(19)日本国特許庁(JP)　(12)特　許　公　報(B2)　(11)特許番号
特許第6720641号
(P6720641)
(45)発行日　令和2年7月8日(2020.7.8)　(24)登録日　令和2年6月22日(2020.6.22)

(51)Int.Cl.　F I
G06F 16/215 (2019.01)　G06F 16/215

請求項の数 17　外国語出願　(全 26 頁)

(21)出願番号　特願2016-68461(P2016-68461)
(22)出願日　平成28年3月30日(2016.3.30)
(65)公開番号　特開2016-212839(P2016-212839A)
(43)公開日　平成28年12月15日(2016.12.15)
審査請求日　平成31年1月15日(2019.1.15)
(31)優先権主張番号　1507301.8
(32)優先日　平成27年4月29日(2015.4.29)
(33)優先権主張国・地域又は機関
英国(GB)
(31)優先権主張番号　16153241.1
(32)優先日　平成28年1月28日(2016.1.28)
(33)優先権主張国・地域又は機関
欧州特許庁(EP)

(73)特許権者　000005223
富士通株式会社
神奈川県川崎市中原区上小田中４丁目１番１号
(74)代理人　100107766
弁理士　伊東　忠重
(74)代理人　100070150
弁理士　伊東　忠彦
(74)代理人　100192636
弁理士　加藤　隆夫
(72)発明者　メンディ・ロジャー
イギリス国，ジーユー１　２ワイビー，サリー，ギルフォード，チャーロック・ウェイ　３７番

最終頁に続く

(54)【発明の名称】多言語データティアのデータ制約

(57)【特許請求の範囲】
【請求項１】
複数の異種データストアを有する多言語データティアにおいてデータ制約を施行するコンピュータの作動方法であって、
前記複数の異種データストアのうちの１つに格納されたデータをシリアライズすることにより、データストアの種類に依存しない共通形式のレコードを生成するステップと、
前記レコードを抽出するステップと、
前記レコードに対応するレコードシェイプを見付けるステップであって、各々のレコードシェイプは、オントロジで表され、レコードの構造を決定する、ステップと、
前記対応するレコードシェイプの中で定められる複数の基準の各々に対して前記レコードをチェックすることにより、前記レコードにデータ制約を適用するステップと、
前記チェックの結果に応じて、前記レコードの有効又は無効を制御するステップと、
を有する方法。
【請求項２】
前記制御するステップにおいて前記レコードが有効とされた場合、前記データストアの中に前記レコードを作成する、前記データストアから前記レコードを読み出す、前記レコードを用いて前記データストアを更新する、及び前記データストアから前記レコードを削除する、のうちの１又は複数を含む操作を前記レコードに対して実行するステップ、を更に有する請求項１に記載の方法。
【請求項３】

指定データを含む要求を受信するステップと、前記指定データに基づき検証されるべきレコードを抽出するステップと、を更に有する請求項１又は２に記載の方法。
【請求項４】
前記レコードは、前記要求に含まれる、又は
前記レコードは、前記データストアのうちの１つに含まれ前記要求の中で指定される、
請求項３に記載の方法。
【請求項５】
各々のデータストアを、データソース識別子を有する抽象データソースとして表すステップであって、前記要求は、前記指定データに対応するデータソース識別子を特定できる情報を含む、ステップ、を更に有する請求項３又は４に記載の方法。
【請求項６】
各々のレコードは、コンマで区切られた値のｎ要素タプルであり、ｎは零より大きい非負整数である、請求項１乃至５のいずれか一項に記載の方法。
【請求項７】
前記データストアは、
（ｉ）トリプルストアであって、該トリプルストアの中のデータのレコードの中で、各々のコンマで区切られた値はＲＤＦ述語の目的語に対応する、トリプルストア、
（ｉｉ）ＲＤＢＭＳであって、該ＲＤＢＭＳの中のデータのレコードの中で、各々のコンマで区切られた値はテーブルに格納された属性を表す、ＲＤＢＭＳ、
（ｉｉｉ）文書指向型データベース、
（ｉｖ）列指向型テーブルに基づくデータベース、
（ｖ）キー値ペアに基づくデータベース、
のうちのいずれかを有する、請求項３に記載の方法。
【請求項８】
新しい型のデータストアが前記多言語データティアに追加されるとき、前記データストアに格納されたデータの構造を定める新しいレコードシェイプを定めるために、前記オントロジを用いるステップ、を更に有する請求項１乃至７のいずれか一項に記載の方法。
【請求項９】
各々のレコードシェイプは、レコードのデータ型、濃度、及びフィールドフォーマットに関する情報を含む、請求項１乃至８のいずれか一項に記載の方法。
【請求項１０】
各々のレコードシェイプは、ＲＤＦ（Resource Description Framework）ｎ要素タプルのセットであり、ｎは零より大きい非負整数であり、前記オントロジはＲＤＦＳ／ＯＷＬに基づく、請求項１乃至９のいずれか一項に記載の方法。
【請求項１１】
複数の異種データストアを有する多言語データティアにおいてデータ制約を施行するデータ制約エンジンであって、
前記複数の異種データストアのうちの１つに格納されたデータをシリアライズすることにより、データストアの種類に依存しない共通形式のレコードを生成する手段と、
前記レコードを抽出する手段と、
前記の抽出されたレコードに基づき、シェイプカタログからのレコードシェイプにアクセスする手段であって、各々のレコードシェイプは、オントロジで表され、レコードの構造を決定する、手段と、
対応するレコードシェイプの中で定められる複数の基準に対して前記レコードをチェックし、前記チェックの結果に応じて前記レコードの有効又は無効を制御する複数の検証器と、
を有するデータ制約エンジン。
【請求項１２】
入ってくる要求を受信するインタフェースであって、各々の要求はデータを指定する、インタフェースを更に有し、前記抽出する手段は、前記要求の中で指定されたデータに基

づき、前記レコードを抽出するよう構成される、請求項１１に記載のデータ制約エンジン。

【請求項１３】

前記検証器により前記レコードが有効とされた場合、前記データストアの中に前記レコードを作成する、前記データストアから前記レコードを読み出す、前記レコードを用いて前記データストアを更新する、及び前記データストアから前記レコードを削除する、のうちの１又は複数を含む操作を前記レコードに対して実行するレコードディスパッチャ、を更に有する請求項１１又は１２に記載のデータ制約エンジン。

【請求項１４】

前記複数の検証器は、それぞれ

スロット数、

濃度、

データ型、

ＨＴＭＬ、ＸＭＬ、及びＪＳＯＮのうちの１又は複数のようなフォーマット、

である個々の検証器を有する、請求項１１、１２、又は１３に記載のデータ制約エンジン。

【請求項１５】

各々のレコードシェイプは、ＲＤＦＳ／ＯＷＬ語彙で表されるＲＤＦ（Resource Description Framework）トリプルである、請求項１１乃至１４のいずれか一項に記載のデータ制約エンジン。

【請求項１６】

請求項１１乃至１５のうちのいずれか一項に記載のデータ制約エンジンとして機能するよう構成されるコンピューティング装置。

【請求項１７】

コンピューティング装置により実行されると、前記コンピューティング装置に請求項１６に記載のコンピューティング装置として動作させる、コンピュータプログラム。

【発明の詳細な説明】

【技術分野】

【０００１】

本発明は、データ記憶の分野に関する。特に、本発明の実施形態は、複数の異種データベースを有するデータティア（所謂、「多言語データティア」）におけるデータ制約をモデル化し及び施行するメカニズムに関する。

【背景技術】

【０００２】

「データティア」の概念は、ソフトウェア工学において広く用いられる。マルチティアアーキテクチャは、プレゼンテーション、アプリケーション処理、及びデータ管理機能が物理的に分離しているクライアント－サーバアーキテクチャである。一般的にｎティアアーキテクチャが考えられるが、最もよく用いられるアーキテクチャは３ティアアーキテクチャである。３ティアアーキテクチャは、標準的に、プレゼンテーションティア、ロジック又は処理ティア、及びデータ記憶ティアで構成される。

【０００３】

図１は、このような３ティアアーキテクチャを簡略化した形式で示す。（図１に示すように）個々のティアが異なるハードウェアに実装されると考えることは有用である場合があるが、これは必須ではない。

【０００４】

本例では、ティア１は、図１のクライアントにより示されるデスクトップＰＣ若しくはワークステーション上で動き及び標準的なグラフィカルユーザインタフェースを使用し得る最上位のアプリケーションのユーザインタフェースを含むクライアントティアである。このティアは、中間のティアであるティア２（ロジックティアとも呼ばれる）に（クエリのような）データを供給する。ティア２は、アプリケーションの機能を提供するために、

（図１のサーバにより示される）ワークステーション若しくはアプリケーションサーバ上で動き１又は複数の別個のモジュールで構成され得る機能プロセスロジックを有する。ティア３は、上位ティアからクエリを受け取り、図１のデータベースにより概略的に示されるコンピュータデータ記憶ロジックを含むデータベースサーバ又はメインフレームに実装され得るデータティアである。このティアは、アプリケーションにより参照されるデータセットと、データを管理し及びデータへのアクセスを提供するデータベース管理システムと、を有する。ＡＰＩ（Application Program Interface）は、個々のティアの間に存在しても良い。各々のＡＰＩは、異なるティアの中のソフトウェアが互いに相互作用する仕様である。したがって、ＡＰＩラッパーがティア３データベースに理解可能なクエリのフォーマットに要求を変換する場合、ティア１から生じる要求又はデータ操作が与えられる。

【０００５】

実際には、マルチティアアーキテクチャは、各々のレベルにおける複数のシステム又はノードの使用を含み得る。このように、アーキテクチャの各々のティアは、分散形式で提供されても良い（実施には、各々のティアの要素は、例えばインターネット上のどこに位置しても良い）。また、ノードは同一のハードウェアシステムとして図示されるが、より一般的には、各々のティアは、ハードウェア及びソフトウェアレベルの両方において異種であっても良い。このようなマルチシステム実装は、個々のノード又はシステムが異種標準若しくは技術を利用する所謂「多言語」ティアの可能性を生じる。例えば、クライアントティアは、ウェブに基づくインタフェースを提供するためにＨＴＭＬ、ＣＳＳ及びＪａｖａＳｃｒｉｐｔ(登録商標)を利用しても良く、ｉＯＳ又はＡｎｄｒｏｉｄのようなモバイルプラットフォームはモバイルインタフェースを利用しても良い。中間のティアは、Ｊａｖａ、．ＮＥＴ、又は多くの他の利用可能なプラットフォームのうちの１つを利用しても良い。

【０００６】

本発明に特に関連するものとして、分散型データベースを形成するために種々のデータベース技術を結合する多言語データティアの可能性がある。データベース技術の２つの主なクラスは、次の通りである。

【０００７】

（ｉ）ＳＱＬ（Structured Query Language）を用いるＲＤＢＭＳ（traditional relational database）アプローチ。これは、関係型データベースに格納されたデータを記憶し、操作し、及び読み出すためのコンピュータ言語である。ＳＱＬに基づく言語の例は、ＭｙＳＱＬ、Ｏｒａｃｌｅ、又はＭＳ　ＳＱＬを含む。

