変換，補助リレーション，SQL プログラムとが導出されることを確認する評価実験を行った．

表 1 に結果を示す．表の見方として，たとえば SMO の演算子 `DROP COLUMN` は，ソーススキーマの表 S1 からターゲットスキーマの表 T1 へと，projection によりカラムを削除するスキーマ進化を行う．逆方向の更新伝播は表 T1 での更新をすべて表 S1 と共有する．このようなスキーマの共存戦略を実現するために，ビューと 1 つの補助リレーションを含む物理データ間の変換を定める，6 LOC（Lines Of Code）からなるプログラムが設計されている．本論文が提案する手法によりこのスキーマの共存戦略を記述すると 8 LOC の記述量となり，導出される双方向変換の Datalog プログラムが 19 LOC，SQL プログラムが 386 LOC，補助リレーションが 1 つとなる．

SMO では `DECOMPOSE TABLE`/`JOIN TABLE` はペアとなっており，ソーススキーマとターゲットスキーマを入れ替えた際に同等のスキーマの共存戦略を実現するものとなっている．そのため，任意のカラムによる outer join がスキーマ進化となる `JOIN TABLE` は，`DECOMPOSE TABLE` によって評価し，主キーおよび外部キーによる inner/outer join と任意のカラムによる inner join が逆方向の更新伝播となる `DECOMPOSE TABLE` は，`JOIN TABLE` により評価した．同様に，表 S1 から表 T1 と T2 の 2 つに分割するスキーマ進化となる `SPLIT TABLE` は，`MERGE TABLE` により評価した．また Cartesian product と difference によるスキーマ進化に対しては，SMO では直接に対応する演算子は用意されていないものの，これらに対してもスキーマの共存戦略が記述，実現可能であることを確認した．

表 1 に示すように，提案した手法によりすべての SMO に対してそのスキーマの共存戦略を記述でき，実現できることを確認した．さらに SMO とは異なったスキーマの共存戦略も記述でき，実現できることを確認した．スキーマの共存戦略の記述量は 3〜22 LOC であるため，極端に大きなプログラムとはならずに様々な共存戦略の記述が可能になっている．先行研究 [16] において SMO ごとに設計されているビューと物理データ間の変換のプログラムと比較しても，同程度の記述量となっており，むしろ提案する手法では補助リレーションが自動的に導出されるため，共存戦略の設計，記述するうえでの負荷を軽減している．導出された補助リレーションの数は，先行研究で設計されているものと同程度となっている．ターゲットスキーマの 1 つの表に対して逆方向に更新を共有する場合はたかだか 1 つ，共有しない場合は 2 つに限定して導出している．そのため補助リレーションで扱う物理データを限定的なものにしている．これらにより，提案した手法の有効性が確認された．

7.　関連研究

情報システムの継続的な発展において，データベーススキーマの複数バージョンの共存は実際的な要件となっている．データベースリファクタリング [1] は，スキーマの改修を類型化し，新旧スキーマバージョンを共存させるための SQL プログラムを実例集としてまとめている．そのような SQL プログラムをスキーマバージョンに応じて効率的に管理，実行するツールとして，Flyway [14] や Liquibase [20] が広く使われている．

これらの手法では直接 SQL プログラムを記述する必要があり開発者への負荷が高いことから，効率化，自動化する手法が広く研究されてきている [7], [21]．Curino らによる PRISM/PRISM++ [9], [10] では，スキーマの改修におけるカラムの追加やリレーションの統合などの基本的な操作を SMO（Schema Modificaiton Operator）として用意し [11]，SMO により新しいスキーマバージョンを定義することで，古いスキーマバージョンに対して発行された SQL クエリを，新しいスキーマバージョンへ自動的に変換する．ただし，新しいデータベースとして新しいスキーマバージョンのデータのみが存在しており，スキーマの共存は実現されていない．これに対して，Herrmann らは MSVDB を提案している [16], [17]．同様に SMO により新しいスキーマバージョンを定義し，SMO ごとにあらかじめ用意されたスキーマの共存戦略により，それぞれのスキーマバージョンでデータを有しながらスキーマバージョン間で更新データを共有できるスキーマの共存を実現して

[*2] 文献 [16] に共存戦略が記載されている SMO を抽出している．共存戦略が記載されていない SMO はビューと物理データの間の恒等写像となっており，容易に実現されるためである．

表 1　評価の結果
Table 1　Evaluation results.

