(19)日本国特許庁(JP)　(12)特　許　公　報(B2)　(11)特許番号
特許第5255000号
(P5255000)
(45)発行日　平成25年8月7日(2013.8.7)　(24)登録日　平成25年4月26日(2013.4.26)

(51)Int.Cl.　F I
G06F 17/30 (2006.01)　G06F 17/30 340D
G06F 17/30 330Z
G06F 17/30 180D

請求項の数 18　(全 64 頁)

(21)出願番号　特願2009-549210(P2009-549210)
(86)(22)出願日　平成20年2月5日(2008.2.5)
(65)公表番号　特表2010-518516(P2010-518516A)
(43)公表日　平成22年5月27日(2010.5.27)
(86)国際出願番号　PCT/US2008/053095
(87)国際公開番号　WO2008/098009
(87)国際公開日　平成20年8月14日(2008.8.14)
審査請求日　平成23年2月4日(2011.2.4)
(31)優先権主張番号　11/671,414
(32)優先日　平成19年2月5日(2007.2.5)
(33)優先権主張国　米国(US)

(73)特許権者　500046438
マイクロソフト　コーポレーション
アメリカ合衆国　ワシントン州　９８０５２－６３９９　レッドモンド　ワン　マイクロソフト　ウェイ
(74)復代理人　100115624
弁理士　濱中　淳宏
(74)代理人　100077481
弁理士　谷　義一
(74)代理人　100088915
弁理士　阿部　和夫

最終頁に続く

(54)【発明の名称】要素型の型フローを可能にするためのクエリパターン

(57)【特許請求の範囲】
【請求項１】
コンピュータシステムにおいてクエリ式を処理する方法であって、該方法は、前記コンピュータシステムに記憶されたコンピュータ実行可能命令を前記コンピュータシステムが実行することによって実施され、
前記クエリ式にアクセスすることであって、前記クエリ式は複数のクエリ句を含み、前記複数のクエリ句は、次のクエリ句にリンクした少なくとも第１のクエリ句を含み、前記複数のクエリ句の順序は、該順序がクエリコンプリヘンションの基礎のコンピューティング言語とは独立するように、該クエリコンプリヘンション内で定義され、
前記第１のクエリ句は、照会可能ソース型、第１のクエリ演算子、および前記ソース型の要素型を含み、前記要素型は制御変数に関連し、前記制御変数は前記クエリ演算子によって決定される関連するスコープを有し、前記照会可能ソース型は、照会されるソースデータのタイプを示し、前記要素型は、前記第１のクエリ句の結果の要素型を示し、前記クエリ演算子は、クエリ演算子パターンに従って定義されたメソッドにマッピングし、
前記次のクエリ句は、前記第１のクエリ句の結果に適用される次の要素型および次のクエリ演算子を含み、前記次の要素型は前記次のクエリ句の結果の要素型を示し、前記次のクエリ演算子は次のクエリ演算子パターンに従って定義された次のメソッドにマッピングすること、
前記照会可能ソース型に基づいて前記ソースデータ内から前記要素型の１組の要素を得るために、前記第１のクエリ句を評価することであって、前記ソースデータ上の前記クエ

リ演算子を実装するためのメソッドを呼び出すことを含むこと、
前記次のクエリ句の次のソース型が前記要素型であることを推論することであって、前記推論は、前記クエリ演算子を考慮した前記ソース型に基づき、前記クエリコンプリヘンション全体を変換する必要なしに、リアルタイムで実施されること、
前記１組の要素を前記次のクエリへ渡すことであって、前記関連する制御変数は、前記次のクエリ句が前記制御変数の前記スコープをサポートする場合に渡されること、
前記次のクエリ句への前記推論した要素型に基づいて、インクリメンタルにコンテキスト情報を提供すること、および、
前記次のソース型および前記コンテキスト情報に基づいて前記１組の要素内から前記次の要素型の次の１組の要素を得るために、前記次のクエリ句を評価することであって、前記１組の要素上の前記次のクエリ演算子を実装するために前記次のメソッドを呼び出すことを含むこと
を含むことを特徴とする方法。
【請求項２】
前記第１のクエリ演算子はカルテシアン積構造であり、前記要素型は、前記ソース型に関する全てのスコープ内制御変数の集約である匿名型であることを特徴とする請求項１に記載の方法。
【請求項３】
前記次のクエリ演算子パターンは、前記ソースデータの引数を含み、前記要素型の照会可能型を返す関数を引数として受け取るように構成された、射影またはマッピングクエリ演算子を含むことを特徴とする請求項１に記載の方法。
【請求項４】
前記次のクエリ演算子パターンは、前記要素型の引数を含み、前記要素型の照会可能型を返す関数を引数として受け取るように構成された、フィルタリングまたは制限クエリ演算子を含むことを特徴とする請求項１に記載の方法。
【請求項５】
前記次のクエリ演算子パターンは、前記要素型の引数を含み、前記要素型の順序付けされたコレクションを返す関数を引数として受け取るように構成された、ソートまたは順序付けクエリ演算子を含むことを特徴とする請求項１に記載の方法。
【請求項６】
前記次のクエリ演算子パターンは、前記要素型の照会可能型を返すように構成された重複除去クエリ演算子を含むことを特徴とする請求項１に記載の方法。
【請求項７】
前記次のクエリ演算子パターンは、前記ソース型としての前記要素型のコレクションを引数として受け取るように構成され、前記要素型の照会可能型を返す、集合クエリ演算子を含むことを特徴とする請求項１に記載の方法。
【請求項８】
前記次のクエリ演算子パターンは、前記要素型の引数を含み、前記要素型の照会可能型を返す関数を引数として受け取るように構成された、条件付き選択またはフィルタリングクエリ演算子を含むことを特徴とする請求項１に記載の方法。
【請求項９】
前記次のクエリ演算子パターンは、前記要素型の引数を含み、数値型を返す関数を引数として受け取るように構成された、集約クエリ演算子を含むことを特徴とする請求項１に記載の方法。
【請求項１０】
前記次のクエリ演算子パターンは、前記要素型の引数を含み、前記次の要素型の照会可能型を返す関数を引数として受け取るように構成された、集約クエリ演算子を含むことを特徴とする請求項１に記載の方法。
【請求項１１】
前記次のクエリ演算子パターンは、前記要素型の引数を含み、ブールに暗黙的に変換可

能な型を返す関数を引数として受け取るように構成された、マッチングクエリ演算子を含むことを特徴とする請求項１に記載の方法。
【請求項１２】
前記次のクエリ演算子パターンは、前記要素型の引数と追加の要素型の引数とを含み、前記次の要素型の照会可能型を返す関数を引数として受け取るように構成された、結合クエリ演算子を含むことを特徴とする請求項１に記載の方法。
【請求項１３】
前記次のクエリ演算子パターンは、前記要素型の引数と追加の要素型のコレクションである引数とを含み、前記次の要素型の照会可能型を返す関数を引数として受け取るように構成された、ネストされた結合クエリ演算子を含むことを特徴とする請求項１に記載の方法。
【請求項１４】
前記次のクエリ演算子パターンは、型Ｋの引数と前記要素型のコレクションである引数とを含み、前記次の要素型の照会可能型を返す関数を引数として受け取るグループ化クエリ演算子を含み、型Ｋはキーであることを特徴とする請求項１に記載の方法。
【請求項１５】
前記クエリ演算子を考慮した前記ソース型に基づき、前記次のクエリ句の次のソース型が前記要素型であることを推論することは、コンテキストコンポーネントが、前記次のソース型が前記要素型であることをリアルタイムで推論することを含むことを特徴とする請求項１に記載の方法。
【請求項１６】
前記コンテキストコンポーネントは、前記推論した要素型に基づいてインクリメンタルにコンテキスト情報を提供することを特徴とする請求項１５に記載の方法。
【請求項１７】
クエリ式を処理する方法をコンピュータシステムに実行させるコンピュータプログラムであって、前記方法は、
前記クエリ式にアクセスすることであって、前記クエリ式は複数のクエリ句を含み、前記複数のクエリ句は、次のクエリ句にリンクした少なくとも第１のクエリ句を含み、前記複数のクエリ句の順序は、該順序がクエリコンプリヘンションの基礎のコンピューティング言語とは独立するように、該クエリコンプリヘンション内で定義され、
前記第１のクエリ句は、照会可能ソース型、第１のクエリ演算子、および前記ソース型の要素型を含み、前記要素型は制御変数に関連し、前記制御変数は前記クエリ演算子によって決定される関連するスコープを有し、前記照会可能ソース型は、照会されるソースデータのタイプを示し、前記要素型は、前記第１のクエリ句の結果の要素型を示し、前記クエリ演算子は、クエリ演算子パターンに従って定義されたメソッドにマッピングし、
前記次のクエリ句は、前記第１のクエリ句の結果に適用される次の要素型および次のクエリ演算子を含み、前記次の要素型は前記次のクエリ句の結果の要素型を示し、前記次のクエリ演算子は次のクエリ演算子パターンに従って定義された次のメソッドにマッピングすること、
前記照会可能ソース型に基づいて前記ソースデータ内から前記要素型の１組の要素を得るために、前記第１のクエリ句を評価することであって、前記ソースデータ上の前記クエリ演算子を実装するためのメソッドを呼び出すことを含むこと、
前記次のクエリ句の次のソース型が前記要素型であることを推論することであって、前記推論は、前記クエリ演算子を考慮した前記ソース型に基づき、前記クエリコンプリヘンション全体を変換する必要なしに、リアルタイムで実施されること、
前記１組の要素を前記次のクエリへ渡すことであって、前記関連する制御変数は、前記次のクエリ句が前記制御変数の前記スコープをサポートする場合に渡されること、
前記次のクエリ句への前記推論した要素型に基づいて、インクリメンタルにコンテキスト情報を提供すること、および、
前記次のソース型および前記コンテキスト情報に基づいて前記１組の要素内から前記次

の要素型の次の１組の要素を得るために、前記次のクエリ句を評価することであって、前記１組の要素上の前記次のクエリ演算子を実装するために前記次のメソッドを呼び出すことを含むこと、
を含むことを特徴とするコンピュータプログラム。
【請求項１８】
クエリ式を処理するユーザインタフェースを備えたコンピュータシステムであって、前記ユーザインターフェースは、
関係データベースをクエリする前記クエリ式にアクセスすることであって、前記クエリ式は複数のクエリ句を含み、前記複数のクエリ句は、次のクエリ句が続く少なくとも第１のクエリ句を含み、前記複数のクエリ句の順序は、該順序がクエリコンプリヘンションの基礎のコンピューティング言語とは独立するように、該クエリコンプリヘンション内で定義され、
前記第１のクエリ句は、照会可能ソース型、第１のクエリ演算子、および前記ソース型の要素型を含み、前記要素型は制御変数に関連し、前記制御変数は前記クエリ演算子によって決定される関連するスコープを有し、前記照会可能ソース型は、照会されるソースデータのタイプを示し、前記要素型は、前記第１のクエリ句の結果の要素型を示し、前記クエリ演算子は、クエリ演算子パターンに従って定義されたメソッドにマッピングし、
前記次のクエリ句は、前記第１のクエリ句の結果に適用される次の要素型および次のクエリ演算子を含み、前記次の要素型は前記次のクエリ句の結果の要素型を示し、前記次のクエリ演算子は次のクエリ演算子パターンに従って定義された次のメソッドにマッピングし、
前記照会可能ソース型に基づいて前記ソースデータ内から前記要素型の１組の要素を得るために、前記第１のクエリ句を評価することであって、前記ソースデータ上の前記クエリ演算子を実装するためのメソッドを呼び出すことを含み、
前記次のクエリ句の次のソース型が前記要素型であることを推論することであって、前記推論は、前記クエリ演算子を考慮した前記ソース型に基づき、前記クエリコンプリヘンション全体を変換する必要なしに、リアルタイムで実施され、
前記１組の要素を前記次のクエリへ渡すことであって、前記関連する制御変数は、前記次のクエリ句が前記制御変数の前記スコープをサポートする場合に渡され、
前記次のクエリ句への前記推論した要素型に基づいて、インクリメンタルにコンテキスト情報を提供し、
前記次のソース型および前記コンテキスト情報に基づいて前記１組の要素内から前記次の要素型の次の１組の要素を得るために、前記次のクエリ句を評価することであって、前記１組の要素上の前記次のクエリ演算子を実装するために前記次のメソッドを呼び出すことを含む
ように構成されたことを特徴とするコンピュータシステム。
【発明の詳細な説明】
【技術分野】
【０００１】
本発明は、式の第１の句中の演算子から式の次の句中の演算子への要素型の型フローを容易にすることのできる演算子パターンの形式化に関する。
【背景技術】
【０００２】
コンピュータ言語の領域では、従来、多くの異なる言語種類が存在してきた。例えば、多くのプログラミング言語は、ほとんどのエンドユーザプログラミング目標に対する一般的な解決法を提供することを目指すが、一方、クエリ言語は、データベースに対するクエリに基づく情報マイニングを専ら対象とすることが多い。何年もの間、プログラミング言語にクエリ機能を追加するための多くの試みがなされてきたが、非常に多くの困難が存在する。
【０００３】

そのような困難の１つは、現代のほとんどのプログラミング言語がオブジェクト指向モデルに基づくことであり、オブジェクト指向モデルは、プログラミング作業の複雑さを低減するように意図された階層、抽象化、モジュール性、カプセル化、および他のいくつかのパラダイムを可能にする。一方、現代のデータベースは主として関係データベースであり、したがって、クエリ言語はオブジェクト指向モデルよりも関係モデルに基づく傾向がある。

【０００４】

もう１つの困難は、プログラミング言語に対する開発環境（例えば、統合開発環境（ＩＤＥ））が進化して、非常に精巧な補助手段が開発者にもたらされ、それなしでプログラミングするのが難しくなったことである。一例は、オートコンプリートユーティリティまたは機構などの、インラインコンテキスト情報である。これらのオートコンプリート機構は、クラス、変数名、およびその他の構造があらかじめ定義されるオブジェクト指向の世界ではうまく働くが、クエリ言語には利用できないことが多い。というのは、クエリがコンパイルまたは変換され終えるまでは、オートコンプリートに必要な要素型の型チェックが利用可能でないからである。さらに、クエリが完了するまではクエリを変換することができず、それによりオートコンプリート機構は無意味になる。さらに、クエリ式が不適切に形成されている場合、コンパイルエラーは難解で修復が難しい可能性がある。

【発明の概要】

【発明が解決しようとする課題】

【０００５】

本発明は、上述したような問題に鑑みてなされたものであり、その目的とするところは、要素型の型フローを可能にするシステム及び方法を提供することにある。

【課題を解決するための手段】

【０００６】

本発明のいくつかの態様に関する基本的な理解を提供するために、本発明の単純化した概要を以下に提示する。この概要は、本発明の広範な概観ではない。この概要は、本発明の鍵となる要素またはクリティカルな要素を識別するものとはせず、本発明の範囲を線引きするものともしない。この概要の唯一の目的は、本発明のいくつかの概念を、後で提示するより詳細な説明の前置きとして単純化した形で提示することである。

【０００７】

本発明は、その一態様では、式中の演算子間における要素型の型フローを容易にするための、コンピュータによって実施される技法を含む。演算子はクエリ演算子とすることができ（ただしこれに限定されない）、したがって型は、クエリ式全体を通して、あるクエリ句から次のクエリ句にフローすることができる。

【０００８】

本発明の一態様によれば、演算子を、関連するメソッド呼出しにマッピングすることができ、メソッド呼出しは演算子パターンに従って定義することができる。したがって、式中のいずれか所与の演算子は、形式化された演算子パターンに従うものと予想することができる。よって、演算子の項ならびに式全体を、予想されるメソッド呼出しの点から表現することができる。メソッドは、インスタンスメソッド、静的メソッド、仮想メソッド、または拡張メソッドとすることができることを理解されたい。

【０００９】

これにより、型フローを容易にし、演算子が演算子パターンに準拠するように制約することによって、本明細書に開示するアーキテクチャは、式の各句につき要素型をインクリメンタルに推論することができる。例えば、ソース型を演算子と組み合わせることによって要素型を決定することができ、これは、式の連続的な句ごとにリアルタイムで達成することができる。このように、式全体を完全に変換する必要なしに、型情報をローカルに解決することができる。したがって、要素型の型チェックを、（リソース利用の点で）より安価な方式で、より素早く実施することができ、また、式が仕上げられた後のみではなく式の構築中に達成することができ、このことは追加の利益を生むことができる。

【００１０】
このような追加の利益の１つは、また本発明の一態様によれば、式と共にオートコンプリート機構を利用できることである。例えば、実行中に要素型を推論することによって、利用可能な型に基づくコンテキスト情報を提供することができ、このことは式構築を助けることができる。
【００１１】
以下の説明および添付の図面で、本発明の例示的ないくつかの態様について詳細に述べる。しかし、これらの態様は、本発明の原理を利用できる様々な方式のうちの少数を示すに過ぎず、本発明は、そのような態様およびそれらの均等物を全て含むものとする。他の利点および顕著な特徴は、本発明に関する以下の詳細な説明を図面と共に考えることから明らかになるであろう。
【図面の簡単な説明】
【００１２】
【図１】クエリパターンを利用して要素型の型フローを容易にすることのできる、コンピュータによって実現されるシステムのブロック図である。
【図２Ａ】例示的なクエリ式に関する、限定でない代表的な第１のクエリ句および限定でない様々な代表的な次のクエリ句を示す図である。
【図２Ｂ】例示的なクエリ式のクエリ句、ならびに要素型の型フローをより詳細に示す図である。
【図３】例示的な入力と出力を伴うクエリ演算子パターンの限定でない様々な例を示す図である。
【図４】要素型をリアルタイムで推論することができ、かつ／または、推論された要素型に基づいてコンテキスト情報をインクリメンタルに提供することができる、コンピュータによって実現されるシステムのブロック図である。
【図５】クエリ式内における要素型の型フローを容易にするための、コンピュータによって実施される方法を定義する手順の例示的なフローチャートを示す図である。
【図６】構成可能なクエリコンプリヘンションおよび／または拡張可能なクエリ式を容易にすることのできる、コンピュータによって実現されるシステムのブロック図である。
【図７】例示的なクエリ式の例示的なクエリ句の結果として生じる制御変数のスコープに関する例示のブロック図である。
【図８】クエリコンプリヘンションの構築を合成的方式で容易にするための、コンピュータによって実施される方法を定義する手順の例示的なフローチャートである。
【図９】開示したアーキテクチャを実行するように動作可能なコンピュータのブロック図である。
【図１０】例示的なコンピューティング環境の概略ブロック図である。
【発明を実施するための形態】
【００１３】
次に、本発明について図面を参照しながら述べるが、図面全体を通して、同じ要素を指すのには同じ参照番号を使用する。以下の説明では、説明上、本発明の完全な理解を提供するために多くの具体的な詳細を述べる。しかし、本発明は、これらの具体的な詳細がなくても実施できることは明白であろう。他の場合では、本発明の説明を容易にするために、周知の構造およびデバイスはブロック図の形で示す。
【００１４】
本出願においては、用語「コンポーネント」、「モジュール」、「システム」、「インタフェース」などは一般に、ハードウェア、ハードウェアとソフトウェアの組合せ、ソフトウェア、または実行中ソフトウェアのいずれかである、コンピュータ関連エンティティを指すものとする。例えば、コンポーネントは、プロセッサ上で稼動するプロセス、プロセッサ、オブジェクト、実行ファイル、実行のスレッド、プログラム、および／またはコンピュータであってよいが、これらに限定されない。例示として、コントローラ上で稼動するアプリケーションもコントローラも両方とも、コンポーネントとすることができる。