【０００８】

（ｉｉ）ＮｏＳＱＬ（Not only SQL））データベース。これは、関係型データベースで用いられる表形式の関係以外の手段により構造化されるデータの記憶及び読み出しのためのメカニズムを提供する。ＮｏＳＱＬデータベースの例は、ＭｏｎｇｏＤＢ及びＣａｓｓａｎｄｒａを含む。

【０００９】

余談として、留意すべきことに、関係型データベースは、データを格納する前に定められる必要のあるテーブルを形成するために、行及び列にデータを格納する。テーブルの定義及びこれらのテーブルに含まれるデータ間の関係は、スキーマと称される。関係型データベースは、固定スキーマを用いる。

【００１０】

グラフデータベースは、データをノード及びアークの形式で格納することにより、関係型データベースの重要な拡張を表す。ここで、ノードはエンティティ又はインスタンスを表し、アークは任意の２個のノード間の特定種類の関係を表す。幾つかの種類のグラフ表現がある。グラフデータは、多次元アレイとして又は他のシンボルにリンク付けされたシンボルとしてメモリに格納されても良い。別の形式のグラフ表現は、各々指定された種類のオブジェクトの有限シーケンス又は順序付きリストである「タプル」の使用である。ｎ

個のオブジェクトを含むタプルは、「ｎタプル」として知られる。ここで、ｎは零より大きい任意の非負整数である。長さ２のタプル（２タプル）は、通常、ペアと呼ばれる。３タプルはトリプルと呼ばれ、４タプルはクワドラプルと呼ばれ、以降同様である。

【００１１】

データベース技術の選択は、記憶エンジン、データモデル、及びクエリ言語を選択する必要がある。関係型データベースは、通常、クエリ言語としてＳＱＬにより、関係型データモデルをサポートする。他方で、ＮｏＳＱＬデータベースは、それぞれ、専門クエリ言語と一緒に、文書、グラフ、キー値、又は列指向モデルのような単一データモデルをサポートする。例えば、ＭｏｎｇｏＤＢは文書データモデルを用い、Ｃａｓｓａｎｄｒａは列指向モデルを用いる。キー値は、アプリケーション開発者がスキーマレスデータを格納できるようにする。このデータは、通常、キーを表すストリングと、「キー値」関係の中の値と考えられる実際のデータと、を有する。

【００１２】

したがって、多言語データティアは、異なるデータモデル（例えば、関係型、文書に基づく、グラフに基づく、等）を採用する自律データストアのセットである。

【００１３】

ここで、後にＲＤＦ、オントロジ、ＲＤＦＳ、ＯＷＬ、ＯＳＬＣ、及びＱＵＤＴを参照するので、これらの用語の幾らかの簡単な説明が与えられる。

【００１４】

ＲＤＦ（Resource Description Framework）は、ウェブリソースの中に実装される情報の概念的記述又はモデル化のための一般的方法として用いられるＷ３Ｃ（World Wide Web Consortium）仕様のファミリである。ＲＤＦは、主語－述語－目的語表現の形式で、リソース（特にウェブリソース）に関するステートメントを作成する概念に基づく。これらの表現は、上述のトリプルの例である。主語はリソースを示し、述語はリソースの特性又は特長を示し並びに主語と目的語との間の関係を表す。

【００１５】

ＲＤＦは、データを表現するための柔軟なモデルを提供する、ラベル付きノード及び有向ラベル付きエッジを有するグラフに基づくデータモデルである。ＲＤＦの基本ユニットは、ステートメントであり、グラフの中のエッジに対応する。ＲＤＦステートメントは、３つのコンポーネント、つまり主語、述語、及び目的語を有する。主語は、エッジの起点であり、リソースでなければならない。ＲＤＦでは、リソースは、ＵＲＩ（Uniform Resource Identifier）によりユニークに識別可能ないかなるものでも良い。標準的に、この識別子は、ＵＲＩの特定の例である、インターネット上のＵＲＬ（Uniform Resource Locator）であるしかしながら、ＵＲＩは、ＵＲＬよりも一般的である（ＵＲＩがインターネット上の文書の場所を特定するために使用できるという要件はない）。

【００１６】

ステートメントの目的語は、エッジの宛先である。主語と同様に、これはＵＲＩにより識別されるリソースであるが、代替でストリング又は数のようなリテラル値であり得る。ステートメントの述語（これもＵＲＩにより識別される）は、主語と目的語との間をどんな関係が保つかを決定する。言い換えると、述語は、目的語へのリンクを提供することにより主語に関する何かを主張する一種のプロパティ又は関係である。

【００１７】

図２は、３つのステートメントを有する例示的なＲＤＦグラフを示す。１つのステートメントは、主語http://example.org/~jdoe#jane、述語p:knows、及び目的語Jane Doeを有する。言い換えると、このステートメントは、「JaneがJohnを知っている（Jane knows John）」を表す。述語p:nameを有するステートメントは、目的語としてリテラル値（つまり「Jane Doe」）を有するステートメントの一例である。このステートメントは、Janeの名前が「Jane Doe」であることを示す。ここで、p:knows及びp:nameは修飾名と呼ばれる。第３のステートメントは、Janeが人であることを言明する。

【００１８】

上述のトリプルは、グラフデータを符号化するために用いることができる。各々のトリプルは、主語－述語－目的語表現を表す。したがって、ＲＤＦグラフは、ＲＤＦトリプルのセットとして表すことができる。また、ＲＤＦトリプルは、ネストされた（nested）データ構造のシリーズとして書き出すことができる（シリアライズ、serialised）。ＲＤＦトリプルをシリアライズする種々の方法がある。例えば、ＸＭＬ（Extensible Markup Language）又はＪＳＯＮ（JavaScript Object Notation）を用いて、種々のファイルフォーマット（シリアライゼーション（serialisation）フォーマット）を生じる。

【００１９】

一例として、以下のＸＭＬコードは、図２のＲＤＦグラフのシリアライゼーションである。

【数１】

```
<rdf:RDF
 xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
 xmlns:p="http://example.org/pers-schema#">
 <rdf:Description rdf:about="http://example.org/~jdoe#jane">
 <p:knows rdf:resource="http://example.org/~jsmith#john" />
 <p:name>Jane Doe</p:name>
 <rdf:type
 rdf:resource=”http://example.org/pers-schema#Person”/>
 </rdf:Description>
</rdf:RDF>
```

【００２０】

リソースを記述するＲＤＦメカニズムは、キー概念が「リンク付きデータ」である、Ｗ３Ｃの「Semantic Web」の取り組みにおける主要なコンポーネントである。リンク付きデータは、基本的に、インターネットリソースを、機械及び人間による使用のために設計されたグローバルデータベースに組織化することを目的とする。ここで、リンクは、文書間ではなくオブジェクト（又はオブジェクトの記述）間に設けられる。リンク付きデータのためのＷ３ＣのSemantic Web技術スタックの鍵となる部分は、上述のＲＤＦ及びＵＲＩに加えて、ＲＤＦＳ及びＯＷＬを含む。

【００２１】

ＲＤＦＳ（RDF Schema）は、ＲＤＦの意味的拡張であり、ＲＤＦで記述される。これは、関連するリソースのグループ及びこれらのリソース間の関係を記述するためのメカニズムを提供する。これらのリソースは、プロパティのドメイン及び範囲のような、他のリソースの特徴を決定するために用いられる。したがって、ＲＤＦＳは、ＲＤＦリソースを構造化する目的で、オントロジの記述、又はＲＤＦ語彙と呼ばれる、のための基本要素を提供する（因みに、用語「オントロジ」と「語彙」との間の区別が引き出されても良いが、本願明細書では、特に文脈上必要でない限り、これらの用語は同義的に用いられる）。ＲＤＦを用いるリソースに関する記述は、トリプルストアに保存され、ＲＤＦクエリ言語ＳＰＡＲＱＬを用いて読み出され操作され得る。ＲＤＦＳ及びＳＰＡＲＱＬの両者は、Ｗ３ＣのSemantic Web技術スタックの部分である。

【００２２】

ＲＤＦスキーマクラス及びプロパティシステムは、Ｊａｖａのようなオブジェクト指向プログラミング言語のタイプシステムと似ている。しかしながら、ＲＤＦスキーマは、クラスのインスタンスが有するプロパティの観点で該クラスを定める代わりに、ＲＤＦスキーマはそれらが適用するリソースのクラスの観点でプロパティを記述する点で、このようなシステムと異なる。ＲＤＦスキーマアプローチは、これらのクラスの元の記述を再定義

する必要がなく、追加プロパティを他者が続いて定義することが簡単であるという意味で、「拡張性がある」。

【００２３】

一方で、ＯＷＬ（Web Ontology Language）のようなより豊かな語彙／オントロジ言語は、データの構造及び意味に関する追加情報をキャプチャすることを可能にする。

【００２４】

ＯＳＬＣ（Open Service for Lifecycle Collaboration）は、関連リソース間のリンクによりデータレベルにおける統合を可能にするためにＲＤＦの上に構築される別のオントロジである。ＯＷＬのように、ＯＳＬＣは、ＲＤＦ上に構築されＲＤＦを拡張する。つまり、ＯＳＬＣリソースは、ＲＤＦプロパティの観点で定められる。

【００２５】

ＱＵＤＴ（Quantity, Unit, Dimension and Type）オントロジは、物理量をモデル化するために用いられる制限及び基本クラスプロパティ、種々の測定システムにおける測定単位及びそれらの次元、を定める。ＯＷＬを基礎として、ＱＵＤＴオントロジの目標は、測定可能な量、異なる種類の量を測定するための単位、異なる測定ユニットの量の数値、並びに、ソフトウェアにいてこれらのオブジェクトを格納し操作するために用いられるデータ構造及びデータ型、の統一モデルを提供することである。

【００２６】

データ検証は、ソフトウェア工学におけるもう１つの重要な概念である。例えば、図１のクライアントティアを参照すると、データは、標準的に、複数のデータ入力フィールドを有するデータ入力フォームを満たすユーザにより入力される。入力されたデータを下位ティアに渡す前に、各々のデータ入力フィールドは、所定の基準について検証される。この検証処理は、データが適正なフォーマットで入力されたこと及び期待値の妥当な範囲であることを保証する。データベースを用いる全てのアプリケーションの間で検証の一貫性を保証するために、検証基準は、データ制約セットにより定められても良い。制約定義言語は、データ制約を定めることができるように定められても良い。しかし、これらは、従来、特定のデータベース技術に固有であり及び／又は独自仕様である（例えば、Oracle CorpによるＣＤＬ）。

【００２７】

留意すべきことに、データ検証は、ユーザにより入力されるデータの上述の例に限られない。より一般的には、データ制約は、関係型データベースに基づき構築されるマルチティアアーキテクチャにおいて広く採用されるメカニズムである。それらは、宣言型アプローチによるデータ検証を可能にし、したがってプログラミング労力を削減する。データ制約は、以下のような異なるレベルにおいてプログラミング言語に依存する検証コードから開発者を解放する。

