タプル重複のもとで外部結合および準結合を含むビューの増分メンテナンスとそのPostgreSQLへの実装

長田　悠吾† 　星合　拓馬† 　石井　達夫† 　増永　良文††

† SRA OSS, Inc. 日本支社　〒 171–0022 東京都豊島区南池袋 2–32–8
†† お茶の水女子大学（名誉教授）　〒 112–8610 東京都文京区大塚 2–1–1
E-mail: †{nagata,hoshiai,ishii}@sraoss.co.jp, ††yoshi.masunaga@gmail.com

あらまし　マテリアライズドビューはその実体をデータベースに格納することで高速な応答を可能にする一方で，実テーブルに更新があった際にはビューの内容を最新の状態に更新する必要がある．これを完全な再計算により行うのは時間がかかるため，実テーブルへの変化分に応じた更新のみをビューに適用する増分メンテナンスと呼ばれる方法が研究されてきた．関係データベースで広く使用される外部結合および準結合を定義に含むビューをメンテナンスするアルゴリズムは過去に提案されているが，これは一意キーの存在を前提としているためタプルの重複が存在するバッグ意味論の下では動作しない．そこで本論文ではタプル重複を含むビューにおいても外部結合および準結合をサポートする増分メンテナンスを提案する．また本手法をオープンソースの RDBMS である PostgreSQL に実装し開発コミュニティに提案中である．本論文ではその実装についても報告する．

キーワード　マテリアライズドビュー，ビューメンテナンス, 外部結合, 準結合, バッグ意味論，リレーショナルデータベース，PostgreSQL

1 はじめに

リレーショナルデータベースにおけるビュー（view）は，それを定義するクエリを実体とし，ビューに対する問合せが発生したときにこのクエリを実行し，その結果をクライアントに返すものである．一方，マテリアライズドビュー（materialized view）では問合せの結果がデータベースに格納されており，これにより通常のビューよりも高速な応答を可能にしている．この機能は例えばデータウェアハウスに格納された大量のデータの解析結果を短いレスポンスタイムで取得したい場合などに有用である．一方で，ビュー定義に含まれる実テーブルが更新された場合にはマテリアライズドビューに格納されているデータは古いものとなり実テーブルの内容との一貫性が失われる．そのため，実テーブルが更新された後にはマテリアライズドビューの内容を最新の状態に更新するメンテナンスが必要となる．ただし，これを更新後の実テーブルから完全な再計算によって行うのは時間がかかりコストが高い．そのため，実テーブルに発生した変化分に応じてデータベースに格納されているマテリアライズドビューの一部分だけを更新する増分ビューメンテナンス（incremental view maintenance, IVM）という手法が研究されてきた [3], [4], [1].

選択（selection），射影（projection），そして内部結合（inner join）はリレーショナルデータベースの問合せにおける代表的な演算であり，これらのみを含むビューは Selection-Projection-Join（SPJ）ビューと呼ばれている．SPJ ビューおよび集約（aggregate）を含むビューの増分メンテナンスについては古くから研究されている．一方でリレーショナルデータベースでは外部結合（outer join）および準結合（semijoin）も広く使用されている．外部結合および準結合を含むビューの増分メンテナンスについても先行研究があるが，これらはビューにタプルの重複がない集合（set）意味論を前提としている．現実の RDBMS はタプルの重複を許容するバッグ（bag）意味論の下に実装されているため，これらの方法を適用することはできない．そこで，本論文ではタプルの重複を含むビューにおいても外部結合および準結合をサポートする増分メンテナンス法を提案する．

また本手法はオープンソースの RDBMS である PostgreSQL に実装することでその動作を確認した．PostgreSQL への増分メンテナンスの実装を目的とした研究は過去にいくつか存在する [8] [10] が，現状の PostgreSQL にはこの機能はまだ実装されていない．そこで，著者らは提案手法による増分メンテナンスの実装を PostgreSQL の機能として加えることを開発コミュニティに提案中である．本論文ではその実装法についても報告する．

2 増分ビューメンテナンス

本節では増分ビューメンテナンスの概要を述べる．

2.1 増分メンテナンスと再計算

増分ビューメンテナンスと再計算との違いを図 1 に示す [1]. ここで，マテリアライズドビューを定義する問合せを Q_v，ある時点におけるビューの内容を T_v，その時点でのデータベースの状態を D とする．このとき $T_v = Q_v(D)$ が成り立つ．添え字の v はビューを意味する．データベース D に更新 δD が発生して新しい状態が D' となったとき，更新後のマテリアライズ

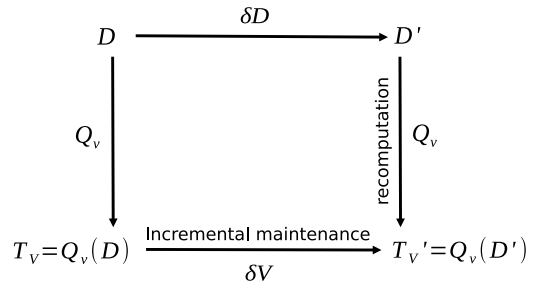

図 1　増分ビューメンテナンスと再計算によるメンテナンス

ドビューの状態 $Q_v(D')$ を D' と Q_v を用いて求めるのが再計算（recomputation）によるビューメンテンナンスである．これに対し，δD を利用して T_v に対する変化分 δV を求め，これを T_v に適用することで新しいビューの状態 T_v' を求める方法が増分メンテナンスである．

