(19)日本国特許庁(JP)　(12)特　許　公　報(B2)　(11)特許番号
特許第5673246号
(P5673246)
(45)発行日　平成27年2月18日(2015.2.18)　(24)登録日　平成27年1月9日(2015.1.9)

(51)Int.Cl.　F I
G06F 12/00 (2006.01)　G06F 12/00 513J
G06F 12/00 512
G06F 12/00 547M

請求項の数 7　(全 19 頁)

(21)出願番号　特願2011-55965(P2011-55965)
(22)出願日　平成23年3月14日(2011.3.14)
(65)公開番号　特開2012-194602(P2012-194602A)
(43)公開日　平成24年10月11日(2012.10.11)
審査請求日　平成25年11月29日(2013.11.29)

(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　100089118
弁理士　酒井　宏明
(72)発明者　溝渕　裕司
神奈川県川崎市中原区上小田中４丁目１番１号　富士通株式会社内
(72)発明者　宗像　聡
神奈川県川崎市中原区上小田中４丁目１番１号　富士通株式会社内
(72)発明者　小高　敏裕
神奈川県川崎市中原区上小田中４丁目１番１号　富士通株式会社内
最終頁に続く

(54)【発明の名称】データストア制御装置、データストア制御プログラムおよびデータストア制御方法

(57)【特許請求の範囲】
【請求項１】
第１データストアに配置されるテーブルと、前記第１データストアが管理方式の異なる第２データストアに配置されるテーブルとを定義した設定ファイルを参照し、前記第１データストアに対する処理を記述したプログラムで定義される永続化対象のうち、前記第２データストアのテーブルを参照することを示す第１属性を、前記プログラムの永続化対象から除外する除外部と、
前記設定ファイルを参照して、前記除外部が除外した前記第１属性によって指定される、前記プログラムの参照先を識別する第２属性を永続化対象として、前記プログラムに追加する追加部と、
前記追加された第２属性に対応する定義情報として、前記参照先を特定する特定情報を前記プログラムに定義する定義部と、
前記プログラムに定義される前記第１属性に対応した定義情報を、前記特定情報を用いて前記第２データストアから取得する処理の実行を示す情報に変更する変更部と
を有することを特徴とするデータストア制御装置。
【請求項２】
前記定義部は、前記定義情報として、前記第２属性に所定値を設定する設定関数と、前記第１属性に値が設定されている場合には、前記第２データストアに対する処理を記述した別プログラムに定義される第２属性の値を応答し、前記第１属性に値が設定されていない場合には、前記設定関数によって設定された所定値を応答する応答関数とを定義するこ

とを特徴とする請求項１に記載のデータストア制御装置。
【請求項３】
前記変更部は、前記第１属性に所定値を設定し、設定された所定値がＮＵＬＬであれば前記第２属性の値をＮＵＬＬに設定し、設定された所定値がＮＵＬＬでなければ前記第１属性の識別子を前記第２属性に設定する処理の実行を示す情報に、前記プログラムに定義される前記第１属性に対応した設定関数に定義される情報を変更するとともに、前記第１属性の値がＮＵＬＬでありかつ前記第２属性の値がＮＵＬＬでなければプライマリキーを用いて取得したオブジェクトを応答し、前記第１属性の値がＮＵＬＬでない又は前記第２属性の値がＮＵＬＬであれば前記第１属性に設定されている値を応答する処理の実行を示す情報に、前記プログラムに定義される前記第１属性に対応した応答関数に定義される情報を変更することを特徴とする請求項１に記載のデータストア制御装置。
【請求項４】
前記設定ファイルに基づいて、前記各テーブルごとにアクセスするデータストアのアクセッサを生成する生成部と、
前記生成されたデータストアごとのアクセッサを用いて、前記第１データストアまたは前記第２データストアに対してアクセスを実行するアクセス実行部と
をさらに有することを特徴とする請求項１～３のいずれか一つに記載のデータストア制御装置。
【請求項５】
前記アクセス実行部は、前記第１データストアまたは前記第２データストアに対する複数のアクセスを同時に開始して同時に終了させることを特徴とする請求項４に記載のデータストア制御装置。
【請求項６】
コンピュータに、
第１データストアに配置されるテーブルと、前記第１データストアが管理方式の異なる第２データストアに配置されるテーブルとを定義した設定ファイルを参照し、前記第１データストアに対する処理を記述したプログラムで定義される永続化対象のうち、前記第２データストアのテーブルを参照することを示す第１属性を、前記プログラムの永続化対象から除外し、
前記設定ファイルを参照して、除外した前記第１属性によって指定される、前記プログラムの参照先を識別する第２属性を永続化対象として、前記プログラムに追加し、
前記追加した第２属性に対応する定義情報として、前記参照先を特定する特定情報を前記プログラムに定義し、
前記プログラムに定義される前記第１属性に対応した定義情報を、前記特定情報を用いて前記第２データストアから取得する処理の実行を示す情報に変更する
処理を実行させることを特徴とするデータストア制御プログラム。
【請求項７】
コンピュータが実行する制御方法であって、
第１データストアに配置されるテーブルと、前記第１データストアが管理方式の異なる第２データストアに配置されるテーブルとを定義した設定ファイルを参照し、前記第１データストアに対する処理を記述したプログラムで定義される永続化対象のうち、前記第２データストアのテーブルを参照することを示す第１属性を、前記プログラムの永続化対象から除外し、
前記設定ファイルを参照して、除外した前記第１属性によって指定される、前記プログラムの参照先を識別する第２属性を永続化対象として、前記プログラムに追加し、
前記追加した第２属性に対応する定義情報として、前記参照先を特定する特定情報を前記プログラムに定義し、
前記プログラムに定義される前記第１属性に対応した定義情報を、前記特定情報を用いて前記第２データストアから取得する処理の実行を示す情報に変更する
ことを特徴とするデータストア制御方法。