SMOs			スキーマの共存戦略			提案する手法			
演算子（ソーススキーマの表 → ターゲットスキーマの表）	ビュー/物理データの変換 [LOC]	補助	スキーマ進化	逆方向の更新伝播	SMO の戦略	共存戦略 [LOC]	双方向変換 [LOC]	SQL [LOC]	補助
ADD COLUMN (S1 → T1)	6	1	outer join（主キー）	すべて S1 と共有	✓	16	44	752	0
				S1 と共有しない		9	35	622	2
DROP COLUMN (S1 → T1)	6	1	projection	すべて S1 と共有	✓	8	19	386	1
				S1 と共有しない		3	17	367	2
DECOMPOSE TABLE (S1→ T1, T2)	18	1	projection	outer join（任意のカラム）として S1 と共有	✓	22	40	644	0
				S1 と共有しない		10	40	622	4
JOIN TABLE (S1, S2 → T1)	8	0	outer join（主キー）	すべて S1，S2 と共有	✓	16	44	752	0
				S1，S2 と共有しない		9	35	622	2
	14	0	outer join（外部キー）	すべて S1，S2 と共有	✓	20	52	1,187	0
				S1，S2 と共有しない		5	26	553	2
	10	0	inner join（主キー）	すべて S1，S2 と共有	✓	11	39	748	1
				S1，S2 と共有しない		9	31	586	2
	18	2	inner join（任意のカラム）	すべて S1，S2 と共有	✓	21	48	1,089	1
				S1，S2 と共有しない		4	23	530	2
SPLIT TABLE (S1 → T1)	7	1	selection	条件を定めず S1 と共有	✓	7	23	467	1
				selection と同じ条件で S1 と共有		5	21	407	1
				S1 と共有しない		3	17	370	2
MERGE TABLE (S1, S2 → T1)	18	1	union	条件を定めて S1，S2 と共有	✓	13	42	835	1
				挿入は S1 と，削除は S1，S2 と共有		8	22	514	0
				S1，S2 と共有しない		5	26	537	2
- (S1, S2 → T1)	-	-	Cartesian product	すべて S1，S2 と共有		20	47	1,085	0
				S1，S2 と共有しない		5	24	536	2
- (S1, S2 → T1)	-	-	difference	S1 と共有し，S2 とは共有しない		6	28	588	0
				S1，S2 と共有しない		4	23	536	2

いる．

一方 Herrmann らの MSVDB ではスキーマの共存戦略があらかじめ用意されたものに限られるため，本論文では共存戦略をプログラム可能にするための手法を提案し，既存 RDBMS 上での実現を可能にしている．

スキーマの共存戦略はスキーマバージョン間でのデータ変換を定めている．このような異種のデータモデル間での整合性あるデータ変換については，プログラミング言語における双方向変換や，データベース分野におけるビュー更新問題 [3], [12] として研究されてきている．

双方向変換では，Foster らによる lenses [15] は，用意された双方向変換となる基本的な演算子を組み合わせ，より大きな双方向変換を設計する手法を提案している．ビュー更新問題では，Bohannon らによる relational lenses [5] は，リレーショナル代数を双方向変換とするような演算子を用意し，それらを組み合わせてより更新可能なビューを設計する手法を提案している．しかしながら個々の演算子のビュー更新戦略があらかじめ用意されたものに限定的されるため，これをプログラム可能にする手法として，Tran らは BIRDS [23] を提案している．

双方向変換やビュー更新問題では，ビュー定義により得られるようなタプルに限定して，その更新をデータベースインスタンスへと反映するビュー更新戦略を定めている．一方スキーマの共存では，スキーマ進化では得られないタプルも含め，任意の更新が発生しうる．本論文では，補助リレーションを導出し任意の更新に対応することで，様々なスキーマの共存戦略を実現を可能にしている．

