DEIM Forum 2022 J23-5

世代管理されたディメンション表を対象とする結合処理の効率的実行

高田　実佳† 　合田　和生† 　喜連川　優†

† 東京大学　〒 153–8503 東京都目黒区駒場 4-6-1
E-mail: †{mtakata,kgoda,kitsure}@tkl.iis.u-tokyo.ac.jp

あらまし　様々なドメインの業務でビッグデータを蓄積し分析する事により業務の改善・新たな知見発見への期待が高まっている．そうした分析対象データには，刻一刻と増加するファクト表と，複数のディメンション表が含まれ，分析者はファクト表のデータを正しく理解する為に，適切なディメンション表を参照する必要がある．その為，分析者はファクト表と適切なディメンション表を結合処理した結果を必要とするが，データの運用機関が長くなるにつれ，ディメンション表が更新を繰り返しその世代が増えると，ファクト表との結合処理が複雑になりその処理時間が増大する．本論文では，そうした世代管理されたディメンション表とファクト表の結合における効率的な実行処理を提案し評価結果を示す．

キーワード　データベース，ディメンション表，世代管理，結合処理

1　はじめに

多くの企業や組織では，日々様々なデータが生成され，そうした業務データの管理には，複数世代のディメンション表とファクト表を用いたスタースキーマが利用されている [1,2]．ファクト表には，商品の購買記録の様な刻一刻と変化する業務プロセスを記録され，ディメンション表には商品の名前や値段といった業務の属性情報が登録されている．業務の改善や新しい知見の発見の為に分析が頻繁に行われており，分析クエリを実行する際には，この様なディメンション表とファクト表との結合が必要になる．

しかし，ある商品の値段が季節や年によって変化することがある様に，ディメンション表の内容は頻繁ではないが時折更新される．その為，刻一刻と記録される業務プロセスを正しく理解する為には，ディメンション表が世代管理されている必要があり，ファクト表のレコードに対して適切な世代のディメンション表を参照することが必要になる．こうした世代管理されたディメンション表とファクト表との結合を従来のデータベースを用いても実現はできる．１つの方式は，各世代のディメンション表とファクト表のバイナリ結合した後ユニオンし，最後に有効なレコード以外を排除するという方式である．また別の方式は，複数世代のディメンション表をユニオンし，そのユニオンしたディメンション表とファクト表を結合し，最後に有効なレコード以外を排除するという方式である．ただし，このような従来方式では不要な計算処理を含み，非効率である．

本論文では，結合条件に結合キーと世代の適合性を考慮することで，世代管理されたディメンション表とファクト表の結合を効率的に実行する μ-join オペレータを提案する．μ-join は，データベースエンジンに初期段階で結合条件を判断させることで，ディメンション表が複数世代存在することによる不要な計算処理を低減する．著者らは，本提案オペレータが従来のデータベースによる処理方式よりも優位であることを示す為，合成データとリアルな医療データを用いて実験を行った．

本論文の構成は，以下の通りである．第 2 章では，ディメンション表の世代管理とその課題について言及する．第 3 章では，μ-join について説明し，第 4 章では，その優位性を示す実験結果を示す．第 5 章では，関連文献について言及し，第 6 章では，今後に向けた課題と将来展望について纏める．

2　世代管理されたディメンション表を対象とする結合処理

本章では医療分野において複数世代のディメンション表が管理される事例を述べる．

日本では国民皆保険制度が適用され，全国民が基本的には何らかの公的保険に加入することが義務付けられている．医療保険が適用される医療サービスは，医療機関毎に毎月，患者事に医療従事者の施した医療サービスの記録が記録され，その医療費の請求が保険事業者に申請される [3]．その際の請求内容を適正を審査する為に審査支払い機関に送られ，審査後に保険者は保険者負担分の支払い分を医療機関に支払う．その際の請求書を医療レセプトといい，請求が行われる月毎に追加され，日々の医療記録はファクト表として記録される．この様な医療レセプトを把握する為には，傷病名マスタ，医科診療行為マスタ，歯科診療行為マスタ，調剤行為マスタ，医薬品マスタ，特定機材マスタ，コメントマスタ，修飾語マスタ，歯式マスタの９種類があり [4]，医療業務の属性情報が登録されている [6–8]．これらはディメンション表として管理され，内容に更新がある毎にその世代が更新されていく．図 1 に，3 世代の傷病名マスタと医療レセプトの例を示す [5]．R_1 が 2020 年以前，R_2 が 2021 年 1 月〜6 月，R_3 が 2021 年 6 月以降，に有効な 3 世代の傷病名マスタ．S が医療レセプトの例である．R_1,R_2,R_3 は傷病コードと傷病名が記載されており，世代毎に傷病コードが変更，廃止，あるいは追加といった更新が起こる．例えば，傷病名が偽斜視の傷病コードは 2020 年には 8831256 だったが，2021 年以

図 1: 3 世代のディメンション表とファクト表の例

降は 3789010 に変更されている．医療レセプトには患者毎に請求年月毎の医療サービスが各レコードに記録されており，医療サービスの内容を正しく把握する為には，各レコードに対して有効な傷病名マスタを参照する必要がある．また，こうしたディメンション表の更新は，傷病名マスタだけではなく，診療行為マスタや調剤行為マスタなど他のマスタにも当てはまる．この様に，データベースが医療レセプトの記録を保持する為には，全ての世代のディメンション表を管理し，ファクト表の各レコードに対して有効なディメンション表と結合することが必要になる．