【発明の詳細な説明】
【技術分野】
【０００１】
本発明は、データストア制御装置、データストア制御プログラムおよびデータストア制御方法に関する。
【背景技術】
【０００２】
従来より、テーブル単位でデータを格納し、SQL（Structured Query Language）文を用いてデータ操作が実行されるRDB（Relational Database）が利用されている。例えば、Java（登録商標）では、RDBを扱うフレームワークとしてJPA（Java（登録商標） Persistence Application Programming Interface）が提供されている。
【０００３】
図１７は、Java（登録商標）によるテーブル作成を説明する図である。図１７に示すように、Java（登録商標）プログラム上で永続化対象のクラスはRDB上の１テーブルに相当し、各オブジェクトはRDB上のテーブルの各行に相当し、各フィールドはRDB上のテーブルの項目に相当する。つまり、図１７に示したEmployeeクラスは、RDB上で社員テーブルを定義するものである。また、Employeeクラス内で定義される「id」、「name」、「position」各々は、社員テーブル上のカラム「社員ID」、「名前」、「役職」に対応する。
【０００４】
続いて、RDBの性質を利用してテーブル間で関係付ける例を説明する。図１８は、参照関係を有するテーブル作成例を示す図である。図１８に示すように、Java（登録商標）プログラム上で、「社員ID」と「名前」と「役職」を定義するEmployeeクラスと、「部署ID」と「部署名」を定義するDepartmentクラスとが参照関係にあるクラス図を実行する。この場合、データストアアクセッサは、「社員ID」、「名前」、「役職」、「部署ID＜FK（foreign key）＞」から形成される社員テーブルと、「部署ID」および「部署名」を有する部署テーブルをRDB上に生成する。つまり、社員テーブルの「部署ID＜FK＞」は、部署テーブルの「部署ID」を参照することが定義され、社員テーブルと部署テーブルとは参照関係にあることが定義される。このように、Java（登録商標）では、JPAを用いてRDBで管理可能な各種テーブルが作成されている。
【０００５】
近年では、クラウドコンピューティングの普及に伴って、従来から広く利用されているRDBの他に、分散KVS（Key-Value Store）等の新しいデータストアの利用が普及している。分散KVSは、サーバ間でのデータの一貫性保証を即時に実行する機能を有しないものの、複数のサーバでのデータ管理によるスケーラビリティや一部のサーバがダウンしてもサービスを継続できる高可用性を有し、RDBに比べてコストが安い。このようなことから、データの一貫性保証を即時にできるがコストが高いRDBと、コストが安く高可用性を有する分散KVSとの両方を用いてシステムを構築することが行われている。
【０００６】
このように、管理方式の異なるデータストアを利用したシステムでは、データストアを連携する技術やデータストアの参照アクセスを最適化する技術などが利用されている。例えば、管理方式の異なるデータストアを連携する技術として、各データストアに共通する基本操作を共通SQLと定義し、各データストア固有の操作については固有の言語を用いて操作する技術が利用されている。また、参照アクセスを最適化する技術として、Java（登録商標）におけるfinderメソッドのアクセスキーの依存関係と、getterメソッドやsetterメソッドによるアクセス状況とに基づいて、データストアへのアクセス回数を減らしたりすることが実施されている。
【先行技術文献】
【特許文献】
【０００７】
【特許文献１】特開２００２－２９７４１８号公報