8. 結論

本論文では，スキーマの共存戦略を記述するドメイン特化言語と，記述された戦略を双方向変換により実現する手法とを提案した．先行研究におけるスキーマの共存戦略やそれ以外の戦略を記述，実現することができ，本論文の手法の有用性が確認された．

一方，本論文が提案するスキーマの共存戦略の記述と実現手法の完全性は主張していない．進化後の複数の表が同

じ属性やタプルを持ち，1 つのタプルの更新に対して表の間で依存関係が発生する場合や，関数従属性など表の中で依存関係がある場合などの対応を検討していないためである．これらは今後の検討課題である．さらに 1 つのスキーマバージョンからスキーマ進化を重ね段階的に複数のスキーマバージョンを得る場合，このような依存関係が複数のスキーマバージョンに波及しうる．Herrmann らが提案する MSVDB [16], [17] では，あらかじめ定められたスキーマの共存戦略により複数のスキーマバージョンへの進化と共存へと対応しているが，本論文ではユーザ任意のスキーマの共存戦略でのこのような依存関係の波及に対する対応を検討していないため，スキーマ進化の連続における複数スキーマの共存実現は，今後の検討課題としている．

謝辞　本研究の一部は JSPS 科研費 JP17H06099 の助成を受けたものです．

参考文献

[1] Ambler, S.W. and Sadalage, P.J.: *Refactoring Databases: Evolutionary Database Design* (*paperback*), Pearson Education (2006).

[2] Aulbach, S., Seibold, M., Jacobs, D. and Kemper, A.: Extensibility and Data Sharing in Evolving Multi-Tenant Databases, *2011 IEEE 27th International Conference on Data Engineering*, pp.99–110, IEEE (2011).

[3] Bancilhon, F. and Spyratos, N.: Update semantics of relational views, *ACM Trans. Database Systems* (*TODS*), Vol.6, No.4, pp.557–575 (1981).

[4] Bárány, V., Ten Cate, B. and Otto, M.: Queries with guarded negation, *Proc. VLDB Endow.*, Vol.5, No.11, pp.1328–1339 (2012).

[5] Bohannon, A., Pierce, B.C. and Vaughan, J.A.: Relational lenses: A language for updatable views, *Proc. 25th ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems*, pp.338–347 (2006).

[6] Brodie, M.L. and Liu, J.T.: *The power and limits of relational technology in the age of information ecosystems*, On The Move Federated Conferences and Workshops (2010).

[7] Cauruccuio, L., Polese, G. and Tortoga, G.: Synchronization of queries and views upon schema evolutions: A survey, *ACM Trans. Database Systems* (*TODS*), Vol.41, No.2, pp.1–41 (2016).

[8] Ceri, S., Gottlob, G., Tanca, L., et al.: What you always wanted to know about Datalog (and never dared to ask), *IEEE Trans. Knowledge and Data Engineering*, Vol.1, No.1, pp.146–166 (1989).

[9] Curino, C.A., Moon, H.J., Deutsch, A. and Zaniolo, C.: Update rewriting and integrity constraint maintenance in a schema evolution support system: PRISM++, *Proc. VLDB Endowment*, Vol.4, No.2, pp.117–128 (2010).

[10] Curino, C.A., Moon, H.J. and Zaniolo, C.: Graceful database schema evolution: The prism workbench, *Proc. VLDB Endowment*, Vol.1, No.1, pp.761–772 (2008).

[11] Curino, C.A., Tanca, L., Moon, H.J. and Zaniolo, C.: Schema evolution in wikipedia: Toward a web information system benchmark, *International Conference on Enterprise Information Systems* (*ICEIS*), Citeseer (2008).

[12] Dayal, U. and Bernstein, P.A.: On the correct translation of update operations on relational views, *ACM Trans. Database Systems* (*TODS*), Vol. 7, No.3, pp.381–416 (1982).

[13] Fitzgerald, B. and Stol, K.J.: Continuous software engineering: A roadmap and agenda, *Journal of Systems and Software*, Vol.123, pp.176–189 (2017).

