異種情報源の統合を支援するシステムの実現

寺川 文乃　宝珍 輝尚　野宮 浩輝

京都工芸繊維大学 〒606-8585 京都府京都市左京区松ヶ崎橋上町

E-mail: {hochin, nomiya}@kit.ac.jp

あらまし これまでに，異種情報源の統合を目的として，異種情報源統合システムを試作してきた．このシステムにはメモリ使用量が多いことと，GUI も含めたシステムになっており汎用性が高くない，という問題があった．本論文では，メモリ使用量の問題を，結合結果のサイズ推定式を用いて適切な結合順序を決定することによりメモリ使用量を削減し解決する．そして，GUI も含めたシステムになっており汎用性が高くないという問題を，異種情報源の統合機能を持つ Java Database Connectivity(JDBC)を作成し，GUI 部分と分離することで解決する．実機による評価の結果，結合操作の実行時間を増やすことなく，メモリ使用量の大幅な削減ができることを示す．

キーワード 異種情報源，JDBC，統合処理

1. はじめに

コンピュータ技術の発展と普及により，自身の所持するデータを電子媒体としてコンピュータ上で取り扱うユーザが増えてきた．考古学者もその内の 1 人である．彼らも自身の所持する考古学データを電子媒体とし,コンピュータ上で取り扱うようになってきている．

多くの場合，考古学者は自らの判断でどのデータベースまたはファイルにデータを蓄積するかを決定する．従って,データは統一のシステムに蓄積されていない．このようにして格納されたデータは異種情報源となる．

異種情報源を統合利用する一つの方法は，データベースのデータを別のデータベースに変換する方法である．しかし,データ変換は変換の手間がかかる．また，データ変換は，元データが頻繁に変更される場合，整合性の維持が困難である．

ラッパーとメディエータに基づくシステム[1]は，データを変換することなく異種情報源の統合を可能にする．ラッパーはアプリケーション固有のクエリをソース特有のコマンドやクエリに変換することで異種情報源へのアクセスを供給する．メディエータは異種情報源を統合するために使われる．この方法はネットワーク上に分散している異種情報源を前提としている．そのため，異種情報源はサーバコンピュータ上にある必要がある．しかし，考古学者には PC のサーバ化は困難なことが多い．また，行いたいのは，一台の PC 中の異種情報を扱うことである．

王ら[2]はユーザのデータをサーバにコピーすることなく,様々な情報源を使うことができるようにした．MySQL，PostgreSQL，SQLite，Excel ファイル，CSV ファイルを Java Database Connectivity(JDBC)を通してコネクションを作ることで，データベースとファイルを統一の方法で使うことができる．

筆者らはデータの変換，計算機のサーバ化，統合サーバへのコピーを行わない異種情報源統合システムを実現した[3]．しかし，このシステムにはメモリ使用量が多いことと，GUI も含めたシステムになっており汎用性が高くない，という問題があった．

そこで本論文では，このシステムのメモリ使用量の問題を，結合結果のサイズ推定式を用いて適切な結合順序を決定することによりメモリ使用量を削減し解決する．また，GUI も含めたシステムになっており汎用性が高くないという問題を，異種情報源の統合機能を持つ JDBC を作成し，GUI 部分と分離することで解決する．また，実機による評価の結果，結合操作の実行時間を増やすことなく，メモリ使用量の大幅な削減ができることを示す．

以降，2.で筆者らが以前作成した異種情報源統合システムについて述べ，3.で今回作成した統合支援システムについて述べる．次に，4.でメモリ使用量の計測実験とその結果について述べ，最後に 5.でまとめる．

2. 異種情報源統合システム

著者らは Java と JDBC を使用し，3 種類のデータベース MySQL，PostgreSQL，SQLite，2 種類のファイル Excel，CSV への同時接続を行い，テーブルの等結合および射影を行う異種情報源統合システムを実現した[3]．このシステムでは，JDBC を用いてデータベースやファイルへの接続を行うために必要な情報,および,結合条件の入力，実行結果の出力を全て 1 つのウィンドウで行う．このウィンドウを図 1 に示す．

処理の流れを図 2 に示す．まず初めに入力された結合条件を分解し，各データベースやファイルに合わせたクエリに変更する．次に，各データベースやファイルへの接続を行い，テーブルのスキーマ情報の取得を行った後，使用するテーブル，カラムがあるかの照合を行う．そして，結合に必要なデータの取得を行う．次に，どのテーブルのカラムかを判別できるよう，テーブル名を付与する．そして，結合操作を行う．結合