【特許文献２】特開２００１－５１８７９号公報
【特許文献３】特開２００７－６６０１７号公報
【発明の概要】
【発明が解決しようとする課題】
【０００８】
しかしながら、従来の技術では、管理方式が異なるデータストアに跨ってデータを登録する場合に、外部キーを用いてデータストア間が関連付けられるので、データの配置を変更するなどのデータストアの保守作業を効率的に実行できないという問題があった。
【０００９】
上述したJPAを例にして説明すると、JPAでは、RDBにテーブルを作成するプログラム内に、RDBのデータが分散KVSを参照していることを外部キーで定義する。すなわち、JPAで複数のデータストアを扱う場合、データストアを跨るリレーションを有するテーブル間での外部参照は、アプリ側で実装することになる。このため、データの配置変更に伴ってアプリを修正するなどデータベース以外の保守を行うことになり、保守を行う範囲が広くなるので、データベースの保守作業を容易に行うことができず、効率的であるとは言い難い。
【００１０】
第１の側面では、データストアの保守作業を効率化するデータストア制御装置、データストア制御プログラムおよびデータストア制御方法を提供することを目的とする。
【課題を解決するための手段】
【００１１】
第１の案では、データストア制御装置は、第１データストアに対する処理を記述したプログラムで定義される永続化対象のうち、前記第１データストアが管理方式の異なる第２データストアを参照することを示す第１属性を永続化対象から除外する除外部を有する。データストア制御装置は、前記プログラムの参照先を識別する第２属性を永続化対象として、前記プログラムに追加する追加部を有する。データストア制御装置は、前記追加された第２属性に対応する定義情報として、前記参照先を特定する特定情報を前記プログラムに定義する定義部を有する。データストア制御装置は、前記プログラムに定義される前記第１属性に対応した定義情報を、前記特定情報を用いて前記第２のデータストアから取得する処理の実行を示す情報に変更する変更部を有する。
【発明の効果】
【００１２】
データストアの保守作業を効率化できる。
【図面の簡単な説明】
【００１３】
【図１】図１は、実施例１に係るアプリケーションサーバを含むシステムの全体構成を示す図である。
【図２】図２は、テーブルの参照関係を示す図である。
【図３】図３は、実施例２に係るアプリケーションサーバの構成を示すブロック図である。
【図４】図４は、設定ファイルテーブルに記憶される設定ファイルの例を示す図である。
【図５】図５は、識別属性およびリレーションが共に設定されている例を示す図である。
【図６】図６は、識別属性には値が設定されているがリレーションが設定されていない例を示す図である。
【図７】図７は、識別属性には値が設定されていないがリレーションが設定されている例を示す図である。
【図８】図８は、識別属性およびリレーションが共に設定されていない例を示す図である。
【図９】図９は、データアクセス部が実行するプログラムの例を示す図である。
【図１０】図１０は、図９に示したトランザクションインスタンスの取得の実装例を示す

