データベースシステム 101－4
（1995. 1. 26）

PERCIO/C++ の宣言的問合せ機能とその実現

波内 みさ　　木村 裕
C&C 研究所　　基本ソフトウェア事業部
NEC

本稿では、オブジェクト指向データベース操作言語 PERCIO/C++ に導入した宣言的問合せ機能と、その処理を行う問合せ処理系について述べる。PERCIO/C++ は C++ の構文を拡張した DDL および DML であるが、宣言的な問合せを記述可能とすることにより、SQL 以上の問合せ記述能力を持つ。また、その宣言的問合せの解釈・実行を行う問合せ処理系は、問合せ最適化手法の拡張性を考慮し、オブジェクト指向の枠組により設計・実現されている。これにより、オブジェクト指向データーベースに対して提案されている様々な最適化手法を柔軟に取り込むことが可能となる。

Declarative Facility for PERCIO/C++ and Its Implementation

Misa Namiuchi　　Yutaka Kimura
C&C Research Laboratories　　Basic Software Division
NEC Corporation

This paper presents a declarative query facility introduced into an object-oriented database manipulation language, PERCIO/C++, and a query processor which processes the query. PERCIO/C++ includes both DDL and DML based on C++ language extension, and has a high descriptive power by extending C++ syntax for a declarative query. The proposed query processor has been designed and implemented in the object-oriented paradigm, considering extensibility of query optimization methods. In this point of view, the query processor can easily adopt various new optimization methods proposed in the world.

1 はじめに

オブジェクト指向データベース(OODB)の実運用が進んでいる昨今、そのデータベース操作言語は、C++のような汎用のプログラミング言語拡張系、SQL拡張系に大別される。プログラミング言語拡張系には元言語の持つ広範な機能が継承されているが、高度な問合せを記述するための関数が用意されている例は少なく、比較的低レベルの関数を組み合わせて利用する場合が多い。一方、SQL拡張系ではSQLのselect文を利用できるため、宣言的でありかつ可読性の高い問合せを記述することが出来る。しかしSQLの機能は限られており、実際のアプリケーション開発時には、開発言語とのインピーダンス・ミスマッチの問題は依然解決されない。

本稿では、オブジェクト指向データベース操作言語PERCIO/C++に導入した宣言的問合せ機能と、その処理を行う問合せ処理系について述べる。

2 PERCIO/C++の宣言的問合せ

PERCIO/C++は、データベースの操作記述が容易なようにオブジェクト指向言語C++の仕様を拡張した、オブジェクト指向データベース管理システムPERCIO ([7], [8]) 用のデータベース操作言語である。PERCIO/C++では、オブジェクトの集合をコレクションクラス(Od_Collection)のサブクラスで表現する。これらのクラスは、集合(Od_Set)、バッグ(Od_Bag)、エクステント(Od_Extent)、リスト(Od_List)、可変長文字列(Od_String)、可変長配列(Od_Varray)がある。このうちエクステントは、あるクラスに属するすべてのオブジェクト(インスタンス)を保持する集合である。

コレクションクラスには、集合の要素を直接扱うメソッド(例: `insert_element`)、オブジェクト間の演算(例: `intersect`)や条件検索のためのメソッド(`select / join / group`)が定義されている。PERCIO/C++による問合せは、このコレクションクラスのメソッド(メンバ関数)で行う。

一般に条件検索では、少なくとも条件の対象となる複数のコレクションと条件式を指定する必要がある。この指定をCやC++の構文にそのまま反映させるためには、可変個のコレクションオブジェクトと文字列として表現した条件式をそれぞれ引数としてselect関数に渡すか、それらを構造体に詰め込んで渡す必要がある。これらの記述は、プログラムのステップ数を増やし読解性を悪くする。また、条件式が文字列として表現されてしまうため、文字列内の式の構文の意味検査が行われない。このため、実行時のエラーやDB破壊が生じる恐れがある。