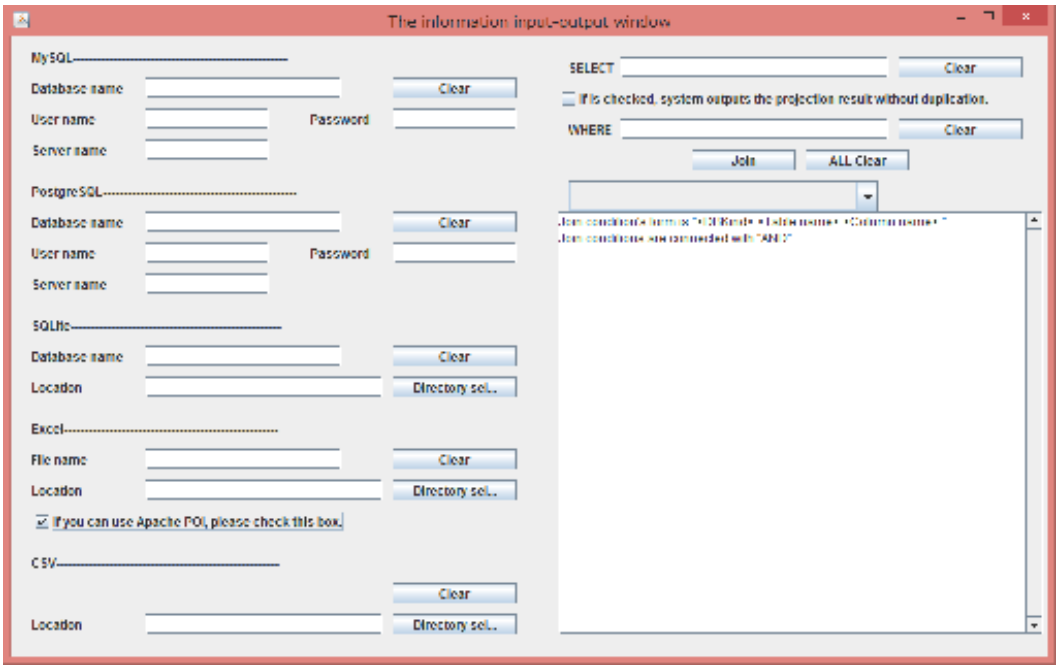

図 1　情報入出力ウィンドウ

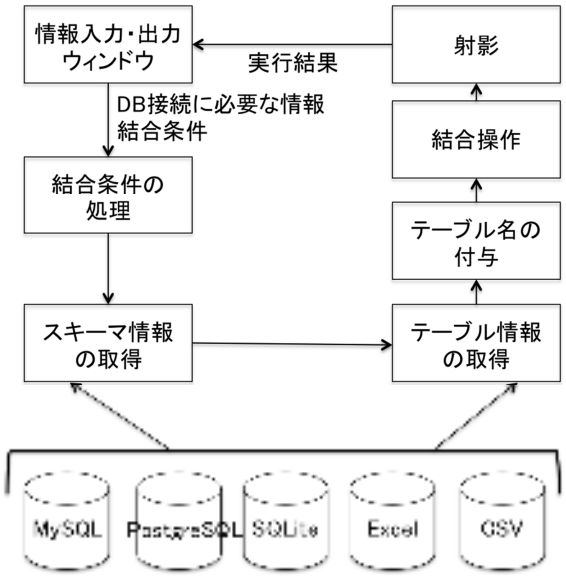

図 2　異種情報源統合システムにおける処理の流れ

操作にはソート・マージ結合[4]を使用し，結合後のサイズが分からないことから，データの管理には ArrayList を用いている．最後に，射影を行う場合は射影をし，結果をウィンドウに出力する．

実機による性能評価を行い，100,000 行の 3 テーブルの結合は実用上問題ない時間で処理できることを確認した．

しかし，メモリ使用量が多いことと，GUI も含めたシステムになっており汎用性が高くないという問題が残った．

3. 統合支援システム

2.で述べた異種情報源統合システムにおける，メモリ使用量が多いという問題を，結合結果のサイズ推定と left-deep tree[5]を用いて，メモリ使用量を削減し解決する．これらについては付録で概説する．また，2.で述べた異種情報源統合システムは GUI も含めたシステムになっており汎用性が高くない．そこで，異種情報源の統合機能を持つ JDBC を作成し，GUI 部分と分離することとする．