図である。
【図１１】図１１は、変換前のEmployeeクラスの例を示す図である。
【図１２】図１２は、リレーションを永続化対象から除外して識別属性を記述した例を示す図である。
【図１３】図１３は、変換後のEmployeeクラスの例を示す図である。
【図１４】図１４は、アプリケーション変換処理の流れを示すフローチャートである。
【図１５】図１５は、クラス変換後のプログラムの実行によって作成されるテーブルの例を示す図である。
【図１６】図１６は、データストア制御プログラムを実行するコンピュータのハードウェア構成例を示す図である。
【図１７】図１７は、Java（登録商標）によるテーブル作成を説明する図である。
【図１８】図１８は、参照関係を有するテーブル作成例を示す図である。
【発明を実施するための形態】
【００１４】
以下に、本願の開示するデータストア制御装置、データストア制御プログラムおよびデータストア制御方法の実施例を図面に基づいて詳細に説明する。なお、この実施例によりこの発明が限定されるものではない。
【実施例１】
【００１５】
図１は、実施例１に係るアプリケーションサーバを含むシステムの全体構成を示す図である。図１に示すように、このシステムは、データストアマシン１およびデータストアマシン２各々とアプリケーションサーバ３とがネットワークを介して接続される。このシステムは、アプリケーションサーバ３がデータストアマシン１またはデータストアマシン２に記憶されるデータを用いて各処理を実行するクラウドコンピューティングを実現する。
【００１６】
データストアマシン１は、RDB（Relational Database）を用いてデータを記憶するサーバである。データストアマシン２は、KVS（Key-Value Store）を用いてデータを記憶するサーバである。
【００１７】
アプリケーションサーバ３は、除外部３ａ、追加部３ｂ、定義部３ｃ、変更部３ｄを有し、データストアマシン１またはデータストアマシン２に対して、テーブル作成、データ登録、データ更新、データ削除等を実行するサーバである。
【００１８】
例えば、アプリケーションサーバ３がJPA（Java（登録商標） Persistence Application Programming Interface）を用いて、データストアマシン１とデータストアマシン２とが参照関係にあるテーブルを作成する例について説明する。
【００１９】
ここで、はじめに、データストアマシン１とデータストアマシン２にあるテーブルが参照関係にある例を説明する。図２は、テーブルの参照関係を示す図である。図２に示すように、RDBは、「社員ID、役職、名前、部署ID」から形成されるレコードを有する社員テーブルを有し、KVSは、「部署ID、部署名」から形成されるレコードを有する社員テーブルを有する。このような状況において、RDBの社員テーブルとKVSの部署テーブルは「部署ID」で関連付けられており、社員テーブルから関連する部署テーブルのレコードを取得することが出来る。このような関係を参照関係にあるという。
【００２０】
次に、アプリケーションサーバ３がJPAを用いて、データストアマシン１にデータストアマシン２を参照するテーブルを生成する第１プログラムを実行したとする。この場合、除外部３ａは、データストアマシン１に作成するテーブルに関して定義した永続化対象のうち、データストアマシン１のテーブルが他のテーブルを参照することを示す第１属性をプログラムの永続化対象から除外する。続いて、追加部３ｂは、データストア１のテーブ

ルからデータストア２のテーブルを識別する第２属性を永続化対象としてプログラムに追加する。そして、定義部３ｃは、追加された第２属性に対応する定義情報として参照先を特定する特定情報をプログラムに含めるようにする。さらに、変更部３ｄは、プログラムに定義される第１属性に対応した定義情報を、特定情報を用いてデータストアマシン２から取得する処理の実行を示す情報に変更する。

【００２１】

このように、アプリケーションサーバ３は、外部キーが定義されたプログラムが実行された場合でも、外部キーを使わずに、管理方式が異なるデータベース間が結びつくように、プログラム内容を書き換えることができる。したがって、データストアマシン１やデータストアマシン２のデータ配置を変更した場合でも、プログラムを修正しなくて済むので、管理方式が異なるデータストアに跨ってデータが登録されている状況でも、データストアの保守作業を効率的に実行できる。

【実施例２】

【００２２】

次に、実施例２に係るアプリケーションサーバについて説明する。ここでは、アプリケーションサーバの構成、具体例、処理の流れ、実施例２による効果を説明する。なお、システム構成は図１と同様とするので、システム構成およびアプリケーションサーバ以外は省略する。また、実施例２では、JPAを用いたプログラムで記述されるアプリケーションを例にして説明する。

【００２３】

［アプリケーションサーバの構成］

図３は、実施例２に係るアプリケーションサーバの構成を示すブロック図である。図３に示すように、アプリケーションサーバ１０は、変換前アプリテーブル１１、変換後アプリテーブル１２、設定ファイルテーブル１３、変換前実行部１４、永続化クラス変換部１５、変換後実行部１６、データアクセス部１７、データアクセッサ１８を有する。

【００２４】

なお、変換前アプリテーブル１１、変換後アプリテーブル１２、設定ファイルテーブル１３は、半導体メモリ素子やハードディスクなどに設けられる。また、上記テーブル以外の各制御部は、CPU（Central Processing Unit）などによって実行される。

【００２５】

変換前アプリテーブル１１は、開発者が生成したアプリケーションを記憶する。例えば、変換前アプリテーブル１１は、RDBでデータを管理するデータストアマシン１に対して、テーブル作成、データ登録、データ更新、データ削除等の処理を実行するアプリケーションを記憶する。また、変換前アプリテーブル１１は、KVSでデータを管理するデータストアマシン２に対して、テーブル作成、データ登録、データ更新、データ削除等の処理を実行するアプリケーションを記憶する。