[14] Flayway: Flyway by Redgate (2020).

[15] Foster, J.N., Greenwald, M.B., Moore, J.T., Pierce, B.C. and Schmitt, A.: Combinators for bi-directional tree transformations: A linguistic approach to the view update problem, *ACM SIGPLAN Notices*, Vol.40, No.1, pp.233–246 (2005).

[16] Herrmann, K., Voigt, H., Behrend, A., Rausch, J. and Lehner, W.: Living in Parallel Realities: Co-Existing Schema Versions with a Bidirectional Database Evolution Language, *Proc. 2017 ACM SIGMOD International Conference on Management of Data*, pp.1101–1116 (2017).

[17] Herrmann, K., Voigt, H., Pedersen, T. and Lehner, W.: Multi-schema-version data management: Data independence in the twenty-first century, *The VLDB Journal*, Vol.27, No.4, pp.547–571 (2018).

[18] Humble, J. and Farley, D.: *Continuous Delivery: Reliable Software Releases through Build, Test, and Deployment Automation* (*Adobe Reader*), Pearson Education (2010).

[19] Keller, A.M.: Algorithms for translating view updates to database updates for views involving selections, projections, and joins, *Proc. 4th ACM SIGACT-SIGMOD Symposium on Principles of Database Systems*, pp.154–163 (1985).

[20] Liquibase: Liquibase by Datical (2020).

[21] Roddick, J.F.: A survey of schema versioning issues for database systems, *Information and Software Technology*, Vol.37, No.7, pp.383–393 (1995).

[22] Terwilliger, J.F., Delcambre, L.M., Maier, D., Steinhauer, J. and Britell, S.: Updatable and Evolvable Transforms for Virtual Databases, *Proc, VLDB Endow.*, Vol.3, No.1-2, pp.309–319 (2010).

[23] Tran, V.D., Kato, H. and Hu, Z.: Programmable View Update Strategies on Relations, *Proc. VLDB Endow.*, Vol.13, No.5, pp.726–739 (2020).

[24] Weissman, C.D. and Bobrowski, S.: The Design of the force.com Multitenant Internet Application Development Platform, *Proc. 2009 ACM SIGMOD International Conference on Management of Data*, pp.889–896 (2009).

付　　録

A.1　命題 3 の証明

証明.　GetPut を証明する．

$$
\begin{aligned}
&put_S(S_b, get_S(S_b)) \\
&= \{get_S \text{と } put_S \text{の定義}\} \\
&\quad (S_b \cap \neg(\neg S_b \cap S_b)) \cup (S_b \cap \neg S_b) \\
&= \{X \cap \neg X = \emptyset\} \\
&\quad S_b
\end{aligned}
$$

PutGet を証明する．

$$get_S(put_S(S_b, S'))$$

$= \{get_S$ と put_S の定義$\}$
$(S_b \cap \neg(\neg S' \cap S_b)) \cup (S' \cap \neg S_b)$
$= \{$ド・モルガンの法則と分配法則と $X \cap \neg X = \emptyset\}$
$S' \cup (S_b \cap \neg S_b)$
$= \{X \cap \neg X = \emptyset\}$
S' □

A.2 $put_{T.S_b}$ の導出

スキーマ進化を定める F と逆方向の更新伝播を定める F' から，$put_{T.S_b}$ を導出する．

(1) 更新後の S'_b は $S'_b = (S_b \cap \neg\Delta^-_{S_b}) \cup (\Delta^+_{S_b})$ であるため，次が成り立つ．

$$S'_b = put_{T.S_b}(S_b, T') = (S_b \cap \neg\Delta^-_{S_b}) \cup (\Delta^+_{S_b}) \quad (A.1)$$