3.1. データ操作処理

3.1.1. 設計

(1)　結合処理

以前のシステムではユーザから入力されたクエリを前から順になぞり結合操作を行うものであった．また，全テーブルデータを ArrayList に格納し，システム内部で保持していた．この中には一度結合操作に使用し，二度と参照されないデータも含まれていた．ユーザが必要としているのは全ての結合操作が完了した最終結果である．

そこで，本システムでは left-deep tree を採用することにより，結合操作に使用するテーブルを 1 度の参照で済ませる．また，結合操作の過程でできるリレーション（中間リレーション）のサイズを抑えるため，式（A.1）を用いて，中間リレーションのサイズが最小になる結合順序を導き出す．

システム内部の処理の流れは図 2 に上記の操作を加えたものであり，図 3 に示す．

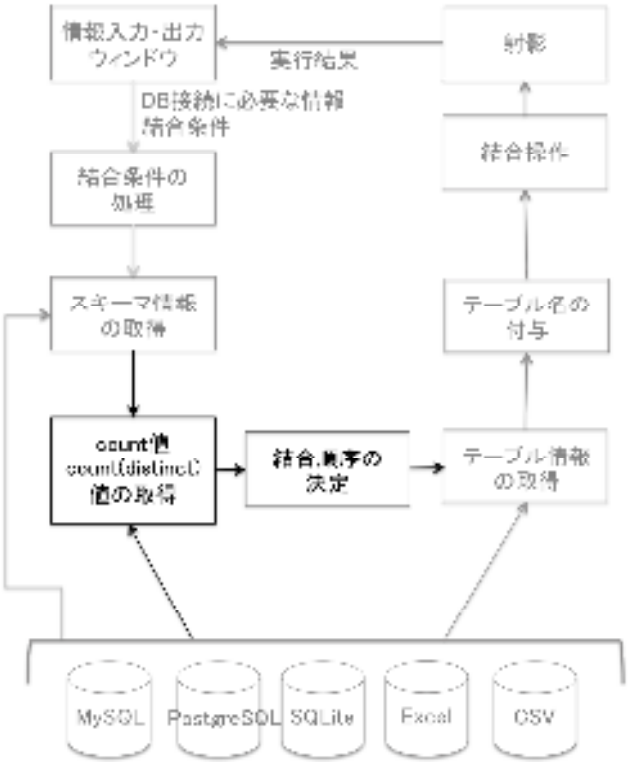

図 3　統合支援システムにおける処理の流れ

(2)　集合演算処理

次に，集合演算を行うためにはあらかじめタプルがソートされていると都合が良い[6]．また，集合演算において，最大で両リレーションのタプル数の和の結果を保持する必要があり，メモリ量の不足が考えられる．そこで，集合演算を実行するためには，以下の条件を設ける．

条件1.　複数データベースの等結合を行うクエリ同士の集合演算の実行は不可とする．

条件2.　Excel，CSV ファイルに関しては，あらかじめ集合演算を行うカラムの順にソートしているものとする．

条件3.　クエリを()でくくることは不可とする．

条件4.　集合演算子の混合は不可とする．

条件 1 は複数データベースの等結合を行った最終結果を複数持つことで，メモリ使用量が増大してしまうことを防ぐためである．条件 2 は，システム内部で使用している Excel と CSV の JDBC がソートに対応していないためである．条件 3 は，クエリの構文に制約を加えることで，クエリ分解時の処理数を減らすためである．条件 4 は，条件 3 より集合演算の優先順位を指定することができないためである．

3.1.2. 実装

本システムにおいて，left-deep tree を採用することで，結合操作に使用するテーブルは 1 度参照するだけで済むようになった．そのため，結合操作に使用するテーブルの情報をあらかじめ ArrayList に保持する必要がなくなった．本システムではソート・マージ結合を採用しているため，初めに結合操作で使用する結合列で ORDER BY することにより，直接 ResultSet から情報を抽出できる．しかし，本システムで採用している Excel，CSV の JDBC は ORDER BY 句に対応していないため，Excel，CSV ファイルの場合は，ソートするために ResultSet から ArrayList にデータを取り出す必要がある．