2.2　バッグ代数の記法

ビューを定義する問合せ Q_v はバッグ代数（bag algebra）上の演算の組み合わせとして記述することができる [3]．バッグ代数とはタプルの重複が存在しない集合意味論の下で定義される関係代数（relational algebra）をタプルの重複を許すバッグ意味論の下で拡張したものであるが，本論文では関係代数の演算子と同じ記法を用いる．すなわち，選択，射影，直積，内部結合はそれぞれ σ_p，π_C，$\times$，$\bowtie_q$ を用いて表す．ここで p は選択の条件を表す述語，C は射影で取り出される属性のリスト，q はテーブルを結合する条件を表す述語を意味する．これらの演算を用いてテーブル R と S の結合 $R \bowtie S$ は $\sigma_p(R \times S)$ と表すこともできる．また，ここで射影演算はタプルの重複を除去しないものとする．タプルの重複を除去する演算は δ で表す．

$\uplus$ は加法和（additive union）を表し，タプルの重複を除去せずに 2 つのテーブルの和集合を求める演算である．たとえば 2 つのテーブル $R = \{a, a, b, b\}$，$S = \{b, c\}$ があるとき $R \uplus S = \{a, a, b, b, b, c\}$ となる．$\dot{-}$ は 2 つのテーブルのモーナス（monus）を表し，タプルの重複を除去せずに 2 つのテーブルの差集合を求める演算である．上記の例では，$R \dot{-} S = \{a, a, b\}$ となる．

2.3　SPJ ビューの増分メンテナンス

ここでテーブル R，S 上の SPJ ビュー $V_1 = \pi_C \sigma_{p1}(R \bowtie_{p2} S)$ が定義されているときに，あるトランザクションにおいてテーブル R が更新されたとする．このとき R の変化は $R \leftarrow R \dot{-} \nabla R \uplus \Delta R$ と表すことができる．ここで ∇R は更新の結果として R から削除されたタプル群，ΔR は R に挿入されたタプル群を表すバッグである．増分ビューメンテナンスはこのときに V_1 に発生する差分，すなわち V_1 から削除すべきタプル群 ∇V_1 と V_1 に挿入すべきタプル群 ΔV_1 を求め，$V_1 \leftarrow V_1 \dot{-} \nabla V_1 \uplus \Delta V_1$ のように適用することで実現できる．これらの差分はビュー定義の R を ∇R，ΔR に置き換えることで求めることが出来る．この例では $\nabla V_1 = \pi_C \sigma_{p1}(\nabla R \bowtie_{p2} S)$，$\Delta V_1 = \pi_C \sigma_{p1}(\Delta R \bowtie_{p2} S)$ となる．

上記の例は 1 つの実テーブルが更新された場合である．次に複数の実テーブルが同時に更新された場合について述べる．3 つのテーブル R,S,T 上の直積ビュー $V_2 = R \times S \times T$ を考える．ここで全ての実テーブルにタプルが挿入されたとする．各テーブルの更新前の状態を R_{pre}，S_{pre}，T_{pre}，更新後の状態を $R_{post} = (R_{pre} \uplus \nabla R)$，$S_{post} = (S_{pre} \uplus \nabla S)$，$T_{post} = (T_{pre} \uplus \nabla T)$ とする．このとき，ビューに発生する差分は

$$\nabla V_2 = (\nabla R \times S_{pre} \times T_{pre}) \uplus (R_{post} \times \nabla S \times T_{pre}) \uplus (R_{post} \times S_{post} \times \nabla T)$$

となる．すなわちテーブルに発生した差分と更新前，更新後の両テーブル状態を用いて実テーブルの数だけの直積演算を行う．タプル削除の場合も同様である．また，同じテーブルがビュー定義の中に複数回現れる自己結合（self join）の場合は，定義上は同じテーブルを「内容が同じ異なるテーブル」とみなすことにより同じ方法で増分メンテナンスが可能である．

2.4　タプル重複の扱い

ビュー，実テーブルおよびこれらの差分はバッグであるためタプルの重複が許される．バッグ内の同じタプルの数はそのタプルの重複度（multiplicity）と呼ばれる．増分メンテナンスにおいてビューに差分を適用する際には加法和 $\uplus$ およびモーナス $\dot{-}$ の演算を用いるが，これらの演算はタプルの重複度同士の加減演算で定義することができる．例えば，バッグ A と B に共通するタプル t があり，その重複度をそれぞれ n，m とする．このとき，加法和 $A \uplus B$ の結果に含まれるタプルの重複度は，A と B に共通するタプル t については $n + m$ となり，片方にのみ存在するタプルについては元の重複度がそのまま使われる．また，モーナス $A \dot{-} B$ に関しては，A と B に共通するタプル t の重複度は $max(n - m, 0)$ となる．このとき重複度が 0 となったタプルは結果に含めない．A にのみ存在するタプルについては元の重複度がそのまま使われる．B のみに含まれるタプルについても演算結果に含まれない．

バッグに重複除去演算 δ を施すと，全てのタプルの重複度は全て 1 となる．一方，重複除去を含むビューの増分メンテナンスを行う際には重複除去を行う前のタプル重複度を保持しておくことが有用である．例えば，テーブル $R = \{a, a, b, c, c\}$ の上にビュー $V_3 = \delta(R)$ が定義されるとき，ビューの内容は $\{a, b, c\}$ となる．このとき R から $\Delta R = \{a, b\}$ が削除されたとする．もしここで ΔR を直接 V_3 から削除すれば $V_3 = \{c\}$ となり正しい状態 $\{a, c\}$ が得られない．しかし，実テーブル R のタプルの重複度が保持されていれば，ΔR を V_3 に適用すると b の重複度が 0 となるため V_3 から削除され，a，c の重複度は 1 以上であるため V_3 に残るということがわかる．

以上のようにタプルの重複度をビュー内に保持することで，タプルの重複および重複除去演算を含むビューの増分メンテナンスを実現する手法は counting アルゴリズムと呼ばれており [6]，本論文における実装ではこの手法を用いている．