１つまたは複数のコンポーネントが１つのプロセスおよび／または実行のスレッド内にあってよく、コンポーネントは、１つのコンピュータ上に局所化されてもよく、かつ／または２つ以上のコンピュータ間で分散されてもよい。別の例として、インタフェースは、入出力コンポーネントならびに関連するプロセッサ、アプリケーション、および／またはＡＰＩコンポーネントを含むことができ、インタフェースは、コマンドラインのように単純であってもよく、またははより複雑な統合開発環境（ＩＤＥ）であってもよい。

【００１５】

さらに、本発明は、開示する主題を実施するようにコンピュータを制御するために標準的なプログラミングおよび／またはエンジニアリング技法を用いてソフトウェア、ファームウェア、ハードウェア、またはこれらの任意の組合せを生み出す方法、装置、または製品として実現することができる。用語「製品」は、本明細書においては、任意のコンピュータ可読デバイス、キャリア、または媒体からアクセス可能な、コンピュータプログラムを包含するものとする。例えば、コンピュータ可読媒体は、磁気記憶デバイス（例えばハードディスク、フロッピーディスク、磁気ストリップなど）、光学ディスク（例えばコンパクトディスク（ＣＤ）、ディジタル多用途ディスク（ＤＶＤ）など）、スマートカード、およびフラッシュメモリデバイス（例えばカード、スティック、キードライブなど）を含むことができるが、これらに限定されない。加えて、搬送波を利用して、電子メールの送受信で使用されるものや、インターネットまたはローカルエリアネットワーク（ＬＡＮ）などのネットワークへのアクセスで使用されるものなどのコンピュータ可読電子データを搬送できることを理解されたい。当然、本発明の範囲または趣旨を逸脱することなく、この構成に多くの修正を加えてよいことは、当業者なら理解するであろう。

【００１６】

さらに、単語「例示的（ｅｘｅｍｐｌａｒｙ）」は、本明細書では、例、事例、または例証としての役割を果たすことを意味するのに使用される。本明細書で「例示的」として述べるどんな態様または設計も、他の態様または設計よりも好まれるか有利であると必ずしも解釈すべきではない。そうではなく、単語「例示的」の使用は、概念を具体的に提示するものとする。本出願においては、用語「または、もしくは、あるいは（ｏｒ）」は、排他的な「または、もしくは、あるいは」ではなく包含的な「または、もしくは、あるいは」を意味するものとする。すなわち、特に指定がない限り、または文脈から明らかでない限り、「ＸはＡまたはＢを利用する」は、自然な包含的順列のいずれかを意味するものとする。すなわち、ＸがＡを利用する場合、ＸがＢを利用する場合、またはＸがＡとＢの両方を利用する場合は、「ＸはＡまたはＢを利用する」は前記のいずれの場合でも満たされる。加えて、冠詞「ａ」および「ａｎ」は、本出願および添付の特許請求の範囲においては、特に指定がない限り、または文脈から単数形を対象とすることが明らかでない限り、「１つまたは複数」を意味するものと一般に解釈すべきである。

【００１７】

本明細書においては、用語「推論する（ｉｎｆｅｒ）」または「推論（ｉｎｆｅｒｅｎｃｅ）」は、イベントおよび／またはデータを介して取り込まれた１組の観察から、システム、環境、および／またはユーザの状態について推理するプロセス、あるいはこれらの状態を推論するプロセスを一般に指す。推論は、特定のコンテキストまたはアクションを識別するのに利用することができ、あるいは、例えば複数の状態にわたる確率分布を生成することができる。推論は確率的とすることができる。すなわち、データおよびイベントの考慮に基づいた、当該の状態にわたる確率分布の計算とすることができる。推論はまた、１組のイベントおよび／またはデータから、より高レベルのイベントを構成するのに利用される技法を指すこともできる。このような推論の結果、イベントが時間的に近接して相関していようがいまいが、またイベントおよびデータの出所であるイベントおよびデータソースが１つであろうが複数であろうが、１組の観察されたイベントおよび／または記憶されたイベントデータから、新しいイベントまたはアクションが構築される。

【００１８】

次に図面を参照するが、最初に図１を参照すると、クエリパターンを利用して要素型の

型フローを容易にすることのできる、コンピュータによって実現されるシステム１００が示されている。一般に、システム１００はユーザインタフェース１０２を備えることができ、ユーザインタフェース１０２は、周知のコンピュータベースのハードウェア（例えばコントローラ）、ソフトウェア（例えばアプリケーション）、ならびに本明細書に開示する他のコンポーネントに、動作可能に結合させることができる。ユーザインタフェース１０２はクエリ式１０４を受け取ることができ、クエリ式１０４は、第１のクエリ句および１つまたは複数の次のクエリ句を含むことができる。第１のクエリ句は、照会可能ソース型（例えばコレクション、ストリームなど）、ならびに、第１のクエリ句によって例えばソース型に関連する制御変数として導入できる要素型を含むことができることを理解されたい。通常、全てのクエリ句はクエリ演算子を含むが、後でわかるように、クエリ式１０４の次のクエリ句は、完全である必要はなく、句がユーザによって入力されている間などに一部分ずつ（例えばインクリメンタルに）受け取ることができる。クエリ式１０４および関連するサブコンポーネントについては、図２Ａおよび２Ｂに関してより詳細に後述する。

【００１９】

システム１００はまた、任意の有効なクエリ演算子（例えば、第１のクエリ句および後続の次のクエリ句に関連するクエリ演算子）に対するクエリ演算子パターン１０６を含むことができる。したがって、例えばシステム１００に動作可能に結合させることのできるパターンストア１０８中に、任意の数のクエリ演算子パターン１０６が存在することができる。特に、クエリ式１０４からのクエリ句（例えばそれぞれ特定のクエリ演算子を含む）は、関連するクエリ演算子パターン１０６に準拠することができ、クエリ演算子パターン１０６は、例えばアクセス可能なインスタンスメソッドを定義することができる。クエリ演算子パターン１０６に準拠することによって、あるクエリ句と次のクエリ句との間に既知の関係があること、したがってあるクエリ演算子から次のクエリ演算子への関係があることを本質的に保証することができる。クエリ演算子パターン１０６については、図３に関してより詳細に述べる。

【００２０】

クエリ式１０４の演算子がクエリ演算子パターン１０６に準拠するという制約をクエリ式１０４に課すことによって、システム１００は、クエリ式１０４のあるクエリ句／演算子から次のクエリ句／演算子への、要素型の型フローを確実にすることができることを理解されたい。さらに、本明細書に述べるように要素型がフローするのを確実にすることによって、従来のように完全なクエリ式１０４を変換／コンパイルする必要なしに、実行中にいくつかの有利な型推論および／またはコンテキスト決定を実施することができる。これについては図４に関してより詳細に論じる。さらに、前述の推論および／または決定は、クエリ式１０４が完了する前に、さらにはクエリ式１０４が不適切に形成されているときでさえ提供することができる。限定ではないがさらに別の説明として、クエリ演算子パターン１０６に準拠する例示的なクエリ式１０４、およびこのスキーマ化に関連する利点のいくつかについて、以下により詳細に述べる。

【００２１】

引き続き図１を参照しながら、しかし図２Ａにも目を向けて、例示的なクエリ式１０４を示す。一般に、クエリ式１０４は、一連のクエリ演算子を特定のコレクションに適用する式とすることができる。クエリ演算子は、値のコレクション全体にわたって一度にコレクションに適用することのできる演算子（例えばＦＲＯＭ、ＷＨＥＲＥ、ＳＥＬＥＣＴなど）である。クエリ式１０４は、第１のクエリ句２０２および任意の数の次のクエリ句２０$４_1$、２０$４_2$などを含むことができ、各クエリ句は、クエリ演算子のうちの１つを含む。

【００２２】

他にも多くのクエリ演算子が存在することができ、それらは本発明の趣旨および範囲の内にあるものとして企図されるが、以下に、いくつかの具体例をそれぞれの簡単な説明と共に提供する。

－ＦＲＯＭ演算子は、１つまたは複数の制御変数を導入することができ、照会するコレクションを指定するか、または制御変数の値を計算する。
－ＲＥＴＵＲＮおよびＳＥＬＥＣＴ演算子は、出力されるコレクションの形状を指定することができる。場合によっては、ＳＥＬＥＣＴ演算子は、新しい制御変数を導入することができる。
－ＷＨＥＲＥおよびＤＩＳＴＩＮＣＴ演算子は、コレクションの値を制限することができる。
－ＯＲＤＥＲＢＹ演算子は、コレクションに順序付けを課すことができる。
－ＳＫＩＰ、ＳＫＩＰＷＨＩＬＥ、ＴＡＫＥ、およびＴＡＫＥＷＨＩＬＥ演算子は、順序または条件に基づいてコレクションのサブセットを返すことができる。
－ＵＮＩＯＮ、ＵＮＩＯＮＡＬＬ、ＥＸＣＥＰＴ、およびＩＮＴＥＲＳＥＣＴ演算子は、２つのコレクションを受け取って単一のコレクションを生み出すことができる。
－ＧＲＯＵＰ演算子は、コレクションを集約することができ、キーでグループ化されたコレクションを返すことができる。
－ＧＲＯＵＰＢＹ演算子は、１つまたは複数のキーに基づいてコレクションをグループ化することができる。場合によっては、ＧＲＯＵＰＢＹ演算子は、新しい制御変数を導入することができる。
－ＡＶＧ、ＳＵＭ、ＣＯＵＮＴ、ＭＩＮ、およびＭＡＸ演算子は、コレクションを集約して値を生み出すことができる。
－ＡＮＹおよびＡＬＬ演算子は、コレクションを集約し、条件に基づいてブール値を返すことができる。
－ＪＯＩＮ演算子は、２つのコレクションを受け取ることができ、要素から導出されたマッチングキーに基づいて単一のコレクションを生み出すことができる。
－ＧＲＯＵＰＪＯＩＮ演算子は、要素から抽出されたマッチングキーに基づいて、２つのコレクションのグループ化結合を実施することができる。

【００２３】

クエリ式１０４中のクエリ演算子の変換は、演算子がクエリ式１０４中で発生する順序で、左から右に行うことができる。通常、クエリ式１０４の第１のクエリ句２０２は、ＦＲＯＭ演算子を含む。というのは、ＦＲＯＭ演算子は、照会するコレクションを導入するからである。加えて、クエリ式１０４中の最後のクエリ句は一般に、クエリの結果として生じるコレクションの最終形状を指定するＲＥＴＵＲＮおよびＳＥＬＥＣＴ演算子を含む。しかし、最後のＳＥＬＥＣＴまたはＲＥＴＵＲＮ演算子は省いてもよいことを理解されたい。例えば、クエリ式１０４がＲＥＴＵＲＮまたはＳＥＬＥＣＴ演算子で終わらない場合、暗黙的なＳＥＬＥＣＴ演算子を想定することができ、この暗黙的なＳＥＬＥＣＴ演算子は、全てのスコープ内制御変数を、返される匿名型のプロパティにリフトすることができる。さらに、クエリ式１０４は、ＳＥＬＥＣＴ演算子が適用された後も継続できることを理解されたい。その場合、後続の演算子からアクセス可能な制御変数は、スコープ内制御変数のみに限定することができる。

【００２４】

本発明の一態様によれば、クエリ式１０４は、以下のようにスキーマ化することができる。

【００２５】

【表１】

QueryExpression ::= *QueryOperatorList*

QueryOperatorList ::=

FromOperator |

QueryOperatorList QueryOperator

QueryOperator ::=

FromOperator |

SelectOperator |

ReturnOperator |

DistinctOperator |

WhereOperator |

OrderByOperator |

PartitionOperator |

SetOperator |

GroupByOperator |

JoinOperator |

GroupJoinOperator

【００２６】

本発明に関する追加の文脈、ならびに関連するプログラム実装形態および／または言語

の高レベルの概観を提供するために、例示的なクエリ式１０４全体を考えてみる。

【００２７】

【表２】

```
Dim names = _
    From C In Customers _
    Where C.State = “WA” _
    Select C.Name
```

このクエリ式１０４は、Ｃｕｓｔｏｍｅｒオブジェクトのコレクションを取り、ワシントンの各顧客につき単一のＮａｍｅプロパティを含む行のコレクションを返すことができる。

【００２８】

クエリ式１０４の構文は、標準的な関係型の構造化クエリ言語（ＳＱＬ）構文に適度に近いものであり得ることはすぐに明らかであり、ＳＱＬに馴染みのある者なら誰でも、これ以上の指示がほとんどなくてもクエリ式１０４を使用できるようになることが意図されている。しかし、構文はＳＱＬによって制約される必要はなく、さらに、クエリ式１０４は、ＳＱＬを別のプログラミング言語に変換したものである必要はない。ＳＱＬは純粋に関係型のモデルに基づいて設計されたので、そのイディオムのいくつかは、階層の概念を含む型システムでは同様に働かない。加えて、ＳＱＬによってしばしば利用される構文要素およびセマンティクス要素のいくつかは、既存のプログラミング言語の構文またはセマンティクスと相容れないか、あるいはうまく融和しない。したがって、クエリ式１０４は、ＳＱＬに馴染みのあるユーザにとって直感的である可能性があるが、いくつかの相違点もある。

【００２９】

例えば、ＳＱＬ、および手続き型言語／構造化クエリ言語（ＰＬＳＱＬ）など他の言語では、クエリおよびクエリ結果は、クエリ全体を評価することによってバインドされる。したがって、適用される特定のクエリ演算子に対する要素型を解釈できることは必要でない。さらに、このような決定は、エラーのないクエリ全体が入力されて比較的高価な変換プロセスによってバインドされるまで、行うことができない。対照的に、本発明は、特定のソース型に対する個々のクエリ演算の点から、クエリ式１０４の入力およびクエリ式１０４の結果をバインドするのを可能にすることができる。これらおよび他の利点について、図２Ｂを参照しながらより深く例示することができる。

【００３０】

図２Ｂに、例示的なクエリ式１０４のクエリ句をより詳細に示す。この例では、第１のクエリ句２０２は第１のクエリ演算子（例えばＦＲＯＭ演算子２０６）を含み、この第１のクエリ演算子は、照会可能ソース型（例えばＣｕｓｔｏｍｅｒｓ２０８）を導入する働きをすることができ、ソース型の要素型に対する制御変数２１０（例えばＣ）を宣言することができる。図示のように、第１のクエリ句２０２の結果は、次のクエリ句２０４_1（本明細書では次のクエリ句２０４と総称する）にフローし（２１２）、それにより、隣接するクエリ句間に既知の関係が存在することができる。特に、要素型は、次のクエリ演算子（例えばＷＨＥＲＥ演算子２１４）にフローして（２１２）、次のクエリ句２０４_1に対するソース型としての働きをすることができる。同様に、次のクエリ句２０４_1の出力

は、次のクエリ句２０４$_2$にフローし（２１２）、ここで次のクエリ演算子（例えばＳＥＬＥＣＴ演算子２１６）に関連付けられる。この次のクエリ演算子もまた、後続の次のクエリ句２０４にフローする（２１２）かまたはクエリ式１０４の出力の最終形状を記述する、要素型を生み出すことができる。

【００３１】

特に、次の各クエリ句２０４のクエリ演算は、特定のクエリ演算子にそれぞれ関連しコレクションまたはシーケンス（例えばソース型）に適用される、１組のメソッド（例えばクエリ演算子パターン１０６）に基づくことができる。これらのクエリ演算子は、コレクションの要素型（または行）との関係で定義することができる。しかし、要素型Ｔである特定のソース型とクエリ演算子の適用とが与えられれば、次のクエリ演算子２０４にフローする（２１２）結果は、要素型Ｓであるコレクションとすることができる。したがって、フローする（２１２）要素型は、所与のクエリ演算子に関連するクエリ演算子パターン１０６に従って変形させることができる。例えば、ＳＥＬＥＣＴおよびＲＥＴＵＲＮなどの射影クエリ演算子に関連するクエリ演算子パターン１０６は、一般に、フロー２１２から受け取った要素型とは異なる要素型を返す。

【００３２】

この例では、要素型Ｔ（例えば、Ｃｕｓｔｏｍｅｒ型２０８に関連する全ての制御変数２１０の集約とすることができる）が、次のクエリ句２０４$_1$のＷＨＥＲＥ演算子２１４にフローする（２１２）。図３に関して述べるように、ＷＨＥＲＥ演算子２１４に関連するクエリ演算子パターン１０６は、コレクションの値をフィルタにかけることはできるが、他の点では、通常は要素型を変更することはない。ここでは、ワシントン州にいないＣｕｓｔｏｍｅｒｓがコレクションからフィルタにかけられるが、次のクエリ演算子であるＳＥＬＥＣＴ演算子２１６にフローする（２１２）結果は、依然として、要素型がＣｕｓｔｏｍｅｒ２０８であるコレクションである。

【００３３】

一方、Ｎａｍｅプロパティが型Ｓｔｒｉｎｇであると仮定した場合、ＳＥＬＥＣＴ演算子２１６の結果は、要素型がＳｔｒｉｎｇであるコレクションである。クエリ結果に対してＳＥＬＥＣＴとＲＥＴＵＲＮの両方の演算子をサポートする組合せは、演算子への入力と出力の予測可能なセマンティクスを提供することができる。加えて、関係型データと対話するとき、より多くのフレキシビリティを容易にして、予想されるＳＱＬセマンティクスをそっくりに模倣することができる。さらに、非テーブル形式の結果を生み出すとき、より高い単純さを容易にすることができ、さらに、クエリ式を反復的に開発するとき、コンパイラエラーおよび必要とされる変更の局所性の改善を容易にすることができる。

【００３４】

加えて、拡張可能マークアップ言語（ＸＭＬ）データのコレクションを照会することのできるいくつかのクエリ言語（例えば、いくつかのプログラミング言語機能を含むクエリ言語であるＸＱｕｅｒｙ）と同様、本発明は、ＸＭＬリテラルと共に利用することもできる。例えば、以下の例示的なクエリ式１０４を利用して、上に論じた例示的なクエリ式１０４の結果と同様の、ワシントンの全ての顧客の名前を含むＸＭＬ要素のコレクションを返すことができる。