【００２６】

別例としては、変換前アプリテーブル１１は、データストアマシン１に対するデータ処理であって、データストアマシン２のテーブルを参照する外部キーが設定されたアプリケーションを記憶する。同様に、変換前アプリテーブル１１は、データストアマシン２に対するデータ処理であって、データストアマシン１のテーブルを参照する外部キーが設定されたアプリケーソンを記憶する。なお、ここで記憶されるアプリケーションは、アプリケーション開発者やアプリケーションサーバ１０の管理者等によって格納される。変換前のアプリケーション１１は、変換前アプリケーションに対応付けて、生成された日時や更新回数などを記憶することもできる。

【００２７】

変換後アプリテーブル１２は、永続化クラス変換部１５によって変換されたアプリケーションを記憶する。なお、ここで記憶されるアプリケーションは、後述する永続化クラス変換部１５の変更部１５ｄによって格納および更新される。また、変換後アプリテーブル１２は、変換後のアプリケーションに対応付けて、変換された日時や変換回数などを記憶

することもできる。

【００２８】

設定ファイルテーブル１３は、データストアごとに配置されるクラスの設定ファイルを記憶する。図４は、設定ファイルテーブル１３に記憶される設定ファイルの例を示す図である。図４に示すように、データストアとして、URL（Uniform Resource Locator）が「http//localdomain/RDB」であり、「RDB」で管理するデータストア名「RDB4Update」が設定されている。そして、このデータストアのクラスとして「Entity Class＝Department」が設定されている。同様に、データストアとして、URLが「http//localdomain/KVS」であり、「KVS」で管理するデータストア名「KVS」が設定されている。そして、このデータストアのクラスとして「Entity Class＝Employee」が設定されている。つまり、図４では、DepartmentクラスがRDBに配置され、EmployeeクラスがKVSに配置されることを、マップで管理することが示されている。

【００２９】

変換前実行部１４は、管理者等の指示操作を受け付けると、指定されたアプリケーションを変換前アプリテーブル１１から読み出して実行する。なお、変換前実行部１４は、マウスなどの入力装置やタッチパネルなどの出力装置等から指示操作を受け付けることができ、さらに、ネットワークを介して他のサーバから指示操作を受け付けることもできる。実施例２では、変換前実行部１４が、データストアマシン１に対するデータ処理であって、データストアマシン２のテーブルを参照する外部キーが設定されたアプリケーションＡを実行したとする。

【００３０】

永続化クラス変換部１５は、除外部１５ａ、追加部１５ｂ、定義部１５ｃ、変更部１５ｄを有し、これらによって、変換前実行部１４によって実行されたアプリケーションを変換して、変換後アプリテーブル１２に格納する。

【００３１】

除外部１５ａは、永続化クラスのクラス間の参照関係を表すリレーション属性を永続化対象から除外する。例えば、除外部１５ａは、データストアマシン１にテーブルを作成するアプリケーションＡにおいて、アプリケーションＡがデータストアマシン１と管理方式が異なるデータストアマシン２を参照することを示すリレーション属性を永続化対象から除外する。

【００３２】

つまり、除外部１５ａは、アプリケーションＡ内のクラス１においてリレーションを表すアノテーションで記述された参照先オブジェクトをリレーション属性として特定し、特定したオブジェクトを永続化対象から除外する。言い換えると、除外部１５ａは、リレーション属性をデータストアマシン１の管理対象から除外する。

【００３３】

追加部１５ｂは、クラス１の参照先を識別する識別属性を永続化対象としてアプリケーションＡのクラス１に追加する。上記例を用いて説明すると、追加部１５ｂは、除外部１５ａによってリレーションが管理対象から除外されたアプリケーションＡに対して、除外されたリレーションによって指定されていた参照先を識別する識別属性をアプリケーションＡに新たに追加する。

【００３４】

定義部１５ｃは、追加された識別属性に対応する定義情報として、参照先を特定する特定情報をアプリケーションＡに定義する。上記例を用いて説明すると、定義部１５ｃは、識別属性が新たに記述されたアプリケーションＡに対して、識別属性のsetterメソッドとgetterメソッドを新たに記述する。

【００３５】

変更部１５ｄは、アプリケーションＡのクラス１に定義されているリレーション属性の定義内容を、定義部１５ｃによって定義されたプライマリキーを用いてデータストアマシン２から参照先のオブジェクトを取得する処理に変更する。上記例を用いて説明すると、