・データレベルに適用されるとき（例えば、データベース管理システムの内部）、それらはデータベース固有検証コードを回避する。

・ＡＰＩ（Application Program Interface）において使用されるとき、それらは、クライアント入力の一貫性チェックを提供し、したがって、ＡＰＩに依存する入力検証コードを置き換える。

【００２８】

例えば、ＳＱＬ ＣＨＥＣＫ制約は、データベーステーブルの中の各々の表により満たされなければならない要件を指定するＳＱＬにおけるインテグリティ制約の種類である。この制約は、述語でなければならず、テーブルの中の単一又は複数の列を参照できる。一方で、データ制約に関連するＷ３Ｃの中の多数の活動がある。これらは、ＲＤＦグラフに対する制約を表現し、プログラマにＲＤＦ文書を検証させ、インタフェースについて期待されるグラフパターンを通信し、ユーザインタフェースフォーム及びインタフェースコードを生成し、及びＳＰＡＲＱＬクエリにコンパイルする言語であるShape Expressionsを含む。同様に、ＯＳＬＣ Ｒｅｓｏｕｒｃｅ Ｓｈａｐｅｓは、許可された値を有するプロパティのリストの仕様、及び該リストのＲＤＦＳクラスとの関連付けを許可する。

【００２９】
他方で、真にスキーマレスなデータベースは、データ型を参照しないでデータを格納でき、データ制約を設けることを困難にする。
【００３０】
先の議論の幾つかを纏めると、Ｗ３Ｃは、ＲＤＦにおける語彙及びオントロジを記述するためにＲＤＦＳ及びＯＷＬを含む標準を提供する。これらの標準は、主に、異なる語彙のリコンシレーション（reconciliation、融和）をサポートして、種々のデータセットと与えられた情報から新しい情報を推測する能力を有する推論エンジンとの統合を実現するために設計される。ＯＳＬＣ　Ｒｅｓｏｕｒｃｅ　Ｓｈａｐｅｓは、ＲＤＦグラフに対する制約を指定し及び検証するために用いることができるＲＤＦ語彙を提供する。Ｒｅｓｏｕｒｃｅ　Ｓｈａｐｅｓは、それらが扱うリソースの種類をクライアントとプログラムで通信し、それらがクライアントから受信するコンテンツを検証する方法をサーバに提供する。
【００３１】
しかしながら、上述のように、マルチティアシステムは、多言語データティアに味方して、純粋な関係型バックエンドから革新的に離れている。現在のデータベース固有制約施行メカニズムは、複数のデータモデルが共存する又はスキーマレスデータベースを有し得るデータティアに適合しない。
【００３２】
例えば、顧客の購入品を追跡し多数の製品製造者のためにレポートを生成する、顧客のネットワークを分析するシステムを検討する。このシステムは、マルチティアアーキテクチャにより実装され、製造者プロファイルを関係型データベースに格納する多言語データティア、及びトリプルストアの顧客のソーシャルネットワークを含む。さらに、このシステムは、種々の製造者の製品カタログを統合すべきである。このようなデータは、製造者により所有されるリモートデータベースに格納され、データベースの事前の知識は与えられない。
【００３３】
このようなシナリオにおいてデータ制約を施行することは、複数の制約定義言語を熟知している必要がる。つまり、データレベルにおいて関係型データベースのテーブルは、おそらくＳＱＬ　ＣＨＥＣＫ制約を含む属性データ型を指定しなければならない。ＯＳＬＣ　Ｒｅｓｏｕｒｃｅ　Ｓｈａｐｅｓ又はＷ３Ｃ　Ｓｈａｐｅ　Ｅｘｐｒｅｓｓｉｏｎｓに関する知識は、トリプルストアデータを制約するために必要である。リモートデータストアは、第三機関により管理され、多言語システム設計者は、データベースレベルの制約を追加するアクセス権を有しない。さらに、このようなリモートデータベースは、スキーマレスであっても良く、したがって検証メカニズムを欠いている。したがって、未知の第三機関データストアをサポートすることは、アプリケーションレベルの検証コードを必要とする。これは、追加の開発労力を意味する。さらに、このような検証コードは、リモートストアが新しいデータモデル及びＡＰＩに基づくかも知れないので、拡張をサポートしなければならない。
【００３４】
したがって、多言語データティアにおける制約の定義及び施行のためのストア非依存（agnostic）メカニズムが必要である。
【発明の概要】
【００３５】
本発明の第１の態様によると、複数の異種データストアを有する多言語データティアにいてデータ制約を施行する方法であって、前記データが格納される方法及び場所に係わらず、前記データストアの中のデータを、前記データをシリアライズするレコードと見なすステップと、検証されるべきレコードを抽出するステップと、前記レコードに対応するレコードシェイプを見付け、レコードの構造を決定するステップであって、各々のレコードシェイプは拡張可能語彙で表される、ステップと、前記対応するレコードシェイプの中で

定められる複数の基準の各々に対して前記レコードをチェックすることにより、前記レコードにデータ制約を適用するステップと、全部の前記基準が満たされる場合、前記レコードを有効であると決定するステップと、を有する方法が提供される。

【００３６】

ここで、異種データストアは、異なる技術、データモデル、等を用いる異なる種類のデータベースであっても良い。

【００３７】

データをレコードと見なすステップは、データベース固有形式で格納されたデータを、「レコード」と呼ばれる共通形式で表すステップを含み得る。したがって、データが格納される（格納されるべき）方法及び場所の詳細は、もはや重要ではない。

【００３８】

検証されるべきレコードを抽出するステップは、データストアから既存のレコードを出力するステップ、又はデータストアに又はそれから特定のデータを生成し、読み出し、更新し、又は削除するユーザ要求からレコードを引き出すステップ、を有し得る。要求からレコードを引き出すステップは、指定されているデータを識別する要求を解析するステップと、レコードの形式で結果を提供するステップと、を有し得る。

【００３９】

レコードシェイプを見付けるステップは、引き出されたレコードに適合するレコードシェイプを見付けるために、定められたレコードシェイプのデポジトリを参照するステップを有し得る。レコードシェイプに対してレコードを検証するステップは、レコードが完全であり期待される形式に従うこと調べるために、後述する多数の基準のいずれかに従ってレコードの形式をチェックすることを意味する。

【００４０】

したがって、統一データモデルは、「レコード」の概念に基づき提供される。各々のレコードは、定められた構造又はそれに関連する「レコードシェイプ」に従ってデータを表す。レコードシェイプは、ＲＤＦＳ／ＯＷＬのような拡張可能語彙で表され、多言語データティアと独立のレポジトリに格納されて、起こり得る予期しないデータモデル、データ型、等を有する追加データストアを扱うために新しいレコードシェイプを定めることを可能にする。データ制約は、レコードが関連するレコードシェイプにより定められた構造に従うことを保証することによりレコードを検証するために、何らかの方法で抽出された（例えば、ＰＯＳＴ、ＧＥＴ、ＰＵＴ、又はＤＥＬＥＴＥのような多言語データティアの中の指定データを操作するために入ってくる要求から抽出された）レコードに適用される。

【００４１】

標準的に、レコードの検証結果は、多言語データティアに関するデータ操作を許可するためである。したがって、方法は、望ましくは、前記レコードが有効であると決定される場合、前記データストアの中に前記レコードを作成する、データストアから前記レコードを読み出す、前記レコードを用いてデータストアを更新する、及びデータストアからレコードを削除する、のうちの１又は複数を含む操作を前記レコードに対して実行するステップ、を更に有する。

【００４２】

方法は、指定データを含む要求を受信するステップと、前記指定データに基づき検証されるべきレコードを抽出するステップと、も有しても良い。

【００４３】

ここで１つの可能性として、例えば要求が新しいレコードをデータストアの中に生成することである場合、上述のレコードは要求に含まれる。

【００４４】

代替で、レコードは、データストアのうちの１つに含まれ、要求の中で指定されても良い。これは、例えば、リモートクライアントにより要求されるリード操作の場合に適用される。

【００４５】

更なる可能性として、レコードは、特定のクライアント要求を有しないで、例えばデータベースのチェック又は発見の処理の中で識別される。

【００４６】

望ましくは、方法は、各々のデータストア（つまり、多数の異なる種類のうちの１つであり得る各々のデータベース）を、データソース識別子を有する抽象データソースとして表すステップを更に有する。要求は、指定されたデータに対応するデータソース識別子を特定できる情報を含む。このように、検証された要求は、適切なデータストアへ容易にルーティングされ得る。

【００４７】

望ましくは、各々のレコードは、コンマで区切られた値のｎ要素タプルである。本発明は、任意の種類のデータストアに適用できる。例えば、データストアのうちの１又は複数は、トリプルストアであっても良い。この場合、トリプルストアの中のデータのレコードの中で、各々のコンマで区切られた値は、ＲＤＦ述語の目的語に対応する。

【００４８】

代替で、データストアは、ＲＤＢＭＳを有しても良く、ＲＤＢＭＳの中のデータのレコードの中で、各々のコンマで区切られた値は、テーブルに格納された属性に対応する。

【００４９】

本発明が適用され得る他の可能なデータストアの種類は（網羅的ではないが）、ＭｏｎｇｏＤＢのような文書指向型データベース、Ｃａｓｓａｎｄｒａのような列指向型テーブルに基づくデータベース、及びキー値ペアに基づくデータベースを含む。ハイブリッドデータベースも存在しても良い。例えば、Ｃａｓｓａｎｄｒａは、ハイブリッド列指向及びキー値ペアデータベースと考えることができる。

【００５０】

未だ開発されていない種類を含む新しい種類のデータストアも、本発明により対応できる。したがって、方法は、望ましくは、新しい型のデータストアが前記多言語データティアに追加されるとき、前記データストアに格納されたデータの構造を定める新しいレコードシェイプを定めるために、前記拡張可能語彙を用いるステップ、を更に有する。

【００５１】

各々のレコードシェイプは、望ましくは、データ型、濃度、及びレコードのフィールドフォーマットに関する情報を含み、ＲＤＦ（Resource Description Framework）ｎタプルのセット（例えば、トリプル）として表されても良い。レコードシェイプは、データモデルと独立であるために、ＲＤＦＳ／ＯＷＬオントロジを用いても良い。方法が各々のデータストアにより用いられるデータモデルの詳細に無関心なので、これは「ストア非依存」アプローチとも呼ばれる。

【００５２】

本発明の第２の態様によると、複数の異種データストアを有する多言語データティアにおいてデータ制約を施行するデータ制約エンジンであって、前記データが格納される方法及び場所に係わらず、前記データストアの中のデータを、前記データをシリアライズするレコードと見なす手段と、検証されるべきレコードを抽出する手段と、前記の抽出されたレコードに基づき、シェイプカタログからのレコードシェイプにアクセスし、レコードの構造を決定する手段であって、各々のレコードシェイプは、拡張可能語彙で表される、手段と、対応するレコードシェイプの中で定められる複数の基準に対して前記レコードを検証し、全部の前記基準が満たされる場合に前記レコードを有効であると決定する複数の検証器と、を有するデータ制約エンジンが提供される。