(2) 逆方向の更新伝播は式 (12) の $\Delta S = F'(S, \Delta T)$ により定められ，単調であることから $\Delta^+_S = F'(S, \Delta^+_T)$，$\Delta^-_S = F'(S, \Delta^-_T)$ である．ソーススキーマに対する双方向変換から S と S_b の関係が恒等写像であるため，S を S_b に置き換えた次が成り立つ．

$$\Delta^+_{S_b} = F'(S_b, \Delta^+_T) \quad (A.2)$$
$$\Delta^-_{S_b} = F'(S_b, \Delta^-_T) \quad (A.3)$$

(3) Δ^+_T と Δ^-_T は，更新前のビュー T と更新後のビュー T' から，次に定められる．

$$\Delta^+_T = T' \cap \neg T \quad (A.4)$$
$$\Delta^-_T = \neg T' \cap T \quad (A.5)$$

(4) 更新前のビュー T は get_T により得られるため，式 (33) と (34) より次に定められる．

$$T = get_T(S_b, A) = get_{T.S_b}(S_b) \cup A = F(S_b) \cup A \quad (A.6)$$

(5) (1)–(4) から，スキーマ進化を定める F と逆方向の更新伝播を定める F' により，$put_{T.S_b}(S_b, T')$ を次に定める．

$$S'_b = put_{T.S_b}(S_b, T') = (S_b \cap \neg(F'(S_b, \neg T' \cap (F(S_b) \cup A)))) \cup F'(S_b, T' \cap \neg(F(S_b) \cup A)) \quad (A.7)$$

A.3 $put_{T.A}$ の導出

$put_{T.A}$ は，$get_{T.S_b}$ と $put_{T.S_b}$ が PutGet を満たさない場合に，そのようなターゲットスキーマのビュー T への挿入を補助リレーション A に挿入する．このような $put_{T.A}$ を次のように導出する．

(1) 更新後の補助リレーション A' は $A' = (A \cap \neg\Delta^-_A) \cup (\Delta^+_A)$ であるため，次が成り立つ．

$$A' = put_{T.A}((S_b, A), T') = (A \cap \neg\Delta^-_A) \cup \Delta^+_A \quad (A.8)$$

(2) PutGet の結果 T'' は，次に定められる．

$$T'' = get_{T.S_b}(put_{T.S_b}(S_b, T')) \quad (A.9)$$

(3) PutGet を満たさず更新されたビュー T' に対して不足するタプルの集合となる $T' \cap \neg T''$ が補助リレーション A に存在しない場合，それを A へ挿入するタプルの集合 Δ^+_A とする．これを次に定める．

$$\Delta^+_A = T' \cap \neg T'' \cap \neg A \quad (A.10)$$

(4) 更新されたビュー T' には存在せず，補助リレーション A に存在するタプルの集合がある場合，それを A から削除するタプルの集合 Δ^-_A とする．れを次に定める．

$$\Delta^-_A = \neg T' \cap A \quad (A.11)$$

(5) (1)–(4) から，$put_{T.A}((S_b, A)T')$ を次に定める．

$$A' = put_{T.A}((S_b, A), T') = (A \cap \neg(\neg T' \cap A)) \cup (T' \cap \neg(get_{T.S_b}(put_{T.S_b}(S_b, T'))) \cap \neg A) \quad (A.12)$$

A.4 命題 4 の証明

証明. GetPut として，get_T を定める式 (33) と put_T を定める式 (35) より，次が成り立つことを証明する．

$$put_T((S_b, A), get_T(S_b, A)) = (put_{T.S_b}(S_b, get_{T.S_b}(S_b) \cup A), put_{T.A}((S_b, A), get_{T.S_b}(S_b) \cup A)) = (S_b, A) \quad (A.13)$$

$put_{T.S_b}(S_b, get_{T.S_b}(S_b) \cup A) = S_b$ となることを示す．

$put_{T.S_b}(S_b, get_{T.S_b}(S_b) \cup A)$
$= \{$式 (A.7) と式 (34) を代入$\}$
$(S_b \cap \neg(F'(S_b, (\neg(F(S_b) \cup A) \cap (F(S_b) \cup A)))))$
$\cup F'(S_b, (F(S_b) \cup A) \cap \neg(F(S_b) \cup A))$
$= \{$式 (33) と (34) より $F(S_b) \cup A = T\}$
$(S_b \cap \neg(F'(S_b, (\neg T \cap T)))) \cup F'(S_b, T \cap \neg T)$
$= \{X \cap \neg X = \emptyset\}$