ディメンション表の世代を管理することで，どの世代のディメンション表がどのファクト表のレコードと結合すべきかを正しく認識することができる．しかし，多くの既存のデータベースでは，ディメンション表の世代管理を明示的にはサポートしておらず，関係データベースのスキーマを用いることで実現している．こうした世代管理の方法は，ストレージのバックアップ方式 [9] と類似した形式で，完全世代管理，差分世代管理，増分世代管理，といった 3 方法に分類される．その例を図 2 に示す．完全世代管理は各世代を 1 表ずつ管理する方法である．差分世代管理は第 1 世代との差分のみを 1 表として管理する方法である．そして，増分世代管理は 1 つ前の世代からの差分を 1 表に管理する方法である．各方法には長所・短所があり，完全世代管理は比較的単純だが，その分容量を多く要しファクト表との結合においては冗長な処理が多くなる可能性が高い．一方，増分世代管理は容量を抑えることができるが，結合処理が複雑になり特定のファクト表とのレコードに対して複数世代を参照する処理が必要となる．

多くのビジネス業務においてその業務改善・新たな知見発見のため，業務データの分析が行われる．その様な業務データの分析において，世代管理されたディメンション表とファクト表の結合を実行される必要がある．この様な結合処理において，ファクト表の各レコードに対し適切な世代のディメンション表が結合されることを保証しなければいけない．既存のデータベースでの 1 つの実現方法に，シンプルにファクト表の各レコードに対して各世代のディメンション表を既存のバイナリ結合オペレータを用いて結合処理を行い，ユニオンし，最後に有効ではない世代との結合レコードを排除するという方法がある．また別の方法は，全世代のディメンション表をユニオンし，その表とファクト表を既存のバイナリ結合オペレータを用いて結合処理を行い，最後に有効ではない世代とファクト表との結合でコードを排除するという方法がある．ただし，両方法とも不要な計算処理を行い，非効率な結合処理といえる．

R_1

Partkey	Retailprice
1	100
2	200
…	…
100	1000

R_2

Partkey	Retailprice
1	100
2	2000
…	…
100	1000

R_3

Partkey	Retailprice
1	1003
2	2000
…	…
100	10003

(a) 完全世代管理

R_1

Partkey	Retailprice
1	100
2	200
…	…
100	1000

R_2

Partkey	Retailprice
2	2000

R_3

Partkey	Retailprice
1	1003
2	2000
100	10003

(b) 差分世代管理

R_1

Partkey	Retailprice
1	100
2	200
…	…
100	1000

R_2

Partkey	Retailprice
2	2000

R_3

Partkey	Retailprice
1	1003
100	10003

(c) 増分世代管理

図 2: ディメンション表の管理方法の例

3 μ-join

本章では，第 2 章で述べた世代管理されたディメンション表とファクト表との結合処理における効率的な実行オペレータ：μjoin を提案する．

3.1 定　　義

まず，世代管理されたディメンション表とファクト表の効率的な結合処理を定義する．

定義 1 (世代管理)
$\check{R}$ を，<$\check{R}$> 世代数の世代管理された表とする．k 番目の世代 $\check{R}_k$ は，

$$\check{R}_k := \{r | r \in \check{R} \wedge e_k(r)\},$$

$e_k(r)$ が true の場合，レコード r は第 k $(0 \leqq k < $<$\check{R}$>$)$ 世代で有効である．また，明らかに

$$\check{R} = \bigcup_{0 \leqq k < <\check{R}>} \check{R}_k.$$

である．

定義 2 (完全世代管理)
図 2(a) に示す様に完全世代管理では，各世代の表 $\check{R}_k$ が 1 表として記録されており， $R_k^{(s)}$ は，

$$R_k^{(s)} \leftarrow \check{R}_k.$$

となる．また，明らかに，ある世代 $\check{R}_k$ は表 $R_k^{(s)}$ から，

$$\check{R}_k \leftarrow R_k^{(s)}.$$

として得ることができる．

定義 3 (差分世代管理)
図 2(b) に示す様に差分世代管理では，初期世代の $\check{R}_0$ 表 からの差分を 1 表の $R_0^{(d)}$ として各世代を管理する．

$$R_0^{(d)} \leftarrow \check{R}_0, \qquad R_k^{(d)} \leftarrow \check{R}_k \ominus \check{R}_0 (k > 0),$$

$\ominus$ は右辺から左辺を引いた差分集合を記す [1]．そして，明らかに，ある世代 $\check{R}_k$ は，$R_0^{(d)}$ と $R_k^{(d)}$ を用いて

$$\check{R}_0 \leftarrow R_0^{(d)}, \qquad \check{R}_k \leftarrow R_0^{(d)} \oplus R_k^{(d)} (k > 0),$$

と表すことができる．$\oplus$ は，右辺の表を左辺の表に追加するオペレーションとする．

定義 4 (増分世代管理)
図 2(c) に示す様に差分世代管理では，初期世代の $\check{R}_0$ は $R_0^{(i)}$ で表され，前の世代の表からの増分を 1 表の $\check{R}_k$ とし，

$$R_0^{(i)} \leftarrow \check{R}_0, \qquad R_k^{(i)} \leftarrow \check{R}_k \ominus \check{R}_{k-1} (k > 0).$$

と表す．また，明らかに，ある世代 $\check{R}_k$ は，$k+1$ 個の $R_0^{(i)}$ から $R_k^{(i)}$ を用いて，

$$\check{R}_0 \leftarrow R_0^{(i)}, \qquad \check{R}_k \leftarrow R_0^{(i)} \oplus \cdots \oplus R_k^{(i)} (k > 0).$$