変更部１５ｄは、識別属性が新たに記述されたアプリケーションＡに対して、外部キーとして設定されていたリレーション属性に対して定義されるsetterメソッドとgetterメソッドを修正する。そして、変更部１５ｄは、変換した変換後のアプリケーションＡを変換後アプリテーブル１２に格納する。

【００３６】

ここで、定義部１５ｃおよび変更部１５ｄは、図５～図８に示したいずれの条件であっても整合性が取れるように、setterメソッドとgetterメソッドの定義または修正を実行する。図５は、識別属性およびリレーションが共に設定されている例を示す図である。図６は、識別属性には値が設定されているがリレーションが設定されていない例を示す図である。図７は、識別属性には値が設定されていないがリレーションが設定されている例を示す図である。図８は、識別属性およびリレーション共に設定されていない例を示す図である。

【００３７】

具体的には、図５は、データストアマシン１のEmployeeテーブルを定義するアプリケーションＡに、「社員ID＝003、役職＝一般社員、名前＝山田、識別属性（dep__id）＝B001」が設定されている。さらに、Employeeテーブルの参照先であるデータストアマシン２のDepartmentテーブルを定義するアプリケーションＢに、「部署ID＝B001、部署名＝購買」が設定されている状況である。

【００３８】

図６は、データストアマシン１のEmployeeテーブルを定義するアプリケーションＡに、「社員ID＝003、役職＝一般社員、名前＝山田、識別属性（dep__id）＝B001」が設定されている。一方、データストアマシン２のDepartmentテーブルを定義するアプリケーションＢには何も設定されていない状況である。

【００３９】

図７は、データストアマシン１のEmployeeテーブルを定義するアプリケーションＡに、「社員ID＝003、役職＝一般社員、名前＝山田、識別属性（dep__id）＝NULL」が設定されている。さらに、Employeeテーブルの参照先であるデータストアマシン２のDepartmentテーブルを定義するアプリケーションＢに、「部署ID＝B001、部署名＝購買」が設定されている状況である。

【００４０】

図８は、データストアマシン１のEmployeeテーブルを定義するアプリケーションＡに、「社員ID＝003、役職＝一般社員、名前＝山田、識別属性（dep__id）＝NULL」が設定されている。さらに、データストアマシン２のDepartmentテーブルを定義するアプリケーションＢに、「部署ID＝B001、部署名＝購買」が設定されているが、EmployeeテーブルとDepartmentテーブルとは参照関係にない状況である。

【００４１】

したがって、定義部１５ｃおよび変更部１５ｄは、アプリケーションＡの変換前と変換後とでテーブル構成等が変わらないように、様々な状況に対応可能なsetterメソッドとgetterメソッドを生成する。なお、具体例を挙げた説明は後述するので、ここでは省略する。

【００４２】

図３に戻り、変換後実行部１６は、変換後アプリテーブル１２から変換後のアプリケーションを読み出して実行する。変換後実行部１６は、変換後のアプリケーションＡを実行した場合、データストアマシン１に対するテーブル作成要求をデータアクセス部１７に出力する。

【００４３】

データアクセス部１７は、設定ファイルテーブル１３に記憶されるクラスごとの設定ファイルに基づいて、クラスごとにアクセスするデータストアを定義するMetaEntityManagerを生成する。例えば、データアクセス部１７は、永続化クラスと配置先データストアに対応するアクセッサをマップで保持する。続いて、データアクセス部１７は、データスト

アと配置されるクラスの一覧ファイルとを設定ファイルテーブル１３から取り込み、マップ内に、クラスをキーにして配置先データストアのデータアクセッサであるEntityManagerを設定する。そして、データアクセス部１７は、作成部１７ａ、更新部１７ｂ、参照部１７ｃ、トランザクション管理部１７ｄを有し、これらと図９に示したプログラムとにしたがって処理を実行する。図９は、データアクセス部が実行するプログラムの例を示す図である。

【００４４】

作成部１７ａは、変換後実行部１６によって実行されたアプリケーションからCreateメッセージを受信した場合、当該アプリケーションのクラスに対応するデータアクセッサを特定する。そして、作成部１７ａは、特定したデータアクセッサを実行するデータアクセッサ１８にCreateメッセージを送信する。例えば、作成部１７ａは、データストアマシン１へのCreateメッセージを受信した場合に、rdb__EntityManagerにCreateメッセージを送信する。また、作成部１７ａは、データストアマシン２へのCreateメッセージを受信した場合に、kvs__EntityManagerにCreateメッセージを送信する。