【００５３】

データ制約エンジンは、望ましくは、クライアント要求のためのインタフェースと、レコードディスパッチャと、を更に備えられる。したがって、一実施形態では、複数の異種データストアを有する多言語データティアにおいてデータ制約を施行するデータ制約エンジンであって、要求を処理するインタフェースであって、各々の要求はデータを指定し、前記インタフェースは、要求から、該要求の中で指定されたデータに対応するレコードを

抽出するよう構成され、レコードは、データが格納される方法及び場所にかかわらず、前記データストアの中のデータをシリアライズする、インタフェースと、前記インタフェースにより抽出されたレコードに基づき、シェイプカタログからのレコードシェイプにアクセスし、レコードの構造を決定する手段であって、各々のレコードシェイプは拡張可能な語彙で表される、手段と、前記指定されたデータをルーティングし、又は前記指定されたデータに対応するレコードが検証器により検証された後に、前記多言語データティアの中の適切なデータストアからデータを検索するレコードディスパッチャと、が提供される。

【００５４】

前記多言語データティアの中の異種データストアの各々は、望ましくは、データソース識別子を有する抽象データソースとして表され、前記要求は、前記指定されたデータに対応する前記データソース識別子を示す情報を含み、望ましくは、前記インタフェースは、前記要求から前記データソース識別子を抽出するよう構成される。

【００５５】

前記複数の検証器は、スロットカウント、濃度、データ型、及びフォーマット（ここで、フォーマットは、例えばＨＴＭＬ、ＸＭＬ、又はＪＳＯＮを含む）の各々のための個々の検証器を有しても良い。スロットカウントは、レコードの中の「スロット」の数を表す（ここで、スロットは、レコードの１又は複数のフィールドのラッパーである）。他の検証器が、各々のスロットに適用されても良い。例えば、濃度は、スロットの中に存在し得る要素の数を表しても良い。データ型は、スロットの各々のフィールドの中で許可されるデータの型を指定しても良い。フォーマットは、ＨＴＭＬ、ＸＭＬ、又はＪＳＯＮのような特定の言語に従ってそれぞれ満たされるシンタックスを定めても良い。

【００５６】

各々のレコードシェイプは、望ましくは、ＲＤＦＳ／ＯＷＬ語彙で表されるＲＤＦ（Resource Description Framework）トリプル（又はｎタプル）である。ＲＤＦトリプルは、識別する前提として、ＵＲＩのようなウェブ識別子を用いて「もの」（つまり、オブジェクト、リソース又はインスタンス）を識別し、それら識別される「もの」を簡易なプロパティ及びプロパティ値の観点で記述する。トリプルの観点では、主語はエンティティを記述するウェブリソースを特定するＵＲＩであっても良く、述語は特性の種類（例えば、色）を特定するＵＲＩであっても良く、目的語は問題のエンティティに起因する特性の種類の特定のインスタンスを指定するＵＲＩであっても良い。

【００５７】

上述のデータ制約エンジンの特徴は、上述の方法のうちの任意のものに適用され得る。逆も同様である。

【００５８】

本発明の第３の態様によると、上述のデータ制約エンジンとして機能するよう構成されるコンピューティング装置が提供される。

【００５９】

本発明の第４の態様によると、コンピューティング装置により実行されると、前記コンピューティング装置に上述のコンピューティング装置として動作させる、コンピュータプログラムが提供される。

【００６０】

本発明の実施形態は、多言語データティアにおけるデータ制約を扱うときに生じる以下の問題を解決する。

【００６１】

Ａ）データ設計者及び開発者は、複数の制約定義言語を扱わなければならない。これは、保守を益々困難にする。

【００６２】

Ｂ）予期しないデータモデルを採用するデータストアが多言語データティアに追加される場合がある。したがって、拡張可能なアプローチが必要である。

【００６３】

Ｃ）多言語データティアは、リモートの第三機関データストアを有する場合が多い。例えば、データベースが直接制御下にない。したがって、多言語データティアアーキテクチャは、代替の制約施行メカニズムを必要とする。

【００６４】

今日のプロトコルは、上述の問題を解決できない。より具体的には、次の通りである。

【００６５】

Ａ）制約を宣言し施行するためのストア非依存アプローチがない。したがって、多言語データティアにおける採用を妨げている。

【００６６】

Ｂ）予期しないデータモデルに適合する拡張可能な設計がない。

【００６７】

Ｃ）これらの大部分は、データストアに対する直接制御を必要とする。したがって、第三機関リモートデータベースをサポートしない。

【００６８】

本発明の実施形態は、データ固有のデータモデルに束縛される制約を置換するのではなく、多言語データティアにおけるデータ検証のための汎用的アプローチを提供する。

【００６９】

ストア非依存エンジンは、多言語データティアにおける制約施行のために提案される。制約は、宣言型アプローチにより記述されるので、データストア固有制約言語が用いられない。さらに、制約は軽量なオントロジの上でモデル化されるので、拡張は自然にサポートされる。制約は、独立型レポジトリに格納され、検証エンジンにより実行時間に施行される。したがって、第三機関データストアを有する多言語データティアが自然にサポートされる。

【００７０】

したがって、本発明の一実施形態は、多言語データティアのためのストア非依存データ制約エンジンである。データ制約エンジンは、ＲＤＦＳ／ＯＷＬを用いて表されるデータ制約（つまり、ルール）を用いて、多言語データティアに格納された（又は格納されるべき）データに関連するデータ操作（要求）をチェックしても良い。

【００７１】

さらに具体的には、本発明の一実施形態は、ＲＤＢＭＳ、トリプルストア、及びＭｏｎｇｏＤＢのような種々の種類の複数のデータベース固有データストアを有する多言語データティアにおいてデータ制約を施行するデータ制約エンジンを提供できる。データ制約エンジンは、ストア非依存の方法で（所謂「レコードシェイプ」を用いて）データ制約が定められるために、「レコード」に基づく統一モデルの概念を用いる。

【００７２】

データ制約エンジンは、例えば、多言語データティアの中のデータにアクセスするためにリモートクライアントから入ってくる要求を処理するＡＰＩを含むことにより、ユーザ要求に提供され得る。ＡＰＩは、各々の要求から、要求の中で指定されたデータに対応するレコード、及び指定されたデータを保持するデータストアを識別するデータソース識別子を抽出する。次に、インタフェースにより抽出されたレコードに基づき、適切なレコードシェイプが、シェイプカタログから抽出される。レコードシェイプは、レコードの構造を決定する。検証器は、それぞれ、フォーマット、データ型、濃度、及びスロット数のような種々の基準に従ってレコードシェイプに対してレコードを検証する。本例では、レコードが検証される場合、レコードディスパッチャは、データソース識別子を用いて適切なデータストアへ指定されたデータを向ける。

【００７３】

上述の及び他の実施形態では、以上で識別された技術的問題は、以下のように解決される。

【００７４】

Ａ）本発明は、「レコードシェイプ」の概念を導入する。これは、データモデルと独立

な、ＲＤＦＳ／ＯＷＬ語彙に基づく宣言型の制約である。既存の提案と異なり、このようなオントロジは、データモデル非依存であるよう設計される。レコードシェイプ及びレコードに基づく統一データモデルにより、データ制約エンジンは、ストア非依存アプローチを保証し、データベース固有の制約言語から開発者を解放する。したがって、多言語データティアのシナリオに適合する。さらに、レコードシェイプは習慣的なＲＤＦトリプルなので、開発者は、新しい制約定義言語を学習する必要がない。

【００７５】

Ｂ）ＲＤＦＳ／ＯＷＬ語彙によりレコードシェイプをモデル化することは、データベース固有制約への拡張性を保証し、したがって、広範なデータストア及び予期しないデータモデルのサポートを可能にする。言い換えると、既存のシェイプは、直ちに変更され、新しいシェイプが追加される。拡張性は、モジュラ及び拡張可能なデータ検証器によっても保証される。

【００７６】

Ｃ）レコードシェイプは、多言語ティアの中の各々のデータストアの内部に格納される必要はない。代わりに、レコードシェイプは、多言語ティアアーキテクチャ（シェイプカタログ）の直接制御の下で独立したレポジトリに格納される。したがって、第三機関データストアのサポートを可能にする。

【図面の簡単な説明】

【００７７】

【図１】マルチティアアーキテクチャの概略図を示す。

【図２】ＲＤＦグラフの一例を示す。

【図３Ａ】データソースとレコードとの間の変換を示し、トリプルからレコードへの変換を示す。

【図３Ｂ】データソースとレコードとの間の変換を示し、関係型テーブルからレコードへの変換を示す。

【図４】本発明の実施形態で用いられるレコードシェイプ語彙を示す。

【図５Ａ】図４のレコードシェイプ語彙を用いて定められる例示的なレコードシェイプを示し、ＲＤＦグラフのレコードシェイプを示す。

【図５Ｂ】図４のレコードシェイプ語彙を用いて定められる例示的なレコードシェイプを示し、関係型ＤＢテーブルのレコードシェイプを示す。

【図６】本発明の一実施形態で提供されるデータ制約エンジンのアーキテクチャを示す。

【図７】本発明の一実施形態で用いられる制約施行アルゴリズムのフローチャートである。

【図８】データ検証器のデータ制約エンジンへの追加を示す。

【図９】本発明のデータ制約エンジンを実装するために適するコンピュータシステムを示す。

【発明を実施するための形態】

【００７８】

本発明の実施形態は、例として図を参照して説明される。

【００７９】

この章は、ｉ）検証制約モデル及びそれらの基準、ｉｉ）検証エンジンアーキテクチャ、ｉｉｉ）検証制約施行メカニズム、を説明する。制約がどのように構築されるかを説明する前に、制約施行エンジンにより使用されるデータモデルが紹介される。