2.5　集　　約

集約を含むビューの増分メンテナンスはテーブルに発生した差分上の集約値を計算することで実現できる．本実装で対応し

ている集約関数は count, sum, avg, min, max である．

ある集約関数 $f(x)$ を用いたビュー $V_4 = \pi_{G,f(x)}R$ がバッグ R 上に定義されているとする．ここで x は R 上の集約対象の属性で，G はグループ化（group-by）属性である．R に差分 ∇R および ΔR が発生したとき，各グループ G に関する差分上の集約値を $\nabla f_G = \pi_{G,f(x)}\nabla R$，$\Delta f_G = \pi_{G,f(x)}\Delta R$ とする．f が count または sum の場合，ビュー内のグループ G の集約値は $f(x) \leftarrow f(x) - \nabla f_G + \Delta f_G$ と計算できる．また f が avg の場合には count および sum の値を別途ビュー内に保持しておくことで $avg(x) \leftarrow sum(x)/count(x)$ と計算できる．f が min または max の場合，タプルが挿入されたときには $min(x) \leftarrow Min(min(x), \nabla min_G)$，$max(x) \leftarrow Max(max(x), \nabla max_G)$ と更新できる．一方でタプルが削除された場合，$min(x) < \Delta min_G$，$max(x) > \Delta max_G$ ならばグループ G のタプルは更新する必要が無いが，それ以外の場合はグループ G の $min(x)$，$max(x)$ を実テーブルから再計算する必要がある．

2.6 メンテナンスのタイミング

ビューの増分メンテナンスを実行するタイミングには大きく 2 つのアプローチがある．即時メンテナンス（immediate maintenance）では実テーブルを更新した同じトランザクションの中でビューが最新の状態に更新される．一方，遅延メンテナンス（deferred maintenance）では実テーブルを更新したトランザクションがコミットされた後にビューの更新が行われる．

本論文で提案する外部結合および準結合の増分メンテナンス法はどちらのアプローチでも使用可能である．ただし，PostgreSQL への最初の実装においては即時メンテナンスを採用した．これは遅延メンテナンスでは複数トランザクションで発生したテーブル差分を管理する機構を実装する必要があるが即時メンテナンスではこれが不要であり，少ない実装量で実現可能だったためである．

3 外部結合を含むビューの増分メンテナンス

本節では外部結合を含むビューの増分メンテナンスの先行研究とその課題，およびこれを解決する提案手法について述べる．

3.1 外 部 結 合

内部結合では結合される 2 つのテーブル間において結合条件が満たすタプルが存在しない場合そのタプルが演算結果に現れない．一方で，テーブル R と S の外部結合では，R のあるタプルに対して結合条件を満たすタプルが S に存在しない場合，R に含まれるタプルが演算結果に現れる（左外部結合または完全外部結合の場合の例）．このとき，タプルのテーブル S に属する属性の値は全て null となる．この操作はタプルの null 拡張（null-extend）と呼ばれ，null 拡張されたタプルはダングリングタプル（dangling tuple）と呼ばれる．外部結合を含むビューの増分メンテナンスでは，タプルの挿入時にはダングリングタプルをビューから削除，タプル削除時にはダングリングタプルをビューに挿入することが状況に応じて追加で必要となる．

外部結合は反結合（antijoin）と外部和集合（outer union) を用いて定義できる．テーブル R と S の左反結合 $R \triangleright_{p(R,S)} S$ はテーブル R のタプルのうち S との間の条件 $p(R,S)$ を「満たさない」タプルを抽出する演算である．

スキーマの異なる 2 つのテーブル R と S の外部和集合 $R \uplus S$ は、互いに共通していない属性については null 拡張した上で加法和をとる演算である．例えば $R(x,y) = \{(1,2)\}$，$S(y,z) = \{(3,4)\}$ の外部和集合は $R \uplus S = \{(1,2,null),(null,3,4)\}$ となる．同じスキーマのテーブル同士の外部和集合は加法和と同じ結果となり矛盾がないため記号は同じものを用いる．

これらの演算を用いて，テーブル R と S の左外部結合（left outer join）は $R \mathbin{⟕_p} S = (R \bowtie_p S) \uplus (R \triangleright_p S)$，右外部結合（right outer join）は $R \mathbin{⟖_p} S = (R \bowtie_p S) \uplus (S \triangleright_p R)$，完全外部結合（full outer join）は $R \mathbin{⟗_p} S = (R \bowtie_p S) \uplus (R \triangleright_p S) \uplus (S \triangleright_p R)$ と定義できる．

3.2 先 行 研 究

外部結合を含むビューの増分メンテナンス法の先行研究として [2] [5] [7] が挙げられる．この中で [2] の手法はビューの中に外部結合が複数含まれる場合に処理が複雑になり性能が良くないことが報告されている [5] [7]．また, [5] で提案されているアルゴリズムは適用できる条件が厳しく，実システムで使用するには十分ではない．例えば，テーブル R, S, T の外部結合ビュー $V_5 = (R \mathbin{⟕_{p(R,S)}} S) \mathbin{⟕_{p(R,T)}} T$ を考える．ここで $p(X,Y)$ はテーブル X と Y の間の条件を表す述語である．この定義のもと R にタプルが挿入された場合，S との結合，T との結合それぞれについてダングリングタプルの削除が必要になる場合があるが, [2] の手法ではこの状況を扱えない．

以下では [7] の手法の概要を説明する．なおこの手法では次の前提を置いている: 全ての実テーブルは null を含まない一意キー（unique key）を持っていること; ビューは同じテーブルを一度だけ参照すること（自己結合の禁止）: ビューは一意キーを出力すること（タプル重複の禁止）; ビューに含まれる全ての述語は null-rejecting であること．すなわち，属性の集合 S 上の述語 $p(S)$ は，S の何れかの属性が null ならば偽（false）を返すこと．