と表せられる．

次に，ファクト表 S と世代管理されたディメンション表 $\check{R}$ を用いて，世代考慮型ファクト-ディメンション表結合オペレータを定義する．

定義 5 (世代考慮型ファクト-ディメンション表結合オペレータ)
ファクト表 S と世代管理されたディメンション表 $\check{R}$ との結合処理 $S \bowtie_{v,Y=X} \check{R}$ は

$$S \bowtie_{v,Y=X} \check{R} := \bigcup_{0 \leqq k < <\check{R}>} \{s \cup r | s \in S \wedge r \in \check{R}_k \wedge s_Y = r_X \wedge v(s) = k\},$$

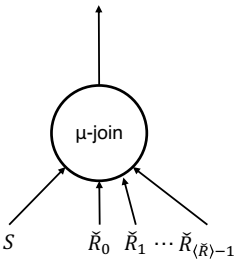
と表す．Y は S の外部キー，X は $\check{R}$ の主キー，$v(s)$ はファクト表 S のあるレコード s が参照すべき世代のディメンション表の世代番号を返す．$Y = X$ は，結合のキー条件， $v(s) = k$ は世代の適合条件を表す．

3.2 実　　装

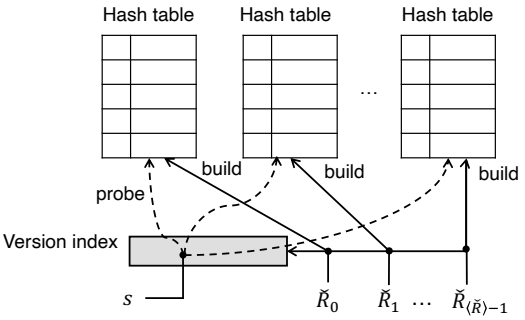
第 3.1 章で述べた世代考慮型ファクト-ディメンション表結合オペレータは，従来データベースに既存するバイナル結合オペレータとユニオンオペレータ，選択オペレータを用いることによっても表す事ができ，計算可能である．しかし，この様な従来のオペレータを用いた複数世代のディメンション表とファクト表との結合処理には非効率な計算処理が発生する．

本論文では，世代管理されたディメンション表とファクト表を直接扱う μ-join というオペレータを提案する．図 3(a) に示す様に，入力に，ファクト表と複数世代のディメンション表として，定義 5 に示した様に結合キーと世代の適合性を考慮した

1：広くデータベースのログに採用されている技術同様，右辺の表から左辺の表の差分となるレコードとは，当該レコードに削除フラグを記録する [13].



(a) μ-join オペレータ



(b) ハッシュ結合ベースの μ-join

図 3: The μ-join

結合条件を満たす結合処理を行い出力する．μ-join はシンプルに，データベースエンジンに結合キーの条件 $s_Y = r_X$ と世代の適合性 $v(s) = k$ を早期に判断するように実装する事で，バイナリ結合による冗長なレコード処理を削減することができる．

さらに，データベースエンジンにバージョンインデックスといったデータ構造を持たせることにより，ファクト表の各レコードに対し適切に世代のディメンション表であるターゲット表を特定する事ができる．このデータ構造によって，差分世代管理や増分世代管理においてもターゲット表を高速に特定することができ，結合処理における複雑さを低減させることができる．例えば，あるファクト表のレコード s が明らかにターゲット表の世代番号が k と特定できれば，レコード s は $\check{R}_k$ と結合できる．

図 3(b) にハッシュ結合ベースの μ-join の処理の様子を示す．従来のバイナリハッシュ結合と同様に，最初に対象となるターゲット表からハッシュ表が生成し，それと同時に，バージョンインデックスもターゲット表に対し生成する．これによって，複数世代のディメンション表の中からファクト表の各レコード s に対して適合する世代番号を高速に特定することができる．このような新たなデータ構造を生成することでオーバーヘッドが懸念されるが，ディメンション表は多くの場合ファクト表に比べかなり小さく，バージョンインデックスの生成はハッシュ表の生成と処理を同時に実行することでオーバーヘッドが生じる可能性は少ないと考える．

4 実　　験

4.1 実験環境

合成データによる実験は広く実施されているが，複数世代のディメンション表とファクト表の結合に関しては著者らの調べた限りほぼ類似研究がなく，適切な評価用の合成データが見つからなかった為，本論文では，TPC-H のデータ生成器 (dbgen) を用いて複数世代のディメンション表を生成し実験を行った．

TPC-H のスキーマは主に 2 つのファクト表 (ORDERS と LINEITEM) と 4 つのディメンション表 (PART，PARTSUPP，SUPPLIER，CUSTOMER) から構成されている．著者らが改変したデータ生成器は，これらのディメンション表を複数世代作成し，各世代のディメンション表が主キー以外の属性情報を更新する様に作成した．例えば，PART 表では，P_RETAILPRICE の値が前世代からランダムに ±5% 変換する様に作成し，次の 3 つのパラメータを変化させたデータを生成した．

- n_v ($n_v \geqq 1$) ：各ディメンション表の世代数．特に指定ない場合，$n_v = 12$ とした．初期世代はオリジナルの TPCH-H 仕様の値とした．T
- u ($0.0 \leqq u \leqq 1.0$)： ディメンション表の各世代間のレコードの変化率．特に指定がない場合，$u = 0.05$ とした．
- s ($0.5 \leqq s$<1.0) ：ディメンション表の各世代の更新するレコードの主キーの偏り．即ち，更新されるレコードの s がランダムに $1 - s$ の主キーの中から選択される事を意味しており，逆も然りである．