【００３５】

【表３】

```
Dim names = _
        From c In Customers _
        Where c.State = “WA” _
        Return <Name><%= c.Name %></Name>
```

【００３６】

いくつかのクエリ演算子、例えばＦＲＯＭ、ＳＥＬＥＣＴ、およびＧＲＯＵＰＢＹなどは、制御変数と呼ばれる特別な種類のローカル変数（例えば制御変数Ｃ２１０）を導入できることを、さらに理解されたい。デフォルトでは、制御変数は、導入する演算子から、その制御変数を隠蔽することができる演算子であってクエリが評価するときにコレクション中の個別の行のプロパティまたは列を表すことができる演算子まで、スコープすることができる。例えば、以下のクエリでは、

【００３７】

【表４】

```
Dim WACusts = _
        From C As Customer In Customers _
        Where C.State = “WA”
```

【００３８】

ＦＲＯＭ演算子は、Ｃｕｓｔｏｍｅｒとして型付けされる制御変数Ｃを導入する。次いで、後続のＷＨＥＲＥクエリ演算子は、制御変数Ｃを参照して、フィルタ式Ｃ．Ｓｔａｔｅ＝”ＷＡ”中で個々の顧客を表す。

【００３９】

ＤＩＳＴＩＮＣＴなどいくつかのクエリ演算子は、制御変数を使用または変更する必要がない。ＳＥＬＥＣＴなど他の演算子は、現在のスコープ内制御変数を隠蔽することができ、新しい制御変数を導入することができる。例えば、以下のクエリでは、

【００４０】

【表５】

```
Dim YourUncles = _

    From C In Customers _

    Select LastName = C.Name _

    Where LastName.StartsWith("Bo")
```

【００４１】
ＷＨＥＲＥクエリ演算子は、ＳＥＬＥＣＴ演算子によって導入されたＬａｓｔＮａｍｅ制御変数へのアクセスしか有さない。ＷＨＥＲＥ演算子がＣを参照しようとした場合、可能性の高い結果としてコンパイルタイムエラーが生じることがある。
【００４２】
次に図３を参照して（図１も引き続き参照しながら）、クエリ演算子パターンの多くの非限定的な例を提供する。クエリ演算子パターン３０２～３２８は、クエリ演算子パターン１０６の具体的な例示であり、本明細書では、参照番号３０２～３２８で個別に参照することもでき、あるいはクエリ演算子パターン１０６として包括的に参照することもできる。クエリ演算子パターン３０２～３２８のうちの単一のクエリ演算子パターンに関係する複数のクエリ演算子のそれぞれが、それ自体を複数のクエリ演算子のうちの他のクエリ演算子と区別する特性を持ち得るにしても、クエリ演算子パターン１０６のいくつかは複数のクエリ演算子に適用可能とすることができることを、さらに理解されたい。さらに、場合によっては、１つの特定のクエリ演算子を、クエリ演算子パターン３０２～３２８のうちの複数と共に利用することもできる（例えばオーバーロード機能）。
【００４３】
一般に、クエリ演算子パターン１０６は、照会可能となるために型Ｃが実装しなければならないメソッドシグネチャとすることができる。クエリ式１０４は一連のクエリ演算子を特定のコレクションに適用できることを想起されたい。各クエリ演算子の制御変数は、その特定のクエリ演算子の適用についての全てのスコープ内制御変数の全部またはサブセットとすることができ、それに対して、あるクエリ演算子から次のクエリ演算子にフローすることのできる要素型Ｔは、全てのスコープ内制御変数の集約である匿名型とすることができる。したがって、クエリ演算子パターン１０６はとりわけ、型Ｃが確実に照会可能型であるようにするのに役立つことができる。いずれか所与のクエリ式１０４で、型Ｃは、以下の少なくとも１つが当てはまる場合に、照会可能型である。
（１）型Ｃは、照会可能型を返すシグネチャＡｓＱｕｅｒａｂｌｅ（）を伴うアクセス可能インスタンスメソッドを含む。
（２）型Ｃは、ＩＥｎｕｍｅｒａｂｌｅ（Ｏｆ　Ｔ）を返すシグネチャＡｓＥｎｕｍｅｒａｂｌｅ（）を伴うアクセス可能インスタンスメソッドを含む。
（３）型Ｃは、クエリ演算子パターン３０２～３２８（以下にさらに詳述する）に準拠する１つまたは複数のクエリ演算子を実装し、型Ｃに対して定義される全てのクエリ演算子について、型Ｔが合致しなければならずオープン型パラメータであってはならない。
【００４４】
クエリ演算子パターン３０２～３２８を参照すると、クエリ演算子パターン３０２は、ＦＲＯＭ演算子を含むことができる。クエリ式１０４内で、ＦＲＯＭ演算子は通常、第１のクエリ句で利用される。したがって、ＦＲＯＭパターン３０２は、クエリのソースならびに使用される制御変数を導入することができ、要素型Ｔの照会可能型を出力することが

できる。クエリ式の評価の前に、型Ｃが設計パターンを満たさない場合（例えばクエリ演算子パターン１０６に準拠しない場合）は、ＡｓＱｕｅｒｙａｂｌｅまたはＡｓＥｎｕｍｅｒａｂｌｅメソッドをクエリ式１０４上で呼び出し、関数の戻り値を一時的な場所に記憶することができることを理解されたい。

【００４５】

上述したように、ＦＲＯＭ演算子は、照会できるコレクションと、コレクションの個々のメンバを表すことのできる制御変数とを導入することができる。クエリ式

【００４６】

【表６】

```
From b As Book In Books ...
```

【００４７】

は、

【００４８】

【表７】

```
For Each b As Book In Books

    ...

Next b
```

【００４９】

と等価であると考えることができる。

【００５０】

コレクション式オペランドは、一般に、値として分類されなければならず、照会可能型でなければならない。ＦＲＯＭ演算子によって宣言される制御変数は、通常、名前付けおよびスコーピングに関してローカル変数を宣言するための通常規則に従わなければならない（上記の構文変換によって暗示されるように）。したがって、制御変数は通常、囲んでいるメソッド中のローカル変数またはパラメータの名前を隠蔽することはできない。

【００５１】

ＦＲＯＭ演算子はまた、コレクションではなく式によってその値を決定できる制御変数を導入できることを理解されたい。このような特徴は、例えば、後のクエリ演算子で複数回使用されることになる値を計算する（例えば、後で使用される度に値を計算するのではなく１回だけ値を計算する）のに有用である可能性がある。例えば、

【００５２】

【表８】

```
Dim TaxedBookPrices = _
        From b in Books _
        From Tax = b.Price * 0.088 _
        Where Tax > 3.50 _
        Select b.Price, Tax, Total = b.Price + Tax
```

【００５３】
は、
【００５４】
【表９】

```
For Each b In Books
        Dim Tax = b.Price * 0.088
        ...
Next b
```

【００５５】
と等価であると考えることができる。
【００５６】
式制御変数を宣言するＦＲＯＭ演算子の構文は、ＦＯＲ　ＬＯＯＰ中の制御変数宣言と同じとすることができるが、例外として、制御変数は一般に、明示的なイニシャライザを介して初期化される。式制御変数は、別の制御変数を参照することが必要とされない。というのは、そうすることの価値が疑わしいからである。式制御変数は普通、クエリ式１０４中で宣言される第１の制御変数となることはできない。
【００５７】
簡潔にするために、かつ／または便利なように、ＦＲＯＭ演算子のオペランドはＡＳ句を省略することができ、その場合、制御変数の型は、変数の範囲が及ぶコレクションまたは式から推論することができる。制御変数の型をクエリ演算子メソッドから導出できない場合、結果としてコンパイルタイムエラーとなる。ＦＲＯＭ演算子は、以下のようにスキーマ的に定義することができる。
【００５８】

【表１０】

FromOperator ::=

From *FromDeclarationList*

FromDeclarationList ::=

FromDeclaration |

FromDeclarationList , *FromDeclaration*

FromDeclaration ::=

VariableIdentifier [As *TypeName*] In *Expression* |

VariableDeclarator

【００５９】

加えて、ＦＲＯＭ演算子は一般に、第１のクエリ句２０２のクエリ演算子として現れるが、クエリ式１０４は複数のＦＲＯＭ演算子を含むことができる。したがって、ＦＲＯＭ演算子が次のクエリ句２０４のクエリ演算子であるときは、クロス積（例えば単純で暗黙的な結合）が結果として得られる。この場合、ＦＲＯＭ演算子は、新しい制御変数を既存の制御変数のセットに結合する（後述のＪＯＩＮ演算子パターン３２４参照）ことができる。この結果、結合されたコレクション中の全ての要素のクロス積が得られる。したがって例えば、式

【００６０】

【表１１】

From b In Books _

From p In Publishers _

...

【００６１】

は、以下のネストされたＦｏｒ　Ｅａｃｈループと等価であると考えることができる。

【００６２】

【表１２】

```
For Each b In Books
    For Each p In Publishers
        ...
    Next p
Next b
```

【００６３】
前のクエリ演算子中で導入された制御変数は、スコープ内とすることができ、次のクエリ演算子２０４に含まれるＦＲＯＭ演算子内で使用することができる。ＳＱＬから見ると、前述の特徴は、「相関サブクエリ」をサポートするものとして考えることができる。例えば、以下のクエリ式では、第２のＦＲＯＭ演算子が、第１の制御変数の値を参照する。

【００６４】
【表１３】

```
From c As Customer In Customers _
From o As Order In c.Orders _
Select c.Name, o
```

【００６５】
加えて、単一のクエリ句中で複数のＦＲＯＭ演算子が現れることができ、その場合は、オペランド間のコンマが、次のクエリ句２０４中の別のＦＲＯＭ演算子とちょうど等価であるものとすることができる。したがって、以下の例
【００６６】
【表１４】

```
From b In Books, p In Publishers _
Select b, p
```

【００６７】
は、

【００６８】
【表１５】

```
From b In Books _

From p In Publishers _

Select b, p
```

【００６９】
と等価とすることができる。
【００７０】
クエリ演算子パターン３０４は、関数を引数として受け取ることのできるＳＥＬＥＣＴまたはＳＥＬＥＣＴＭＡＮＹという名前のアクセス可能インスタンスメソッドであるクエリ演算子を含むことができる。この関数は、型Ｔの引数を含むことができ、型Ｓとなる。このメソッドは、要素型Ｓの照会可能型を返すことができる。
【００７１】
ＳＥＬＥＣＴ演算子は、クエリ式１０４の結果を記述することができる。ＳＥＬＥＣＴ演算子は、宣言のリストを取り、結果として生じるコレクションの要素型を構築することができる。例えば、クエリ式１０４が
【００７２】
【表１６】

```
Dim CustAndOrderNames = _

    From c In Customers, o In c.Orders _

    Select c.Name, o.Product
```

【００７３】
である場合は、Ｎａｍｅは型ＳｔｒｉｎｇでありＰｒｏｄｕｃｔは型Ｓｔｒｉｎｇなので、結果の型はＩＥｎｕｍｅｒａｂｌｅ（Ｏｆ｛Ｎａｍｅ　Ａｓ　Ｓｔｒｉｎｇ，Ｐｒｏｄｕｃｔ　Ａｓ　Ｓｔｒｉｎｇ｝）とすることができる。
【００７４】
以下のクエリ（例えばクエリ式１０４）は、匿名型を明示的に返すことと等価とすることができる。
【００７５】

【表１７】

```
Dim CustAndOrders = _
    From c In Customers, o In c.Orders _
    Return New With { c.Name, o.Product }
```

【００７６】

ＳＥＬＥＣＴ演算子が１つの宣言のみを有する場合、結果は、１つのプロパティを伴う匿名型とすることができる。したがって、Ｎａｍｅは型Ｓｔｒｉｎｇなので、結果の型はＩＥｎｕｍｅｒａｂｌｅ（Ｏｆ｛Ｎａｍｅ　Ａｓ　Ｓｔｒｉｎｇ｝）とすることができる。例えば以下のとおりである。

【００７７】

【表１８】

```
Dim CustNames = _
    From c In Customers
    Select c.Name
```

【００７８】

ＳＥＬＥＣＴ演算子内で使用される宣言内の式は、基礎の匿名型への変換前にバインドされる。より具体的には、ＳＥＬＥＣＴ演算子中の宣言内の式は、メンバアクセスを開始することができ、このメンバアクセスでは、ドット演算子はクエリ中の制御変数にバインドする必要はない。そうではなく、ドット演算子はその代わり、囲んでいるＷＩＴＨブロックがあればそれにバインドすることができる。

【００７９】

ＳＥＬＥＣＴ演算子に加えて、クエリ式１０４は他の射影演算子を含んでもよいことを理解されたい。例えば、クエリ式は、関連するクエリ演算子パターン１０６が存在することのできるＲＥＴＵＲＮ演算子を含むことができる。ＳＥＬＥＣＴ演算子と同様、ＲＥＴＵＲＮ演算子は、クエリ式１０４の結果を記述することができる。ＲＥＴＵＲＮ演算子は、結果として生じるコレクションの要素を生み出す式を取ることができる。例えば、式

【００８０】

【表１９】

```
Dim ReducedBookPrices = _
    From b in Books _
    Return b.Price * .8
```

【００８１】
は、元のコストの８０％に割り引かれた価格のコレクションを結果としてもたらすことができる。
【００８２】
ＲＥＴＵＲＮ演算子で終わるクエリ式１０４の結果は、返された式の型がその要素型であるコレクションとすることができる。例えば、以下のクエリ式では、ｃ．Ｎａｍｅは型Ｓｔｒｉｎｇなので、結果の型はＩＥｎｕｍｅｒａｂｌｅ（ＯＦ　Ｓｔｒｉｎｇ）である。
【００８３】
【表２０】

```
Dim CustNames = _
    From c In Customers _
    Return c.Name
```

【００８４】
クエリ式１０４がＲＥＴＵＲＮまたはＳＥＬＥＣＴ演算子なしで終わる場合は、コレクションの結果的な要素型は、全てのスコープ内制御変数についてのプロパティを伴う匿名型とすることができる。
【００８５】
【表２１】

```
' Result type is IEnumerable(Of {Name As String, Product As String})
Dim CustNames = _
    From c In Customers, O In C.Orders
```

【００８６】
ＳＥＬＥＣＴ演算子がクエリ式１０４の結果を記述することができるにしても、任意のクエリ式１０４が、ＳＥＬＥＣＴ演算子の後で継続することができる。その場合、ＳＥＬＥＣＴステートメントによって導入された制御変数はスコープ内制御変数だが、全ての前の制御変数は、通常はスコープ外である。
【００８７】

【表２２】

```
CustNames = _
    From c In Customers, O In C.Orders _
    Select Name = C.Name, Price = O.Price
    Where Price > 500
    Return Name
```

【００８８】
ＳＥＬＥＣＴ演算子の後で継続するクエリ式１０４は、以下のようなネストされたクエリと等価とすることができる。

【００８９】
【表２３】

```
Dim CustNames = _
    From X In (From c In Customers, O In C.Orders _
        Return New With { C.Name, O.Price }), _
    Name = X.Name, Price = X.Price
    Where Price > 500
    Return Name
```

【００９０】
ＲＥＴＵＲＮおよびＳＥＬＥＣＴ演算子はそれぞれ、以下の例に基づいて定義することができる。

【００９１】

【表２４】

ReturnOperator ::=

Return *Expression*

SelectOperator ::= Select *SelectDeclarationList*

SelectDeclarationList ::=

SelectDeclaration |

SelectDeclarationList , *SelectDeclaration*

SelectDeclaration ::=

VariableDeclarator

【００９２】

クエリ演算子パターン３０６は、関数を引数として取ることのできるＷＨＥＲＥという名前のアクセス可能インスタンスメソッドであるクエリ演算子を含むことができる。型Ｔをこの関数の引数とすることができ、結果は、ブール型に暗黙的に変換可能とすることのできる型のものとすることができる。例えば、結果の型は、ＩＦ、ＷＨＩＬＥ、またはＤＯステートメント中で使用できる式の型と同じ規則を有することができる。したがって、型は、ブール型への暗黙的な変換を有することができるか、あるいは、ＩｓＴｒｕｅおよびＩｓＦａｌｓｅ演算子が定義されている必要があるものとすることができる。ＷＨＥＲＥ演算子は、要素型Ｔの照会可能型を返すことができる。

【００９３】

ＷＨＥＲＥ演算子は、コレクション中の値を、所与の条件を満たす値に制限することができる。ＷＨＥＲＥ演算子は、制御変数値のセットごとに評価されるブール式を受け取ることができる。式の値がＴｒｕｅである場合は、値は出力コレクション中に現れることができ、そうでない場合は、値はスキップすることができる。クエリ式

【００９４】

【表２５】

```
From b In Books, p In Publishers _
Where b.PublisherID = p.PublisherID _
...
```

【００９５】

は、以下のネストされたループと等価であると考えることができる。

【００９６】
【表２６】

```
For Each b In Books
    For Each p In Publishers
        If b.PublisherID = p.PublisherID Then
            ...
        End If
    Next p
Next b
```

【００９７】
ＷＨＥＲＥ演算子は、以下のように定義することができる。
【００９８】
【表２７】

WhereOperator ::= Where *BooleanExpression*

【００９９】
クエリ演算子パターン３０８は、関数を引数として取ることのできるＯＲＤＥＲＢＹまたはＯＲＤＥＲＢＹＤＥＳＣＥＮＤＩＮＧという名前のアクセス可能インスタンスメソッドであるクエリ演算子を含むことができ、この関数は、型Ｔの引数を取り込むことができ、型Ｓとなる。パターン３０８に関連するメソッドは、要素型Ｔの順序付けされたコレクションを返すことができる。
【０１００】
ＯＲＤＥＲＢＹ演算子は、制御変数中に現れる値を順序に基づいて順序付けすることができる。ＯＲＤＥＲＢＹ演算子は、制御変数を順序付けするのに使用すべき値を指定する式を取ることができる。例えば、以下のクエリは、価格でソートされた本の題名を返す。

【０１０１】
【表２８】

```
From book In Books _
Order By book.Price
Select book.Title
```

【０１０２】
順序付けは昇順とすることができ、その場合、より小さい値がより大きい値の前にくる。あるいは順序付けは降順とすることができ、その場合、より大きい値がより小さい値の

前にくる。順序付けのデフォルトは昇順である。例えば、以下のクエリは、最も高価な本を最初にして、価格でソートされた本の題名を返す。

【０１０３】

【表２９】

```
From book In Books _
Order By book.Price Descending
Select book.Title
```

【０１０４】

ＯＲＤＥＲＢＹ演算子はまた、順序付けのために複数の式を指定することもでき、その場合、ネストされた方式でコレクションを順序付けすることができる。例えば、以下のクエリ式は、州、次いで各州内の都市、次いで各都市内のＺＩＰコードで、著者を順序付けする。