3.2.1 準　　備

タプル t_1 と t_2 が同じスキーマ上に定義されており，両方のタプルで null でない属性に関しては値が同じであり，t_1 の null となる属性の数が t_2 より少ないとき，t_1 は t_2 を包摂（subsume）すると言う．テーブル T の「包摂されるタプルの除去（removal of subsumed tuples）」$T\downarrow$ は T のタプルのうち他のタプルに包摂されないものを返す演算である．テーブル R と S の最小和（minimum union）$R \oplus S$ を $(R \uplus S)\downarrow$ と定義する．例えば $R(x,y,z) = \{(1,2,3),(4,5,6)\}$，$S(y,z) = \{(5,6),(7,8)\}$ の場合，$R \oplus S = \{(1,2,3),(4,5,6),(null,7,8)\}$ となる．

3.2.2 結合選言標準形

最小和の演算を用いると，テーブル R と S の左外部結合は $R \mathbin{⟕_p} S = (R \bowtie_p S) \oplus R$，右外部結合は $R \mathbin{⟖_p} S = (R \bowtie_p S) \oplus S$，完全外部結合は $R \mathbin{⟗_p} S = (R \bowtie_p S) \oplus R \oplus S$ と書き換えることができる．これを用いて V_5 は以下のように書き換えることが

できる．

$$
\begin{aligned}
V_5 &= (R \rtimes_{p(R,S)} S) \rtimes_{p(R,T)} T \\
&= (\sigma_{p(R,S)}(R \times S) \oplus R \oplus S) \rtimes_{p(R,T)} T \\
&= \sigma_{p(R,S)\wedge p(R,T)}(R \times S \times T) \oplus \sigma_{p(R,T)}(R \times T) \oplus \\
&\quad \sigma_{p(R,S)}(R \times S) \oplus R \oplus S
\end{aligned}
$$

[7] ではこのような内部結合が最小和で結合された形式のビュー定義を結合選言標準形（join-disjunctive normal form）と呼んでいる．一般的にテーブル集合 U 上のビュー定義表現 E は以下のように標準形で書くことが出来る．

$$E = E_1 \oplus E_2 \oplus ... \oplus E_n$$

ここで各項（term）E_i は

$$E_i = \sigma_{p_i}(T_{i_1} \times T_{i_2} \times ... \times T_{i_m})$$

の形を取る．$T_{i_1}, T_{i_2} ... T_{i_m}$ は U に含まれるテーブルの部分集合，述語 p_i は定義クエリに含まれる選択および結合条件の部分集合の連言（conjunction）である．

3.2.3 包摂グラフとビューメンテナンスグラフ

上述の通り，結合選言標準形の各項はテーブルの部分集合 S を持つ．これを各項のソーステーブル（source tables）と呼ぶ．また各項により生成されるタプルはソーステーブル以外のテーブル（$U - S$）において null 拡張されている．これは，タプルがこれらのテーブルとの反結合により生成されるためである．

ソーステーブル集合が S である項により生成されるタプルは，ソーステーブル集合が S の上位集合であるような項により生成されるタプルによってのみ包摂される．この包摂関係を表した有向非巡回グラフ（directed acyclic graph, DAG）を包摂グラフ（subsumption graph）と呼ぶ．各項 E_i 毎に節（node）n_i を持ち，各節は項のソーステーブル集合 S_i でラベル付けされる．n_i から n_j への辺（edge）は S_i が S_j の最小上位集合（minimal superset）の場合に存在する．$S_j \subset S_k \subset S_i$ となるような節 n_k がグラフに存在しないとき，S_i は S_j の最小上位集合である．このとき，項 E_i は E_j の（直接の）親（parent）と呼ばれる．

このようにして作られた包摂グラフはあるテーブルが更新された時にどの項がどのように影響を受けるかを表している．具体的にはテーブル T が更新されたとき，ソーステーブル集合 S に T を含む項は直接的（directly）に影響を受ける．すなわち T にタプル挿入があったときにはその項により生成されるタプルがビューに挿入され，削除があった場合には対応するタプルがビューから削除される．ソーステーブル集合に T を含まないが親の何れかが直接的に影響を受ける項は間接的に（indirecly）影響を受ける．すなわち，親の項にタプルの挿入があった場合には自身の項からのダングリングタプルの削除，親の項にタプルの削除があった場合には、自身の項へのダングリングタプルの挿入の必要があるかもしれない．それ以外の項は影響を受けない項である．

包摂グラフグラフから影響を受けない項に対応する節を取り除き，残った節に項の影響の受け方のラベル（直接的なら D, 間接的なら I) を付与したものをビューメンテナンスグラフと呼ぶ．外部結合ビューの増分メンテナンスはビューメンテナンスグラフを参照することで行う．

3.2.4 ビューのメンテナンス

ビューのメンテナンスは 2 段階の手順を踏む．まず直接的な影響を受ける項に属するタプルに対する差分を計算してビューに適用する．この差分は一次差分（primary delta）と呼ぶ．次に間接的な影響を受ける項があれば，それに属するタプルに対する差分を計算してビューに適用する．この差分を二次差分（secondary delta）と呼ぶ．

一次差分は基本的に通常の SPJ ビューと同じように計算することができる．すなわち，変更のあったテーブル T を，その差分 ∇T，ΔT に置き換えたビュー定義クエリを実行することで計算可能である．しかし，差分の計算時には変更があったテーブルにおける null 拡張は不要であるため，完全外部結合を左外部結合や右外部結合に，左外部結合と右外部結合は内部結合に置き換える必要がある．例えば，$V_5 = (R \rtimes_{p(R,S)} S) \rtimes_{p(R,T)} T$ において R にタプル挿入があった場合，その一次差分は $\Delta V_5 = (\Delta R \rtimes_{p(R,S)} S) \rtimes_{p(R,T)} T$ と R と S の完全外部結合を左外部結合に置き換えたクエリを実行して計算する．計算された差分は通常の SPJ ビューと同じようにビューに適用する．

二次差分は間接的な影響を受ける項に属するダングリングタプルの挿入や削除を行うための差分である．この差分は間接的な影響を受ける項が複数ある場合，それぞれについて計算し適用する必要がある．