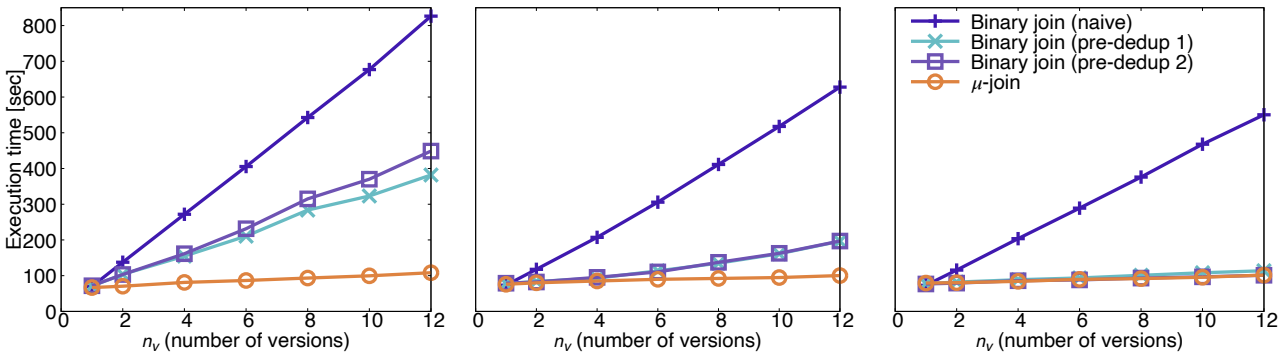
また，改変したデータ生成器では各世代情報をディメンション表の属性に追加し，各ファクト表のレコードがどの世代のディメンション表と適合するはずか判明できるようにした．これにより，ファクト表が与えられた時，ファクト表の日付情報（例えば O_ORDERDATE 等）に応じて，その時に有効な世代のディメンション表を，n_v 世代数分のディメンション表の中から特定できる様にした．

上記の様に生成した合成データを用いて，次のような 3 クエリを用いてファクト表と世代管理されたディメンション表との結合処理を行なった．

- クエリ A (1 ファクト表と 1 ディメンション表) ： LINEITEM と複数世代の PART 表の結合．
- クエリ B (2 ファクト表と 1 ディメンション表) ： LINEITEM と ORDERS と複数世代の PART 表の結合．
- クエリ C (1 ファクト表と 2 ディメンション表) ： LINEITEM と複数世代の PART 表と PARTSUPP の結合．

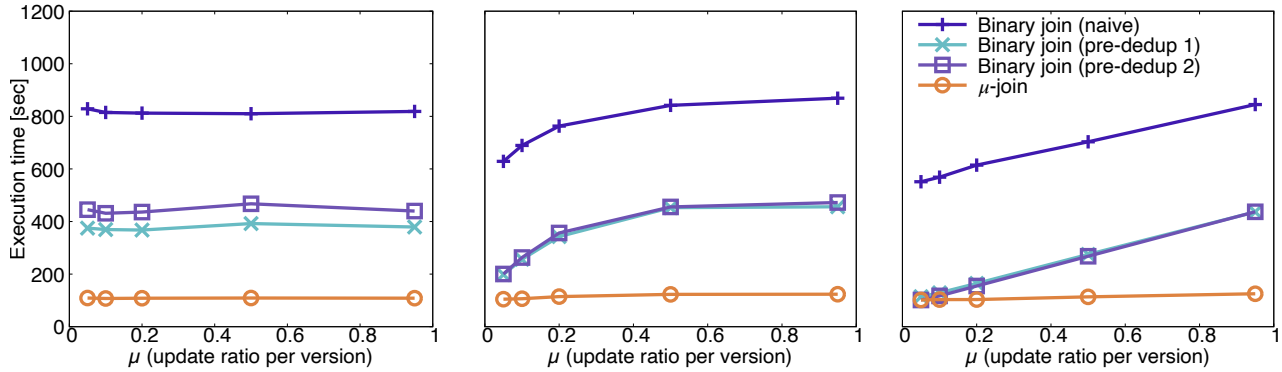
上記のようなデータとクエリを用いて，著者らは従来のバイナリ結合オペレータを用いた方式と，提案する μ-join オペレータを比較した．そのリストを以下に記す．

- **Binary join (naive)** ：従来のバイナリ結合を用いてファクト表と各世代のディメンション表をバイナリ結合し，各結合結果をユニオンし，最後に世代非適合が生じているレコードを排除する．
- **Binary join (pre-dedup 1)**： 全世代のディメンション表を重複排除なくユニオンし，その表とファクト表をバイナリ結合し，最後に世代非適合が生じていたり冗長なレコードを排除する．
- **Binary join (pre-dedup 2)**： 全世代のディメンション表を重複排除なくユニオンし，その表とファクト表をバイナ