上記の問題点を解決するために、PERCIO/C++ではC++の構文を拡張し、「ブロック式」と呼ぶ構文を導入した。図1にその構文規則を示す。

ブロック式とは、ブロック演算子[ ]で囲まれた式の集まりのことである。この式は、大きく2つの部分よりなる。まず最初に、条件式の対象となるコレクションオブジェクトとそれに対応したオブジェクト変数を演算子: を用いて宣言する。演算子: の左辺にコレクションオブジェクト、右辺にオブジェクト変数を指定する。オブジェクト変数は、関係データベースにおけるタプル変数に相当し、左辺に指定されたコレクションをドメインとして、そのコレクションの各要素オブジェクトを値として保持する。この変数に保持された値を使って、次に続く条件式が評価される。

オブジェクト変数宣言の後は条件式部分であり、キーワード`where`の後に記述する。SQLでは条件式には組み込みの演算子しか許されず、また他社OODBでは結合演算が出来なかったり、C++の任意の演算子が使えないなどの制限がある。これに対し、PERCIO/C++ではC++で許されている演算がほとんどすべて、条件式中に記述可能である。これによりPERCIO/C++は、他言語に比べ高い記述能力を持つ。

```
1: Od_Extent<ホテル *> inn;
2: inn.select ([this: S;
3:      where S-> 客室数 >= 200
4:         && strcmp (S-> 最寄り駅 -> 名前,
5:                    "湯沢 ")==0; ]);
```

この例では、行1でホテルクラスのエクステントオブジェクトを定義している。このエクステントオブジェクトは、ホテルクラスのインスタンス群を保持する。selectの引数のブロック式では、まずオブジェクト変数を宣言する(行2)。左辺のthisは、エクステントへのポインタであり、オブジェクト変数Sはこの要素を値として保持する。where文は選択条件式である(行3-5)。行4では関数strcmp()を使用している。このブロック式は、エクステントの個々の要素であるオブジェクトをオブジェクト変数Sに束縛し、whereの条件式を適用する。

3 問合せ処理系の実現

3.1 問合せ処理系の設計

OODBにおける問合せ最適化手法には、以下のような課題があり、それぞれに対して様々な研究が

```
block-expression:
        [ block-statement_opt ]

block-statement:
        sj-block-statement

sj-block-statement:
        block-declaration-list b-compound-statement_opt
        b-compound-statement

b-compound-statement:
        b-statement b-compound-statement
        b-statement

b-statement:
        where-statement
        order-statement

block-declaration-list:
        block-declaration
        block-declaration-list block-declaration

block-declaration:
        expression bind-operator object-variable-list ;

object-variable-list:
        object-variable
        object-variable-list , object-variable

order-statement:
        order_by order-decl-list ;

order-decl-list:
        order-declaration
        order-decl-list , order-declaration

order-declaration:
        order-specifier expression
        expression

order-specifier:
        od_asc
        od_desc

where-statement:
        where expression ;

bind-operator:
        :
```

図 1: ブロック式の構文定義

行われている。

- navigation の探索空間の絞り込み
- RDB における問合せ最適化手法の利用 ([6])
- heuristics の取り込み ([4])
- オブジェクトの等価性の取り扱い ([5])
- ユーザ定義メソッドの取り扱い ([2],[3])

また、任意のユーザ定義の型の取り扱いに際し、それぞれの型ごとの最適化情報の管理を行う必要性もある。したがって、OODB における問合せ処理系は、問合せ最適化手法および最適化情報の管理に柔軟な拡張性を持つことが望ましい。

以上の要因を考慮し、本問合せ処理系を設計するに当たって以下の方針を採った。

- 問合せ処理系自身がオブジェクト指向モデルに基づくこと
- 問合せ最適化方式の適用・管理を各構成オブジェクトに委託し、拡張可能性を持たせること

これらの方針に基づき、問合せ実行制御器、問合せ解析器、問合せ最適化器の三つのモジュールから構成される処理系を設計した (図 2)。それぞれのモジュールの内部情報および操作手続きは、オブジェクト指向モデルにおけるクラスとして定義され、そのインスタンスにより実際の処理が行われる。