$(S_b \cap \neg(F'(S_b,\emptyset))) \cup F'(S_b,\emptyset)$
$= \{F'$は単調であるため $F'(S_b,\emptyset)=\emptyset\}$
S_b

$put_{T.A}((S_b,A), get_{T.S_b}(S_b)\cup A) = A$ となることを示す．

$put_{T.A}((S_b,A), get_{T.S_b}(S_b)\cup A)$
$= \{$式 (A.12) と (A.7) を代入$\}$
$(A\cap\neg(\neg(get_{T.S_b}(S_b)\cup A)\cap A))\cup((get_{T.S_b}(S_b)\cup A)$
$\cap\neg(get_{T.S_b}(put_{T.S_b}(S_b, get_{T.S_b}(S_b)\cup A)))\cap\neg A)$
$= \{put_{T.S_b}(S_b, get_{T.S_b}(S_b)\cup A) = S_b\}$
$(A\cap\neg(\neg(get_{T.S_b}(S_b)\cup A)\cap A))\cup((get_{T.S_b}(S_b)\cup A)$
$\cap\neg(get_{T.S_b}(S_b)\cap\neg A)$
$= \{$ド・モルガンの法則と分配法則と $X\cap\neg X=\emptyset\}$
$(A\cap(get_{T.S_b}(S_b)\cup A))\cup(A\cap\neg A)$
$= \{X\cap(Y\cup X)=X$ と $X\cap\neg X=\emptyset\}$
A

よって (A.13) が成り立つ．

次に PutGet を証明する．

$get_T(put_T((S_b,A),T'))$
$= \{$式 (33) と式 (35) を代入$\}$
$get_{T.S_b}(put_{T.S_b}(S_b,T'))\cup put_{T.A}((S_b,A),T')$
$= \{$式 (A.9) から $get_{T.S_b}(put_{T.S_b}(S_b,T'))=T''\}$
$T''\cup put_{T.A}((S_b,A),T')$
$= \{$式 (A.12) を代入$\}$
$T''\cup((A\cap\neg(\neg T'\cap A))\cup(T'\cap\neg T''\cap\neg A))$
$= \{$ド・モルガンの法則と分配法則と $X\cap\neg X=\emptyset\}$
$T''\cup T'$
$= \{$スキーマの共存戦略の一貫性が成り立つため
$T''\subset T'\}$
T' □

A.5 put_T^{uns} の導出

put_T^{uns} は，ターゲットスキーマのビュー T に対する更新をソーススキーマと共有しない場合に，補助リレーション A_{ins} と A_{del} を更新する．このような put_T^{uns} を次のように導出する．

(1) 更新されたビュー T' に存在し，スキーマ進化の結果の $F(S_b)$ にも補助リレーション A_{ins} にも存在しないタプルは，ビュー T に新たに挿入されたタプルである．その集合を $\Delta^+_{A_{ins}}$ とし，次に定める．

$$\Delta^+_{A_{ins}} = T'\cap\neg F(S_b)\cap\neg A_{ins} \tag{A.14}$$

(2) 更新されたビュー T' に存在せず A_{ins} に存在するタプルは，ビュー T に挿入された後に，削除されたタプルである．その集合を $\Delta^-_{A_{ins}}$ とし，次に定める．

$$\Delta^-_{A_{ins}} = \neg T'\cap A_{ins} \tag{A.15}$$

(3) 更新されたビュー T' に存在せず，スキーマ進化の結果の $F(S_b)$ に存在し補助リレーション A_{del} には存在しないタプルは，$F(S_b)$ により得られるタプルの集合から新たに削除されたタプルである．その集合を $\Delta^+_{A_{del}}$ とし，次に定める．

$$\Delta^+_{A_{del}} = \neg T'\cap F(S_b)\cap\neg A_{del} \tag{A.16}$$

(4) 更新されたビュー T' に存在し A_{del} にも存在するタプルは，$F(S_b)$ により得られるタプルの集合から削除した後に，改めて挿入されたタプルである．その集合を $\Delta^-_{A_{del}}$ とし，次に定める．

$$\Delta^-_{A^{del}} = T'\cap A_{del} \tag{A.17}$$

A.6 補題 5 の証明

証明. ビュー T に更新がある場合に 1 つの更新を 1 ステップとし，帰納法により証明する．ステップ n におけるビュー T を $T(n)$，それにより更新された補助リレーションを $(A_{ins}(n), A_{del}(n))$ とする．