(a) 完全世代管理　(b) 差分世代管理　(c) 増分世代管理

図 4: 異なるディメンション表の世代数における評価



(a) 完全世代管理　(b) 差分世代管理　(c) 増分世代管理

図 5: 異なるディメンション表の各世代間のレコードの変化率における評価

リ結合し，できるだけ重複レコードを排除する．最後に，世代非適合が生じていたり冗長なレコードを排除する．

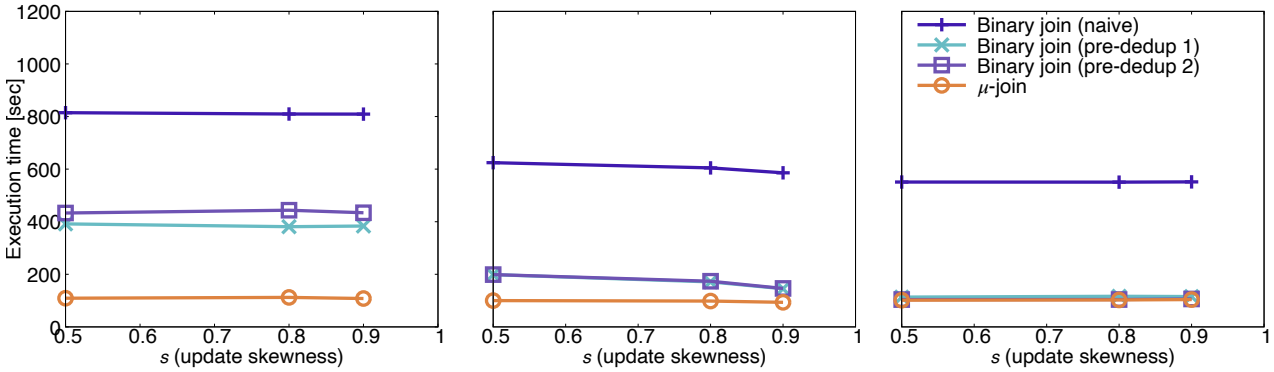
- **μ-join**：μ-join オペレータによって，ファクト表と複数世代のディメンション表との結合を直接処理し，不要な結合処理を必要としない．

著者らはこれらの実験をコード実装し，結合処理の時間を計測し，各評価項目につき，5 回計測した平均値を用いた．また，本実験では単純化の為，選択処理は行わず，バイナリ結合および μ-join オペレータにおいてハッシュ結合をベースとして利用した．全クエリをシングルスレッドで実行し，全ての実験は 56CPU コア，96GB メモリ，31TB RAID ストレージの環境下で行なった．

4.2　合成データを用いた実験結果

まず，クエリ A (1 ファクト表と 1 ディメンション表) を用いて，異なるディメンション表の世代数におけるファクト表と複数のディメンション表との結合処理を評価した．図 4(a) に示す様に，完全世代管理では，μ-join がファクト表と 2 世代のディメンション表との結合において従来方式の最高値よりも 31.7% 高速であることが分かった．さらに世代数が増えるにつれ優位性が高まり，12 世代数の場合では 71.7% もの高速化が確認できた．図 4(b) に示す様に，差分世代管理では，μ-join が世代数 2 のディメンション表との結合において 2.89%，世代数 12 の結合において 48.9% 従来方式の最高値よりも高速である事が確認できた．図 4(c) に示す様に，増分世代管理では，μ-join は従来方式と同等の性能を確認した．本結果より μ-join が従来のバイナリ結合に比べ，結合条件に結合キー条件と世代の適合性を早期に考慮することで不要な処理を削減でき，大幅に高速化が達成されることを確認した．

次に，クエリ A を用いて異なるディメンション表の各世代間のレコードの変化率における評価を行なった．図 5(a) に示す様に，μ-join は一貫してディメンション表の各世代間のレコードの変化率が 0.95 時に従来方式の最高値より 71.4% 高速化を達成する事が分かった．これは，完全世代管理では，各世代のディメンション表のレコード数はディメンション表の各世代間のレコードの変化率に依らず一定の為であるといえる．図 5(b) に示す様に，μ-join が，各世代間のレコードの変化率が 0.05 時には 47.2% 従来方式の最高値より高速である事が確認できた．その優位性は，各世代間のレコードの変化率が大きくなるにつれて大きくなり，0.95 時には 72.9% 高速化を達成した．これは各世代間のレコードの変化率が大きくなるにつれて従来方式における冗長な処理が増えるのに対して，μ-join は増えない為である．図 5(c) に示す様に，μ-join は各世代間のレコードの



(a) 完全世代管理　(b) 差分世代管理　(c) 増分世代管理

図 6: 異なるディメンション表の各世代の更新するレコードの主キーの偏りにおける評価

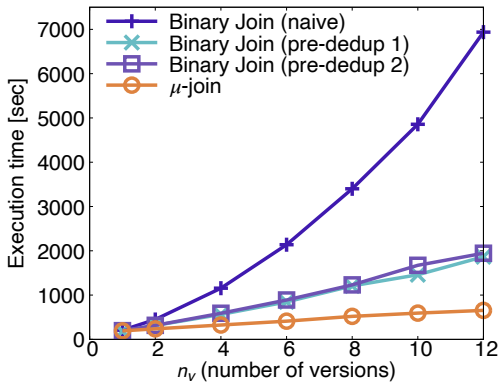
変化率が 0.05 時には同等の性能に対し，各世代間のレコードの変化率 0.95 時には 71.3% 高速化を確認した．

さらに，クエリ A を用いて異なるディメンション表の各世代の更新するレコードの主キーの偏りにおける評価を行なった．図 6(a) に示す様に，μ-join は，完全世代管理においてディメンション表の各世代の更新するレコードの主キーの偏りが大きい場合に 71.9% の高速化を確認した．これは，完全世代管理では各世代のディメンション表のサイズが一定であり，μ-join の優位性が一貫していえる為である．図 6(b) に示す様に，差分世代管理では，ディメンション表の各世代の更新するレコードの主キーの偏りが大きい場合には μ-join の優位性が低減する事が分かった．ディメンション表の各世代の更新するレコードの主キーの偏りが大きくなるにつれ，差分世代管理の各世代の 1 表のサイズは小さくなり，それによって従来方式の冗長な処理コストが軽減するためである．図 6(c) に示す様に，増分世代管理ではディメンション表の各世代の更新するレコードの主キーの偏りが変化しても各世代毎ディメンション表のサイズには影響しない為，μ-join は従来方式とほぼ同等の性能である事を確認した．