【０１０５】

【表３０】

```
From author In Authors _
Order By author.State, author.City, author.ZIP _
Select author.Name, author.State, author.City, author.ZIP
```

【０１０６】

ＯＲＤＥＲＢＹ演算子は、以下の例に基づいてスキーマ化することができる。

【０１０７】

【表３１】

OrderByOperator ::= Order By *OrderExpressionList*

OrderExpressionList ::=

OrderExpression |

OrderExpressionList , *OrderExpression*

OrderExpression ::=

Expression [*Ordering*]

Ordering ::= Ascending | Descending

【０１０８】

加えて、クエリ演算子パターン３１０は、ＤＩＳＴＩＮＣＴという名前のアクセス可能インスタンスメソッドであるクエリ演算子を含むことができる。通常、このメソッドは、ソース型Ｃと同じ要素型の照会可能型を返す。ＤＩＳＴＩＮＣＴ演算子は、コレクション中の値を、異なる値を伴う（例えば、繰り返される値を返さない）値のみに制限することができる。例えば、クエリ

【０１０９】

【表３２】

```
From c In Customers, o In c.Orders _
Select c.Name, o.Price _
Distinct
```

【０１１０】

は、顧客が同価格の注文を複数有する場合であっても、顧客名と注文価格との異なる対合ごとに１つの行のみを返すことになる。ＤＩＳＴＩＮＣＴ演算子は、以下のようにスキーマ化することができる。

【０１１１】

【表３３】

```
DistinctOperator ::= Distinct
```

【０１１２】

次にクエリ演算子パターン３１２を参照するが、パターン３１２は、ＣＯＮＣＡＴ、ＵＮＩＯＮ、ＩＮＴＥＲＳＥＣＴ、またはＥＸＣＥＰＴという名前のアクセス可能インスタンスメソッドであるクエリ演算子を含むことができる。このメソッドは、ソースＣと同じ要素型Ｔのコレクションを引数として受け取ることができ、要素型Ｔの照会可能型を返すことができる。

【０１１３】

次のクエリ演算子パターンであるパターン３１４は、ＴＡＫＥまたはＳＫＩＰという名前のアクセス可能インスタンスメソッドとすることのできるクエリ演算子を含むことができ、このメソッドは、値を引数として受け取ることができ、ソース型Ｃと同じ要素型の照会可能型を返すことができる。

【０１１４】

ＴＡＫＥ演算子は、コレクションからの所与の数の要素を結果としてもたらすことができる。ＷＨＩＬＥ修正子と共に使用されるとき（例えばクエリ演算子パターン３１６参照）、ＴＡＫＥ演算子は、条件が当てはまる間は要素のシーケンスを結果としてもたらすことができる。

【０１１５】

ＳＫＩＰ演算子は、コレクションからの所与の数の要素を無視することができ、次いでコレクションの残りを返す。ＷＨＩＬＥ修正子と共に使用されるとき、ＳＫＩＰ演算子は、条件が当てはまる間は要素をスキップし、次いでコレクションの残りを返す。ＴＡＫＥおよびＳＫＩＰに関する例示的なスキーマを以下に提供する。

【０１１６】

【表３４】

PartitionOperator ::=

TakeExpression |

SkipExpression

TakeExpression ::=

Take [While] *Expression* |

SkipExpression ::=

Skip [While] *Expression*

【０１１７】

クエリ演算子パターン３１６は、ＴＡＫＥＷＨＩＬＥまたはＳＫＩＰＷＨＩＬＥという名前のアクセス可能インスタンスメソッドであるクエリ演算子を含むことができる。このメソッドは、関数を引数として取ることができ、この関数は、型Ｔの引数を受け取ることができ、ブールとなる。このメソッドは、要素型Ｔの照会可能型を返すことができる。

【０１１８】

クエリ演算子パターン３１８は、ＳＵＭ、ＭＩＮ、ＭＡＸ、ＣＯＵＮＴ、またはＡＶＥＲＡＧＥという名前のアクセス可能インスタンスメソッドであるクエリ演算子を含むことができる。このメソッドは、関数を引数として受け取ることができ、この関数は、型Ｔの引数を受け取り、数値型となる。このメソッドは、数値型を返すことができる。

【０１１９】

クエリ演算子パターン３２０は、ＭＩＮまたはＭＡＸという名前のアクセス可能インスタンスメソッドであるクエリ演算子を含むことができ、このメソッドは、型Ｔの引数を含む関数を引数として取ることができ、型Ｓとなる。このメソッドは、要素型Ｓの照会可能型を返すことができる。

【０１２０】

クエリ演算子パターン３２２は、関数を引数として受け取ることのできるＡＮＹまたはＡＬＬという名前のアクセス可能インスタンスメソッドであるクエリ演算子を含むことができる。この関数は、型Ｔの引数を含むことができ、ブールに暗黙的に変換可能とすることのできる型となる。このメソッドは、ブールに暗黙的に変換可能とすることのできる型を返すことができる。

【０１２１】

パターン３１８～３２２に関連するクエリ演算子は、ＡＧＧＲＥＧＡＴＥ演算子との関連で利用できることを理解されたい。集約（ａｇｇｒｅｇａｔｅ）クエリ句内では、標準的な１組のＡＧＧＲＥＧＡＴＥ演算子を、グループ化されたスコープ内制御変数に適用することができる。ＡＧＧＲＥＧＡＴＥ演算子は、ＡＮＹ、ＡＬＬ、ＣＯＵＮＴ、ＬＯＮＧＣＯＵＮＴ、ＳＵＭ、ＭＩＮ、ＭＡＸ、ＡＶＥＲＡＧＥ、および／またはＧＲＯＵＰを含

むことができるが、これらに限定されない。

【０１２２】

ＡＮＹ集約演算子は、所与の条件を満たす要素がグループ中にあるかどうか確認することができる。ＡＬＬ集約演算子は、グループ中の全ての要素が所与の条件を満たすかどうか判定することができる。例えば、以下の例示的なクエリ式１０４は、１８歳未満の顧客がいるかどうかチェックすることができる。

【０１２３】

【表３５】

```
From cust In Customers _
Group By cust.State _
Aggregate YoungerThan18 = Any(cust.Age < 18)
```

【０１２４】

一方、以下の例示的なクエリ式１０４は、所与の州における、少なくとも５つの注文を有する全ての顧客を返すことができる。

【０１２５】

【表３６】

```
From cust In Customers _
Group By cust.State _
Aggregate AtLeast5 = All(cust.Orders.Count() > 5)
```

【０１２６】

ＣＯＵＮＴおよびＬＯＮＧＣＯＵＮＴ集約演算子は、オプションのブール式を取り、グループ中の、所与の条件を満たす要素の数をカウントすることができる。

【０１２７】

【表３７】

```
From cust In Customers _
Group By cust.State _
Aggregate Seniors = Count(cust.Age > 50)
```

【０１２８】

ＳＵＭ集約演算子は、特定のセレクタ式に基づいて、グループの要素の合計を計算することができる。例えば、以下の例示的なクエリ式１０４は、カテゴリでグループ化された全ての注文の総額を計算することができる。

【０１２９】

【表３８】

```
From order In Orders _
Group By order.Category
Aggregate Total = Sum(order.Price)
```

【０１３０】
ＭＩＮおよびＭＡＸ集約演算子は、何らかのセレクタ式に基づいて、グループの要素の最小（または最大）を計算することができる。例えば、以下の例示的なクエリ式１０４は、各州の最年少および最年長の顧客を計算することができる。

【０１３１】
【表３９】

```
From cust In Customers _
Group By cust.State
Aggregate Oldest = Max(cust.Age), Youngest = Min(cust.Age)
```

【０１３２】
ＡＶＥＲＡＧＥ集約演算子は、通常、特定のセレクタ式に基づいて、グループの要素の平均を計算する。例えば、以下の例示的なクエリ式１０４は、カテゴリでグループ化された全ての製品の平均価格を計算することができる。

【０１３３】
【表４０】

```
From prod In Products _
Group By prod.Category
Aggregate AveragePrice = Average(prod.Price)
```

【０１３４】
特別なＧＲＯＵＰ集約演算子は、オプションのセレクタ式に基づいて、グループの全ての要素を明示的なコレクションに蓄積することができる。例えば、以下の例示的なクエリ式１０４は、所与の州の全ての顧客名を単一のコレクションにまとめることができる。

【０１３５】

【表４１】

```
From cust In Customers _
Group By cust.State
Aggregate Names = Group(cust.Name)
```

【０１３６】
クエリ演算子パターン３２４は、ＪＯＩＮという名前のアクセス可能インスタンスメソッドであるクエリ演算子を含むことができる。このメソッドは、型Ｔ’の要素を伴う照会可能型Ｓ；型Ｔの引数を取ることができ、キーＫを表す型となる、内側セレクタとしての働きをする関数；型Ｔ’の引数を取ることができ、キーＫを表す型となる、外側セレクタとしての働きをする関数；ならびに／または、それぞれ型ＴおよびＴ’である２つの引数を取り、型Ｖのハッシュ値となる、結合条件としての働きをする関数を、引数として取ることができる。このメソッドは、要素型Ｖの照会可能型を返すことができる。
【０１３７】
したがって、ＪＯＩＮ演算子は、２つのコレクションを取り、要素から導出されたマッチングキーに基づいて単一のコレクションを生み出すことができる。どれを結合するかに関する条件を指定するとき、いくつかの制限を適用することができる。例えば、条件式に対するオペランドは、暗黙的にブールに変換可能であることが必要とされてよい。ＪＯＩＮスキーマの一例は以下のとおりである。
【０１３８】
【表４２】

JoinOperator ::=

Join *LoopControlVariable* [As *TypeName*] In *Expression* Where *JoinConditionExpression*

JoinConditionExpression ::=

RelationalOperatorExpression |

LikeOperatorExpression

【０１３９】
クエリ演算子パターン３２６は、ＧＲＯＵＰＪＯＩＮという名前のアクセス可能インスタンスメソッドであるクエリ演算子を含むことができ、このメソッドは、型Ｔ’の要素を伴う照会可能型Ｓ；型Ｔの引数を取り、キーＫを表す型となる、外側キーセレクタとしての働きをする関数；型Ｕの引数を取り、キーＫを表す型となる、内側キーセレクタとしての働きをする関数；および／または、結果を生むことのできる関数であって、型Ｔの引数と、要素型Ｕのコレクションである引数とをとることができ、選択された型Ｖとなる関数を、引数として受け取ることができる。このメソッドは、要素型Ｖの照会可能型を返すことができる。
【０１４０】
ＧＲＯＵＰＪＯＩＮ演算子は、要素から抽出されたマッチングキーに基づいて、２つのコレクションのグループ化結合を生み出すことができる。この演算子は、階層型の結果（例えば、外側要素が、合致する内側要素のコレクションと対合されている）を生み出すこ

とができ、関係データベースから見た直接の等価物を必要としない。以下の例示的なクエリ式１０４は、顧客とその注文とのグループ化結合を実施することができ、顧客名およびその顧客の注文の総計を含む匿名型のコレクションを生み出す。

【０１４１】

【表４３】

```
From cust In db.Customers _
Group Join order In db.Orders On cust.ID = order.CustID _
Aggregate TotalOrders = Sum(order.Total) _
Select cust.Name, TotalOrders
```

【０１４２】

固定されたセットに作用して単一の値を計算する、任意の数の追加の演算子があってよい。クエリの文脈では、これらの演算子は、計算された値を集約すると言うことができる。セットにわたって集約するには、一般にそのセットを最初に導入しなければならない。クエリ内では、集約計算のためのコレクションを指定する様々な方法があり得る。グループが導入された後、このグループにわたって集約して単一の値を計算することができる。したがって、グループ化演算子ＧＲＯＵＰＪＯＩＮ（ならびに、クエリ演算子パターン３２８に関して後で論じるＧＲＯＵＰＢＹ）に加えて、他のグループ化演算子が存在することもできる。そのような一例はＯＶＥＲ演算子である。

【０１４３】

ＯＶＥＲ演算子は、集約コンプリヘンションのためにグループを蓄積するよう指定することができる。ＯＶＥＲ演算子はスタンドアロンの式として使用することもでき、あるいは、ＯＶＥＲ演算子を使用して、事前形成済みコレクションを集約計算に利用するよう指定することもできる。例えば、以下の例示的なクエリ式１０４は、２００６年１月１日よりも前の全ての注文総計の合計を集約することができる。

【０１４４】

【表４４】

```
Over order In Orders _
Where order.Date <= #01/01/2006# _
Aggregate Sum(order.Total)
```

【０１４５】

ＯＶＥＲ演算子の各適用は、通常、対応するＡＧＧＲＥＡＴＥ演算子によって閉じられて、コレクションの内容に基づいて単一の値が計算されなければならない。クエリの結果は、集約計算に対応する単一の値とすることができ、この単一の値は、この場合、２００６年よりも前の全ての注文の合計を表す整数である。

【０１４６】

ＯＶＥＲ演算子とＡＧＧＲＥＧＡＴＥ演算子との間では、どんなクエリ演算子を使用することもできる可能性がある。ＡＧＧＲＥＧＡＴＥ演算子内では、前のＯＶＥＲ演算子によって指定される制御変数に対して、ＡＧＧＲＥＧＡＴＥ演算子（例えば、前述のパター

ン３１８～３２２に関連する演算子）のみを使用することができる。
【０１４７】
階層型オブジェクトグラフでしばしばそうであるようにセットがすでに形成されているときは、ＯＶＥＲ演算子を使用して、セットを集約計算に使用するよう指定することができる。例えば、以下の例示的なクエリ式１０４は、２００６年よりも前にワシントンの顧客によって行われた全ての注文の合計を集約することができる。
【０１４８】
【表４５】

```
From cust In db.Customers _
Where cust.State = “WA” _
Over order In Orders _
Where order.Date <= #01/01/2006# _
Aggregate Sum(order.Total)
```

【０１４９】
このクエリの結果は、１）ｃｕｓｔと呼ばれるＣｕｓｔｏｍｅｒプロパティと、Ｓｕｍと呼ばれるＩｎｔｅｇｅｒプロパティとの２つのプロパティを伴う匿名型がその要素型である照会可能型とすることができる。
【０１５０】
クエリ演算子パターン３２８は、ＧＲＯＵＰＢＹという名前のアクセス可能インスタンスメソッドであるクエリ演算子を含むことができ、このメソッドは、型Ｔの引数を受け取り、キーＫを表す型となる、キーセレクタとしての働きをする関数；および／または、結果を生む関数であって、型Ｋの引数と、要素型Ｔのコレクションである引数とを取り、選択された型Ｖとなる関数を、引数として取る。このメソッドは、要素型Ｖの照会可能型で結果を返すことができる。
【０１５１】
ＧＲＯＵＰＢＹ演算子は、１つまたは複数の共通キー式に基づいて、コレクションの要素をグループ化する（例えばローカルに）ことができる。最初のコレクションのこれらの各区分に対して、後続のＡＧＧＲＥＧＡＴＥステートメントが、これらの各グループがどのように単一の値または行に集約されるかを指定することができる。
【０１５２】
例えば、以下の例示的なクエリ式１０４は、全ての顧客をＳｔａｔｅでグループ化し、次いで各グループのカウントおよび平均年齢を計算することができる。
【０１５３】

【表４６】

```
From cust In Customers _
Group By cust.State
Aggregate Total = Count(), AverageAge = Avg(cust.Age)
```

【０１５４】

ＳＥＬＥＣＴ演算子と同様、ＧＲＯＵＰＢＹ演算子は、キーセレクタ中で宣言される変数を制御変数としてスコープ内にすることができ、前にスコープ内であった全ての変数を隠蔽することができる。対照的に、ＳＥＬＥＣＴ演算子のいくつかの態様とは異なり、これらの隠蔽された制御変数は、後続のＡＧＧＲＥＧＡＴＥ演算子の内部で使用されるＡＧＧＲＥＧＡＴＥ演算子によって再びスコープ内にすることができる。さらに、ＯＶＥＲ演算子のいくつかの実装形態とは異なり、ＧＲＯＵＰＢＹ演算子とＡＧＧＲＥＧＡＴＥ演算子との間では、他のクエリ演算子を使用することはできない。

【０１５５】

【表４７】

```
From cust In Customers _
Group CustomersByState By cust.State
```

【０１５６】

この構造は、明示的な集約によってグループを構築することと等価とすることができるが、ＧＲＯＵＰＢＹ演算子句の直後にクエリを終了するときに有用とすることができる。したがって、上の式は、以下の式と等価とすることができる。

【０１５７】

【表４８】

```
From cust In Customers _
Group By cust.State _
Aggregate CustomersByState = Group(cust)
```

【０１５８】

個別の各グループの表現を実際に構築するために、ＧＲＯＵＰＢＹ演算子を実際に実装することは必ずしも必要とされない。そうではなく、実装は、グループ化を後続の集約と結合して単一の演算にする、基礎の実装を使用することができる。

【０１５９】

クエリ式１０４のターゲットおよび適用されているクエリ演算子から、制御変数の型および名前を推論できることを理解されたい（これについては図４に関してより詳細に後述する）。したがって、要素型（スコープ内制御変数の集約である）を同様にして推論することができる。前述の各規則では、要素型の推論された型をＴとすることができ、照会可

能型Ｃが例えばＦＯＲ　ＥＡＣＨ．．．ＮＥＸＴステートメント中で使用されるコレクションであるとき、要素型は通常、要素型Ｔと合致しなければならない。

【０１６０】

以下の演算子のサブセットを実装するコレクションとして、順序付けされたコレクションを定義することができることも、さらに理解されたい。すなわち、型Ｔの引数を取り込み型Ｓとなる関数を引数として取るＴＨＥＮＢＹまたはＴＨＥＮＢＹＤＥＳＣＥＮＤＩＮＧという名前のアクセス可能インスタンスメソッドである。

【０１６１】

クエリ演算子パターン１０６は、特定のクエリ演算を実装するメソッドにクエリ構文をバインドするのを容易にすることができるので、利用される基礎の言語によって順序保存を指示する必要はない。そうではなく、順序保存は、演算子自体の実装によって決定することができる。このことは、例えばユーザ定義の数値型について加算演算子をオーバーロードするための実装が、加算に似たどんなものも実施しない場合があるという点で、ユーザ定義の演算子に類似する可能性がある。しかし、本発明に内在する予測可能性を維持するためには、ユーザ予想に合致しない演算子を実装することは、推奨される方針ではない場合がある。

【０１６２】

本発明のより完全な理解を助けるために、次に、追加の特徴、態様、および／または実装形態についてさらに論じることができる。例えば、以下のセクションのいくつかでは、各クエリ演算子が標準的なクエリ演算子メソッドへの呼出しにマッピングする例示的な方式について詳述する。他のセクションでは、上に紹介したように、また後のセクションで利用されるように（例えば図４および以下の関連する記述に関して）要素型情報がフローする方式について述べる。