テーブルにタプルの挿入があり，一次差分が ΔV であり，項 E_i が間接的な影響を受けるとする．一次差分が適用済なのでビューの内容は $V \uplus \Delta V$ となる．このときに削除されるダングリングタプルは以下で計算できる．

$$
\begin{aligned}
\nabla D_i &= \sigma_{nn(T_i)\wedge n(S_i)}(V \uplus \Delta V) \bowtie^{ls}_{eq(T_i)} \sigma_{P_i}\Delta V \qquad (1) \\
P_i &= \bigwedge_{E_k \in pard(E_i)} nn(T_k)
\end{aligned}
$$

ここで T_i は項 E_i のソーステーブル集合，S_i は E_i 上で null 拡張されているテーブル集合である．$nn(T_i)$ は T_i に含まれる全てテーブルにおいて null 拡張されていない，$n(S_i)$ は S_i に含まれる全てのテーブルにおいて null 拡張されていることを意味する．すなわち，$\sigma_{nn(T_i)\wedge n(S_i)}(V \uplus \Delta V)$ はビューの中から項 E_i に属するダングリングタプルを抽出するための選択演算である．$pard(E_i)$ は項 E_i の「直接的な影響を受けた」親の項の集合を意味しており，$\sigma_{P_i}\Delta V$ は一次差分によって親の項に挿入されたタプルを抽出している．$\bowtie^{ls}_{eq(T_i)}$ は T_i のキー属性が等しいことを条件とした準結合であり，このダングリングタプルが親の項に挿入されたタプル（$\sigma_{P_i}\Delta V$）に包摂されるかをチェックしている．全体として，親の項に新しく結合されたタプルが挿入されることによりビューから削除されるべきダングリングタプルを求める表現となっている．

次にテーブルからタプルが削除されたときを考える．このときの一次差分が ∇V であり，項 E_i が間接的な影響を受けるとする．一次差分が適用済なのでビューの内容は $V \dot{-} \nabla V$ となる．

このときに挿入すべきダングリングタプルは以下で計算できる．

$$\Delta D_i = (\delta\pi_{T_i.*}\sigma_{P_i}\nabla V) \triangleright_{eq(T_i)} (V \dot{-} \nabla V) \quad (2)$$

$\sigma_{P_i}\nabla V$ は一次差分によって削除されたタプルの中から直接的な影響を受けた親の項に属するタプルのみを抽出している．射影により項 E_i のソーステーブルの属性のみ取り出すことで他の属性は null 拡張されており，これが挿入すべきダングリングタプルの表現になっている．重複除去を行っているのはタプルが一意キーを持っているという前提のためである．最後に，得られたダングリングタプルに対して，ビューの内容と T_i のキー属性が等しいことを条件とした反結合を行っている．これは，親の項のタプルに包摂される，すなわち，親の項に結合された結果がまだ残っているダングリングタプルの挿入を避けるためである．

3.2.5 先行研究の課題

前述の通り，先行研究の手法では実テーブルおよびビューに一意キーが存在することを前提としておりタプルの重複を許していない．これには以下の理由があると考えられる．

まずビュー定義クエリを結合選言標準形に変換する際に最小和演算を使用しており，その演算の過程で包摂されるタプルの除去が行われる点である．その際にタプルに一意キーがないと重複してタプルが除去されてしまう可能性がある．例えば，null を含むタプルを含むテーブル $R = \{(a,b,c),(a,b,null)\}$ があるとすると，タプル $(a,b,null)$ は途中で R から除去されることになる．そのため，結合選言標準形がクエリの正しい表現とならない．

次に二次差分の計算において「ダングリングタプルが直接的な影響を受けた親の項のタプルに包摂されるか」をチェックする際に，キー属性が等しいことを条件とした準結合および反結合を用いている点である．この条件の前提には一意キーの存在がある．

もう一つの考慮点は二次差分の計算においてテーブルからタプルが削除されたときにビューに挿入すべきダングリングタプルの重複度である．一意キーの存在を前提とすると挿入されるダングリングタプルも一意であるため，その重複度は 1 と定まる．しかし，タプルの重複を許すとダングリングタプルにも重複が発生するため，挿入時には重複度を考慮しなければならない．これを決定する手段は先行研究では提供されていない．

なお，先行研究では自己結合を許していないが，これは単に包摂グラフを生成する際にソーステーブル集合をラベルとして用いているためだと考えられる．この点に関しては，同じテーブルであってもクエリ内で複数現れる場合は異なる ID を付与することで別のテーブルのように扱い，複数テーブルが更新されたのと同じ処理を行うことで対処可能である．

3.3 提案手法：タプル重複を許す外部結合ビューの増分メンテナンス

本節では先行研究の課題を解決するため，タプル重複を許す外部結合ビューの増分メンテナンス法について提案する．

3.3.1 結合関係からのビューメンテナンスグラフの生成

外部結合を含むビューメンテナンスの要点を整理すると以下の通りである．

外部結合ビューにあるタプル $t1$ が挿入され，$t1$ がテーブル $T_1, T_2, \ldots T_n$ の何れにおいても null 拡張されていない場合，いままで満たされていなかったテーブル T_i と T_k のタプル間の結合条件 p_{ij} が新しく満たされた可能性がある．その場合には結合条件 p_{ij} に基づく反結合によって生成されていたダングリングタプル $t2$ をビューから削除する必要が出てくる．このとき，$t2$ は $t1$ で null 拡張されていないテーブル集合のうち一部において null 拡張されている．一方，$t1$ で null 拡張されているテーブルについては $t2$ でも null 拡張されている．ビューからタプルを削除する場合も同様に考えることができる．