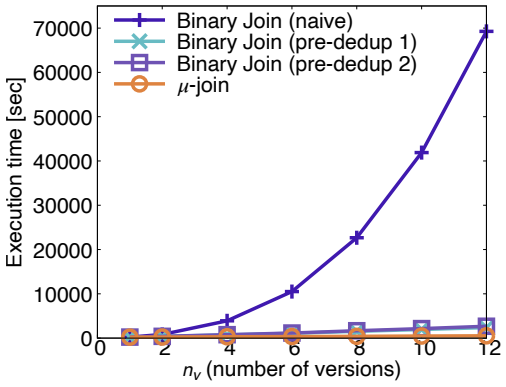
最後に，クエリ B とクエリ C を用いて，異なるディメンション表の世代数におけるファクト表と複数のディメンション表との結合処理を評価した．図 7(a) に示す様に，完全世代管理では，μ-join がファクト表と 2 世代のディメンション表との結合において従来方式の最高値よりも 24.6% 高速であることが分かった．さらに世代数が増えるにつれ優位性が高まり，12 世代数の場合では 65.0% の高速化が確認できた．図 7(b) に示す様に，完全世代管理では，μ-join がファクト表と 2 種類の 2 世代のディメンション表との結合において従来方式の最高値よりも 35.9% 高速である事が確認できた．さらに，各ディメンション表の世代数が増えるにつれ優位性は向上し，12 世代の場合 78.2% の高速化を達成した．

4.3　医療データを用いた実験結果

前章で示した様に，本論文で提案する μ-join の優位性を合成データを用いて確認した．本章では医療分野のリアルデータを用いて評価実験を行う．



(a) クエリ B



(b) クエリ C

図 7: 異なるディメンション表の世代数における評価

本章では，三重県より提供頂いた公的医療保険の審査請求の記録である医療レセプトを用いて評価実験を行う [14, 15]．本データセットには，2018 年 1 月から 2020 年 11 月までの 10,504,087 件の医療レセプトと，6,081,943 件の調剤レセプトを含むんでおり，患者毎の傷病名や各患者への診療行為や調剤といった医療サービスの内容が含まれている．図 1 に示す様に，そのような医療サービスの内容の把握には，傷病，診療行為，調剤の属性情報が記載されている傷病名マスタ，診療行為マスタ，調剤マスタと呼ばれる情報が必要である．本実験では，医療レセプトと同時期に有効であった 2 世代の傷病名マスタ（各

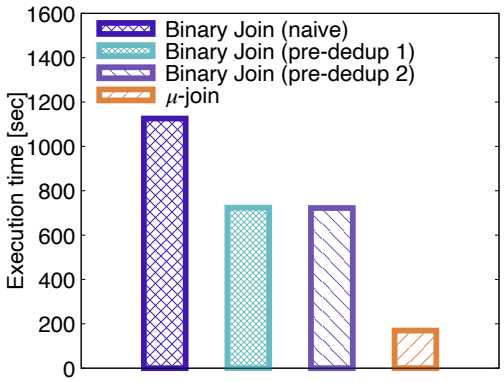


図 8: 医療レセプトを用いた評価

世代 26,164，26,407 の傷病），3 世代の診療行為マスタ（各世代 7,546，7,556，8,729 種の診療行為），3 世代の調剤マスタ（各世代 20,800，20,943，21,934 種類の調剤）をディメンション表として用いた．

上記のリアルデータを用いて，第 4.1 章で述べた実験環境で評価を行い，評価クエリには，

- クエリ D ：医療レセプトと複数世代の傷病名マスタ，診療行為マスタ，調剤マスタの結合

を用いた．図 8 にクエリ D の評価結果を示しており，リアルなファクト表である医療レセプトと，傷病名マスタ，診療行為マスタ，調剤マスタの 3 種類の世代管理されたディメンション表との結合において，μ-join が従来方式の最高値に比べ 76.5% 高速である事を確認した．

5 関 連 文 献

多くの企業や組織で，スタースキーマ [1,2] やスノーフレークスキーマ [16,17] はファクト表とディメンション表を用いて業務データは管理されている．こうして管理されるデータの世代管理については長く研究されており [11]，データのバージョンを追跡できる仕組みが提案されている [12]．関係データベースにおける解析クエリに必要なハッシュ結合，マージ結合，ネステッドループ結合といった結合処理については，多くのデータベースで実用化されている [19–21]．さらに，結合処理の性能を向上する効率的な結合処理も研究されている [23–25]．複数のデータを掛け合わせるビッグデータの分析において結合処理は必要不可欠であり，アプリケーションで実行するには負荷が高く，読み込み量や IO 数が増えるにつれデータベースエンジン内で処理の負荷は大きくなる [18,21]．こうした問題に対し，ハッシュ結合の性能向上に向けた代表的な手法にブルームフィルタが研究されている [26]．これは，ハッシュ関数による変換値が集合に含まれるかどうかを判定することによって探索空間を限定することで，読み込みコストを抑制することができる [22,27,28]．

分析に要する特定の計算処理をデータベース内にオフロードし性能向上を目指す技術として，ストアドプロシージャがある [29,31]．これによってユーザの必要とする計算処理をデータベース内部で実行処理の効率化も研究されている [43,44]．また，従来のローストアでのデータ格納方式ではカラム毎の集計などの分析処理が非効率であることから，従来のローストアでのデータ格納方式ではカラム毎の集計などの分析処理が非効率であることから，列方向の集計に効率的なデータ構造であるカラムストア [33,34] は広く利用されており，カラムストアにおける結合処理の効率的な結合方式が提案されている [35,36,42]．クエリの利用効率のため，頻繁に使う統計処理のオペレータ [30] や，SQL を拡張した形式で分析を実行できるオペレータも研究されている [32,37,38,41]．医療分野の分析で広く行われるコホート分析に必要な計算処理を，データベースオペレータとして定義し，高速にコホート分析を実現する研究が報告されている [39]．また，複数のディメンション表とファクト表や複数表のカラムストアにおける結合処理をオペレーターとして定義し処理の最適化も研究されている [40]．以上の様な関連研究は存在するが，著者らが知る限り，第 2 章で述べたケースのように世代管理されたディメンション表とファクト表における結合については考慮されていない．