【００８０】

本発明の実施形態は、レコードの概念に基づく「ストア非依存」モデルを採用する（定義１）。

【００８１】

定義１：（レコード）レコードは、以下に示すように、コンマで区切られる値のｎ要素タプルで構成される。

【００８２】

値１，値２，値３，．．．，値Ｎ
制約施行エンジンは、情報がデータティアに格納されている方法及び場所に係わらず（例えば、ＲＤＢＭＳの関係型テーブルとして、トリプルストアのグラフとして、ＭｏｎｇｏＤＢの文書として、等）、データをレコードと見なす。
【００８３】
記憶と独立しアプローチを保証するために、レコードは、論理的にデータソースに組織化される（定義２）。
【００８４】
定義２：（データソース）データソースは、データベース固有制約の抽象表現である（例えば、関係型テーブル、ＲＤＦグラフ、ＭｏｎｇｏＤＢ、等）。
【００８５】
前述の企業－製品－顧客の例では、顧客はトリプルストアの中のグラフhttp://customersに格納され、関係型テーブルcompaniesに中の企業プロファイルは、ＲＤＢＭＳに含まれると仮定する（図１）。ＲＤＦグラフの中の及びテーブルの中のタプルは、制約施行エンジンによりレコードにシリアライズされる。
【００８６】
図３Ａでは、レコードは、ＲＤＦグラフhttp://customersの抽象表現であるCustomersと名付けられたデータソースに関連付けられる。例示的なレコードの中の各々のカンマで区切られた値は、ＲＤＦ述語の目的語に対応する（例えば、John　Doeは述語foaf:nameの目的語である）。
【００８７】
図３Ｂで、レコードの中の各々のカンマで区切られた値は、関係型テーブルに格納された属性に対応する。レコードは、Companiesと名付けられたデータソースに関連付けられる。データソースは、関係型テーブルcompaniesの抽象表現である。
【００８８】
各々のデータソースは、データ制約をモデル化するエンティティであるレコードシェイプに関連付けられる（定義３）。
【００８９】
定義３：（レコードシェイプ）レコードシェイプは、各々のレコードがどのように構造化されなければならないかを決定するデータ制約セットである。レコードシェイプに含まれる制約は、レコードフィールドに関連付けられ、以下に関する情報を含む。
【００９０】
・データ型
・濃度（cardinality）（つまり、存在する要素の数）
・フィールドフォーマット
レコードシェイプは、多言語データティアに責任を持つデータ設計者又はバックエンド開発者により手動で作成される。
【００９１】
レコードシェイプは、宣言型アプローチに従う。それらは、ＲＤＦで表され、レコードシェイプ語彙である軽量なＲＤＦＳ／ＯＷＬオントロジでモデル化される。本発明は、既存のオントロジ（例えばＯＳＬＣ、ＱＵＤＴ）のクラス及びプロパティを再利用し拡張するリンク付きデータ理念を採用するが、既存の研究とは異なり、データモデル非依存型の方法で制約をモデル化する語彙が用いられる。この選択は、多言語データストアのサポートを保証する。
【００９２】
さらに、ＲＤＦＳ／ＯＷＬ語彙が設計により拡張できるので、このようなオントロジに基づくアプローチは、拡張可能なデータ制約を保証する。したがって、直接的なモデル追加は、後方互換性を妥協することなく、予期しないデータモデル、データ型、データフォーマット、又は測定単位を有するデータストアをサポートする。
【００９３】

図４は、語彙の主なクラス及びプロパティを示す。以下は、語彙要素の詳細な説明である。

【００９４】

クラス（Classes）

・レコード（Record）。アトミックな意味のあるデータ単位を表す。

・データソース（DataSource）。レコードエンティティの抽象ソースである。ＲＤＢＭＳのテーブル、トリプルストアのＲＤＦ名付きグラフ、ＣＳＶファイル、ＭｏｎｇｏＤＢ文書、Ｃａｓｓａｎｄｒａテーブル、等を有する。

・シェイプ（Shape）。データソース又はレコードを記述するレコードシェイプ。スロットのコンテナを有する。

・スロット（Slot）。スロットは、１又は複数のフィールドのラッパーを有する。

・フィールド（Field）。フィールドは、レコードのコンマで区切られた要素の構造を記述する。

・qudt：単位（qudt:Unit）クラスは、ＱＵＤＴ語彙からインポートされ、測定単位（例えば、メートル）を表すために用いられる。

【００９５】

プロパティ（Properties）

・hasShape。シェイプをレコード又はデータソースに関連ける。

・フィールド（Field）。スロットをフィールドに関連付ける。

・hasSlot。スロットをシェイプに関連付ける。

・Index。レコードの中のスロットのグローバルユニークなインデックスを決定する。

・isKey。スロットがレコードのユニークな識別子であるかどうかを決定する。

・isAutoKey。レコードが「黙示的な」キーを有するかどうかを決定する。プロパティは、ＲＤＦインスタンスのために用いられる。ＲＤＦインスタンスは、それらのＵＲＩによりユニークに識別されるが、このような情報片は、黙示的ＲＤＦプロパティとして現れない。したがって、このような特徴をモデル化するプロパティが必要である。

・isServerDefaultGraph。データソースがトリプルストアの規定グラフに対応するかどうかを記述する。

・Datatype。フィールドのｘｓｄデータ型を示す。

・format。フィールドのフォーマット情報を示す（例えば、ＪＳＯＮ（JavaScript Object Notation）、ＸＭＬ、ＨＴＭＬ、等）。このプロパティは、ＣＬＯＢ（Character Large Object、種々のデータベース管理システムにより用いられるデータ型）にあるフィールドのシンタックスチェックを可能にし、例えば、ＸＭＬ及びＨＴＭＬコンテンツが適格であることを検証し、ＪＳＯＮシンタックス検証を調べる、等である。留意すべきことに、サポートされるフォーマットのリストは、他の文字の大きなオブジェクトに及びバイナリオブジェクト（例えば、ＰＤＦ、画像、等）に拡張される。

・unit。ＱＵＤＴ語彙に従い、スロットの測定単位を示す。

・vann:preferredNamespacePrefix。このプロパティは、ＶＡＮＮ語彙に属する（ＶＡＮＮは、他の語彙の注釈を可能にするために考案された語彙である）。レコードシェイプ語彙では、フィールドの中で用いられる名前空間プレフィックスを示す（このようなレコードがＲＤＦトリプルに対応する場合）。

・vann:preferredNamespaceUri。このプロパティは、ＶＡＮＮ語彙に属する。レコードシェイプ語彙では、フィールドの中で用いられるＵＲＩを示す（このようなレコードがＲＤＦトリプルに対応する場合）。

・oslc:occurs。このプロパティは、元来、ＯＳＬＣ語彙の中で現れる。これは、以下のインスタンスを参照することにより、フィールドの濃度を指定する。

・oslc:Exactly-one

・oslc:One-or-many

・oslc:Zero-or-many

・oslc:Zero-or-one

【００９６】
図５Ａ及び５Ｂは、２つの例示的なレコードシェイプを示す。図５ＡはＲＤＦグラフのシェイプであり、図５Ｂは関係型ＤＢテーブルのシェイプであり（プレフィックスは省略されている）、企業－製品－顧客の例に関する。２つのレコードシェイプはそれぞれ、図４のレコードシェイプ語彙により定められる（語彙は、recshプレフィックスにより示される）。
【００９７】
図５Ａで、シェイプは、顧客を記述するＲＤＦグラフの構造及び制約をモデル化する。データソースCustomersは、CustShシェイプに関連付けられる（第２行）。シェイプは、３個のスロットを有する。つまり、第１のスロット（第７～９行）は「黙示的」キーであり（第９行）、したがってフィールドを含まない。フィールドの値は、ＲＤＦリソースのユニークな識別子として機能するインスタンスのＵＲＩにより自動的に生成される（本例では、このような値はhttp://customers/１である）。第２のスロット（第１１～１３行）は、顧客の名前を記述するフィールドを有する（第１８～２２行）。このフィールドは、顧客の名前のＲＤＦプロパティをモデル化するプレフィックス及び名前空間を指定する（第１９～２０行）。濃度は第２１行で定められ、データ型は第２２行で定められる。第３のスロット（第１１～１４行）は、各々の顧客の知人をモデル化する（第２４～２８行）。顧客は複数の人々を知っている可能性があるので、濃度はゼロ又は多（zero or many）である（第２７行）。顧客は、ＵＲＩとして定義されなければならない（第２８行）。
【００９８】
図５Ｂで、シェイプは、企業関係型テーブルのコンテンツをモデル化する。データソースCompaniesは、レコードシェイプCompanyShに関連付けられる（第１～４行）。シェイプは、５個のスロットを有する（第５～６行）。第１のスロット（第８～１１行）は、各々のタプルのユニークな識別子を特定する（第１０行）。ユニークな識別子のフォーマットは、第２６～２８行のフィールドにより定められる。第２のスロット及びそのフィールドは、企業の名前をモデル化する（第１３～１５及び３０～３２行）。第３のスロット及びそのフィールドは、企業のＵＲＩをモデル化する（第１７～１８及び３４～３６行）。第４のスロット－フィールド対は、設立年をモデル化する（第２０～２１及び３８～４０行）。留意すべきことに、本例では、フィールド型はxsd:dateである。最後のスロット－フィールド対は、企業のＨＴＭＬ記述をモデル化する（第２３～２４及び４２～４５行）。留意すべきことに、このフィールドのデータ型はストリングであり（第４４行）、このようなストリングはＨＴＭＬシンタックスに従わなければならない（第４５行）。
【００９９】
図６は、ソフトウェアの観点からのシステム概略である。システムは、例としてリモートクライアントからの要求を参照することにより記述される。しかし、理解すべきことに、本発明はこのような要求の内容を検証することに限定されない。本発明の実施形態は、クライアント要求に関係なく、データストアからのデータ読み出しの検証、データストアの中のデータを検査すること、及びデータの発見にも適用できる。
【０１００】
データ制約エンジン１００は、２つの主要なコンポーネント、つまりレコードシェイプカタログ１１０と検証器１２０とを有する。
【０１０１】
シェイプカタログ１１０。これは、トリプルストアとして実装されるレコードシェイプレポジトリである。シェイプは、データ設計者により手動で作成され、このコンポーネントに格納される。カタログ１１０のお陰で、シェイプは、多言語ティアの中の各々のデータストアの内部に格納される必要がない。したがって、第三機関データストアのサポートを可能にする。データ制約エンジン１００の部分として示されるが、データ制約エンジンがアクセス可能な限り、シェイプカタログ１１０は、勿論、リモートに格納されても良い。

【０１０２】
検証器１２０。このモジュールは、シェイプに対するレコードの検証を担う。これらは以下を含む。
・スロットカウント検証器１２１。これは、シェイプに対してレコードスロットの数を調べる。
・濃度検証器１２２。これは、シェイプ濃度制約に対して各々のレコードフィールドの濃度を調べる。
・データ型検証器１２３。これは、レコードフィールドデータ型がシェイプデータ型に適合するかどうかを調べる。
・フォーマット検証器１２４。この検証器グループは、レコードシェイプの中のフォーマットプロパティにより指定されるものに従って、レコードフィールドシンタックスを調べる。
【０１０３】
上述の検証器は、シェイプカタログ１１０と一緒に格納され得る検証器リストの中で定められても良い。データ制約エンジンは、例えば、ＨＴＭＬ（検証器１２５）、ＸＭＬ（検証器１２６）、及びＪＳＯＮ（検証器１２７）のための内蔵シンタックス検証を提供される。留意すべきことに、サポートされるフォーマットのリストは、レコードシェイプオントロジのなかで拡張可能であり、したがって、新しいフォーマット検証器が第三機関により追加され得る。
【０１０４】
データ制約エンジン１００の前述のコンポーネントは、２つの外部モジュール、つまりＡＰＩ１３０及びレコードディスパッチャ１４０と連携して動作する。
【０１０５】
ＡＰＩ（又はより正確にはＡＰＩのセット）１３０は、リモートクライアント３０により要求される入来データ操作の処理、及び応答の構築を担うフロントエンドである。「データ操作」は、ここでは、生成、読み出し、更新、及び削除のような一般的な恒久的記憶機能を含む。例えば、ＨＴＴＰに基づくＡＰＩは、このような一般的操作をＰＯＳＴ（生成）、ＧＥＴ（読み出し）、ＰＵＴ（更新）、及びＤＥＬＥＴＥ（削除）にマッピングする。このようなデータ操作は、標準的に、自動的に又はユーザ入力に応答して、リモートクライアントにより実行されるアプリケーションにより生成される。
【０１０６】
レコードディスパッチャ１４０は、多言語データティア２０の中の正しいデータストアにレコードをルーティングし、及びそれからレコードを検索する。図６で、このデータティアは、例としてＲＤＢＭＳ２１、トリプルストア２２、ＭｏｎｇｏＤＢ２３を含むとして図示される。ドットにより示されるように、種々の更なるデータベースも、多言語データティア２０に含まれても良い。
【０１０７】
図７は、図６のデータ制約エンジン１００により実行される制約施行処理のフローチャートである。
【０１０８】
リモートクライアント３０は、例えばオペランドを得る、結果を書き込む、等のために多言語データティアへのアクセスを要求するアプリケーションを動かすことにより、多言語データティアの中のデータに関するデータ操作（アクセス要求）を生成すると仮定する。多言語データティアに対するこのようなデータ操作は、制約評価をトリガする。入ってくる（又は出ていく）レコードは、カタログ１１０に格納されるシェイプに対して検証される。無効なレコードが検証エラーをトリガする。有効なレコードは、要求されたデータストアへ送られる（又はそれから検索される）。
【０１０９】
リモートクライアントから入ってくるデータ操作の例に適用するとき、データ制約エンジン１００により実行される制約施行処理は、以下のように動作する。