【０１６３】

（クエリ演算子の変換）

様々なクエリ演算子のそれぞれは、標準的なクエリ演算子パターン（例えばクエリ演算子パターン１０６）に従って定義できるメソッド呼出しに直接にマッピングすることができることを理解されたい。したがって、クエリ演算子（またはクエリ式全体）の意味は、呼び出されることになるクエリ演算子メソッドの点から表現することができる。メソッドは、照会されているオブジェクトのインスタンスメソッド、またはオブジェクト外部の拡張メソッドとすることができる。例えば、クエリ

【０１６４】

【表４９】

```
Dim names = _
        From c In Customers _
        Where c.State = "WA" _
        Return c.Name
```

【０１６５】

は、

【０１６６】

【表５０】

```
Dim names = Customers.Where(c => c.State).Select(c => c.Name)
```

【０１６７】

と等価とすることができる。

【０１６８】

（複合制御変数）

本発明によって、本明細書に述べる様々な目的に制御変数を利用することができるが、制御変数は一般にエンドユーザから直接にアクセス可能ではないことを理解されたい。ＷＨＥＲＥまたはＳＥＬＥＣＴなど、いくつかのクエリ演算子は、クエリ中でスコープ内である制御変数を参照することのできる式を取る。これらの式は、制御変数を取って式の結果を返すローカル関数として表現される。例えば、クエリ

【０１６９】

【表５１】

```
Dim WACusts = _
    From cust In Customers _
    Where cust.State = "WA"
```

【０１７０】

は、

【０１７１】

【表５２】

```
Dim WACusts = Customers.Where(c => c.State = “WA”) _
    .Select(c => New With {c.Name})
```

【０１７２】

と等価とすることができ、制御変数ｃｕｓｔは、ラムダ式に対するパラメータとして使用される。同様に、複数の制御変数がスコープ内である場合、これらの制御変数は通常、ローカル関数中に渡すことができるように、単一の匿名型に共にグループ化しなければならない。例えば、クエリ

【０１７３】

【表５３】

```
Dim WACustsAndOrders = _
    From cust In Customers _
    From order In cust.Orders _
    Where cust.State = "WA" AndAlso order.Price > 10.50
```

【０１７４】

は、

【０１７５】

【表５４】

```
Function Filter1(it As {c As Customer, o As Order}) As Boolean
    Return it.c.State = "WA" AndAlso it.o.Price > 10.50
End Function
Function Join1(c As Customer) As _
        IEnumerable(Of {c As Customer, o As Order})
    Function Select1(o As Order) As {c As Customer, o As Order}
        Return New {c, o}
    End Function
    Return c.Orders.Select(AddressOf Select1)
End Function
Dim WACustsAndOrders = _
    Customers.SelectMany(AddressOf Join1).Where(AddressOf
    Filter1)
```

【０１７６】

と等価とすることができ、制御変数ｃおよびｏは、型が匿名型｛ｃ　Ａｓ　Ｃｕｓｔｏｍｅｒ，ｏ　Ａｓ　Ｏｒｄｅｒ｝であるＩＴと呼ばれる複合制御変数に、共に結合される。

【０１７７】

ラムダ式に対するサポートも提供することができ、これは本発明の趣旨および範囲の内にあるものと考えるべきであることを、さらに理解されたい。さらなる留意として、複合制御変数は、普通はネストされない。例として、

【０１７８】

【表５５】

```
From b In Books _
From ba In BookAuthors _
From a In Authors _
...
```

【０１７９】

は、

【０１８０】

【表５６】

```
Function Join1(b As Book) As IEnumerable(Of {b As Book, ba As
BookAuthor})
        Function Select1(ba As BookAuthor) As {b As Book, ba As
BookAuthor}
                Return New { b, ba }
        End Function
        Return BookAuthors.Select(AddressOf Select1)
End Function
Function Join2(it As {b As Book, ba As BookAuthor}) As _
                IEnumerable(Of {b As Book, ba As BookAuthor, a As
Author})
        Function Select2(a As Author) As _
                        {b As Book, ba As BookAuthor, a As Author}
                Return New {b, ba, a}
        End Function
        Return Authors.Select(AddressOf Join2)
End Function
Books.SelectMany(AddressOf Join1).SelectMany(AddresOf Join2)
```

【０１８１】

と等価とすることができる。

【０１８２】

（ＦＲＯＭ演算子の変換）

標準的なＦＲＯＭ演算子は、一般に、クエリのソースおよび使用される制御変数を導入するのみである。本質的に、この演算子は、どんな特定のクエリ演算子呼出しにも変換しない。ＪＯＩＮの場合、ＦＲＯＭ演算子は、通常の制御変数と結合するときにはＳＥＬＥＣＴＭＡＮＹクエリ演算子メソッドへの呼出しに変換することができ、式制御変数と結合するときにはＳＥＬＥＣＴクエリ演算子メソッドへの呼出しに変換することができる。したがって、クエリ式

【０１８３】

【表５７】

```
From b In Books, p In Publishers _
Select b.Title, p.Name
```

【０１８４】
は、
【０１８５】
【表５８】

```
Function Join1(b As Book) As IEnumerable(Of {b As Book, p As
Publisher})
    Function Select2(p As Publisher) As {b As Book, p As Publisher}
        Return New With {b, p}
    End Function
    Return Publishers.Select(AddressOf Select2)
End Function
Function Select1(it As {b As Book, p As Publisher}) As _
        {Title As String, Name As String}
    Return New With {it.b.Title, it.p.Name}
End Function
Books.SelectMany(AddressOf Join1).Select(AddressOf Select1)
```

【０１８６】
と等価とすることができ、クエリ式
【０１８７】
【表５９】

```
From b In Books, Tax = b.Price * 0.088 _
Select b.Title, Tax
```

【０１８８】
は、
【０１８９】

【表６０】

```
Function Select1(b As Book) As {b As Book, Tax As Double}
    Return New With {b, Tax = b.Price * 0.088}
    End Function
    Function Select2(it As {b As Book, Tax As Double}) As {Title As String,
    Tax As Double}
        Return New With {it.b.Title, it.Tax}
    End Function
    Books.Select(AddressOf Select1).Select(AddressOf Select2)
```

【０１９０】

と等価とすることができる。

【０１９１】

各クエリ演算子は、特定のパターンに従うソース型に対する基礎のメソッドにバインドすることができる。このパターンは、演算子の結果の要素型を指示し、場合によっては、基礎のメソッド中に渡すことのできる式に対する制約を課す。

【０１９２】

（明示的に型付けされた制御変数）

ＦＲＯＭ演算子がＡｓ句を使用して制御変数に対するターゲット型Ｏｂｊｅｃｔを指定し、Ｔの型をソース型から推論することができない場合は、Ｃａｓｔ（Ｏｆ　Ｏｂｊｅｃｔ）演算子を使用して要素型を変換することができる。クエリ式

【０１９３】

【表６１】

```
Dim nums = _
    From addrNum As Object In publisher.Address _
    Where addrNum < 5
```

【０１９４】

は、

【０１９５】

【表６２】

```
Function Filter1(addrNum As Integer) As Boolean
    Return addrNum < 5
End Function
Dim names = publisher.Address.Cast(Of Object)().Where(AddressOf
Filter1)
```

【０１９６】

と等価である。

【０１９７】
ＦＲＯＭ演算子がＡｓ句を使用して制御変数に対するターゲット型Ｔ（Ｏｂｊｅｃｔではない）を指定し、Ｔの型をソース型から推論することができない場合、またはターゲット型がソースの要素型と合致しない場合は、Ｃａｓｔクエリ演算子を使用して要素型をＯｂｊｅｃｔに変換することができ、後続のＳＥＬＥＣＴ演算子を利用してターゲット型を得ることができる。例えば、クエリ式
【０１９８】
【表６３】

```
Dim nums = _
        From addrNum As Integer In publisher.Address _
        Where addrNum < 5
```

【０１９９】
は、
【０２００】
【表６４】

```
Function Filter1(addrNum As Integer) As Boolean
        Return addrNum < 5
End Function
Function Select1(addrNum As Integer) As String
        Return addrNum
End Function
Dim names = publisher.Address.Cast(Of Object)().Select(AddressOf
Select1).Where(AddressOf Filter1)
```

【０２０１】
と等価である。
【０２０２】
ＦＲＯＭ演算子がＡｓ句を使用して制御変数に対するターゲット型Ｔを指定し、Ｔの型がソースの要素型と合致する場合は、後でＣａｓｔまたはＳＥＬＥＣＴ演算子を適用する必要はない。
【０２０３】
（ＷＨＥＲＥ演算子の変換）
ＷＨＥＲＥ演算子は、Ｗｈｅｒｅクエリ演算子メソッドへの呼出しに変換することができる。例えば、クエリ式
【０２０４】

【表６５】

```
From book In Books _
Where book.PublisherID = 10 _
Return book.Title
```

【０２０５】
は、
【０２０６】
【表６６】

```
Books.Where(b => b.PublisherID = 10).Select(b => b.Title)
```

【０２０７】
と等価とすることができる。
【０２０８】
（ＯＲＤＥＲＢＹ演算子の変換）
ＯＲＤＥＲＢＹ演算子は、第１のソートについてＯＲＤＥＲＢＹまたはＯＲＤＥＲＢＹＤＥＳＣＥＮＤＩＮＧクエリ演算子メソッドへの呼出しに変換することができ（ソートのタイプに応じて）、次いで、後続のソートについてＴＨＥＮＢＹおよびＴＨＥＮＢＹＤＥＳＣＥＮＤＩＮＧクエリ演算子メソッドへの呼出しに変換することができる。例えば、クエリ式
【０２０９】
【表６７】

```
From a In Authors _
Order By a.State, a.City Descending _
Return a.Name
```

【０２１０】
は、
【０２１１】
【表６８】

```
Authors.OrderBy(a => a.State).ThenByDescending(a => a.City). _
        Select(a => a.Name)
```

【０２１２】
と等価とすることができる。
【０２１３】
（ＲＥＴＵＲＮおよびＳＥＬＥＣＴ演算子の変換）

ＲＥＴＵＲＮ演算子は、ＳＥＬＥＣＴクエリ演算子メソッドへの呼出しに変換することができる。例えば、

【０２１４】

【表６９】

```
From e In Employees _
Return e.Name
```

【０２１５】

は、

【０２１６】

【表７０】

```
Employees.Select(e => e.Name)
```

と等価とすることができる。

【０２１７】

ＳＥＬＥＣＴ演算子は、ＳＥＬＥＣＴクエリ演算子メソッドへの呼出しおよび匿名型の構造に変換することができる。例えば、

【０２１８】

【表７１】

```
From e In Employees _
Select e.Name, e.Salary
```

【０２１９】

は、

【０２２０】

【表７２】

```
Employees.Select(e => New With {e.Name, e.Salary})
```

【０２２１】

と等価とすることができる。

【０２２２】

（ＤＩＳＴＩＮＣＴ演算子の変換）

ＤＩＳＴＩＮＣＴ演算子は、ＤＩＳＴＩＮＴクエリ演算子メソッドへの呼出しに変換することができる。例えば、

【０２２３】

【表７３】

```
From e In Employees _
Distinct
```

【０２２４】
は、
【０２２５】
【表７４】

```
Employees.Distinct()
```

【０２２６】
と等価とすることができる。
【０２２７】
（ＴＡＫＥまたはＳＫＩＰ［ＷＨＩＬＥ］演算子の変換）
ＴＡＫＥまたはＳＫＩＰ演算子は、それぞれのＴＡＫＥまたはＳＫＩＰクエリ演算子メソッドを呼び出すことができる。例えば、
【０２２８】
【表７５】

```
From e In Employees _
Order By e.Salary Descending _
Return e.Name
Take 5
```

【０２２９】
は、
【０２３０】
【表７６】

```
Employees.OrderByDescending(e => e.Salary).Select(e => e.Name).Take(5)
```

【０２３１】
と等価とすることができ、クエリ
【０２３２】

【表７７】

```
From e In Employees _
Order By e.Salary Descending _
Return e.Name
Skip 5
```

【０２３３】
は、
【０２３４】
【表７８】

```
Employees.OrderByDescending(e => e.Salary).Select(e => e.Name).Skip(5)
```

【０２３５】
と等価とすることができる。
【０２３６】
ＷＨＩＬＥ修正子と共に使用されるときは、条件式を生み出して、基礎のＴＡＫＥＷＨＩＬＥまたはＳＫＩＰＷＨＩＬＥクエリ演算子に適用することができる。例えば、クエリ
【０２３７】
【表７９】

```
From e In Employees _
Order By e.Salary Descending _
Take While e.Salary > 50000 _
Return e.Name
```

【０２３８】
は、
【０２３９】
【表８０】

```
Employees.OrderByDescending(e => e.Salary). _
        .TakeWhile(e => e.Salary > 50000).Select(e => e.Name)
```

【０２４０】

と等価とすることができる。

【０２４１】

（ＧＲＯＵＰＢＹおよびＡＧＧＲＥＧＡＴＥ演算子の変換）

ＧＲＯＵＰＢＹ演算子、およびそれに続く集約するＳＥＬＥＣＴまたはＲＥＴＵＲＮ演算子は、ＧＲＯＵＰＢＹクエリ演算子メソッドへの呼出し、およびそれに続く、構築されたグループにわたる集約に変換することができる。例えば、クエリ

【０２４２】

【表８１】

```
From C In Customers _
Group By C.State _
Aggregate Youngest = Min(c.Age), Oldest = Max(c.Age)
Select State, Youngest, Oldest
```

【０２４３】

は、

【０２４４】

【表８２】

```
Customers
.GroupBy(c => New With { .State = c.State }, _
(g, k) => New With { .State = k, _
.Youngest = g.Min(c => c.Age),
.Oldest = g.Max(c => c.Age) })
```

【０２４５】

と等価とすることができる。

【０２４６】

（例示的なオブジェクト）

本明細書に含まれる全ての例は、２つの別々のオブジェクトセットのうちの一方を使用する。Ｃｏｍｐａｎｙオブジェクトは、会社に関する情報を表すことができ、階層型に接続することができる。

【０２４７】

【表８３】

```
Class Company
        Dim Name As String
        Dim Employees As List(Of Employees)
        Dim Customers As List(Of Customer)
End Class
Class Customer
        Dim Name As String
        Dim Orders As List(Of Order)
End Class
Class Order
        Dim Product As String
        Dim Price As Integer
End Class
Class Employee
        Dim Name As String
        Dim Birthday As Date
        Dim Position As String
        Dim Salary As Decimal
End Class
Dim Employees As List(Of Employee)
Dim Customers As List(Of Customer)
```

【０２４８】

Ｂｏｏｋオブジェクトは、例えば書店に蓄えられた本に関する情報を表すことができ、関係型に接続することができる。

【０２４９】

【表８４】

```
Class Book
        Dim BookID As Long
        Dim Title As String
        Dim PublisherID As Long
        Dim Price As Double
End Class
Class Author
        Dim AuthorID As Long
        Dim Name As String
        Dim Address As String
        Dim City As String
        Dim State As String
        Dim ZIP As String
End Class
Class BookAuthor
        Dim BookID As Long
        Dim AuthorID As Long
End Class
Class Publisher
        Dim PublisherID As Long
        Dim Name As String
        Dim Address As String
        Dim State As String
End Class
Dim Books As List(Of Book)
Dim Authors As List(Of Author)
Dim BookAuthors As List(Of BookAuthor)
Dim Publishers As List(Of Publisher)
```

【０２５０】

上記を念頭に置いて、次に図４に移ると、要素型をリアルタイムで推論することができ、かつ／または、推論された要素型に基づいてコンテキスト情報をインクリメンタルに提供することができる、コンピュータによって実現されるシステム４００が示されている。一般に、システム４００は、図１に関しておおむね上述したようにクエリ式１０４を受け取ることのできるユーザインタフェース１０２を備えることができる。加えて、システム４００は、ユーザインタフェース１０２および／またはコンテキストコンポーネント４０２に動作可能に結合（図示せず）させることのできるパターンストア１０８を備えることができる。コンテキストコンポーネント４０２は、クエリ式１０４を調べ、要素型に関係する推論をリアルタイムで行うことができる。例えば、前述のように、各クエリ句のクエリ演算子が特定のパターン（例えば図１からのクエリ演算子パターン１０６）に準拠するので、またさらに、要素型があるクエリ句から次のクエリ句にフローすることができるので、この２つの情報を利用して、いずれか所与の次のクエリ句に対する要素型を決定することができる。

【０２５１】

クエリ演算子がメソッド呼出しとして記述され、クエリ演算が純粋な機械的変形として適用されるときは、一般に、型解決がオーバーロード解決の間に行われることが必要とされる。このようにすると、クエリ演算子が型をある演算子から別の演算子にフローできると仮定した場合に、演算子は一般に、適用される次の演算子に型を提供できるように完全にバインドされなければならない。この結果、構文エラーまたは他の何らかの問題のせいでオーバーロード解決が失敗した場合、適用される次の演算子に型情報を提供するための従来の機構はない。この型フロー機構のない合成的クエリ演算子（後で図６からより詳細に論じる）は、結果として、妙なエラーメッセージ、異なる式セマンティクス、オートコンプリートのためのコンテキスト情報の不足、および全体的により遅いコンパイラスループットを引き起こす。

【０２５２】

述べたように、本発明は、要素型が完全なメソッドバインディングに依拠するのではなく、利用されている特定の演算子のパターンに依拠するようにすることのできる、クエリ演算子のパターンの形式化を採用することができる。さらに別の例示として、ユーザインタフェース１０２によって（例えばクエリ式１０４がタイプ入力されているときに）受け取られる、以下の例示的なクエリ式１０４を考えてみる。

【０２５３】

【表８５】

```
Dim q = From cust In Customers _
        Where cust.Name = “XYZ” & errorUnresolvedVarName _
        Select cust.
```

【０２５４】

第１のクエリ句はＦＲＯＭ演算子を含むが、このＦＲＯＭ演算子は、ソース型の要素とすることのできる匿名要素型「ｃｕｓｔ」として、照会可能ソース型「Ｃｕｓｔｏｍｅｒｓ」を導入することができる。論じたように、要素型は次のクエリ句にフローすることができ、この次のクエリ句で、ＷＨＥＲＥ演算子は、コレクションをフィルタにかけることができるが、他の点では要素型を改変しない。最後に、要素型は次のクエリ句にフローし、この次のクエリ句では、ＳＥＬＥＣＴ演算子、およびＳＥＬＥＣＴ演算子メソッドが受け取ると予想する関数の一部が見られるが、この次のクエリ句の残りはまだ不完全である。