3.2. JDBC 化

3.2.1. 設計

本 JDBC では複数データベースへのコネクションを持つ必要があり，そのためには複数データベースへの接続情報を登録・保持するクラスが必要である．そのため，JDBC の基本クラス[7]である Connection クラス，Statement クラス，ResultSet クラスの他に，接続情報を保持する JDBC クラスを作成する．

また，各データベースへの接続情報に alias を設定することで，同じ種類のデータベースでも複数登録ができるようにする．ユーザは登録した alias を用いて問い合わせを行うことになる．

JDBC クラスが持つ public なメソッドを表 1 に示す．

3.2.2. 実装

接続情報を複数登録・保持する JDBC クラスを実装する．表 1 に示した各接続情報を登録･削除する public メソッドを実装する．ユーザは上記のメソッドで登録した alias を用いて問い合わせを行うことになる．システムはこれ以降の全ての動作において，alias を用いて各データベースを判断する．そのため，同じインスタンス内では同じ alias 名を登録することはできない．

また，本 JDBC では問い合わせを行う場合，「<Alias>:<TableName>.<ColumnName>」の形式で入力する必要がある．各要素を区分するために “:”を用いているため，alias には“:”を使用できない．

3.2.3. 使用例

(1)では JDBC を使用する際の接続情報の登録・Statement の作成，Connection の作成例を示し，(2)では結合を行う場合の使用例，(3)では集合演算を行う場合の仕様例を示す．

(1)　JDBC の使用例

本 JDBC の使用例を図 4 に示す．図 4 では，1 行目で，JDBC クラスのインスタンスを作成している．2～4 行目で接続情報を登録している．これによりインスタンス jdbc は alias が“M1”，“S1”，“S2”である接続情報を保持する．5 行目で登録した接続先への connection を持つ Connection クラスのインスタンスを作成している．6 行目で Statement クラスのインスタンスを作成している．

表 1　JDBC クラスの持つ public なメソッド

メソッドの概要	
返り値	説明
boolean	set_MySQL(String dbname, String host, String username, String pass, String alias) 指定されたデータベース，ホスト，ユーザ名，パスワード，alias で MySQL の接続情報を登録するメソッド．接続できない場合，SQLException を返す．また，alias が既に登録されているものと同じ場合は SQLException を，未登録の場合 true を返す．
boolean	set_PostgreSQL(String dbname, String host, String username, String pass, String alias) 指定されたデータベース，ホスト，ユーザ名，パスワード，alias で PostgreSQL の接続情報を登録するメソッド．接続できない場合，SQLException を返す．また，alias が既に登録されているものと同じ場合は SQLException を，未登録の場合 true を返す．
boolean	set_SQLite(String dbname, String location, String alias) 指定されたデータベース，位置，alias で SQLite の接続情報を登録するメソッド．alias が既に登録されているものと同じ場合は SQLException を，未登録の場合 true を返す．
boolean	set_DB2(String dbname, String host, int port, String username, String pass, String alias) 指定されたデータベース，ホスト，ポート番号，ユーザ名，パスワード，alias で DB2 の接続情報を登録するメソッド．接続できない場合，SQLException を返す．また，alias が既に登録されているものと同じ場合は SQLException を，未登録の場合 true を返す．
boolean	set_Excel(String filename, String location, String alias) 指定されたファイル名，位置，alias で Excel の接続情報を登録するメソッド．alias が既に登録されているものと同じ場合は SQLException を，未登録の場合 true を返す．
boolean	set_CSV(String location, String alias) 指定された位置，alias で CSV の接続情報を登録するメソッド．alias が既に登録されているものと同じ場合は SQLException を，未登録の場合 true を返す．
boolean	remove_DB(String alias) 指定された alias で登録された接続情報を探し，登録情報から削除するメソッド． 接続情報が存在する場合は true，存在しない場合は SQLException を返す．
Connection	createConnection() 上記のメソッドで事前に設定した接続情報を用いて，各 RDBMS・ファイルへの接続を行うメソッド．接続できた場合，Connection オブジェクト，接続できない場合，SQLException を返す．

```
1:  JDBC jdbc = new JDBC();
2:  jdbc.set_MySQL(“testdb”, “localhost”, “testuser”, “test”, “M1”);
3:  jdbc.set_SQLite(“testdb”, “/Users/Test”, “S1”);
4:  jdbc.set_SQLite(“testdb2”, “/Users/Test/Document”, “S2”);
5:  Connection con = jdbc.createConnection();
6:  Statement stmt = con.createStatement();
7:  jdbc.remove_DB(“S1”);
8:  jdbc.set_Excel(“test.xlsx”, “/Users/Test/Desktop”, “E1”);
9:  Connection con2 = jdbc.createConnection();
10: Statement stmt2 = con2.createStatement();
11: ResultSet rs1 = stmt.executeQuery(“select * from M1:test, S1:test where M1:test.id = S1:test.id”);
12: ResultSet rs2 = stmt2.executeQuery(“select S2:test.tid, S2:test.tname from S2:test UNION select M1:test.id,
M1:test.name from M1:test);
```