このような $t1$ と $t2$ の null 拡張に基づく関係を先行研究では「$t1$ が $t2$ を包摂する」という関係で表している．すなわち，ダングリングタプル $t2$ を包摂するようなタプル $t1$ がビュー挿入されると $t2$ は削除され，そのような $t1$ がビューから全て削除されるとダングリングタプル $t2$ がビューに挿入されるという関係である．これに対し提案手法では包摂を用いない別の関係を考える．

3.2.1 で定義したビュー V_5 は，反結合と外部和集合を使った表現で以下のように書くことができる．

$$\begin{aligned} V_5 &= (R \rtimes_{p(R,S)} S) \rtimes_{p(R,T)} T \\ &= (\sigma_{p(R,S)\wedge p(R,T)}(R \times S \times T)) \uplus (\sigma_{p(R,T)}(R \times T) \triangleright_{p(R,S)} S) \uplus \\ &\quad (\sigma_{p(R,S)}(R \times S) \triangleright_{p(R,T)} T) \uplus (R \triangleright_{p(R,S)\vee p(R,T)} (S \uplus T)) \uplus \\ &\quad (S \triangleright_{p(R,S)} R) \end{aligned}$$

これを一般化すると，テーブル集合 U 上のビュー定義表現 E は以下のように書くことが出来る．

$$E = D_1 \uplus D_2 \uplus \ldots \uplus D_n$$

ここで各項 D_i は

$$D_i = \sigma_{p_i}(T_{i_1} \times T_{i_2} \times \ldots \times T_{i_m}) \triangleright_{q_i} (T_{i_{m+1}} \uplus T_{i_{m+2}} \uplus \ldots \uplus T_{i_k})$$

の形を取る．$T_{i_1}, T_{i_2} \ldots T_{i_k}$ は U に含まれるテーブルの部分集合，述語 p_i, q_i はそれぞれ定義クエリに含まれる選択および結合条件の部分集合の連言，選言である．

内部結合に参加している $T_{i_1}, T_{i_2} \ldots T_{i_m}$ を項 D_i のソーステーブル集合とする．このとき，ソーステーブル集合が S である項に属するダングリングタプルは，ソーステーブル集合が S の上位集合 $W(\supset S)$ である項からのタプル削除が発生すると「今まで満たされていた条件が満たされなくなる」ことによって現れ，逆に W へのタプル挿入によって「条件が新しく満たされる」ことによって S からはタプルが取り除かれるという包摂と似た関係になる．したがって，この関係を元に先行研究の包摂グラフと同様の方法でグラフを生成することができる．本論文ではこれを結合関係グラフと呼ぶことにする．包摂グラフと同様に，結合関係グラフも「あるテーブルが更新された時にどの項がどのように影響を受けるか」を表しているため，これを用

いてビューメンテナンスグラフを生成することできる．この方法では最小和演算を介さないため，タプルに重複がある場合でも使用可能である．

3.3.2 一意キーに依存しない二次差分の計算

先行研究の２つ目の問題は，二次差分を計算する式 (1) と式 (2) においてキー属性が等しいという条件 $eq(T_i)$ が使用されていることであった．この条件は先行研究では「ダングリングタプルが直接的な影響を受けた親の項のタプルに包摂されるか」をチェックするために使われている．一方，提案手法では包摂関係ではなく「ダングリングタプルの生成に関わる結合条件が直接的な影響を受けた親の項のタプルにて新しく満たされたか，あるいは，満たされなくなったか」をチェックする．そのため提案手法ではキー属性ではなく，項のソーステーブル集合 T_i が参加する内部結合の結合条件 p_i において参照されている属性値が等しいという条件を $eq(T_i)$ として使用する．

3.3.3 挿入するダングリングタプルの重複度の特定

最後に，タプル重複を許す場合には挿入するダングリングタプルの重複度が決められないという問題を解決する．

一次差分によりビューの項 D_p からタプル t_p が削除されたときに，二次差分として項 D_i にダングリングタプル t_d を挿入する場合を考える．D_i のソーステーブル集合を $T_i = \{R_1, R_2, ..., R_n\}$ とする．項 D_p のソーステーブルは T_i の上位集合であるため，削除されたタプル t_p には T_i 内の各テーブルから来たタプルの結合が含まれている．ここでテーブル R_j から来たタプル t_j^i の重複度を m_j とすると，挿入すべきダングリングタプルの重複度は $\prod_{k=1}^{n} m_k$ と求まる．

ここで「あるタプル t_p を構成しているタプル群の重複度 m_j をどう求めるか」が問題となる．本手法では，一次差分を計算する時に実テーブルごとに各タプルに付与された ID の種類を数えることでこれを実現する．

例としてテーブル $R = \{a, a\}$ と $S = \{b, c\}$ の直積を考える．このとき，R, S に一時的に ID が付与可能だとし，ID を付与した後のテーブルは $R = \{a(id_R = 001), a(id_R = 002)\}$, $S = \{b(id_S = 101), c(id_S = 102)\}$ とする．直積の結果は $R \times S = \{(a,b),(a,b),(a,c),(a,c)\}$ となり，(a,b) と (a,c) の重複度はそれぞれ 2 である．その重複度の内訳は各値に付与された ID の種類を数えると求まる．この例で (a,b) に関して言えば，a に付与された id_R は 001 と 002 の 2 種類，b に付与された id_S は 101 の 1 種類である．すなわち，重複度 2 のタプル (a,b) は，R から来た重複度 2 のタプル (a) と，S から来た重複度 1 のタプル (b) から構成される，ということが言える．

なお，ここで付与される ID は差分の計算時にタプルを区別する用途にのみ用いられるもので，テーブルに格納されるデータに何かしら意味を与えるものではなく，一意キーとは異なるものである．この意味において，これは文献 [9] [10] において提案された，ユーザに不可視な物理 ID をタプルに付与する操作である「OID の顕在化」と似ている．本論文の実装においては，PostgreSQL においてタプルの物理位置を表す TID（タプル ID）と，クエリ実行時に行に自動で連番を生成する row_number() 関数をこの用途に用いた．