【０２５５】

ＳＥＬＥＣＴ句中にフローする要素型は、今やこの特定のクエリ句に対するソース型としての働きをすることができる。このように、コンテキストコンポーネント４０２は、ソ

ース型およびクエリ演算子に基づいて、このクエリ句に対する要素型（例えばｃｕｓｔ）をリアルタイムで推論することができる。したがって、推論は完全にローカルとすることができ、クエリ式１０４を完全に変換する必要はない。そうではなく、コンテキストコンポーネント４０２は、バックグラウンドコンパイラと同様にして推論を実施することができる。

【０２５６】

加えて、コンテキストコンポーネント４０２の推論は、いくつかの方法でより大きい効率を促進することができる。例えば、推論は、変換が実施されるときに１回だけ実施すれば済む。したがって、クエリ式１０４中に２つのＷＨＥＲＥ演算子がある場合は、第２のＷＨＥＲＥ演算子についての変換は必要とせずに済む。加えて、推論は、一般に実施しなければならない型発見を避けるために、完全な、かつ通例はより高価なコンパイルと共に利用することもできる。したがって、要素型を前もって推論することを様々な方法で利用して、コンパイル時間を速くすることができる。

【０２５７】

さらに、コンテキストコンポーネント４０２が要素型（ここでは「Ｓｅｌｅｃｔ　ｃｕｓｔ．」クエリ句中の「ｃｕｓｔ」）を推論すると、コンテキストコンポーネント４０２は、コンテキスト情報４０４をインクリメンタルに提供することができる。例えば、「ｃｕｓｔ．」中でドットがタイプ入力されたとき、コンテキスト情報４０４を、図示のようなオートコンプリートを可能にするための便利で馴染みのあるポップアップウィンドウの形で、または別の形で、動的に提供することができる。このような特徴は、従来はクエリ式に利用可能でないことを理解されたい。というのは、クエリ式は（従来）、要素型を決定できるようになる前にまず変換しなければならないからである。また、完全でエラーのないクエリ式を供給することが従来の変換／コンパイルの前提であるため、これまではどんな同様のコンテキストおよび／またはオートコンプリート機構も排除されてきた。

【０２５８】

コンテキスト情報４０４は、クエリ式に対して非常に有用である可能性があり、また主としてＩＤＥにおいてコンテキスト上のフィードバックを提供する文脈で例示されているが、いずれの場合も必須ではない。例えば、本明細書に述べることは、クエリ式のクエリ演算子だけでなくそれ以外にも適用可能とすることができる。特に、ある演算子から次の演算子への型フローの概念、および、ある演算子と次の演算子との間の関係の概念は、型フローを使用できる場合のどんな演算子またはどんなＡＰＩにも適用することができる。そのような一例は、ＡＰＩ呼出しを共に連鎖させる際に存在する（例えば、型があるコマンドから次のコマンドにフローすることが意図される場合のコマンドラインに含まれるパイプライン）。別の例は、型はないが、遅延バインドされる最適化などのために何らかの形のフローを利用できる場合の、遅延バインドとすることができる。

【０２５９】

コンテキストコンポーネント４０２によって実施される推論は、受け取った既知の情報（例えば、既知かつ定義済みのクエリ演算子に関連するソース型）に基づいて完全に事前定義済みとすることができ、あるいは、他の態様によれば、推論は確率的とすることができることをさらに理解されたい。例えば、種々の型の演算子と共に利用されるとき、コンテキストコンポーネント４０２を利用して、利用可能なデータの全体またはサブセットを調べることができ、イベントおよび／またはデータを介して取り込まれた１組の観察から、システム、環境、および／またはユーザの状態についての推理を可能にするかまたはこれらの状態を推論することができる。推論は、特定のコンテキストまたはアクションを識別するのに利用することができ、あるいは、例えば複数の状態にわたる確率分布を生成することができる。推論は確率的とすることができる。すなわち、データおよびイベントの考慮に基づいた、当該の状態にわたる確率分布の計算とすることができる。推論はまた、１組のイベントおよび／またはデータから、より高レベルのイベントを構成するのに利用される技法を指すこともできる。

【０２６０】

このような推論の結果、イベントが時間的に近接して相関していようがいまいが、またイベントおよびデータの出所であるイベントおよびデータソースが１つであろうが複数であろうが、１組の観察されたイベントおよび／または記憶されたイベントデータから、新しいイベントまたはアクションを構築することができる。様々な分類（明示的および／または暗黙的に訓練された）手法および／またはシステム（例えばサポートベクターマシン、ニューラルネットワーク、エキスパートシステム、ベイズ信念ネットワーク、ファジィ論理、データ融合エンジンなど）を、本発明に関連する自動的なおよび／または推論されたアクションの実施と共に利用することができる。

【０２６１】

分類器は、入力属性ベクトルｘ＝（ｘ１，ｘ２，ｘ３，ｘ４，ｘｎ）を、あるクラスにこの入力が属する信頼度にマッピングする関数とすることができ、すなわちｆ（ｘ）＝ｃｏｎｆｉｄｅｎｃｅ（ｃｌａｓｓ）である。このような分類は、確率ベースおよび／または統計ベースの分析（例えば分析の効用およびコストを考慮に入れる）を利用して、自動的に実施されるようユーザが望むアクションを予測または推論することができる。サポートベクターマシン（ＳＶＭ）は、利用できる分類器の例である。ＳＶＭは、可能性のある入力の空間における超曲面を見つけることによって動作し、超曲面は、トリガ基準を非トリガイベントから分離することを試みる。直感的に、これによって分類は、訓練データと近いが同一ではないテスト用データに対して正しいものになる。他の有向および無向モデル分類手法は、例えばナイーブベイズ、ベイズネットワーク、決定樹、ニューラルネットワーク、ファジィ論理モデルを含み、また、種々の独立パターンを提供する確率分類モデルを利用することができる。分類は、本明細書においては、優先順位のモデルを開発するのに利用される統計的回帰も含む。

【０２６２】

図５に、本発明による方法を示す。説明を簡単にするために、本明細書におけるこの方法および他の方法を一連の動作として図示および記述するが、いくつかの動作は本明細書に図示および記述するのとは異なる順序で、かつ／または他の動作と同時に実施することができるので、本発明は動作の順序によって限定されないことを理解および認識されたい。例えば、別法として方法は、状態図などで、相互に関係付けられる一連の状態またはイベントとして表すこともできることは、当業者なら理解および認識するであろう。さらに、本発明により方法を実施するのに、例示する全ての動作が必要とはされない場合もある。加えて、以下におよび本明細書全体を通して開示する方法は、そのような方法をコンピュータに搬送および転送するのを容易にするために、製品に記憶することができることもさらに理解されたい。用語「製品」は、本明細書においては、任意のコンピュータ可読デバイス、キャリア、または媒体からアクセス可能な、コンピュータプログラムを包含するものとする。

【０２６３】

次に図５を参照すると、要素型の型フローを容易にするための、コンピュータによって実施される方法が示されている。一般に参照番号５０２で、クエリ演算子を含むクエリ句の一部を受け取ることができる。クエリ演算子は、関連するメソッド呼出しにマッピングすることができ、メソッド呼出しは、引数（例えば関数、値、コレクションなど）を受け取ることができ、型付けされた結果（例えば照会可能型、数値型、順序付けされたコレクションなど）を返すことができる。メソッド呼出しは、クエリ演算子パターンに従って定義することができる。

【０２６４】

参照番号５０４で、要素型を決定するためにソース型およびクエリ演算子を利用することができる。例えば、クエリ演算子パターンを形式化することができるので、クエリ演算子のパターンをソース型と共に利用して要素型を推論することができる。参照番号５０６で、前のクエリの要素型を、参照番号５０４で論じた利用する動作のためのソース型として採用することができる。これにより、要素型は、あるクエリ句から次のクエリ句に再帰的にフローすることができる。

【０２６５】
前述のことを念頭に置けば、他の方法でプログラミング言語のクエリ機能を拡張するためにいくつかの特徴および／または態様を利用できることは認識および理解されるであろう。一例として、従来はＡＰＩ呼出しに関連していた合成的能力によってコンプリヘンションなどコンピュータベースおよび／または言語ベースの構造を提供するために、本明細書に展開する態様を効果的に利用することができる。これらおよび他の関連する概念についての記述を、図６に関して見ることができる。
【０２６６】
次に図６に移ると、構成可能なクエリコンプリヘンションおよび／または拡張可能なクエリ式を容易にすることのできる、コンピュータによって実現されるシステム６００のブロック図が提供されている。一般に、システム６００は、初期化データ６０４を受け取ることのできる変形（ｔｒａｎｓｆｏｒｍａｔｉｏｎ）コンポーネント６０２を含むことができる。図示のように、初期化データ６０４は、例示的なクエリ式６０６中の第１のクエリ句とすることができ、第１のクエリ句は通常、ＦＲＯＭ句であるように制限される（例えば、第１のクエリ句のクエリ演算子がＦＲＯＭ演算子である）。初期化データ６０４は、コレクション（例えば図示のＣｕｓｔｏｍｅｒｓ）、または式（例えば、図３に関して詳述したＦＲＯＭ演算子に関連する前述のＴａｘ＝ｂ．Ｐｒｉｃｅ＊０．０８８）を含むことができる。いずれの場合も、初期化データは制御変数を含むこともでき、制御変数は、適用例に応じて、コレクションに関連するか（例えばＣが制御変数）、あるいは式に関連する（例えばＴａｘが制御変数）。
【０２６７】
変形コンポーネント６０２はまた、クエリ式６０６によって特徴付けられるシーケンス中のクエリ句のセット６０８を受け取ることができる。クエリ句のセット６０８は空集合とすることができ、その場合は、後でより詳細に述べるように、クエリ式６０６は単一のクエリ句（例えば、初期化データ６０４として利用されるＦＲＯＭ句）からなることを理解されたい。初期化データ６０４で行えると同様に、変形コンポーネント６０２はまた、セット６０８中の各クエリ句につき、スコープ内制御変数を解決する（例えばポピュレートまたは変形する）ことができる。例えば、変形コンポーネント６０２は、利用可能な型に各クエリ句のクエリ演算子を適用することによって、セット６０８中の各クエリ句を、クエリ式６０６によって指示される順序で順番に処理することができる。
【０２６８】
本発明によれば、クエリ句のセット６０８は、例えばＸＱｕｅｒｙブランドの言語の各バージョンによって定義されるＦＬＷＯＲ／ＦＬＷＲや、ＳＱＬの実装形態で予想されるＳｅｌｅｃｔ－Ｆｒｏｍ－Ｗｈｅｒｅなどのユビキタスな構文テンプレートによって制限する必要はないことを理解されたい。そうではなく、演算子の順序が自由裁量であるときでも、個々のクエリ句を有効に供給し、これらをモジュール方式で継ぎ合わせ、これらが直感的な結果を生むことができるように、セット６０８は、性質上、合成的とすることができる。
【０２６９】
これにより、システム６００はまた、クエリ式６０６によって導入される全ての制御変数のスコープを管理することのできるコンプリヘンションコンポーネント６１０を含むことができる。制御変数のスコープは、クエリ句の演算子に基づいて決定することができ、スコープ内制御変数は、次のクエリ句に渡すことができる。例えば、変形コンポーネント６０２は、ある種の情報がパイプラインによってあるクエリ句から次のクエリ句にフローできるような、パイプラインと考えることのできるものを確立することができる。したがって、変形コンポーネントは、クエリ句と共にコレクションを受け取り、クエリ句に関連する演算子に従ってコレクションを変形（例えばフィルタリング、射影など）し、変形したコレクションを、次のクエリ句に利用可能なようにパイプラインに出力することができる。
【０２７０】

同様に、スコープ内制御変数もまた次のクエリ句に渡すことができ、受け取ったクエリ句のタイプに基づいて、正確にどの制御変数がスコープ内なのかをコンプリヘンションコンポーネント６１０によって定義することができる。通常、次のクエリ句は、スコープ内である変数へのアクセスしか有することができない。この特徴ならびに他の特徴について、図６に加えて図７も参照してさらに述べることができる。

【０２７１】

図７は、コンプリヘンションコンポーネント６１０によって決定することのできる、例示的なクエリ式６０６の制御変数のスコープに関係する例示的な図である。本発明の一態様によれば、コンプリヘンションコンポーネント６１０は、クエリ句／演算子のタイプに基づいて、新しい制御変数を宣言することができる（そして新しい制御変数をスコープ内にすることができる）。例えば、いくつかの型のクエリ句は、すでにスコープ内である制御変数に加えて、新しい制御変数を導入することができる。すなわち、ＦＲＯＭ、ＬＥＴ、ＳＥＬＥＣＴ、またはＧＲＯＵＰＢＹクエリ句は、この結果を生むことができる。ＦＲＯＭ句によって導入された制御変数は、通常、次のＳＥＬＥＣＴまたはＧＲＯＵＰＢＹ句まで蓄積され、その後、コンプリヘンションコンポーネント６１０によってスコープ外にすることができる。このような場合を、ＦＲＯＭ句７０２、７０４（これらは、初期化データ６０４、およびクエリ句のセット６０８に含まれる最初のクエリ句にマッピングする）によって例示するが、これらの句の制御変数は、ＳＥＬＥＣＴ句７１０の適用までスコープ内に留まる。

【０２７２】

句７０２は、Ｃｕｓｔｏｍｅｒコレクションおよび制御変数Ｃを導入する。したがって、この情報は、パイプラインによって次のクエリ句に供給することができる。よって、{Ｃ　Ａｓ　Ｃｕｓｔｏｍｅｒ}を句７０４に利用可能にすることができる。句７０４もまたＦＲＯＭ句であり、このことは、本発明の合成的性質によって可能である。句７０４は、制御変数Ｃを利用して、新しいコレクションＣ．Ｏｒｄｅｒｓ（例えば全ての顧客の全ての注文）ならびに新しい制御変数Ｏを導入し、この制御変数Ｏは、コンプリヘンションコンポーネント６１０がスコープ内にすることができる。したがって、次のクエリ句７０６は、スコープ内制御変数ＣとＯの両方へのアクセスを有する。

【０２７３】

おおむね上述したように、いくつかのクエリ句型はスコープ内変数に影響を及ぼすことができるが、他のクエリ句型（例えば、ＷＨＥＲＥ、ＯＲＤＥＲＢＹ、ＤＩＳＴＩＮＣＴなど）は必ずしもそうではない。このため、クエリ句７０６および７０８はコレクションに影響を及ぼすことはできるが、スコープ内変数（例えばＣおよびＯ）は変わらないものとすることができ、したがって、クエリ句７０８および７１０それぞれが変形コンポーネント６０２によって受け取られたとき、各句によってそれぞれこれらのスコープ内変数にアクセス可能とすることができる。

【０２７４】

本発明の一態様によれば、コンプリヘンションコンポーネント６１０は、受け取ったクエリ句のタイプに基づいて、既存の制御変数をスコープ外にすることができる。ＳＥＬＥＣＴクエリ句７１０は、この特徴の特質的な例示を提供する。クエリ句７１０は、制御変数ＣおよびＯにエクスポーズされるが、ＳＥＬＥＣＴ句の特徴の１つは、現在のスコープ内制御変数を隠蔽し、新しいスコープ内制御変数（例えばＮａｍｅ　Ａｓ　Ｓｔｒｉｎｇおよびｐｒｉｃｅ　Ａｓ　Ｉｎｔ）を導入することとすることができる。クエリ句７１２には、これらの新しい制御変数ＮａｍｅおよびＰｒｉｃｅを供給することができるが、クエリ句７１２は一般に、コンプリヘンションコンポーネント６１０がＳＥＬＥＣＴ句７１０との関連でスコープ外にした前の制御変数（例えばＣおよびＯ）にはアクセスできない。完全を期すために、少数の特徴に留意されたい。第１に、クエリ句７１０は、コンマで境界を定められた複合オペランドのもう１つの例を示している。

【０２７５】

留意すべき第２の特徴は、ＷＨＥＲＥ句７１２がＳＥＬＥＣＴ句７１０に続くことであ

る。従来のクエリ言語は、ＳＥＬＥＣＴ、ＲＥＴＵＲＮ、または同様の演算子などの射影演算子を含むクエリ句が適用されると、クエリを終了する。このような従来のクエリ言語のユーザが射影後もクエリ演算を継続したい場合は、前のクエリの結果を参照できる新しいクエリを実装しなければならない。しかし、例えば本発明の合成的性質により、クエリ式６０６は複数のＳＥＬＥＣＴ句を含むことができ、クエリ式６０６は、ＳＥＬＥＣＴ句が発生した後も継続することができる（例えば句７１２で示すように）。同様の趣旨で、前の章で触れたが、クエリ式６０６は２つのＦＲＯＭ句（例えば句７０２および７０４）も含むことを指摘しておくべきであり、これは従来のクエリ言語が可能にすることのできないさらに別の態様である。

【０２７６】

明確な例によって示されてはいないが、クエリ式６０６はまた、他のクエリ句タイプを含むこともでき、特に、これらの他のクエリ句タイプの多くは、制御変数のスコープに影響を及ぼすことができる。例えば、ＲＥＴＵＲＮ句を受け取ると、コンプリヘンションコンポーネント６１０は、全ての制御変数をスコープ外にすることができる。別の例として、クエリ句のタイプに基づいて、コンプリヘンションコンポーネント６１０は、新しい制御変数を宣言およびスコープし（例えばスコープ内にし）、後続のクエリ句の特定のセットについて、既存の制御変数（例えば前のクエリ句のスコープ内制御変数）をスコープすることができる。ＧＲＯＵＰＢＹ、ＡＧＧＲＥＡＴＥ、その他は、このような挙動を容易にすることができ、それにより、既存の制御変数を、各ＧＲＯＵＰ演算子に及ぶＡＧＧＲＥＧＡＴＥクエリ句の範囲にわたってスコープすることができる。

【０２７７】

本発明の別の態様によれば、コンプリヘンションコンポーネント６１０は、受け取ったクエリ句のタイプに基づいて、全てのスコープ内制御変数のタプルに対するエイリアスを生成することができる。例えば、ＩＮＴＯクエリ句は、例えばコンプリヘンションコンポーネント６１０による、エイリアスの作成を容易にすることができるが、エイリアスはそれ自体が制御変数である必要はない。

【０２７８】

引き続き図６および７を参照して、本発明の様々な追加の態様を強調することができる。いくつかの利点は、有効なクエリ式６０６の拡張可能性に関係する（例えば、ほぼ任意のタイプのほぼ任意の数のクエリ句を使用してクエリ式６０６を拡張することができる）が、別の利点は、クエリ式６０６がどんな時点でも終了できることとすることができる。したがって、クエリ式６０６は不定数のクエリ句を含む可能性を有するが、クエリ式６０６全体が、単一の制御変数を導入した後で終了する単一のクエリ句のみで構成されてもよい。したがって、完全かつ有効なクエリ式６０６の最も単純な例の１つは、以下のような初期化データ６０４のみを含むクエリ式６０６であり、クエリ式６０６が明示的なＳＥＬＥＣＴ、ＲＥＴＵＲＮ、または他の最終クエリ演算子を含むことは必要とされない。