3.2 問合せ処理系の概要

本問合せ処理系は、問合せ実行制御器、問合せ解析器、問合せ最適化器の三つのモジュールから構成される。

問合せ実行制御器は、実際の問合せ処理の実行を制御する。ここでは、与えられた問合せをまず問合せ解析器に渡してその内部表現を得、次に問合せ最適化器により問合せ評価計画 (Query Evaluation Plan: QEP) を得た後、実際の検索処理を開始する。実行制御器に検索処理実行条件を与えることにより、最適化器のコスト予測値による検索処理制御 (検索実行の中止など) も可能である。

問合せ解析器は、与えられた問合せ条件を解析し、属性値の制約として表現されたオブジェクトの条件およびオブジェクト間の関連・制約を処理系内の内部表現に変換する。この内部表現には、DB オブジェクトの 1 クラスをノードとする循環を含むグラフを用いる。これを以下、条件グラフとよぶ。

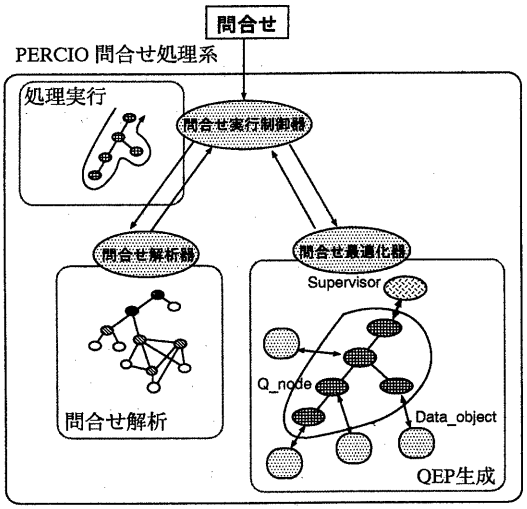


図 2: 問合せ処理系の構成

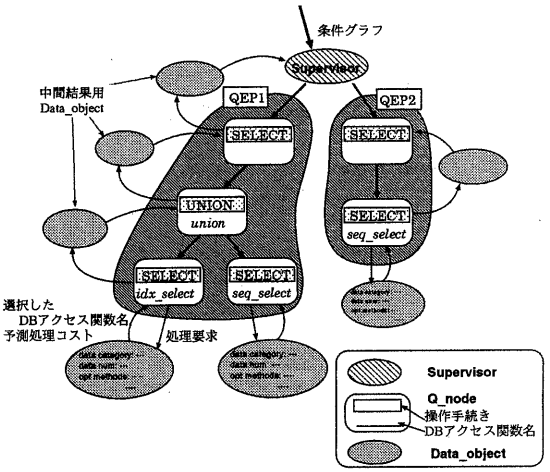


図 3: 問合せ最適化器の構成

問合せ最適化器は、問合せ解析器によって生成された問合せの内部表現から QEP を生成する。この時、応答時間最小、メモリ使用量最小などの最適化基準のうち、予め設定された基準に従って QEP の最適化を行う。本最適化器では、これらを実現するために、DB 中のデータの物理情報 (格納形式、オブジェクト数、インスタンス変数に対するインデックスの有無など) や統計情報 (選択率など)、検索処理を行う計算機の性能情報 (命令実行速度、I/O 速度、利用可能メモリ量など) 等を利用して、問合せ実行コストの予測計算を行う。

問合せ実行制御器、問合せ解析器、問合せ最適化器は、問合せがネストしていない通常の場合にはシステムに一つずつ存在するが、問合せがネストした場合に複数生成され、それぞれの副問合せに対して QEP を生成する。

3.2.1 問合せ最適化器

本問合せ最適化器は、最適化処理を、(i) 問合せ木の構築、(ii) 問合せ木の各ノードで実行する DB アクセス関数 (DB オブジェクトを操作するために DBMS により提供されている下位の関数) の決定、の二つのフェイズに分け、これを三種類のモジュール ("Supervisor", "Q_node", "Data_object") によって実現する。これらのモジュールはそれぞれ一つのクラスとして実現し、各段階の最適化処理をそれらのメソッドとして定義することによって、最適化戦略の拡張性を実現する。