図 4　JDBC の使用例

結合演算の問い合わせ文…「select <SELECT_LIST> from <TABLE_LIST> where <WHERE>」−①
<SELECT_LIST>…「<Alias>:<TableName>.<ColumnName>, <Alias>:<TableName>.<ColumnName>, …」or「*」
<TABLE_LIST>…「<Alias>:<TableName>, <Alias>:<TableName>,…」
<WHERE>…「<Alias>:<TableName>.<ColumnName> = <Alias>:<TableName>.<ColumnName> (AND …)」
※複数条件の場合 AND で条件を繋ぐ．

図 5　結合演算を行う場合の問い合わせ構文

集合演算・和の問い合わせ文…図 5 の① UNION 図 5 の① UNION …
集合演算・積の問い合わせ文…図 5 の① INTERSECT 図 5 の① INTERSECT 図 5 の① INTERSECT …

図 6 集合演算を行う場合の問い合わせ構文

また，接続情報を削除する例を続いて示す．7 行目のように remove_DB()メソッドを使用し接続情報を削除している．これにより，インスタンス jdbc から alias が“S1”である接続情報が除去される．次に，8 行目で alias が“E1”である接続情報を新たに追加している．これにより，インスタンス jdbc は alias が“M1”，“S2”，“E1”である接続情報を保持する．9 行目で登録した接続先への connection を持つ Connection クラスのインスタンスを作成している．

(2) 結合

結合演算を行う場合の問い合わせ文の形式を図 5 に示す．<Alias>は事前に登録した alias，<TableName>はテーブル名，<ColumnName>はカラム名を表す．結合演算を行う場合の例を図 4 の 11 行目に示す．ここでは事前に登録した alias が M1 である MySQL と S1 である SQLite の等結合を行っている．

(3) 集合演算

集合演算を行う場合の問い合わせ文の形式を図 6 に示す．和集合を求める場合は図 5 で示した問い合わせ文を「UNION」で繋ぎ，積集合を求める場合は「INTERSECT」で繋ぐ．和集合を求める例を図 4 の 12 行目に示す．ここでは事前に登録した alias が S2 である SQLite のテーブル test のカラム tid, tname と alias が M1 である MySQL のテーブル test のカラム id, name の和集合を求めている．

4. 実験

筆者らが以前作成した異種情報源統合システムと，今回作成したシステムのメモリ使用量を比較する．以降，以前作成したシステムを旧システム，今回作成したシステムを新システムと記す．

4.1. 実験方法

メモリ使用量とシステム実行時間を 6 回，実機（Windows 8.1 Pro，2.93GHz Intel Core 2 Duo, 4GB Memory）にて計測する．結果の単位は KB と msec である．

実験に用いた MySQL，PostgreSQL，SQLite のテーブルの仕様を以下に示す．

1) 行数は 10,000 行．
2) カラムは id，name，loc を持つ．
3) カラム id には 1 から 10,000 の数が昇順に入る．
4) カラム name には長さ 20 のランダムな文字列が入る．
5) カラム loc には長さ 40 のランダムな文字列が入る．

Excel のシートの仕様を以下に示す．

1) 行数は 5 行．
2) カラムは id，evaluate，test を持つ．
3) カラム id には 1 から 5 の数が昇順に入る．
4) カラム evaluate には長さ 3 もしくは 4 の文字列が入る．
5) カラム test には 2，3，5 の整数が 1 つ入る．

CSV ファイルは，Excel のカラム test を除いたもので構成される．全テーブルはインデックス付けを行っていない．MySQL，PostgreSQL，SQLite のテーブルはプログラムで作成した．