【０２７９】

【表８６】

From C In Customers

【０２８０】

説明として、変形コンポーネント６０２は、受け取った各クエリ句に暗黙的なＳＥＬＥＣＴまたはＲＥＴＵＲＮを付加するように構成されてよい。したがって、パイプラインに含まれる結果的なコレクションが所望の形状である限り、明示的なステートメントがこれらの中間結果を選択または返却する必要はない。そうではなく、クエリ式６０６が所与の句、例えば７０２～７０８のいずれか１つの後で終了する場合、パイプライン中のコレクションを結果として出力することができ、出力のフォーマットは、スコープ内制御変数の数に基づいて推論することができる。

【０２８１】

例えば、クエリ式６０６が句７０４～７０８のいずれかの後で終了した場合（それぞれの場合に、２つのスコープ内制御変数ＣおよびＯがある）、所望の出力が名前－値の対のコレクションとして存在するはずであると推論することができる。したがって、出力は、各制御変数のフィールドを有するタプルとすることができる。一方、句７０２の後など、スコープ内の制御変数が１つしかない場合は、クエリ式６０６の実施者は、フィールドｃを有するタプルを返されることを通常は予想しないであろう。そうではなく、実施者は、顧客のコレクションを望む可能性が高いであろう。したがって、スコープ内制御変数が１つしかない場合は、出力は単に、コレクションの基礎の値とすることができる。

【０２８２】

本発明の一態様によれば、ＦＲＯＭ句によって導入される制御変数は、式ならびにコレクションに関連することができる。代表例として、上に紹介した以下のクエリ部分を考えてみる。

【０２８３】

【表８７】

```
From b in Books _
From Tax = b.Price * 0.088 _
```

【０２８４】

第１のＦＲＯＭ句は、コレクションＢｏｏｋｓに対する制御変数ｂを導入するが、制御変数ｂは、スコープ内にされ、次のクエリ句によってアクセス可能である。次のＦＲＯＭ句は、式（コレクションではなく）ｂ．Ｐｒｉｃｅ＊０．０８８に関連する制御変数Ｔａｘを導入する。２つのＦＲＯＭ句のうちの後者により、式の結果として、アクセス可能な名前が提供される効果が得られる。この名前は、例えば後続のＳＥＬＥＣＴ句までスコープ内に留まることのできる、制御変数ｔａｘによって参照することができる。この特徴は、クエリ内のプロシージャステートメントに類似すると考えることができ、クエリ内のプロシージャステートメントは、式の値が後で何回も利用されるかもしれないにもかかわらず値が１回計算されれば済むようにすることができ、値を再計算する必要はない。

【０２８５】

本発明の別の態様によれば、単一のクエリ式６０６のコンテキスト内で、集合演算を利用することができる。従来のクエリ言語は通常、入力として単一のコレクションだけしか可能にしないが、変形コンポーネント６０２は、入力として２つのコレクションを受け取ることができる。例えば、変形コンポーネント６０２は、複数のコレクションを受け取るように構成されてよく、複数のコレクションに関連する制御変数がクエリ句のタイプ（例えばＵＮＩＯＮ句、ＩＮＴＥＲＳＥＣＴ句など）に従ってマージされた単一のコレクションを出力することができる。マージされた制御変数は、スコープ内にされ、関連する集合演算子の中を通り、次のクエリ句からアクセス可能にされてよく、したがってクエリ式６０６は継続することができる。制御変数は、一般に同じ型でなければならず、一般に同じ名前を有さなければならないが、当然、２つの異なるコレクションに関連するものとすることができる。

【０２８６】

次に図８に移ると、クエリコンプリヘンションの構築を合成的方式で容易にするための、コンピュータによって実施される方法８００が示されている。参照番号８０２で開始し、コレクション（または式）、およびコレクション（または式）に関連する制御変数を得ることができる。一般に、この得られたデータは、クエリ式の第１のクエリ句の産物であり、第１のクエリ句は、普通はＦＲＯＭ句である。参照番号８０４で、クエリ式に含まれるクエリ句のセットからの現在のクエリ句を受け取ることができる。コンプリヘンション

がモノリシックであることを必要とする従来の言語とは異なり、現在のクエリ句に対して、あるいは大抵はクエリ式全体に対して、構文上の順序付け制限を課す必要はないことを理解されたい。

【０２８７】

参照番号８０６で、現在のクエリ句に従ってコレクションを修正することができる。単純に言えば、入力として受け取ったコレクションを、現在のクエリ句に関連するクエリ演算子のガイドラインに基づいて変形することができる。現在のクエリ句は、モノリシックな構文テンプレートに基づいてではなく、現在のクエリ句に関連する予想される型に従って評価できることを理解されたい。次に、参照番号８０８で、修正されたコレクションを次のクエリ句に渡すことができる。したがって、現在のクエリ句の出力を、次のクエリ句の入力として利用することができる。

【０２８８】

参照番号８１０で、現在のクエリ句に基づいて制御変数のスコープを決定することができる。例えば、いくつかの型のクエリ句は、既存の制御変数に対するどんな変更も容易にしないが、他のクエリ句は、新しい制御変数の導入を容易にすることができ（場合によっては、次のクエリ句の特定の連続に対してのみ）、さらに他のクエリ句は、既存の制御変数をスコープ外にするとともに新しい制御変数を導入（オプションで）するのを容易にすることができる。参照番号８１２で、現在のクエリ句に対するスコープ内制御変数へのアクセスを、次のクエリ句に提供することができる。

【０２８９】

次に図９を参照すると、開示したアーキテクチャを実行するように動作可能な例示的なコンピュータシステムのブロック図が示されている。本発明の様々な態様に関する追加の文脈を提供するために、図９および後続の考察では、本発明の様々な態様を実施できる適切なコンピューティング環境９００の簡単かつ一般的な記述を提供するものとする。加えて、１つまたは複数のコンピュータ上で稼動することのできるコンピュータ実行可能命令の一般的な文脈で本発明を上述したが、本発明を他のプログラムモジュールと共に、かつ／またはハードウェアとソフトウェアの組合せとして実施することもできることは、当業者なら理解するであろう。

【０２９０】

一般に、プログラムモジュールは、特定のタスクを実施するか特定の抽象データ型を実装するルーチン、プログラム、コンポーネント、データ構造などを含む。さらに、本発明の方法は他のコンピュータシステム構成で実施することもできることは、当業者なら理解するであろう。これら他のコンピュータシステム構成は、シングルプロセッサまたはマルチプロセッサのコンピュータシステム、ミニコンピュータ、メインフレームコンピュータ、ならびにパーソナルコンピュータ、ハンドヘルドコンピューティングデバイス、マイクロプロセッサベースのまたはプログラム可能な消費者電子機器などを含み、これらはそれぞれ、１つまたは複数の関連するデバイスに動作可能に結合させることができる。

【０２９１】

例示した本発明の態様はまた、通信ネットワークを介してリンクされたリモート処理デバイスによっていくつかのタスクが実施される分散コンピューティング環境で実施してもよい。分散コンピューティング環境では、プログラムモジュールは、ローカルとリモートの両方のメモリ記憶デバイス中に位置することができる。

【０２９２】

コンピュータは通常、様々なコンピュータ可読媒体を備える。コンピュータ可読媒体は、コンピュータによってアクセスできる任意の利用可能な媒体とすることができ、揮発性と不揮発性の媒体、取外し可能と取外し不可能の媒体の両方を含む。限定ではなく例として、コンピュータ可読媒体は、コンピュータ記憶媒体および通信媒体を含むことができる。コンピュータ記憶媒体は、コンピュータ可読命令、データ構造、プログラムモジュール、または他のデータなどの情報を記憶するための任意の方法または技術で実現された、揮発性と不揮発性、取外し可能と取外し不可能の両方の媒体を含むことができる。コンピュ

ータ記憶媒体は、ＲＡＭ、ＲＯＭ、ＥＥＰＲＯＭ、フラッシュメモリ、または他のメモリ技術、ＣＤ－ＲＯＭ、ディジタル多用途ディスク（ＤＶＤ）、または他の光学ディスク記憶装置、磁気カセット、磁気テープ、磁気ディスク記憶装置、または他の磁気記憶デバイス、あるいは、所望の情報を記憶するのに使用できコンピュータによってアクセスできる他の任意の媒体を含むが、これらに限定されない。

【０２９３】

通信媒体は通常、コンピュータ可読命令、データ構造、プログラムモジュール、または他のデータを、搬送波や他のトランスポート機構などの被変調データ信号に組み入れるものであり、任意の情報送達媒体を含む。用語「被変調データ信号」は、信号中の情報を符号化するようにしてその特性の１つまたは複数が設定または変更される信号を意味する。限定ではなく例として、通信媒体は、配線式ネットワークや直接配線式接続などの配線式媒体と、音響、無線周波数、赤外線などのワイヤレス媒体および他のワイヤレス媒体とを含む。以上のいずれかの組合せも、コンピュータ可読媒体の範囲に含めるべきである。

【０２９４】

再び図９を参照すると、本発明の様々な態様を実施するための例示的な環境９００はコンピュータ９０２を含み、コンピュータ９０２は、処理装置９０４、システムメモリ９０６、およびシステムバス９０８を備える。システムバス９０８は、限定しないがシステムメモリ９０６を含めたシステムコンポーネントを、処理装置９０４に結合する。処理装置９０４は、様々な市販のプロセッサのいずれかとすることができる。デュアルマイクロプロセッサおよび他のマルチマイクロプロセッサアーキテクチャを、処理装置９０４として利用してもよい。

【０２９５】

システムバス９０８は、いくつかのタイプのバス構造のいずれかとすることができ、これらのバス構造は、様々な市販のバスアーキテクチャのいずれかを使用してメモリバス（メモリコントローラありまたはなし）、周辺バス、およびローカルバスにさらに相互接続することができる。システムメモリ９０６は、読取専用メモリ（ＲＯＭ）９１２およびランダムアクセスメモリ（ＲＡＭ）９１０を含む。ＲＯＭ、ＥＰＲＯＭ、ＥＥＰＲＯＭなどの不揮発性メモリ９１２にはＢＩＯＳ（ｂａｓｉｃ　ｉｎｐｕｔ／ｏｕｔｐｕｔ　ｓｙｓｔｅｍ）が記憶され、このＢＩＯＳは、起動中などにコンピュータ９０２内の要素間で情報を転送するのを助ける基本ルーチンを含む。ＲＡＭ９１０は、データをキャッシュするためのスタティックＲＡＭなどの高速ＲＡＭを含むこともできる。

【０２９６】

コンピュータ９０２はさらに、内部ハードディスクドライブ（ＨＤＤ）９１４（例えばＥＩＤＥ、ＳＡＴＡ）を備え、この内部ハードディスクドライブ９１４はまた、適切なシャーシ（図示せず）中で外部使用されるように構成されてもよい。コンピュータ９０２はさらに、磁気フロッピーディスクドライブ（ＦＤＤ）９１６（例えば取外し可能ディスケット９１８に対して読取りまたは書込みを行うため）、および光学ディスクドライブ９２０（例えばＣＤ－ＲＯＭ９２２の読取り、またはＤＶＤなど他の大容量光学媒体に対して読取りまたは書込みを行うため）を備える。ハードディスクドライブ９１４、磁気ディスクドライブ９１６、および光学ディスクドライブ９２０は、ハードディスクドライブインタフェース９２４、磁気ディスクドライブインタフェース９２６、および光学ドライブインタフェース９２８によってそれぞれシステムバス９０８に接続することができる。外部ドライブ実装形態の場合のインタフェース９２４は、ユニバーサルシリアルバス（ＵＳＢ）とＩＥＥＥ１３９４インタフェースとのうちの少なくとも一方または両方の技術を含む。他の外部ドライブ接続技術も、本発明の企図の内にある。

【０２９７】

ドライブおよびそれらに関連するコンピュータ記憶媒体は、データ、データ構造、コンピュータ実行可能命令などの不揮発性記憶を提供する。コンピュータ９０２の場合、ドライブおよび媒体は、適切なディジタルフォーマットの任意のデータの記憶に対応する。上記のコンピュータ可読媒体の記述では、ＨＤＤ、取外し可能磁気ディスケット、およびＣ

ＤやＤＶＤなどの取外し可能光学媒体に言及しているが、ｚｉｐドライブ、磁気カセット、フラッシュメモリカード、カートリッジなど、コンピュータによって読取り可能な他のタイプの媒体を例示的な動作環境で使用してもよいこと、さらに、このような媒体はどれも、本発明の方法を実施するためのコンピュータ実行可能命令を含んでよいことは、当業者には理解されるはずである。

【０２９８】

ドライブおよびＲＡＭ９１０には、オペレーティングシステム９３０、１つまたは複数のアプリケーションプログラム９３２、他のプログラムモジュール９３４、およびプログラムデータ９３６を含めて、いくつかのプログラムモジュールを記憶することができる。ＲＡＭ９１０には、オペレーティングシステム、アプリケーション、モジュール、および／またはデータの全部または一部をキャッシュすることもできる。本発明は、様々な市販のオペレーティングシステム、またはオペレーティングシステムの組合せを使用して実施できることを理解されたい。

【０２９９】

ユーザは、１つまたは複数の配線式／ワイヤレス入力デバイス、例えばキーボード９３８や、マウス９４０などのポインティングデバイスを介して、コンピュータ９０２にコマンドおよび情報を入力することができる。他の入力デバイス（図示せず）は、マイクロホン、赤外線リモートコントロール、ジョイスティック、ゲームパッド、スタイラスペン、タッチスクリーンなどを含んでよい。これらおよび他の入力デバイスは、システムバス９０８に結合される入力デバイスインタフェース９４２を介して処理装置９０４に接続されることが多いが、パラレルポート、ＩＥＥＥ１３９４シリアルポート、ゲームポート、ＵＳＢポート、赤外線インタフェースなど、他のインタフェースで接続することもできる。

【０３００】

モニタ９４４または他のタイプの表示デバイスも、ビデオアダプタ９４６などのインタフェースを介してシステムバス９０８に接続される。モニタ９４４に加えて、コンピュータは通常、スピーカやプリンタなど、他の周辺出力デバイス（図示せず）も備える。

【０３０１】

コンピュータ９０２は、リモートコンピュータ９４８など１つまたは複数のリモートコンピュータへの配線式および／またはワイヤレス通信を介した論理接続を用いて、ネットワーク化された環境で動作することができる。リモートコンピュータ９４８は、ワークステーション、サーバコンピュータ、ルータ、パーソナルコンピュータ、ポータブルコンピュータ、マイクロプロセッサベースの娯楽機器、ピアデバイス、または他の一般的なネットワークノードとすることができ、通常は、コンピュータ９０２に関して述べた要素の多くまたは全てを備えるが、簡単にするためにメモリ／記憶デバイス９５０のみが示してある。図示の論理接続は、ローカルエリアネットワーク（ＬＡＮ）９５２、および／またはより大きいネットワーク、例えばワイドエリアネットワーク（ＷＡＮ）９５４への、配線式／ワイヤレス接続性を含む。このようなＬＡＮおよびＷＡＮネットワーキング環境は、オフィスおよび会社で一般的であり、イントラネットなど企業全体のコンピュータネットワークを容易にするが、これらのネットワークは全て、大域的な通信ネットワーク、例えばインターネットに接続することができる。

【０３０２】

ＬＡＮネットワーキング環境で使用されるときは、コンピュータ９０２は、配線式および／またはワイヤレス通信ネットワークインタフェースあるいはアダプタ９５６を介して、ローカルネットワーク９５２に接続される。アダプタ９５６は、ＬＡＮ９５２への配線式またはワイヤレス通信を容易にすることができ、ＬＡＮ９５２上には、ワイヤレスアダプタ９５６と通信するためのワイヤレスアクセスポイントが配置されてもよい。

【０３０３】

ＷＡＮネットワーキング環境で使用される場合、コンピュータ９０２は、モデム９５８を備えることができ、あるいは、ＷＡＮ９５４上の通信サーバに接続され、あるいは、インターネットなどによってＷＡＮ９５４を介した通信を確立するための他の手段を有する

。モデム９５８は内蔵または外付けとすることができ、また配線式またはワイヤレスデバイスとすることができ、シリアルポートインタフェース９４２を介してシステムバス９０８に接続される。ネットワーク化された環境では、コンピュータ９０２に関して示したプログラムモジュールまたはその一部をリモートのメモリ／記憶デバイス９５０に記憶することができる。図示のネットワーク接続は例であり、コンピュータ間で通信リンクを確立する他の手段を使用してもよいことは理解されるであろう。

【０３０４】

コンピュータ９０２は、ワイヤレス通信において動作可能に配置された任意のワイヤレスデバイスまたはエンティティ、例えばプリンタ、スキャナ、デスクトップおよび／またはポータブルコンピュータ、ポータブルデータアシスタント、通信衛星、ワイヤレスに検出可能なタグに関連する任意の機器または場所（例えばキオスク、新聞売店、手洗所）、ならびに電話機と通信するように動作可能である。これは、少なくともＷｉ－ＦｉおよびＢｌｕｅｔｏｏｔｈ（商標）ワイヤレス技術を含む。したがって、通信は、従来のネットワークと同様に事前定義済みの構造とすることもでき、あるいは単に、少なくとも２つのデバイス間のその場限りの通信とすることもできる。

【０３０５】

Ｗｉ－Ｆｉまたはワイヤレスフィデリティ（Ｗｉｒｅｌｅｓｓ　Ｆｉｄｅｌｉｔｙ）は、家庭のソファ、ホテルの部屋のベッド、または職場の会議室からワイヤなしでインターネットに接続するのを可能にする。Ｗｉ－Ｆｉは、例えばコンピュータなどのデバイスが室内でも室外でも、すなわち基地局の範囲内のどこででもデータを送受信できるようにする、セルホン中で使用される技術に類似するワイヤレス技術である。Ｗｉ－Ｆｉネットワークは、ＩＥＥＥ８０２．１１（ａ、ｂ、ｃなど）と呼ばれる無線技術を使用して、安全かつ信頼性があり高速なワイヤレス接続性を提供する。Ｗｉ－Ｆｉネットワークを使用して、コンピュータを相互に、インターネットに、および配線式ネットワーク（ＩＥＥＥ８０２．３またはＥｔｈｅｒｎｅｔを使用する）に接続することができる。Ｗｉ－Ｆｉネットワークは、免許不要の２．４および５ＧＨｚ無線帯域で、例えば１１Ｍｂｐｓ（８０２．１１ａ）または５４Ｍｂｐｓ（８０２．１１ｂ）のデータレートで、あるいは、両方の帯域を含む製品と共に（デュアルバンド）動作し、したがって、ネットワークは、多くの職場で使用される基本的な９ＢａｓｅＴ配線式ＥｔｈｅｒｎｅｔネットワークとＥｔｈｅｒｎｅｔネットワークと同様の実世界性能を提供することができる。