Supervisor クラスのインスタンスは一つの問合せに対して一つだけ存在し、QEP の候補となる問合せ木を生成し管理する。Q_node クラスのインスタンスは、その問合せ木の各ノードとなり、問合せ木を構成する。一つの Q_node で実行される処理は、元の検索条件の一部分 (ここではこれを「部分問合せ」とよぶ) の処理に相当する。Data_object クラスのインスタンスは、最適化に用いる情報と部分問合せの最適化戦略を管理する。Data_object は、扱う対象となるデータのメディアの種類に応じたサブクラスを持ち、それぞれのメディア特有の最適化情報を管理する。

最適化器では、まず Supervisor により、問合せ解析器において生成された条件グラフを基にいくつかの問合せ木が生成される。そして、問合せ木上の Q_node とそれに対応する Data_object が最適化処理情報をやり取りすることによって、各 Q_node が部分問合せの処理実行に利用する DB アクセス関数が決定され、同時にその予想処理コストが計算される。

この処理をすべての候補の問合せ木について実行することによって、最適化器では最終的に、複数個の QEP とその予想処理コストが得られる。これらの中から、問合せ実行制御器においてその予想処理コスト値をもとに一つだけが選択され、それ以外は破棄される。ここで選択された QEP を実行することにより、実際の検索処理が行われる (図 3)。

3.3 問合せ処理例

DB中に次のクラス「ホテル」、「リゾート」、「温泉」、「スキー場」、「駅」に属するオブジェクトが存在するとする。

```
class ホテル : public Od_Pobject{
public:
    persistent char*        名称;
    int                     客室数;
    Od_Set<リゾート *>      近隣リゾート地;
    駅 *                    最寄り駅;
};

class リゾート : public Od_Pobject{
public:
    persistent char*        名称;
    駅 *                    最寄り駅;
};

class 温泉 : public リゾート {
public:
    char                    効能 [1024];
};

class スキー場 : public リゾート {
public:
    unsigned int            ゲレンデ数;
};

class 駅 : public Od_Pobject{
public:
  char 名前 [16];
};
```

このDBに対し、次のような二つの問合せが発行された場合を考える。

問合せ 1 ゲレンデ数5以上のスキー場と温泉が近くにあり、電車で行ける、客室数100以上のホテルを探し、ホテルのデータを出力する

問合せ 2 ゲレンデ数5以上のスキー場と温泉が近くにあり、電車で行ける、客室数100以上のホテルを探し、ホテルとスキー場、温泉のデータを出力する

二つの問合せは、検索条件は同じであるが出力が異なっている。これらに対して、PERCIO/C++によって記述した問合せの例を、図4に示す。問合せ2で利用しているメソッドjoinは、検索条件を満たすすべてのオブジェクトの組合せを生成し、それを格納した集合を返却するものである。

以下、この問合せに対する処理を示す。

(1) 条件グラフの生成

問合せ処理系は、問合せ解析器を生成して問合せの内部表現である条件グラフを生成する。問合せ1, 2の検索条件は同じなので、問合せ解析器はどちらを受け取った場合にも、図5に示す条件グラフを生成する。

(2) QEP 候補の生成

次に問合せ処理系は、問合せ最適化器を生成する。問合せ最適化器は、条件グラフから図6, (1),(2)の二つのQEP候補を生成する。ここで、図中の同じ番号を持つQ_nodeは、同じ検索処理を行うものとする。

QEP候補1とQEP候補2は、Q_node 4の入力となる集合が違い、これによりQEP候補1には、最終結果を生成するQ_node 5が追加されている。

(3) QEP の選択

問合せ最適化器では、得られたQEP候補に対して、DBの物理情報・統計情報を基に処理コストの予測計算を行う。物理情報には、稼働マシンにおける平均I/O時間、演算速度などの値を使い、統計情報としては、DB中の各コレクションの各属性に対し、値の最大値、最小値、平均値、属性が集合型の場合には平均要素数などの値を用いる。