【００４５】

更新部１７ｂは、JPAのpersist関数を複数データストアに対応付けたものであり、図９に示した更新用メソッドを実行する。例えば、更新部１７ｂは、変換後実行部１６によって実行されたアプリケーションがpersist関数を実行した場合、当該persist関数の実行先アプリケーションのクラスに対応するデータアクセッサを特定する。そして、更新部１７ｂは、特定したデータアクセッサを実行するデータアクセッサ１８に処理実行依頼を送信する。

【００４６】

一例を挙げると、更新部１７ｂは、persist関数におけるEntityの配置先がデータストアマシン１である場合に、rdb__EntityManagerに更新処理の実行を指示する。また、更新部１７ｂは、persist関数におけるEntityの配置先がデータストアマシン２である場合に、kvs__EntityManagerに更新処理の実行を指示する。

【００４７】

参照部１７ｃは、JPAのfind関数を複数データストアに対応付けたものであり、図９に示した参照用メソッドを実行する。例えば、参照部１７ｃは、変換後実行部１６によって実行されたアプリケーションがfind関数を実行した場合、当該find関数の実行先アプリケーションのクラスに対応するデータアクセッサを特定する。そして、参照部１７ｃは、特定したデータアクセッサを実行するデータアクセッサ１８に処理実行依頼を送信する。

【００４８】

一例を挙げると、参照部１７ｃは、find関数におけるクラスの配置先がデータストアマシン１である場合に、rdb__EntityManagerに参照処理の実行を指示する。また、参照部１７ｃは、find関数におけるクラスの配置先がデータストアマシン２である場合に、kvs__EntityManagerに参照処理の実行を指示する。

【００４９】

トランザクション管理部１７ｄは、JPAにおけるgetTransactionを複数データストアに対応付けたものであり、図９に示したトランザクションインスタンスの取得を実行する。このトランザクション管理部１７ｄは、図１０に例示した複数のトランザクションを管理するMetaTransactionクラスを実装し、各データストアマシンへのトランザクションを同時に実行して終了させるように、各トランザクションを管理する。図１０は、図９に示したトランザクションインスタンスの取得の実装例を示す図である。

【００５０】

データアクセッサ１８は、各データストアマシンに対応したアクセッサを保持し、データアクセス部１７から入力された各種処理を対応するアクセッサに振り分ける。そして、データアクセッサ１８は、各アクセッサを用いてデータストアに処理を実行する。例えば、データアクセッサ１８は、データストアマシン１へのテーブル作成処理、言い換えると、Employeeクラスの実行要求を受信した場合に、データストアマシン１に対応したアクセ

ッサを用いて、テーブル作成処理をデータストアマシン１に実行する。同様に、データアクセッサ１８は、データストアマシン２へのテーブル作成処理、言い換えると、Departmentクラスの実行要求を受信した場合に、データストアマシン２に対応したアクセッサを用いて、テーブル作成処理をデータストアマシン２に実行する。

【００５１】

［具体例］

次に、図１１～図１３を用いて、プログラム変換の具体例を説明する。図１１は、変換前のEmployeeクラスの例を示す図であり、図１２は、リレーションを永続化対象から除外して識別属性を記述した例を示す図であり、図１３は、変換後のEmployeeクラスの例を示す図である。ここでは、JPAで記述されるプログラムを例にして説明する。

【００５２】

変換前実行部１４は、図１１に示したプログラムを有するアプリケーションを実行する。すると、除外部１５ａは、図１１の（Ａ）に示したアノテーション「@OneToOne」に記述される「private Department dep;」を、図１２に示すように「private transient Department dep;」に変更する。この結果、「Department dep」が永続化対象から除外され、各データストアマシンの管理対象外となる。

【００５３】

追加部１５ｂは、図１２に示すように、図１１の状態では記述されていなかった「private String dep__id」をプログラム内に新たに記述する。この結果、識別属性としてString型の「dep__id」が永続化対象として新たに定義される。

【００５４】

そして、定義部１５ｃは、図１３の（Ｂ）に示すように、識別属性「dep__id」のgetterメソッドをプログラム内に新たに定義する。具体的には、定義部１５ｃは、Employeeクラス内のリレーション属性「this.dep」に値が設定されていれば、リレーション先の識別属性の値を返し、クラス内のリレーション属性「this.dep」に値が設定されていなければ、識別属性の値を返す。つまり、定義部１５ｃは、リレーション先であるDepatmentクラスに値が設定されている場合には、Depatmentクラスで設定されている識別属性の値、言い換えると、参照先のオブジェクトのプライマリキーを返す。また、定義部１５ｃは、リレーション先であるDepatmentクラスに値が設定されている場合には、Employeeクラスの識別属性の値を返す。