6 おわりに

本論文では，世代管理されたディメンション表とファクト表を直接結合する μ-join オペレータを提案した．そして，μ-join オペレータを用いることにより，複数世代のディメンション表とファクト表の結合処理を従来のバイナリ結合オペレータを用いた場合よりも合成データで最大 71.7%，医療データを用いて 76.5% 高速に実行できることを示した．今後は，大規模データでの実験や提案オペレータへの問合せ言語，内部処理の最適化について追求して行く予定である．

謝　　辞

本研究の一部は，日本学術振興会科学研究費補助金 20H04191 の助成を受けたものである．診療報酬明細情報は三重県各市町村，後期高齢者医療制度，国民健康保険連合会より提供を受けたものであり，その取扱いに関しては満武巨裕博士（医療経済研究機構）に指導いただいた．感謝する次第である．

文　　献

[1] Ralph Kimball and Margy Ross. *The data warehouse toolkit: the complete guide to dimensional modeling.* John Wiley & Sons, 2011.

[2] Patrick O'Neil, Elizabeth O'Neil, Xuedong Chen, and Stephen Revilak. The star schema benchmark and augmented fact table indexing. TPCTC, Vol.5895, pp.237-252, 2009.

[3] 厚生労働省. 我が国の医療保険について. https://www.mhlw.go.jp/stf/seisakunitsuite/bunya/ kenkou_iryou/iryouhoken/iryouhoken01/index.html, (2022-01-10 参照).

[4] 社会保険診療報酬支払基金 編集. レセプト電産処理システム マスタファイル仕様説明書 本編. https://www.ssk.or.jp/seikyushiharai/tensuhyo/kihonmasta/index.files/master_1_20211216.pdf, (2022-02-09 参照).

[5] 社会保険診療報酬支払い基金. 傷病名マスター. https://www.ssk.or.jp/seikyushiharai/tensuhyo/kihonmasta/kihonmasta_07.html, (2022-02-10 参照).

[6] 関総研グループ. 医療需要の実態把握に活用 NDB オープンデータの概要. 医業経営情報 REPORT (2018). http://www.sekisoken.co.jp/wp-content/uploads/2018/10/Report201810_02.pdf. (2022-01-10 参照).

[7] 藤森研司. レセプトデータベース (NDB) の現状とその活用に対する課題. 医療と社会, Vol.26, No.1, pp.15–24, 2016.

[8] 福田治久, 佐藤 大介, 白岩 健, 福田 敬. NDB 解析用データセットテーブルの開発. 保健医療科学, Vol.68, No.2, pp.158–167, 2019.

[9] Prakai Nadee and Preecha Somwang. Efficient incremental data backup of unison synchronize approach. *BEEI*, Vol.10, No.5, pp.2707-2715, 2021.

[10] TPC. TPC-H Standard Specification Revision 3.0.0. http://tpc.org/TPC_Documents_Current_Versions/pdf/tpc-h_v3.0.0.pdf/ (2022-01-10 参照).

[11] Peter Dadam, Vincent Y. Lum, and H.-D. Werner. Integration of Time Versions into a Relational Database System. In VLDB. pp.509–522, 1984.

[12] Anant P. Bhardwaj, Amol Deshpande, Aaron J. Elmore, David R. Karger, Sam Madden, Aditya G. Parameswaran, Harihar Subramanyam, Eugene Wu and, and Rebecca Zhang. Collaborative Data Analytics with DataHub. *Proc. VLDB Endow.*, Vol.8, No.12, pp.1916–1919, 2015.

[13] Jim Gray and Andreas Reuter. *Transaction processing: concepts and techniques.* Elsevier, 1992.

[14] Jumpei Sato, Hiroyuki Yamada, Kazuo Goda, Masaru Kitsuregawa, and Naohiro Mitsutake. Enabling patient traceability using anonymized personal identifiers in Japanese universal health insurance claims database. *AMIA Jt Summits Transl Sci Proc*, Vol.2019, pp.345–352, 2019.

[15] Kazutoshi Umemoto, Kazuo Goda, Naohiro Mitsutake, and Masaru Kitsuregawa. A prescription trend analysis using medical insurance claim big data. In ICDE. pp.1928–1939, 2019.

[16] Michael Krippendorf and Il-Yeol Song. The translation of star schema into entity-relationship diagrams, In DEXA Workshop. pp.390–395, 1997.

[17] Mark Levene and George Loizou. Why is the snowflake schema a good data warehouse design? *Information Systems.* Vol.28, No.3, pp.225–240, 2003.

[18] Shimin Chen, Anastassia Ailamaki, Phillip B. Gibbons, and Todd C. Mowry. Improving hash join performance through prefetching. *ACM Trans. Database Syst.*, Vol.32, No.3, pp.17, 2007.

[19] Masaru Kitsuregawa, Hidehiko Tanaka, and Tohru Moto-Oka. Application of hash to data base machine and its architecture. *New Generation Computing*, Vol.1, No.1, pp.63–74, 1983.