まず、問合せ1の場合の処理コストを考える。この場合、QEP候補1とQEP候補2の処理コストを式で表すと、次のようになる。Q_node 4の処理コストを表す関数 F(n) は、nの数に比例して値が大きくなるものとする。

```
QEP 候補 1 の処理コスト =
        Q_node 1, 2, 3 の処理コスト
          + F(Q_node 1 での結果数)
          + Q_node 5 の処理コスト
QEP 候補 2 の処理コスト =
        Q_node 1, 2, 3 の処理コスト
          + F(Q_node 3 での結果数)
```

Q_node 3の出力集合数は、Q_node 1の出力集合数と比べ、少ないかまたは同じである。

```
Od_Extent<ホテル*> hotel_ext;
Od_Extent<スキー場*> ski_ext;
Od_Extent<温泉*> spa_ext;

// 問合せ 1
Od_Collection<ホテル*>* hotel_set =
  hotel_ext.select([ this: H; &ski_ext: R; &spa_ext: S;
                     where H->客室数 >= 100 && H->最寄り駅 != Od_NULL && H->最寄り駅 == R->最寄り駅 &&
                           R->ゲレンデ数 >= 5 && H->近隣リゾート地.contains_element(S); ]);
// 問合せ 2
Od_Collection<Od_Join<ホテル*, スキー場*, 温泉*>*>* resort_pack_set =
  hotel_ext.join([ this: H; &ski_ext: R; &spa_ext: S;
                   where H->客室数 >= 100 && H->最寄り駅 != Od_NULL && H->最寄り駅 == R->最寄り駅 &&
                         R->ゲレンデ数 >= 5 && H->近隣リゾート地.contains_element(S); ]);
```

図 4: PERCIO/C++ による問合せ例

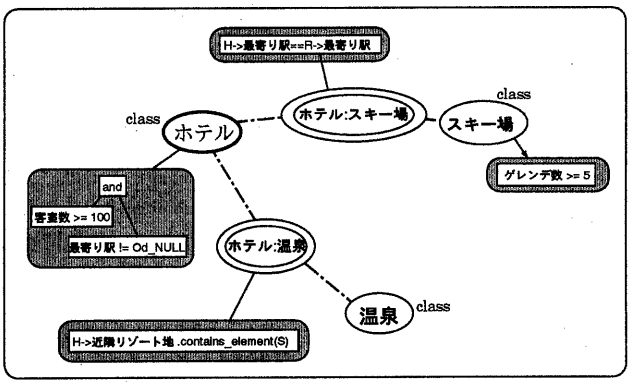


図 5: 条件グラフ

Q_node 1 での結果数 ≧ Q_node 3 での結果数

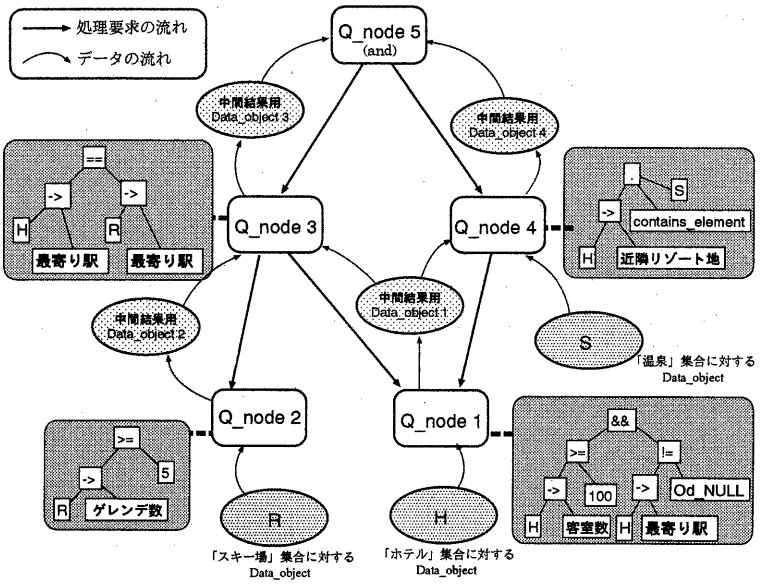
したがって、問合せ 1 の場合、QEP 候補 2 の処理コストの方が小さくなると予想されることから、QEP 候補 2 が選択される。