結合条件は前から順に MySQL, PostgreSQL, SQLite, Excel, CSV の順に結合するように書くこととし，カラム id で等結合を行うものとする．これにより，旧システムでは結合条件を前から順に実行するため，MySQL（10,000 行）と PostgreSQL（10,000 行），その結果と SQLite（10,000 行），その結果と Excel（5 行），その結果と CSV（5 行）の順に結合を行う．つまり，中間リレーションのサイズ推移は 10,000 行, 10,000 行, 5 行, 5 行となる．新システムでは中間リレーションが小さくなるような結合順序で結合を行うため，中間リレーションのサイズ推移は 5 行，5 行，5 行，5 行となる．

実験1. システムの結合操作実行時間を計測する．

実験2. 結合操作終了時点でのシステム全体で使用するメモリ使用量を計測する．

4.2. 実験結果

実験1. 図 7 にシステムの実行時間を示す．全ての回において，新システムの実行時間が，旧システムよりも短くなっている．

実験2. 図 8 に結合操作が終了した時点でのシステム全体で使用しているメモリ使用量を示す．全ての回において，新システムのメモリ使用量が，旧システムよりも少なくなっている．

4.3. 考察

A) 結合操作実行時間

結合操作実行時間の平均は旧システムでは 91.6[msec]であり，新システムでは 6[msec]である．旧システムでは 10,000 行と 10,000 行の結合操作が 2 回あるのに対し，新システムではその部分が 5 行と 10,000 行の結合操作になっている．比較する行数が少ないほど，結合操作にかかる時間は短くなると考えられる．結合する

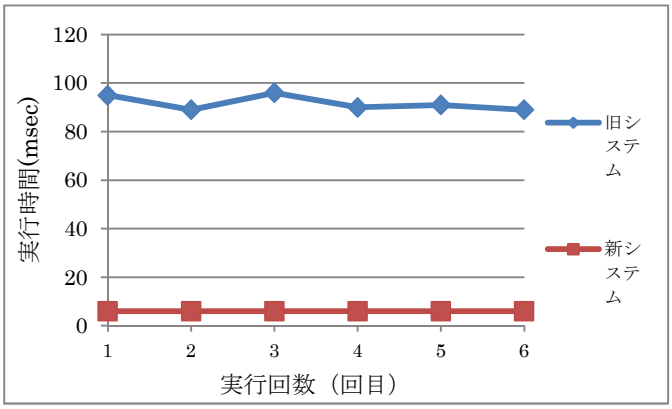

図 7 システムの実行時間

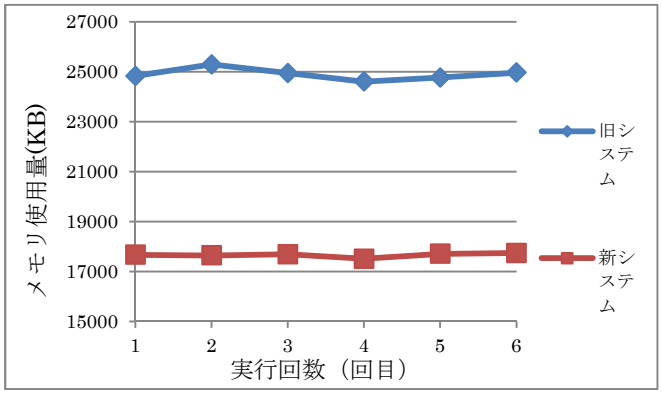

図 8　システムのメモリ使用量

リレーションのタプル数を S，T とすると，ソート・マージ結合の時間計算量は O((S+T)log(S+T))で表される．旧システムでの時間計算量は $O(20{,}000log(20{,}000)*2+10{,}005log(10{,}005)+10\log(10))$であるのに対し，新システムでは $O(10{,}005log(10{,}005)*3+10log(10))$である．そのため，結合操作自体にかかる時間は削減できていると考えられる．

B)　メモリ使用量

図 8 より，新システムのメモリ使用量が旧システムより大幅に少なくなっている．これは，旧システムでは全リレーションのデータ，および，中間リレーションのデータを全て保持しているのに対し，新システムでは使用したデータを全て破棄していることから大幅に下がったと考えられる．

5. まとめ

本論文では，これまでに試作してきた異種情報源統合システムにおいて，メモリ使用量が多いこと，GUI も含めたシステムになっており汎用性が高くないこと，という 2 つの問題点を結合結果のサイズ推定式を用いて適切な結合順序を決定すること，異種情報源の統合機能を持つ JDBC を作成し，GUI 部分と分離することで解決した．実機による評価の結果，結合操作の実行時間を増やすことなく，メモリ使用量の大幅な削減ができることを示した．