4 準結合を含むビューの増分メンテナンス

本節では準結合を含むビューの増分メンテナンスの先行研究と提案手法について述べる．

4.1 準 結 合

テーブル R と S の準結合（semijoin）$R \ltimes^{ls}_{p(R,S)} S$ は，テーブル R のタプルのうち S との間の条件 $p(R,S)$ を満たすタプルのみを抽出する演算である．バッグ意味論においては単純に結合 $Join_{p(R,S)}$ を行った結果を射影 π_R しただけではないないことに注意を要する．通常の結合の場合，R の１行のタプルと結合条件を満たすタプルが S に複数存在する場合には複数行の結果が得られるが，準結合では R のタプルが１行返るだけである．そのため射影と重複除去を用いて $R \ltimes^{ls}_{p(R,S)} S = \pi_R(R \bowtie_{p(R,S)} (\delta\pi_{C_p} S))$ と定義することができる．ここで $C_p \in S$ は結合条件 p で使用されている S 上の属性集合である．

先行研究として [2] が準結合を含むビューの増分メンテナンスを扱っているが，この手法は集合意味論に基づくものでタプル重複が存在する状況は考慮されていない．

4.2 提案手法：タプル重複を許す準結合ビューの増分メンテナンス

テーブル R と S の準結合 $R \ltimes^{ls}_{p(R,S)} S$ において，テーブル R に更新が発生した場合は通常の SPJ ビューと同じように増分メンテンナンスが可能である．しかし，テーブル S には重複除去演算が行われているため，S に更新が発生した場合には重複度を考慮した増分メンテナンスが必要となる．そこで，提案手法では重複除去演算 δ ビューに対するアプローチと同様にタプルの重複度をビュー内に保持する counting アルゴリズムにより解決する．

R の各タプルについて，これと条件 $p(R,S)$ で結合する S のタプルの行数を数え，これを R のタプル毎に保存しておく．このカウンタ値が 1 以上であれば，そのタプルはビュー内に存在できる．もし S にタプルの削除が発生しカウンタ値が 0 となれば，そのタプルと結合するタプルはもう S に存在しないことを意味するのでタプルをビューから削除する．S にタプルを挿入した時に R のタプルとの結合が発生した場合，もしその R のタプルがまだビューに存在していなければ，ビューにタプルを挿入しカウンタ値を結合した S のタプルの行数に設定する．もし，すでにタプルがビューに存在していれば，新たに発生した結合の行数の分だけカウンタ値を増やす．

5 PostgreSQL への実装

本節では本論文で提案した増分ビューメンテナンスの PostgreSQL への実装について述べる．

5.1 概　　要

現在の PostgreSQL ではマテリアライズドビューを作成する機能はサポートしているが，そのメンテナンス法として用意されているのはビュー定義クエリを実行し直すことでビューを更新

する，すなわち再計算による方法のみであり，増分ビューメンテナンスの機能は実装されていない．そこで我々は PostgreSQL への増分メンテナンスの機能を実装を進めており，本論文で提案した外部結合および準結合のサポートもこの実装に含まれている．実装は PostgreSQL の開発コミュニティに提案され議論中である．開発中のコードについては GitHub にて公開されている [11].

提案中の実装は即時メンテナンスアプローチを採用しており，実テーブルを更新した時に AFTER トリガ関数が起動し，マテリアライズドビューはそのトリガの中で即時に自動的に更新される．

現在の実装では，選択，射影，内部結合，外部結合，集約，DISTINCT，タプルの重複，GROUP BY，EXISTS，単純なサブクエリを含むビューに対応している．

5.2 基本的な動作

増分メンテナンス可能なマテリアライズドビュー（Incrementally maintainable materialized view: IMMV）は CREATE INCREMENTAL MATERIALIZED VIEW という構文で作成する．ここで INCREMENTAL というキーワードが本実装において既存の PostgreSQL に加えた拡張である．

マテリアライズドビューの作成時にはビュー定義クエリが実行されその結果がデータベースに格納される．IMMV の定義時にはクエリはユーザが指定したものと同一ではなく内部的に書き換えられたクエリが実行される．具体的には重複除去を行う DISTINCT 句やタプルの重複を扱うために，タプルの重複度を計算する count(*) 集約関数と GROUP BY 句がクエリに追加される．重複度の計算結果は__ivm_count__という名前のカラムでビューに保存される．__ivm_ で始まるカラム名は IMMV において隠しカラムとして扱われ，SELECT 文のターゲットリストに明示的に指定されない限りクエリ結果には現れない．クエリで集約や EXISTS 句が使用されている場合にはこれとは別の隠しカラムも追加される．

IMMV の定義時にはビューを定義する全ての実テーブルに対して AFTER トリガが作成され，ビューの更新はこの AFTER トリガの中で行われる．選択，射影，内部結合の増分メンテナンス法については 2.3 の例で述べた通りである．また DISTINCT およびタプル重複の対応のため，2.4 で述べたように__ivm_count__に格納されているタプル重複度の更新が行われる．集約が含まれる場合は 2.5 で述べた方法で集約値の更新が行われる．集約値を計算するために count および sum の値が必要な場合には，これらの値は IMMV 作成時に追加された隠しカラムに格納される．

5.3 外部結合ビューの増分メンテナンス

IMMV の定義に外部結合が含まれている場合，ビューをメンテナンスする際にビューメンテナンスグラフを作成する．まずビュー定義に含まれる結合の構造を解析することで，3.3.1 で述べたような標準形に変換する．その後，標準形の各項のソーステーブル集合の包含関係に基づいて結合関係グラフを作成する．このとき，各項のソーステーブルの内部結合で使われる結合条件において参照されているカラムの情報を収集しておく．この情報は 3.3.2 で述べたように二次差分を計算する時に使用される．