【０３０６】

次に図１０を参照すると、開示したアーキテクチャを実行するように動作可能な例示的なコンピュータコンパイルシステムの概略ブロック図が示されている。システム１０００は、１つまたは複数のクライアント１００２を含む。クライアント１００２は、ハードウェアおよび／またはソフトウェア（例えばスレッド、プロセス、コンピューティングデバイス）とすることができる。クライアント１００２は、例えば、本発明を利用することによって、クッキー（複数可）および／または関連するコンテキスト情報を収容することができる。

【０３０７】

システム１０００は、１つまたは複数のサーバ１００４も含む。サーバ１００４もまた、ハードウェアおよび／またはソフトウェア（例えばスレッド、プロセス、コンピューティングデバイス）とすることができる。サーバ１００４は、例えば、本発明を利用することによって、変形を実施するためのスレッドを収容することができる。クライアント１００２とサーバ１００４との間の可能な通信の１つは、２つ以上のコンピュータプロセス間で伝送されるように適合されたデータパケットの形とすることができる。データパケットは、例えば、クッキーおよび／または関連するコンテキスト情報を含むことができる。システム１０００は、クライアント１００２とサーバ１００４との間の通信を容易にするために利用することのできる通信フレームワーク１００６（例えば、インターネットなどの大域的な通信ネットワーク）を含む。

【０３０８】

通信は、配線式（光ファイバを含む）および／またはワイヤレス技術を介して容易にすることができる。クライアント１００２は、クライアント１００２にとってローカルな情報（例えばクッキー（複数可）および／または関連するコンテキスト情報）を記憶するのに利用することのできる１つまたは複数のクライアントデータストア１００８に動作可能に接続される。同様に、サーバ１００４は、サーバ１００４にとってローカルな情報を記憶するのに利用することのできる１つまたは複数のサーバデータストア１０１０に動作可能に接続される。

【０３０９】

上述したことは、様々な実施形態の例を含む。当然、実施形態について述べるために、コンポーネントまたは方法の考えられるあらゆる組合せを記述することは不可能だが、さらに多くの組合せおよび置換が可能であることは当業者なら理解するであろう。したがって、この詳細な説明は、添付の特許請求の趣旨および範囲の内に入るあらゆる改変、修正、および変形を包含するものとする。

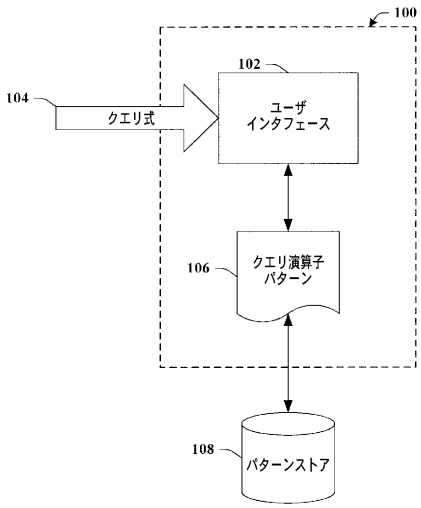
【０３１０】

特に、前述のコンポーネント、デバイス、回路、システムなどによって実施される様々な機能に関して、このようなコンポーネントを記述するのに使用される用語（「手段」への言及を含む）は、開示した構造に対して構造的に等価でなくても、本明細書に例示した実施形態の例示的な態様における機能を実施する述べたコンポーネントの指定の機能を実施する任意のコンポーネント（例えば機能的均等物）に、特に指示がない限り対応するものとする。これに関して、実施形態は、システム、ならびに、様々な方法の動作および／またはイベントを実施するためのコンピュータ実行可能命令を有するコンピュータ可読媒体を含むこともまた理解されるであろう。

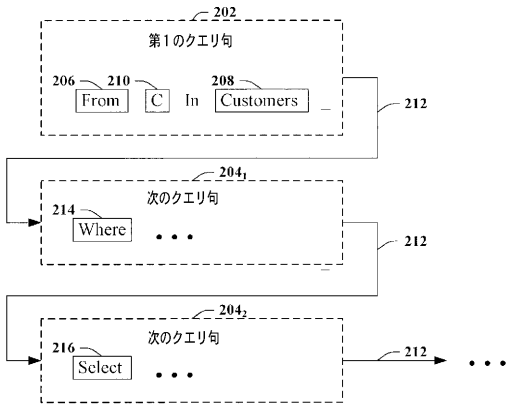
【０３１１】

加えて、特定の特徴をいくつかの実装形態のうちの１つのみに関して開示した場合もあるが、このような特徴は、いずれか所与のまたは特定の適用例に望まれ有利であろうように、他の実装形態の他の１つまたは複数の特徴と組み合わせてもよい。さらに、詳細な説明または特許請求の範囲で用語「ｉｎｃｌｕｄｅ（含む、備える）」および「ｉｎｃｌｕｄｉｎｇ」ならびにその異形が使用される限り、これらの用語は、用語「ｃｏｍｐｒｉｓｉｎｇ（含む、備える）」と同様にして包含的とする。

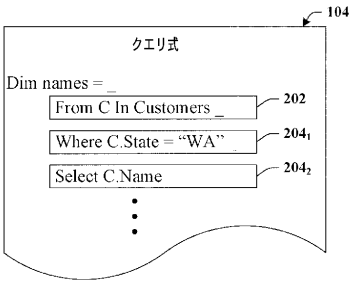
【図１】



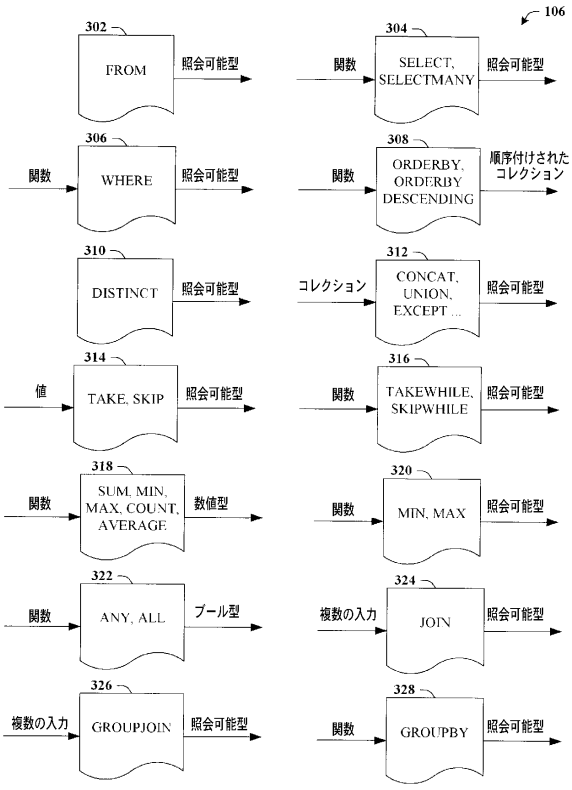
【図２Ｂ】



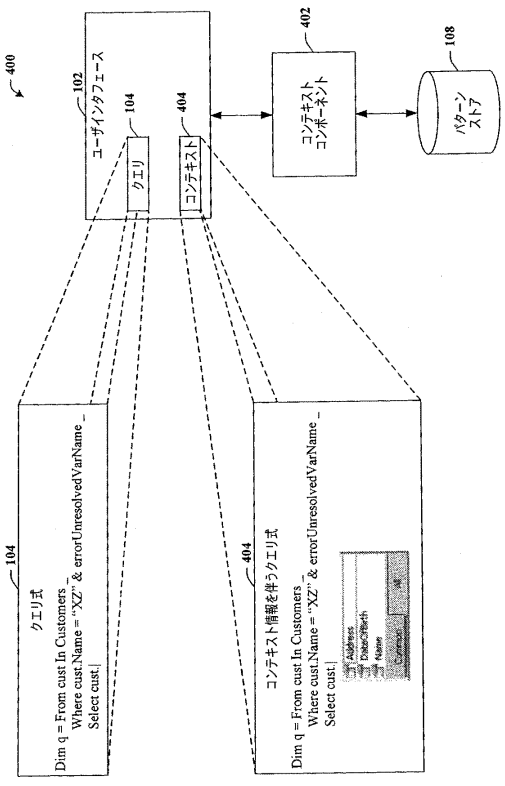
【図２Ａ】



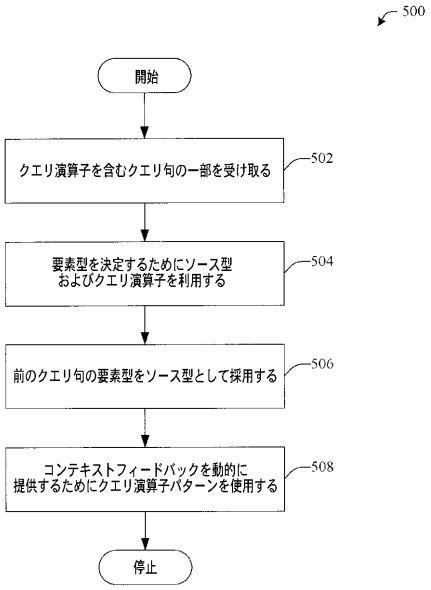
【図３】



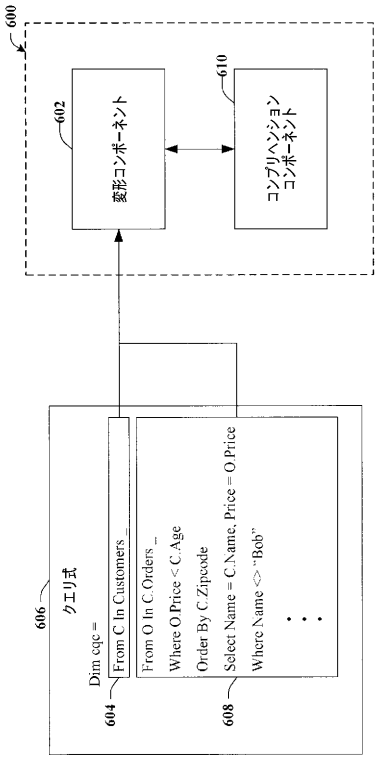
【図４】



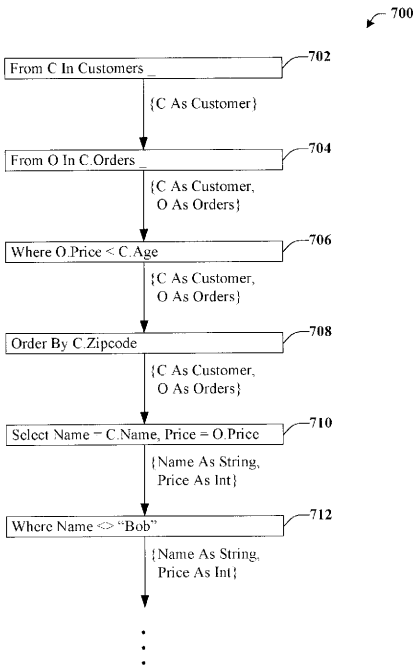
【図５】


500
開始
クエリ演算子を含むクエリ句の一部を受け取る
502
要素型を決定するためにソース型
およびクエリ演算子を利用する
504
前のクエリ句の要素型をソース型として採用する
506
コンテキストフィードバックを動的に
提供するためにクエリ演算子パターンを使用する
508
停止

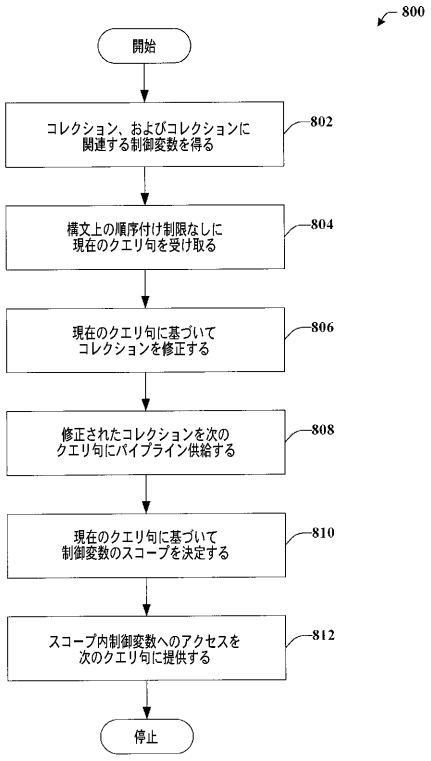
【図６】


600
変形コンポーネント
602
610
コンプリヘンション
コンポーネント
606
クエリ式
Dim cqc =
From C In Customers _
From O In C.Orders _
Where O.Price < C.Age
Order By C.Zipcode
Select Name = C.Name, Price = O.Price
Where Name <> "Bob"
. . .
604
608

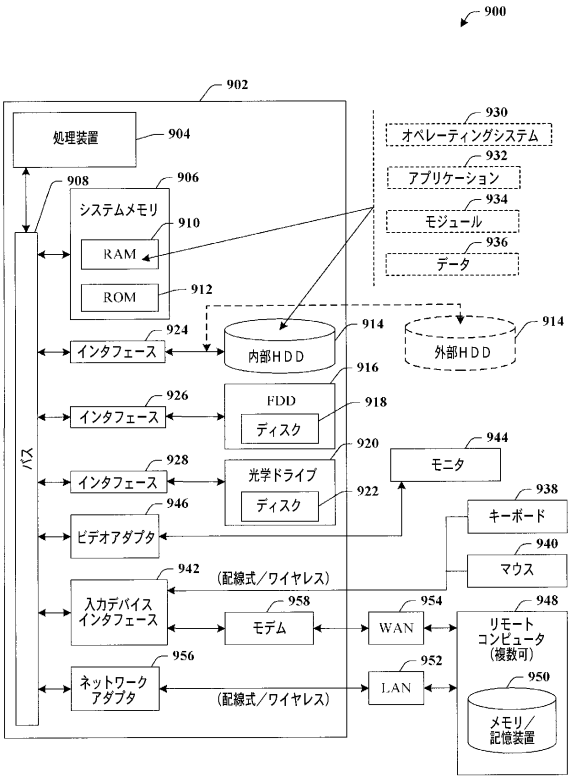
【図７】


700
From C In Customers _
702
{C As Customer}
From O In C.Orders _
704
{C As Customer,
O As Orders}
Where O.Price < C.Age
706
{C As Customer,
O As Orders}
Order By C.Zipcode
708
{C As Customer,
O As Orders}
Select Name = C.Name, Price = O.Price
710
{Name As String,
Price As Int}
Where Name <> "Bob"
712
{Name As String,
Price As Int}

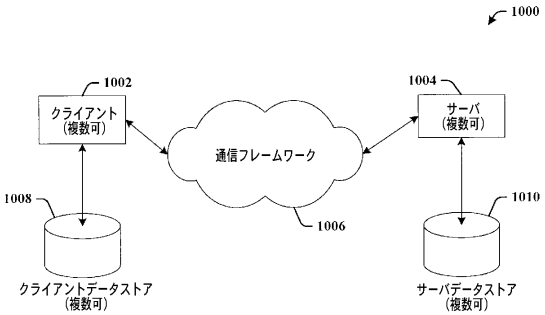
【図８】


800
開始
コレクション、およびコレクションに
関連する制御変数を得る
802
構文上の順序付け制限なしに
現在のクエリ句を受け取る
804
現在のクエリ句に基づいて
コレクションを修正する
806
修正されたコレクションを次の
クエリ句にパイプライン供給する
808
現在のクエリ句に基づいて
制御変数のスコープを決定する
810
スコープ内制御変数へのアクセスを
次のクエリ句に提供する
812
停止

【図９】


900
902
処理装置
904
908
906
システムメモリ
910
RAM
ROM
912
930
オペレーティングシステム
932
アプリケーション
934
モジュール
936
データ
924
インタフェース
914
内部ＨＤＤ
914
外部ＨＤＤ
916
FDD
926
インタフェース
918
ディスク
920
光学ドライブ
928
インタフェース
922
ディスク
944
モニタ
938
キーボード
946
ビデオアダプタ
940
マウス
942
（配線式／ワイヤレス）
入力デバイス
インタフェース
958
モデム
954
WAN
948
リモート
コンピュータ
（複数可）
956
ネットワーク
アダプタ
952
LAN
（配線式／ワイヤレス）
950
メモリ／
記憶装置
バス

【図１０】


1000
1002
クライアント
（複数可）
1004
サーバ
（複数可）
通信フレームワーク
1006
1008
1010
クライアントデータストア
（複数可）
サーバデータストア
（複数可）

フロントページの続き

(72)発明者 ヘンリクス ヨハネス マリア マイヤー
アメリカ合衆国 ９８０５２ ワシントン州 レッドモンド ワン マイクロソフト ウェイ マイクロソフト コーポレーション インターナショナル パテンツ内
(72)発明者 アマンダ ケー. シルバー
アメリカ合衆国 ９８０５２ ワシントン州 レッドモンド ワン マイクロソフト ウェイ マイクロソフト コーポレーション インターナショナル パテンツ内
(72)発明者 ポール エー. ヴィック
アメリカ合衆国 ９８０５２ ワシントン州 レッドモンド ワン マイクロソフト ウェイ マイクロソフト コーポレーション インターナショナル パテンツ内
(72)発明者 エブゲニ ザボクリトスキ
アメリカ合衆国 ９８０５２ ワシントン州 レッドモンド ワン マイクロソフト ウェイ マイクロソフト コーポレーション インターナショナル パテンツ内
(72)発明者 アレクセイ ブイ. ツインガウス
アメリカ合衆国 ９８０５２ ワシントン州 レッドモンド ワン マイクロソフト ウェイ マイクロソフト コーポレーション インターナショナル パテンツ内

審査官 打出 義尚

(56)参考文献 関谷寛幸, Ｒａｐｉｄ ＳＱＬ ５．７ Ｃｌｏｓｓ－Ｐｌａｔｆｏｒｍ 日揮情報ソフトウェア株式会社, Ｖｉｓｕａｌ Ｂａｓｉｃ ｍａｇａｚｉｎｅ, 日本, 株式会社翔泳社, ２００２年 ７月 １日, 第８巻, 第１３号, ｐｐ．２３５－２４１
岸上ユウコ, 使うぞ！アクセス, ＹＯＭＩＵＲＩ ＰＣ, 日本, 読売新聞社, ２００２年 ３月 １日, 第７巻, 第４号, ｐｐ．４５－５２
Mads Torgersen, Language integrated query: unified querying across data sources and programming languages, Companion to the 21st ACM SIGPLAN symposium on Object-oriented programming systems, languages, and applications, 米国, ACM, ２００６年, pages 736-737, ＵＲＬ, http://doi.acm.org/10.1145/1176617.1176700

(58)調査した分野(Int.Cl., ＤＢ名)
Ｇ０６Ｆ １７／３０