【０１１０】
処理はステップＳ１００で開始する。ステップＳ１０２で、ＡＰＩ１３０は、データ操作を解析し（parse）、レコード及びデータソース識別子を抽出する。一方で、ステップＳ１０４で、エンジン１００は、カタログ１１０に問い合わせ、前のステップで抽出されたデータソースに関連するレコードシェイプをフェッチする。
【０１１１】
ステップＳ１０６で、レコードシェイプが存在するか否かが調べられる。シェイプが見付からない場合（Ｓ１０６で「ｎｏ」）、検証手順は進むことができず、レコードは、無効とマークされる（Ｓ１１６）。
【０１１２】
シェイプが見付かる（Ｓ１０６で「ｙｅｓ」）と仮定すると、Ｓ１０８で、シェイプのスロットの数に対してレコードのスロット数が一致するかが調べられる。不一致の場合（Ｓ１０８で「ｎｏ」）、レコードは無効である（Ｓ１１６）。その他の場合（Ｓ１０８で「ｙｅｓ」）、Ｓ１１０で、エンジンは、シェイプの中で指定された候補に対して、各々のレコードフィールドの候補を調べる。不一致が検出された場合（Ｓ１１０で「ｎｏ」）、レコードは無効である（Ｓ１１６）。
【０１１３】
次に、Ｓ１１２で、データ制約エンジン１００は、各々のレコードフィールドがシェイプに含まれるデータ型と一致するデータ型を有することを検証する。不一致が検出された場合（Ｓ１１２で「ｎｏ」）、レコードは無効である（Ｓ１１６）。その他の場合、処理はＳ１１４へ進み、フォーマットプロパティに従って（このようなプロパティがレコードシェイプの中に存在する場合）各々のフィールドのシンタックスを調べる。固有フォーマット検証器が実行される（追加データフォーマットに対するＨＴＭＬ、ＸＭＬ、ＪＳＯＮ、又は第三機関拡張シンタックスチェック）。シンタックス検証が成功しない場合（Ｓ１１４で「ｎｏ」）、レコードは無効である（Ｓ１１６）。その他の場合、レコードは有効であり（Ｓ１１８）、要求されたデータストアへディスパッチされる（又はそれから検索された対応するデータである）。
【０１１４】
例えば、「生成（Create）」操作（例えば、ＨＴＴＰ　ＰＯＳＴ）と共に５個のレコードが多言語データティアへ送られ、それらはデータ制約エンジン１００により検証される仮定する。各々の操作も、レコードに関連するデータソースの名前を制約する。
i) http://customers/１, “John Doe”, http://customers/２（レコードはデータソースCustomersに属する）
ii) http://customers/１, http://customers/２（レコードはデータソースCustomersに属する）
iii) http://customers/１, , http://customers/２（レコードはデータソースCustomersに属する）
iv) ２, “ACME inc.”, http://acme.com, ２００６, “<html><head>…”（レコードはデータソースCompaniesに属する）
v) ２, “ACME inc.”, http://acme.com, １９９０-１１-０１, “<html<head>…”（レコードはデータソースCompaniesに属する）
レコード（ｉ）は、Customersデータソースに属する。エンジンは、このようなデータソースに関連するレコードシェイプを検索するためにカタログに問い合わせる。レコードシェイプが存在し（CustSh、図５Ａを参照）、次にレコードを検証するために使用される。先ず、スロット数が調べられる。レコード（ｉ）は、レコードシェイプCustShのような３つのコンマで区切られたスロットを有する。各々のフィールドの候補が検証される。それらは全て正しいので、エンジンは、データ型検証と共に進む。レコード（ｉ）はＵＲＩで始まる。これは、黙示的キーの正しいデータ型である（図５Ａ、第９行）。スロット２は、有効な値（ストリング）を有し、最後のスロットは、ＵＲＩフィールドを有する。これは、シェイプと一致する。したがって、レコード（ｉ）は有効である。

【０１１５】
レコード（ｉｉ）は、Customersデータソースに属する。エンジンは、このようなデータソースに関連するレコードシェイプを検索するためにカタログに問い合わせる。レコードシェイプが存在し（CustSh、図５Ａを参照）、次にレコードを検証するために使用される。先ず、スロット数が調べられる。レコード（ｉｉ）は、レコードシェイプCustShにより要求される３つのスロットの代わりに、２つのコンマで区切られたスロットを有する。したがって、レコード（ｉｉ）は有効ではない。
【０１１６】
レコード（ｉｉｉ）は、Customersデータソースに属する。エンジンは、このようなデータソースに関連するレコードシェイプを検索するためにカタログに問い合わせる。レコードシェイプが存在し（CustSh、図５Ａを参照）、次にレコードを検証するために使用される。先ず、スロット数が調べられる。レコード（ｉｉｉ）は、レコードシェイプCustShのような３つのコンマで区切られたスロットを有する。各々のフィールドの候補が検証される。そのレコードシェイプが正確に１つの要素が存在しなければならないと規定するにも係わらず（図５Ａ、第２１行）、第２のフィールドは空である。よって、レコード（ｉｉｉ）は有効ではない。
【０１１７】
レコード（ｉｖ）は、Companiesデータソースに属する。カタログは、データソースに関連するシェイプについて問い合わせられる。１つのシェイプが見付かる（CompanySh、図５Ｂ）。スロット数チェックの後、フィールド候補が検証される。それらは正しいので、エンジンはデータ型のチェックに進む。１つのエラーが第３のフィールドで検出される（「２００６」）。このような値は、xsd:dateのＹＹＹ－ＭＭ－ＤＤフォーマットに適合しない。したがって、レコード（ｉｖ）は有効ではない。
【０１１８】
レコード（ｖ）は、Companiesデータソースに属する。カタログは、データソースに関連するシェイプについて問い合わせられる。１つのシェイプが見付かる（CompanySh、図５Ｂ）。スロット数チェックの後、フィールド候補が検証される。それらは正しいので、エンジンはデータ型のチェックに進む。データ型は全て正しい。CompanyShシェイプは最後のフィールドが有効なＨＴＭＬコンテンツを有しなければならないと記載する。シンタックス検証は、「<html<head>…」ストリングに対して実行される。<htmlタグが閉じられていないので、シンタックスは正しくない。したがって、レコード（ｖ）は有効ではない。
【０１１９】
ＰＯＳＴ操作の場合には、次に、有効であると分かったレコードは、記憶のために多言語データティアへ転送される。レコードが無効であると分かった場合、エラーメッセージが、要求の生じたリモートクライアント３０へ返される。
【０１２０】
多の種類のアクセス要求は、例えば命令が多言語データティアに渡される前に検証されるＧＥＴ命令により指定されたデータと共に、同様の方法で処理され得る。
【０１２１】
さらに、データ制約エンジンの仕様は、多言語データティアに追加される又はそれから検索されるデータを指定する入ってくるデータ操作の検証に限定されない。同様に、多言語データティアに既に格納されているデータの検証に適用され得る。
【０１２２】
一例として、データ制約エンジンは、（ＧＥＴ要求に応答してのような）任意の理由で多言語データティアから読み出されるレコードを検証するために使用され得る。
【０１２３】
別の例として、データ制約エンジンは、各々のレコードが特定のデータストアのために定義されたレコードシェイプに従うか否かを調べるために、該データストアに（又はインテグリティの疑われるその部分に）体系的に適用され得る。本例では、ＡＰＩ１３０及び

リモートクライアント３０は、チェックを開始し、結果をリモートクライアントに報告する以外に、処理に含まれる必要がない。

【０１２４】

データ制約エンジンがデータストアのコンテンツを発見し又はあるデータストアから別のデータストアへデータを転送するために用いられ得る別の例。

【０１２５】

図８は、データ検証器リスト（及び／又はシェイプカタログ１１０）に拡張を追加する処理を示す。

【０１２６】

データ制約エンジン１００（図６）の検証器リストは、第三機関により拡張可能である。したがって、予期しないデータモデルに基づくデータストア及び追加データフォーマット（例えば、ＰＤＦ、画像、等のようなバイナリオブジェクト）をサポートする。以下のステップが実行されない限り、データフォーマットに対する制約がサポートされないことに留意する。新しいデータ検証器を追加する処理は、以下のように図８に纏められる。

【０１２７】

処理はステップＳ２００で開始する。ステップＳ２０２で、データ制約エンジンは、レコードシェイプオントロジの現在のバージョンが更新されるか否かを調べる。検証器リストの拡張は、（例えば追加プロパティを追加することにより）オントロジ編集を必要とする場合がある。したがって、データ制約エンジンは、最新バージョンを参照しなければならない。留意すべきことに、レコードシェイプオントロジは、レコードシェイプと一緒に、カタログに格納される。レコードシェイプオントロジが古い場合（Ｓ２０２で「ｙｅｓ」）、エンジンは、Ｓ２０４で最新バージョンを検索するために、カタログに問い合わせる。ステップＳ２０６で、オントロジが更新されると（必要な場合）、エンジンは、任意の追加検証器（例えば、新しいレコードシェイプ）を追加することにより検証器リストを更新する。処理はステップＳ２０８で終了する。　留意すべきことに、図８に記載の手順は、ブートストラップ時間に実行される、又はシステム管理者により手動でトリガされ得る。したがって、検証器は、いつでも、データ制約エンジンにプラグインされ得る。

【０１２８】

図９は、本発明を実施するのに適するコンピュータシステム１０又はその部分を概略的に示す。コンピュータシステム１０は、図６に示すデータ制約エンジン１００のプログラムコードを含む種々のプログラム及びデータを格納するメモリ１４を有する。メモリは、メモリに保持されるプログラムを実行するＣＰＵ１２に接続される（当業者により理解されるように、ＣＰＵは実際には多くの別個のＣＰＵ又はコアであっても良い）。入力／出力部１６は、コンピュータシステム１０の外部のエンティティ、特にリモートクライアント３０及び２つのデータベース２５及び２６により例示される多言語データティア２０と、（インターネットのような）ネットワーク４０を介して通信を実行する。