現在の JDBC では MySQL, PostgreSQL, SQLite, DB2, Excel, CSV の利用に限られている．また，集合演算において()を使用できず，集合演算子は 1 種類のみの使用に制限されている．この制限を取り払うことで，より様々な用途に利用してもらえると考えている．そのため，この制限を取り払うことが今後の課題である．

参　考　文　献

[1] H. Garcia-Molina, Y. Papakonstantinou, D. Quass, A. Rajaraman, Y. Sagiv, J. Ullman, V. Vassalos and J. Widom, “The TSIMMIS Approach to Mediation: Data Models and Languages”, Journal of Intelligent Information Systems, vol8, no.2, pp,117-132, 1997.

[2] X. Wang, T. Hochin and H. Nomiya, “Feasibility of Unified Usage of Heterogeneous Databases Storing Private Information”, Proc. of 1st ACIS International Symposium on Applied Computing & Information Technology (ACIT 2013), pp.337-342, 2013.

[3] A. Terakawa, T. Hochin and H. Nomiya, “Integrated Usage of Heterogeneous Databases for Novice Users”, Proc. of International Conference on Software Engineering Research, Management, and Applications (SERA2014), pp. 705-710, 2014.

[4] DK. Shin and AC. Meltzer, “A New Join Algorithm”, ACM SIGMOD Record, vol.23, no.4, pp.13-20, 1994.

[5] H. Garcia-Molina, J. Ullman and J. Widom, “Database Systems The Complete Book”, Pearson Education, pp. 826-832, 847-856, 862-864, 2002.

[6] A. Silberschatz, H. Korth and S. Sudarshan, “Database system concepts 4th edition”, McGraw-Hill Education, pp.515-516, 2002.

[7] Lance Andersen and Specification Lead, “JDBC™ 4.2 Specification”, 2014.

A.　付録

A.1. 結合結果のサイズ推定

属性 X,Y を持つリレーション R を R(X,Y)と表し，リレーションのタプル数を T(R)，R の属性 X の distinct 値を V(R,Y)と表す[5].

R(X,Y)と S(Y,Z)を属性 Y で等結合したサイズは以下のようにして推定することができる．

$V(R,Y) \leq V(S,Y)$と仮定する．

1. R の全タプルが，与えられた S のタプルと結合する確率は$1/V(S,Y)$.
2. S は T(S)タプルあるため，結合されるタプル数の期待値は$T(S)/V(S,Y)$.
3. R は T(R)タプルあるため，R と S を等結合した時の結合推定サイズは$T(R)T(S)/V(S,Y)$.

一般的には V(R,Y)と V(S,Y)の大きい方で割ることで推定サイズを求めるため，一般式は以下のようになる．

$$T(R \bowtie S) = T(R)T(S)/\max\big(V(R,Y),V(S,Y)\big) \quad (A.1)$$

Y がいくつかの属性を表すと仮定する．R(x, y1, y2)と S(y1, y2, z)の結合サイズは以下のようにして推定する．

$$\frac{T(R)T(S)}{\max\big(V(R,y1),V(S,y1)\big)\max\big(V(R,y2),V(S,y2)\big)} \quad (A.2)$$

A.2. Left-deep tree

木の形は図 A.1 に示すように 3 種類ある[5].

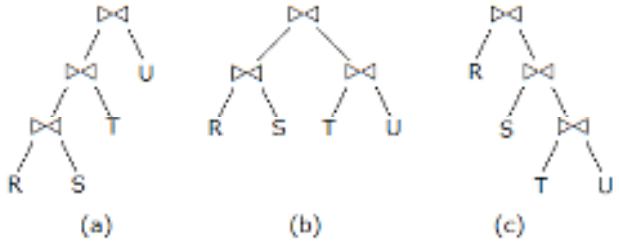

図 A.1 木の形

(a)を left-deep tree，(b)を bushy tree，(c)を right-deep tree と呼ぶ．結合順序を left-deep tree に制限することで次のような利点がある．

1. 木の形を制限することで探索数が減る．
2. 一般的な結合アルゴリズム(特に nested-loop join, one-pass join)では非 left-deep tree を使った同じアルゴリズムよりも，left-deep tree を使った方が，効率が良い傾向にある．