一次差分を計算する際には 3.3.3 で述べた通り，差分のタプルを構成する各実テーブルのタプル重複度の内訳を求める必要がある．そのためにクエリを書き換えて以下のような集約関数を加える．

```
jsonb_buid(
  "r", count(DISTINCT r.ctid),
  "s", count(DISTINCT s.ctid)
)::jsonb AS __ivm_meta__
```

ここで，r,s は一次差分の計算に用いる実テーブルの名前の例であり，実際の実装では現実に即したものとなる．ctid は PostgreSQL においてタプルの物理的な ID を出力する特別なシステムカラムであり，物理的に異なるタプルでは互いに異なる値を取る．count(DISTINCT) によりこの値の種類の数を取得することで，一次差分のタプルを構成する各タプルの重複度の内訳を求める．最終的にはこれらの値は JSON オブジェクトに格納される．この情報は二次差分として挿入するダングリングタプルの重複度を決めるのに用いられる．

5.4 準結合ビューの増分メンテナンス

SQL では準結合は EXISTS 句として実装されている．4.2 で述べた手法により，本実装では EXISTS 句を含む IMMV を定義可能である．

以下は EXISTS 句を含むクエリの例である．

```
SELECT t1.*
FROM test AS t1
WHERE EXISTS(SELECT 1
             FROM test2 AS t2
             WHERE t1.id = t2.id);
```

上記のような EXISTS 句を含む IMMV の増分メンテナンスを行う際には，以下のように LATERAL サブクエリに変換する．

```
SELECT t1.* ,
       exist_query."__ivm_exists_count_0__"
FROM test AS t1 ,
  LATERAL(SELECT 1,
            COUNT(*) AS "__ivm_exists_count_0__"
          FROM test2 AS t2
          WHERE t1.id = t2.id
          HAVING __ivm_exists_count_0__ > 0
  ) AS exist_query;
```

そうすることで，準結合しているタプルの行数を隠しカラムとしてビューに格納でき，4.2 で述べた counting を用いた準結合ビューのサポートが可能になる．

5.5 動 作 例

ここではタプル重複を含む外部結合ビューの増分メンテナンスの簡単な動作例を示す.

```
=# SELECT * FROM r;
 i
---
 1
 1
(2 rows)

=# SELECT * FROM s;
 i
---
 1
 2
(2 rows)
```

上記のようなテーブル r, s があるときに，以下のように完全外部結合ビュー mv1 を定義する.

```
=# CREATE INCREMENTAL MATERIALIZED VIEW mv1(r,s) AS
    SELECT * FROM r FULL OUTER JOIN s on r.i=s.i;
SELECT 2
=# select * from v1;
 r | s
---+---
 1 | 1
 1 | 1
   | 2
(3 rows)
```

ここでテーブル s にタプル (2) を挿入すると，以下のように mv1 からダングリングタプルが削除される.

```
=# INSERT INTO r VALUES (2);
INSERT 0 1
=# select * from mv1;
 r | s
---+---
 1 | 1
 1 | 1
 2 | 2
(3 rows)
```

次にテーブル s からタプル (1) を削除すると，以下のように mv1 にダングリングタプルが正しい重複度で挿入される.

```
=# DELETE FROM s WHERE i = 1;
DELETE 1
=# select * from mv1;
 r | s
---+---
 2 | 2
 1 |
 1 |
(3 rows)
```

6 おわりに

本論文ではタプルの重複を含むビューにおいて外部結合および準結合をサポートする増分メンテナンスの手法を提案した．外部結合ビューの増分メンテナンスに関しては，先行研究の課題を，なぜ一意キーの存在が前提とされているのかという点から分析し，その制限を回避する手法を提案した．また，準結合ビューに関しては，タプルの重複度を数える conting アルゴリズムの手法を用いることでタプル重複を含む場合でも増分メンテナンスを可能とする手法を提案した．本論文で述べた手法はオープンソースの RDBMS である PostgreSQL に実装し，その動作が確認された．より実践的なクエリや条件における性能調査，および最適化による性能改善が今後の課題としてあげられる．

文 献

[1] R. Chirkova and J. Yang, “Materialized Views,” Foundations and Trends in Databases 4(4), pp.295-405, 2012.

[2] T. Griffin and B. Kummar, “Algebraic change propagation for semi-join and outerjoin queries,” SIGMOD Record, 27(3), September 1998.

[3] T. Griffin and L. Libkin, “Incremental Maintenance of Views with Duplicates,” Proc. ACM SIGMOD ’95, pp.328–339, 1995.

[4] A. Gupta and I. S. Mumick, “Maintenance of materialized views: Problems, techniques, and applications,” IEEE Data Engineering Bulletin, vol. 18, no. 2, pp. 3-18, 1995.

[5] H. Gupta and I. S. Mumick, “Incremental Maintenance of Aggregate and Outerjoin Expressions,” Information Systems, 31(6), 2006.

[6] A. Gupta, I. S. Mumick and V.S. Subrahmanian, “Maintaining Views Incrementally,” In SIGMOD Conference, 1993.

[7] P-Å Larson and J. Zhou, “Efficient Maintenance of Materialized Outer-Join Views,” In ICDE, pp.56-65, 2007.

[8] Quoc Vinh, N. T. “Synchronous incremental update of materialized views for PostgreSQL,” Programming and Computer Software 42(5), pp.307-315, 2016.

[9] 増永良文，長田悠吾，石井達夫, “OID を用いた新規 Incremental View Maintenance 方式の提案,” 第 11 回 Web とデータベースに関するフォーラム (WebDB Forum), 2018.

[10] 長田悠吾, 石井達夫, 増永良文, “OID を用いた Incremental View Maintenance 方式の PostgreSQL への試験的実装,” 第 11 回データ工学と情報マネジメントに関するフォーラム (DEIM Forum), 2019.

[11] IVM (Incremental View Maintenance) development for PostgreSQL https://github.com/sraoss/pgsql-ivm/