【０１２９】

纏めると、本発明の実施形態は、多言語データティアにおける制約施行のためのストア非依存エンジンを提供できる。制約は、宣言型アプローチにより記述されるので、データストア固有制約言語が用いられない。さらに、制約は軽量なオントロジの上でモデル化されるので、拡張は自然にサポートされる。制約は、独立型レポジトリに格納され、検証エンジンにより実行時間に施行される。したがって、第三機関データストアを有する多言語データティアが自然にサポートされる。

【０１３０】

上述の態様の何れにおいても、種々の特徴は、ハードウェアで、又は１若しくは複数のプロセッサで動作するソフトウェアモジュールとして実施されても良い。ある態様の特徴は、他の態様の特徴に適用されても良い。

【０１３１】

本発明は、上述の任意の方法を実行するコンピュータプログラム又はコンピュータプログラムプロダクト、及び上述の任意の方法を実行するプログラムを格納しているコンピュ

ータ可読媒体も提供する。本発明を実施するコンピュータプログラムは、コンピュータ可読媒体に格納されてもよい。或いは、例えば、インターネットウェブサイトから提供されるダウンロード可能なデータ信号のような信号形式又は任意の他の形式であってもよい。

【０１３２】

上述の実施形態に加え、更に以下の付記を開示する。

（付記１）　複数の異種データストアを有する多言語データティアにおいてデータ制約を施行する方法であって、

データが格納される方法及び場所に係わらず、前記データストアの中のデータを、前記データをシリアライズするレコードと見なすステップと、

検証されるべきレコードを抽出するステップと、

前記レコードに対応するレコードシェイプを見付け、レコードの構造を決定するステップであって、各々のレコードシェイプは拡張可能語彙で表される、ステップと、

前記対応するレコードシェイプの中で定められる複数の基準の各々に対して前記レコードをチェックすることにより、前記レコードにデータ制約を適用するステップと、

全部の前記基準が満たされる場合、前記レコードを有効であると決定するステップと、

を有する方法。

（付記２）　前記レコードが有効であると決定される場合、前記データストアの中に前記レコードを作成する、データストアから前記レコードを読み出す、前記レコードを用いてデータストアを更新する、及びデータストアからレコードを削除する、のうちの１又は複数を含む操作を前記レコードに対して実行するステップ、を更に有する請求項１に記載の方法。

（付記３）　指定データを含む要求を受信するステップと、前記指定データに基づき検証されるべきレコードを抽出するステップと、を更に有する請求項１又は２に記載の方法。

（付記４）　前記レコードは、前記要求に含まれる、又は

前記レコードは、前記データストアのうちの１つに含まれ前記要求の中で指定される、

請求項３に記載の方法。

（付記５）　各々のデータストアを、データソース識別子を有する抽象データソースとして表すステップであって、前記要求は、前記指定データに対応するデータソース識別子を特定できる情報を含む、ステップ、を更に有する請求項３又は４に記載の方法。

（付記６）　各々のレコードは、コンマで区切られた値のｎ要素タプルである、請求項１乃至５のいずれか一項に記載の方法。

（付記７）　前記データストアは、

（ｉ）トリプルストアであって、該トリプルストアの中のデータのレコードの中で、各々のコンマで区切られた値はＲＤＦ述語の目的語に対応する、トリプルストア、

（ｉｉ）ＲＤＢＭＳであって、該ＲＤＢＭＳの中のデータのレコードの中で、各々のコンマで区切られた値はテーブルに格納された属性を表す、ＲＤＢＭＳ、

（ｉｉｉ）ＭｏｎｇｏＤＢのような文書指向型データベース、

（ｉｖ）Ｃａｓｓａｎｄｒａのような列指向型テーブルに基づくデータベース、

（ｖ）キー値ペアに基づくデータベース、

のうちのいずれかを有する、請求項３に記載の方法。

（付記８）　新しい型のデータストアが前記多言語データティアに追加されるとき、前記データストアに格納されたデータの構造を定める新しいレコードシェイプを定めるために、前記拡張可能語彙を用いるステップ、を更に有する請求項１乃至７のいずれか一項に記載の方法。

（付記９）　各々のレコードシェイプは、レコードを形成するデータ型、濃度、及びフィールドに関する情報を含む、請求項１乃至８のいずれか一項に記載の方法。

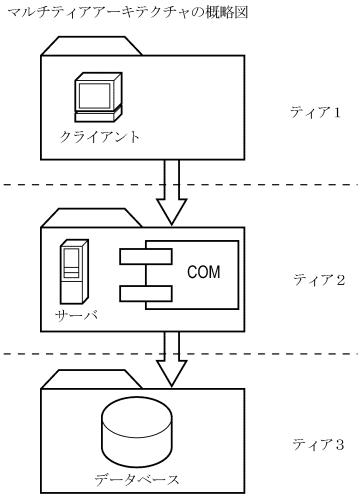
（付記１０）　各々のレコードシェイプは、ＲＤＦ（Resource Description Framework）ｎタプルのセットであり、望ましくは前記拡張可能語彙はＲＤＦＳ／ＯＷＬに基づく、請求項１乃至９のいずれか一項に記載の方法。

（付記１１）　複数の異種データストアを有する多言語データティアにおいてデータ制約

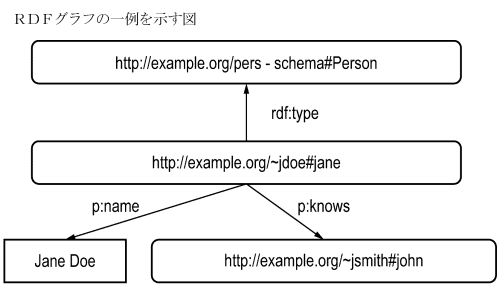
を施行するデータ制約エンジンであって、
データが格納される方法及び場所に係わらず、前記データストアの中のデータを、前記データをシリアライズするレコードと見なす手段と、
前記レコードを抽出する手段と、
前記の抽出されたレコードに基づき、シェイプカタログからのレコードシェイプにアクセスし、レコードの構造を決定する手段であって、各々のレコードシェイプは、拡張可能語彙で表される、手段と、
対応するレコードシェイプの中で定められる複数の基準に対して前記レコードを調べ、全部の前記基準が満たされる場合に前記レコードを有効であると決定することにより前記レコードを検証する複数の検証器と、
を有するデータ制約エンジン。
（付記１２） 入ってくる要求を受信するインタフェースであって、各々の要求はデータを指定する、インタフェースを更に有し、前記抽出する手段は、前記要求の中で指定されたデータに基づき、前記レコードを抽出するよう構成される、請求項１１に記載のデータ制約エンジン。
（付記１３） 前記レコードが有効であると決定される場合、前記データストアの中に前記レコードを作成する、データストアから前記レコードを読み出す、前記レコードを用いてデータストアを更新する、及びデータストアからレコードを削除する、のうちの１又は複数を含む操作を前記レコードに対して実行するレコードディスパッチャ、を更に有する請求項１１又は１２に記載のデータ制約エンジン。
（付記１４） 前記複数の検証器は、それぞれ
スロット数、
濃度、
データ型、
ＨＴＭＬ、ＸＭＬ、及びＪＳＯＮのうちの１又は複数のようなフォーマット、
である個々の検証器を有する、請求項１１、１２、又は１３に記載のデータ制約エンジン。
（付記１５） 各々のレコードシェイプは、ＲＤＦＳ／ＯＷＬ語彙で表されるＲＤＦ（Resource Description Framework）トリプルである、請求項１１乃至１４のいずれか一項に記載のデータ制約エンジン。
（付記１６） 請求項１１乃至１５のうちのいずれか一項に記載のデータ制約エンジンとして機能するよう構成されるコンピューティング装置。
（付記１７） コンピューティング装置により実行されると、前記コンピューティング装置に請求項１６に記載のコンピューティング装置として動作させる、コンピュータプログラム。
【産業上の利用可能性】
【０１３３】
レコードシェイプ及びレコードに基づく統一データモデルに依存することにより、本発明は、データ制約を施行するストア非依存アプローチを可能にし、データベース固有の制約言語から開発者を解放する。したがって、多言語データティアのシナリオに適合する。さらに、レコードシェイプは習慣的なＲＤＦトリプルなので、開発者は、新しい制約定義言語を学習する必要がない。ＲＤＦＳ／ＯＷＬに基づくオントロジの使用は、予期しないデータモデル及び型を扱うために、新しいレコードシェイプを追加することを容易にし、アプリケーションレベルの検証コードの必要性を低減又は除去する。したがって、本発明は、プログラミング労力の低減に貢献する。
【符号の説明】
【０１３４】
２０ データティア
２１ ＲＤＢＭＳ
２２ トリプルストア

２３　ＭｏｎｇｏＤＢ
３０　リモートクライアント
１００　検証器
１１０　シェイブカタログ
１２１　スロット数検証器
１２２　濃度検証器
１２３　データ型検証器
１２４　フォーマット検証器
１２５　ＨＴＭＬ検証器
１２６　ＸＭＬ検証器
１２７　ＪＳＯＮ検証器
１３０ＡＰＩ
１４０　レコードディスパッチャ