(基礎) ステップ 1 を $(A_{ins}(1), A_{del}(1)) = (\emptyset,\emptyset)$ とする．$A_{ins}(1)=\emptyset$ と $A_{del}(1)=\emptyset$ が互いに素であることは明らか．

(帰納) 帰納法の仮定として，$A_{ins}(n-1)$ と $A_{del}(n-1)$ が互いに素であり，$A_{ins}(n-1)\cap A_{del}(n-1)=\emptyset$ が成り立つとする．$A_{ins}(n)\cap A_{del}(n)=\emptyset$ となることを次に示す．

$A_{ins}(n)\cap A_{del}(n)$
$= \{$式 (54) により $A_{ins}(n-1), A_{del}(n-1)$ を更新$\}$
$((A_{ins}(n-1)\cap\neg\Delta^-_{A_{ins}})\cup\Delta^+_{A_{ins}})\cap$
$((A_{del}(n-1)\cap\neg\Delta^-_{A_{del}})\cup\Delta^+_{A_{del}})$
$= \{$式 (A.14)–(A.17) を代入$\}$
$((A_{ins}(n-1)\cap\neg(\neg T(n)\cap A_{ins}(n-1)))$
$\cup(T(n)\cap\neg F(Sb)\cap\neg A_{ins}(n-1)))$
$\cap((A_{del}(n-1)\cap\neg(T(n)\cap A_{del}(n-1)))$
$\cup(\neg T(n)\cap F(Sb)\cap\neg A_{del}(n-1)))$
$= \{$集合の演算による式変形と $A\cap\neg A=\emptyset$ を適用$\}$
$\emptyset$

よって A_{ins} と A_{del} は互いに素． □

A.7　命題 6 の証明

証明.　get_T^{uns} を定める式 (53) と put_T^{uns} を定める式 (54)–(58) より，GetPut として次が成り立つことを証明する．

$$
\begin{aligned}
&put_T^{uns}((S_b,(A_{ins},A_{del})),get_T^{uns}(S_b,(A_{ins},A_{del})))\\
&=(S_b,(A_{ins},A_{del}))
\end{aligned}
$$

put_T^{uns} を定める式 (54) より，$A_{ins}=(A_{ins}\cup\neg\Delta^-_{A_{ins}})\cap\Delta^+_{A_{ins}}$ となることを示す．

$$
\begin{aligned}
&(A_{ins}\cap\neg\Delta^-_{A_{ins}})\cap\Delta^+_{A_{ins}}\\
&=\{\text{式 (55)–(56) と式 (53) を代入}\}\\
&\quad(A_{ins}\cap\neg(\neg((F(S_b)\cap\neg A_{del})\cup A_{ins})\cap A_{ins}))\\
&\quad\cup(((F(S_b)\cap\neg A_{del})\cup A_{ins})\cap\neg F(S_b)\cap\neg A_{ins})\\
&=\{\text{集合の演算による式変形}\}\\
&\quad(A_{ins}\cap F(S_b)\cap\neg A_{del})\cup A_{ins}\\
&=\{X\cup(X\cap Y)=X\}\\
&\quad A_{ins}
\end{aligned}
$$

put_T^{uns} を定める式 (54) より，$A_{del}=(A_{del}\cup\neg\Delta^-_{A_{del}})\cap\Delta^+_{A_{del}}$ となることを示す．