[20] Leonard D. Shapiro. Join processing in database systems with large main memories. *ACM Trans. Database Syst.*, Vol.11, No.3, pp.239–264, 1986.

[21] Jignesh M. Patel, Michael J. Carey, and Mary K. Vernon. Accurate modeling of the hybrid hash join algorithm, In SIGMETRICS, pp.56–66, 1994.

[22] Ronald Barber, Guy M. Lohman, Ippokratis Pandis, Vijayshankar Raman, Richard Sidle, Gopi K. Attaluri, Naresh Chainani, Sam Lightstone, and David Sharpe. Memory-efficient hash joins. *Proc. VLDB Endow.*, Vol.8, No.4, pp.353–364, 2014.

[23] Huiju Wang, Xiongpai Qin, Xuan Zhou, Furong Li, Zuoyan Qin, Qing Zhu, and Shan Wang. Efficient query processing framework for big data warehouse: an almost join-free approach. *Fontiers Comput. Sci.*, Vol.9, No.2 pp.224–236, 2015.

[24] Vivek Tiwari and Ramjeevan Singh Thakur. Contextual snowflake modelling for pattern warehouse logical design. *Sadhana*, Vol. 40, No.1, pp.15–33, 2015.

[25] Mohammed Benjelloun, Mohamed El Merouani, and El Amin Aoulad Abdelouarit. Impact of using snowflake schema and bitmap index on data warehouse querying. *IJCA*, Vol.180, No.15, pp.33–35, 2018.

[26] Burton H. Bloom. Space/time trade-offs in hash coding with allowable errors. *Commun. ACM*, Vol.13, No.7, pp.422-426, 1970.

[27] Sukriti Ramesh, Odysseas Papapetrou, and Wolf Siberski. Optimizing distributed joins with bloom filters. In ICDCIT. pp.145–156, 2008.

[28] Thi-To-Quyen Tran, Thuong-Cang Phan, Anne Laurent, and Laurent d'Orazio. Improving Hamming distance-based fuzzy join in MapReduce using Bloom Filters. In FUZZ-IEEE. pp.1–7, 2018.

[29] Andrew Eisenberg. New standard for stored procedures in SQL. *SIGMOD Record*, Vol.25, No.4, pp.81–88, 1996.

[30] Jim Gray, Surajit Chaudhuri, Adam Bosworth, Andrew Layman, Don Reichart, Murali Venkatrao, Frank Pellow, and Hamid Pirahesh. Data cube: A relational aggregation operator generalizing group-by, cross-tab, and sub-totals. *Data Min. Knowl. Discov.*, Vol.1, No.1, pp.29–53, 1997.

[31] Jim Melton and Alan R. Simon. *SQL: 1999: understanding relational language components.* Elsevier, 2001.

[32] Patrick Bosc, Didier Dubois, Olivier Pivert, and Henri Prade. Flexible queries in relational databases–the example of the division operator. *Theor. Comput. Sci.*, Vol.171, No.1-2, pp.281–302, 1997.

[33] Peter A. Boncz, Marcin Zukowski, and Niels Nes. MonetDB/X100: Hyper-Pipelining Query Execution. In CIDR. pp.225–237, 2005.

[34] Stefan Manegold, Peter A. Boncz, and Martin L. Kersten. Optimizing database architecture for the new bottleneck: memory access. *VLDB J.*, Vol.9, No.3, pp.231–246, 2000.

[35] Michael Stonebraker, Daniel J. Abadi, Adam Batkin, Xuedong Chen, Mitch Cherniack, Miguel Ferreira, Edmond Lau, Amerson Lin, Samuel Madden, Elizabeth J. O'Neil, Patrick E. O'Neil, Alex Rasin, Nga Tran, and Stanley B. Zdonik. C-store: a column-oriented DBMS. In VLDB. pp.553–564, 2005.

[36] Daniel J. Abadi, Samuel Madden, and Nabil Hachem. Column-stores vs. row-stores: how different are they really? In SIGMOD. pp.967–980, 2008.

[37] Damianos Chatziantoniou and Kenneth A. Ross. Querying multiple features of groups in relational databases. In VLDB. pp.296–306, 1996.

[38] Shazzad Hosain and Hasan Jamil. Algebraic operator support for semantic data fusion in extended sql. In CIS. pp.1–6 , 2010.

[39] Dawei Jiang, Qingchao Cai, Gang Chen, H. V. Jagadish, Beng Chin Ooi, Kian-Lee Tan, Anthony K. H. Tung. Cohort Query Processing. *Proc. VLDB Endow.*, Vol.10, No.1, pp.1–12, 2016.

[40] Sungheun Wi, Wook-Shin Han, Chu-Ho Chang, and Kihong Kim. Towards Multi-way Join Aware Optimizer in SAP HANA. *Proc. VLDB Endow.*, Vol.13, No.12, pp.3019–3031, 2020.

[41] Bin Yao, Feifei Li, and Piyush Kumar. K nearest neighbor queries and kNN-Joins in large relational databases (almost) for free. In. ICDE. pp.4–15, 2010.

[42] Daniel J. Abadi, Daniel S. Myers, David J. DeWitt, and Samuel Madden. Materialization Strategies in a Column-Oriented DBMS. In ICDE. pp.466–475, 2007.

[43] Surajit Chaudhuri and Kyuseok Shim. Optimization of queries with user-defined predicates. *ACM Trans. Database Syst.*, Vol.24, No.2, pp.177–228, 1999.

[44] Rheinländer, Astrid and Leser, Ulf and Graefe, Goetz. Optimization of complex dataflows with user-defined functions. *CSUR*, Vol.50, No.3, pp.1–39, 2017.