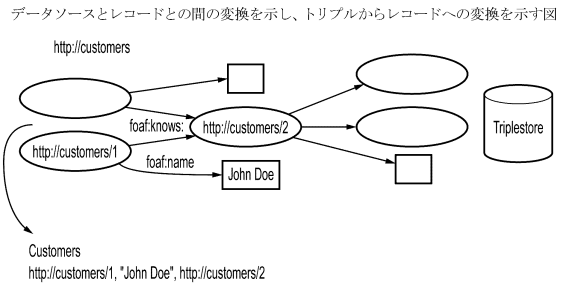
【図１】



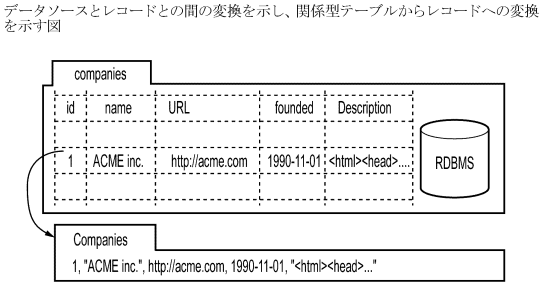
【図２】



【図３Ａ】

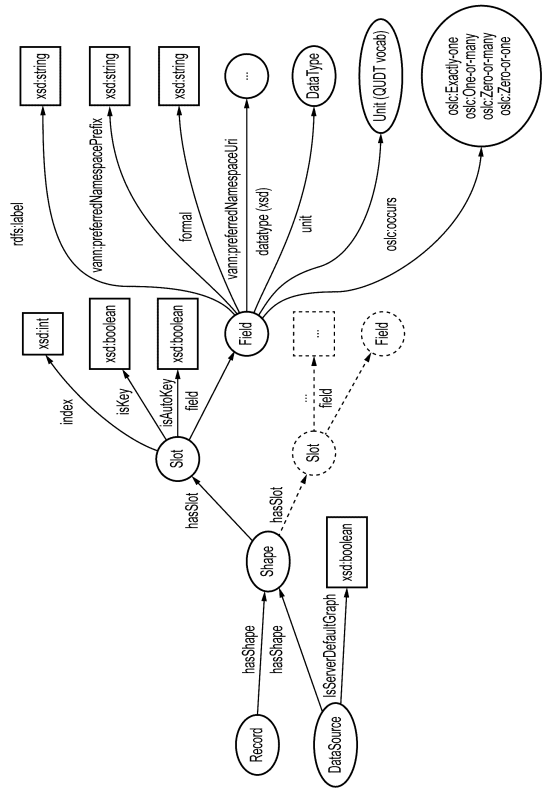


【図３Ｂ】



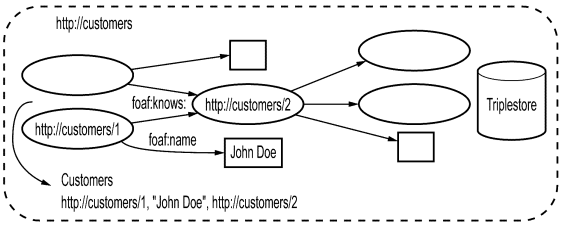
【図４】

本発明の実施形態で用いられるレコードシェイプ語彙を示す図



【図５Ａ】

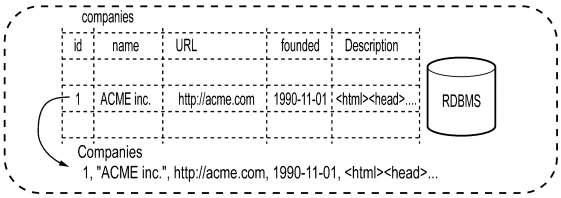
図４のレコードシェイプ語彙を用いて定められる例示的なレコードシェイプを示し、ＲＤＦグラフのレコードシェイプを示す図




```
:Customers a recsh DataSource;
        recsh:hasShape :custSh.

:custSh a recsh:Shape;
        recsh:hasSlot  :custShSlot0,  :custShSlot1,  :custShSlot2.

:custShSlot0 a recsh:Slot;
        recsh:index "0"^^xsd:integer;
        recsh:isAutoKey "true"^^xsd:boolean.

:custShSlot1 a recsh:slot;
        recsh:index "1"^^xsd:integer;
        recsh:field :custField1.

:custShSlot2 recsh:index "2"^^xsd:integer;
        recsh:field :custField2.

:custField1 rdfs:label "name";
        vann:preferredNamespacePrefix "foaf";
        vann:preferredNamespaceUri <http://xmlns.com/foaf/0.1/>;
        oslc:occurs oslc:Exactly-one;
        recsh:dataType xsd:string.

:custField2 rdfs:label "knows";
    vann:preferredNamespacePrefix "foaf";
    vann:preferredNamespaceUri <http://xmlns.com/foaf/0.1/>;
    oslc:occurs oslc:Zero-or-many;
    recsh:dataType xsd:anyURI.
```

【図５Ｂ】

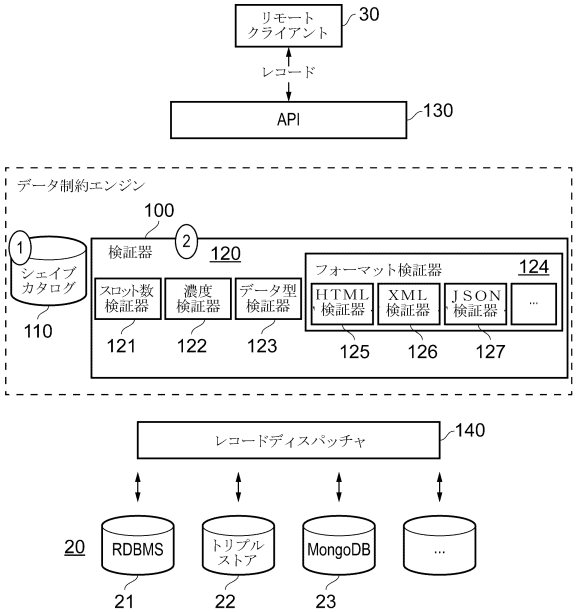
図４のレコードシェイプ語彙を用いて定められる例示的なレコードシェイプを示し、関係型ＤＢテーブルのレコードシェイプを示す図




```
:Companies a recsh:DataSource;
        recsh:hasShape :CompanySh.

:CompanySh a recsh:shape;
    recsh:hasSlot   :Compslot0,  :Compslot1,
                    :Compslot2, :Compslot3,  :Compslot4

:Compslot0 a recsh:slot;
        recsh:index "0"^^xsd:integer;
        recsh:isKey "true"^^xsd:boolean;
        recsh:field   :CompField0.

:Compslot1 a recsh:Slot;
        recsh:index "1"^^xsd:integer;
        recsh:field   :CompField1.

:Compslot2 recsh:index "2"^^xsd:integer;
        recsh:field :CompField2.

:Compslot3 recsh:index "3"^^xsd:integer;
        recsh:field :CompField3.

:Compslot4 recsh:index "4"^^xsd:integer;
        recsh:field :CompField4.

:CompField0 rdfs:label "id";
        oslc:occurs oslc:Exactly-one;
        recsh:dataType xsd:integer.

:CompField1 rdfs:label "name";
        oslc:occurs oslc:Exactly-one;
        recsh:dataType xsd:string.

:CompField2 rdfs:label "URL";
    oslc:occurs oslc:Zero-or-one;
    recsh:dataType xsd:anyURI.

:CompField3 rdfs:label "founded";
    oslc:occurs oslc:Zero-or-one;
    recsh:dataType xsd:date.

:CompField4 rdfs:label "description";
        oslc:occurs oslc:Zero-or-one;
        recsh:dataType xsd:string;
        recsh:format "HTML".
```

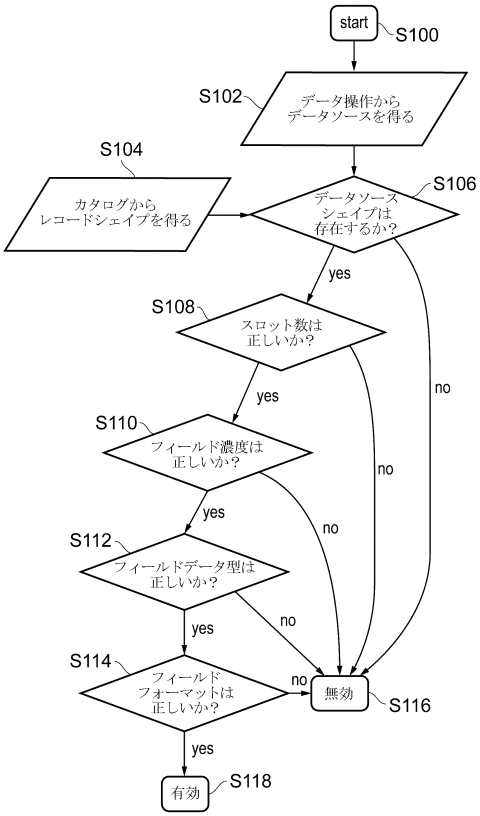
【図６】

本発明の一実施形態で提供されるデータ制約エンジンのアーキテクチャを示す図



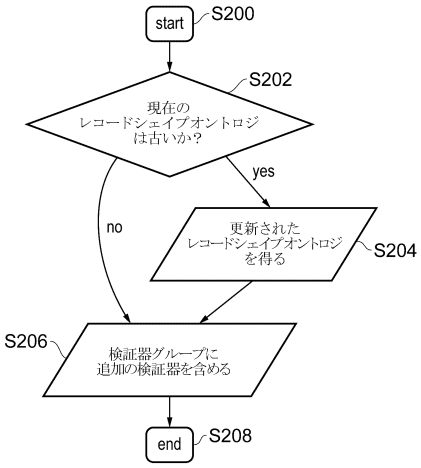
【図７】

本発明の一実施形態で用いられる制約施行アルゴリズムのフローチャート



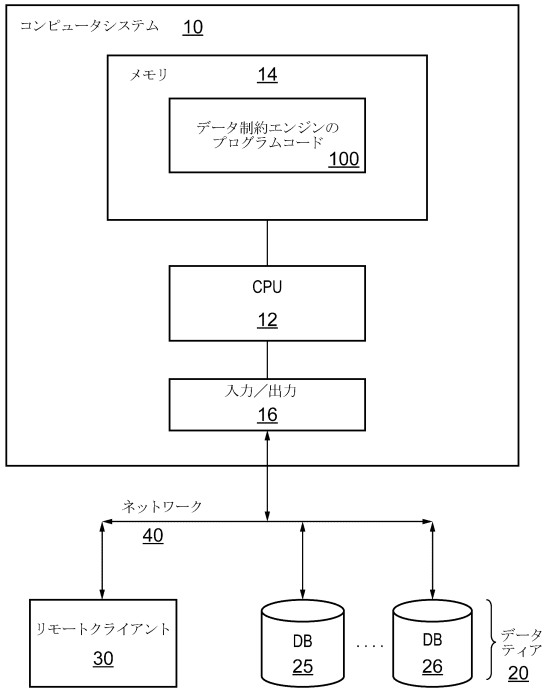
【図８】

データ検証器のデータ制約エンジンへの追加を示す図



【図９】

本発明のデータ制約エンジンを実装するために適するコンピュータシステムを示す図



フロントページの続き

(72)発明者　コスタベッロ・ルカ
アイルランド，ゴールウェイ，ボアーモア，キョート・セオイジ　４４番
(72)発明者　ヴァンデンブッシェ・ピエール－イヴ
アイルランド，ゴールウェイ，ブラックロック，ダン・ナ・カレッジ　２３番
(72)発明者　ウムブリヒ・ユルゲン
オーストリア，１０２０　ウィーン，ヴェルトハンデルシュプラッツ　１番，ビルディング・ディー２，エントランス・シー，セカンド・フロアー　ウィーン　ユニヴァーシティ　オブ　エコノミクス　アンド　ビジネス　インスティチュート　フォー　インフォーメーション　ビジネス内

審査官　早川　学

(56)参考文献　特開２００６－３１８３７１（ＪＰ，Ａ）
韓国公開特許第１０－２０１４－００６９６６９（ＫＲ，Ａ）
米国特許出願公開第２００８／０３０６９２６（ＵＳ，Ａ１）
渡辺祐太、他５名，ユビキタス環境におけるマルチデータベースの仮想化技術，情報処理学会シンポジウムシリーズ［ＣＤ－ＲＯＭ］，日本，社団法人情報処理学会，２０１０年　６月３０日，Ｖｏｌ.２０２０，Ｎｏ.１，ｐ．１２６２～１２６７

(58)調査した分野(Int.Cl.，ＤＢ名)
Ｇ０６Ｆ　１６／００－１６／９５８