$$
\begin{aligned}
&(A_{del}\cup\neg\Delta^-_{A_{del}})\cap\Delta^+_{A_{del}}\\
&=\{\text{式 (57)–(58) と式 (53) を代入}\}\\
&\quad(A_{del}\cap\neg(((F(S_b)\cap\neg A_{del})\cup A_{ins})\cap A_{del}))\\
&\quad\cup(\neg((F(S_b)\cap\neg A_{del})\cup A_{ins})\cap F(S_b)\cap\neg A_{del})\\
&=\{\text{集合の演算による式変形}\}\\
&\quad(A_{del}\cap\neg F(S_b)\cap\neg A_{ins})\cup(A_{del}\cap\neg A_{ins})\\
&=\{\text{補題 5 より } A_{ins}\cap A_{del}=\emptyset\}\\
&\quad A_{del}
\end{aligned}
$$

よって GetPut が成り立つ．

PutGet として次が成り立つことを証明する．

$$
\begin{aligned}
&get_T^{uns}(put_T^{uns}((S_b,(A_{ins},A_{del})),T'))=T'\\
&get_T^{uns}(put_T^{uns}((S_b,(A_{ins},A_{del})),T'))\\
&=\{\text{式 (54)–(58) と式 (53) を代入}\}\\
&\quad(F(S_b)\cap\neg((A_{del}\cap\neg(T'\cap A_{del}))\cup\\
&\quad(\neg T'\cap F(S_b)\cap\neg A_{del}))))\\
&\quad\cup((A_{ins}\cap\neg(\neg T'\cap A_{ins}))\cup(T'\cap\neg F(S_b)\cap\neg A_{ins}))\\
&=\{\text{集合の演算による式変形}\}\\
&\quad(F(S_b)\cap\neg A_{del}\cap T')\cup(F(S_b)\cap T'\cap A_{del})\\
&\quad\cup(A_{ins}\cap T')\cup(T'\cap\neg F(S_b)\cap\neg A_{ins})\\
&=\{\text{分配法則}\}\\
&\quad T'\cap(F(S_b)\cup A_{ins}\cup(\neg F(S_b)\cap\neg A_{ins}))\\
&=\{\text{ド・モルガンの法則}\}\\
&\quad T'\cap((F(S_b)\cup A_{ins})\cup\neg(F(S_b)\cup A_{ins}))\\
&=\{X\cap(Y\cup\neg Y)=X\}\\
&\quad T'
\end{aligned}
$$

よって PutGet が成り立つ．　□

田中 順平

1974 年生．1998 年電気通信大学電気通信学部機械制御工学科卒業．2000 年東京工業大学大学院総合理工学研究科知能システム科学博士前期課程修了．2002 年同大学院総合理工学研究科知能システム科学博士後期課程退学．同年野村證券株式会社入社．2018 年総合研究大学院大学複合科学研究科情報学専攻博士課程入学．

バン ダン トラン

1994 年生．2017 年 School of Information and Communication Technology, Hanoi University of Science and Technology (Vietnam) 卒業．同年総合研究大学院大学複合科学研究科情報学専攻博士課程入学．

加藤 弘之

1965 年生．1990 年東京理科大学理学部物理学科卒業．同年日本ユニシス株式会社入社．1999 年奈良先端科学技術大学院大学情報科学研究科博士後期課程修了．同年学術情報センター助手．2001 年国立情報学研究所アーキテクチャ科学研究系助教．博士（工学）．日本ソフトウェア科学会，ACM 各会員．

胡 振江　（正会員）

1966 年生．1988 年中国上海交通大学計算機科学系を卒業．1996 年東京大学大学院工学系研究科情報工学専攻博士課程修了．同年日本学術振興会特別研究員を経て，1997 年東京大学大学院工学系研究科情報工学専攻助手，同年 10 月同専攻講師，2000 年同専攻准教授，2008 年国立情報学研究所教授，2019 年東京大学情報理工学研究科教授（併任），2019 年より北京大学計算機科学技術系専攻長・教授，国立情報学研究所特任教授．博士（工学）．日本工学会フェロー，日本工学アカデミー会員，IEEE フェロー，ACM Distinguished Scientist，欧州科学アカデミー会員．