問合せ 2 の場合は、Q_node 3, 4 および Q_node 5 で結合演算が行われ、その結果、複数集合の要素の組合せが中間結果として生成される。QEP 候補 1 と QEP 候補 2 の処理コストの計算式は問合せ 1 の場合と同じだが、Q_node 1、Q_node 3 での結果数と Q_node 5 での処理時間如何で、両者間での処理コスト予測値の大小関係が変わってくる。Q_node 3 での結果数が Q_node 5 での結果数を上回り、それによる処理コストが Q_node 5 での結合演算処理の予測コストを上回った場合、問合せ最適化器は、QEP 候補 1 を QEP として選択する。

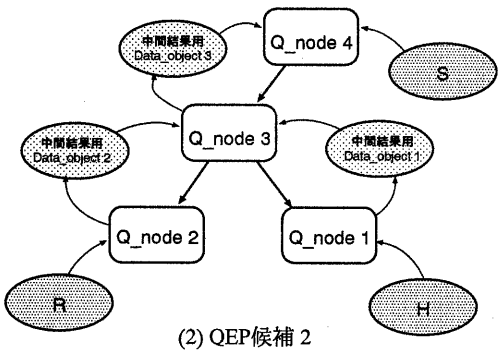
DB の統計情報が得られていない場合には、処理コスト予測計算による QEP 選択は行わず、次のような heuristics を導入して選択を行う。

- 利用する Q_node の数 (中間結果の数) はなるべく少なくする
- 一つの集合に対する複数の検索条件は、一度に (= 一つの Q_node によって) 評価する
- インデックスが利用可能な場合は、優先して利用する

この方針にしたがった場合、問合せ 1,2 両者に対



(1) QEP候補 1



(2) QEP候補 2

図 6: QEP 候補

して、 QEP 候補 2 が選択される。

3.4 処理系の実現

以上の問合せ処理系の設計を基に、オブジェクト指向データベース管理システム PERCIO の問合せ処理系を実装した。設計・実装の過程で明らかとなった本処理系の問題点・課題を以下に示す。

QEP の生成

本問合せ処理系では、まず問合せを内部表現である条件グラフに変換する。この条件グラフの生成段階では、同じ集合に対する条件をまとめるなどの heuristic な最適化を実現している。しかし、条件グラフという内部表現を生成し、さらに問合せ木生成を行うという二段階の操作は、処理上の冗長性を含んでいるので、この冗長性を可能な限り排除することが今後の検討課題である。また、条件グラフの段階での代数的最適化や意味的最適化も、最適化手段として今後考慮する必要があると思われる。

QEP 生成手法は、条件グラフから QEP 候補となる問合せ木を生成する過程の最適化と DB アクセス関数の選択に分類し、それぞれ、問合せ最適化器の Supervisor と Data_object のメソッドとして実現した。 QEP 生成の二つの過程を分離し、それぞれの処理を分担するモジュールを分割することによ

り、処理系の実現および管理を容易にした。しかしこれにより、生成される問合せ木すべてについてDBアクセス関数決定とコスト計算を行わなくてはならない。この両者の融合によって、生成する問合せ木の数を減少させることも検討中である。

メソッドを含む問合せのコスト計算

コスト計算による問合せ最適化を行う場合、メソッドの実行コストをどのように見積もるかが問題となる。メソッドを含む問合せの処理方式として、現在、次の方式が提案されている。

- メソッド定義を問合せ中に展開し、コスト計算を行う
- メソッドの処理コストが関数として与えられることを前提とする

本処理系では、このうち後者の方式を採用した。あらゆるメソッドについて、マシン依存の処理コストを与えることは一般に困難であるが、開発が終了して実稼働するアプリケーションでは、コストを実測し関数化することが可能である。コスト関数は、DBの統計情報を測定する場合に同時に測定・設定するよう、自動化することも考えられる。

コスト計算による最適化

本問合せ処理系では、コスト計算を行うために、DBの統計情報と稼働マシンの物理情報を利用する。このうち、統計情報を設定するためには、DB中のオブジェクトすべてにアクセスし、さらにその属性値に対して演算を行わなければならず、DBが大規模になるにつれ、その作業コストは非常に大きくなる。
PERCIOでは、この統計情報設定機能が保守機能の一つとして用意されている。

しかし、問合せ処理の中間結果の統計情報を予想するために利用する選択率に関しては、その設定のために属性値の値ごとの計算が必要があり、実際の実施は一般に困難である。このため、値の最大値、最小値から一様分布あるいは正規分布を仮定して選択率を計算するが、常にこれが有効であるとは限らない。値の分布に規則性がない場合と値の更新の頻度がある程度を越えるアプリケーションでは、コスト計算以外のheuristicsな問合せ最適化手法を積極的に利用する必要があると思われる。この二つの最適化方式をどのように切替えて利用するかは、今後の課題である。

4 まとめ

PERCIO/C++では、SQLのような宣言的な問合せを可能にするために、C++の言語仕様に「ブロック式」と呼ぶ構文を導入した。これにより、C++で集合検索を直接記述した場合に比べ記述量を減らすことができる。また、その条件式には更新式を除くほとんどすべての式を書くことができ、高い記述能力が実現されている。

さらに、PERCIO/C++の宣言的問合せを評価する処理系を、問合せ最適化手法の拡張性を意識して、オブジェクト指向の枠組で実現した。

今後は、本文中に挙げた問合せ処理系の課題について検討を行っていく予定である。

謝辞

この仕事を進めるにあたり、有益な助言を下さった弊社C&C研究所 鶴岡 邦敏課長に深謝いたします。

参考文献

[1] Beeri, C. and Kornatzky, Y., "Algebraic Optimization of Object-Oriented Query Languages", Proc. 3rd Int. Conf. on Database Theory, LNCS 470, pp.72-88, Dec. 1990.

[2] Gardarin, G. and Lanzelotte, R.S.G., "Optimizing Object-Oriented Database Queries using Cost-Controlled Rewriting", Proc. 4th Int. Conf. on Extending Database Technology (EDBT'92), LNCS 580, pp.534-545, 1992.

[3] Jiao, Z. and Gray, P.M.D., "Optimization of Methods in a Navigational Query Language", Proc. 2nd Int. Conf. on Deductive and Object-Oriented Databases (DOOD'91), LNCS 566, pp.22-42, Dec. 1991.

[4] Kim, K.-C., "Performance of Query Optimization Heuristics in Object-Oriented Databases", Proc. 2nd Int. Symp. on Database Systems for Advanced Applications (DASFAA'91), pp.99-108, Apr. 1991.

[5] Shaw, G.M. and Zdonik, S.B., "Object-Oriented queries: Equivalence and Optimization", Proc. 1st Int. Conf. on Deductive and Object-Oriented Databases (DOOD'89), pp.281-296, Dec. 1989.

[6] Valduriez, P. and Danforth, S., "Query Optimization for Database Programming Languages", Proc. 1st Int. Conf. on Deductive and Object-Oriented Databases (DOOD'89), pp.553-571, Dec. 1989.

[7] 鶴岡、「C++をベースに柔軟さを採り入れた「PERCIO」」、日経インテリジェントシステム別冊『オブジェクト指向データベース - 基礎知識から製品の評価まで -』、pp. 208-215, 日経BP社, 1994.

[8] 鶴岡、木村、「オブジェクト指向データベース PERCIOの機能と構成」、NEC技報、Vol. 47, No. 6, pp. 36-41, 1994.

[9] 波内、木村、「オブジェクト指向モデルによるデータベース問合せ処理系の実現」、第46回情報処理学会全国大会予稿集4, pp. 139-140, 1993.