【００５５】

また、定義部１５ｃは、図１３の（Ｃ）に示すように、識別属性「dep__id」のsetterメソッドをプログラム内に新たに定義する。具体的には、定義部１５ｃは、識別属性「dep__id」に「this.dep__id」の値を設定する。

【００５６】

続いて、変更部１５ｄは、図１３の（Ｄ）に示すように、Employeeクラス内に既に定義されているリレーション属性「dep」のgettrメソッドを変更する。具体的には、変更部１５ｄは、Employeeクラス内のリレーション属性「this.dep」にnullが設定され、かつ、Employeeクラス内の識別属性「this.dep__id」にnullが設定されていなければ、識別属性を用いてデータストアからオブジェクトを取得して応答する。また、変更部１５ｄは、Employeeクラス内のリレーション属性「this.dep」にnullが設定されていない、または、Employeeクラス内の識別属性「this.dep__id」にnullが設定されている場合、リレーション属性「dep」の値を応答する。

【００５７】

つまり、変更部１５ｄは、クラス内のリレーション属性がnullで、かつ、クラス内の識別属性がnullでなければ、データアクセス部１７やデータストアクセッサ１８を介して、参照先のオブジェクトを取得するように、getterメソッドを変更する。

【００５８】

また、変更部１５ｄは、図１３の（Ｅ）に示すように、Employeeクラス内に既に定義されているリレーション属性「dep」のsettrメソッドを変更する。具体的には、変更部１５

ｄは、Employeeクラス内のリレーション属性「this.dep」にリレーション属性「dep」の値を設定する。そして、変更部１５ｄは、設定された値がnullであれば、識別属性「dep__id」にnullを設定する。また、変更部１５ｄは、設定された値がnullでなければ、リレーション属性のIDを識別属性「dep__id」に設定する。

【００５９】

つまり、変更部１５ｄは、リレーション属性に参照先のオブジェクトを設定した結果、nullでなければ、識別属性に参照先のオブジェクトのプライマリキーを設定し、nullであれば、識別属性にnullを設定するように、setterメソッドを変更する

【００６０】

このようにして変換された変換後のアプリケーションは、変換後実行部１６によって実行される。すると、データアクセス部１７の作成部１７ａは、実行された変換後のアプリケーションを解析して、データストアマシン１へのテーブル作成処理をデータアクセッサ１８に出力する。データアクセッサ１８は、データストアマシン１に対応したデータアクセッサを介して、データストマシン１にテーブル作成を実行する。

【００６１】

［処理の流れ］

次に、アプリケーションプログラムの変換処理の流れを説明する。図１４は、アプリケーション変換処理の流れを示すフローチャートである。

【００６２】

図１４に示すように、変換前実行部１４が変換前のアプリケーションを実行すると（Ｓ１０１肯定）、追加部１５ｂは、実行された変換前のアプリケーションのリレーションを切断し、当該変換前のアプリケーションに識別属性を追加する（Ｓ１０２）。

【００６３】

続いて、定義部１５ｃは、変換前のアプリケーションに追加した識別属性に対して、図５～図８に示したどの条件でも整合性が取れるように、getterメソッドおよびsetterメソッドを定義した定義内容を変換前のアプリケーションに追加する（Ｓ１０３）。さらに、変更部１５ｄは、変換前のアプリケーションに既に記述されているリレーション属性のgetterメソッドおよびsetterメソッドを、図５～図８に示したどの条件でも整合性が取れるように修正する（Ｓ１０４）。

【００６４】

ここまでの処理を実行することで、アプリケーションサーバ１０は、図１１に示した変換前のEmployeeクラスを図１３に示した変換後のEmployeeクラスに変換できる。

【００６５】

その後、データアクセス部１７は、設定ファイルテーブル１３に記憶されるクラスごとの設定ファイルに基づいて、クラスごとにアクセスするデータストアを定義するMetaEntityManagerを生成する（Ｓ１０５）。このとき、データアクセス部１７は、トランザクション管理部１７ｄが実行するトランザクションインスタンスも生成する。この処理によって、アプリケーションサーバ１０は、図４に示した設定ファイル等を基に、図９に示したMetaEntityManagerおよび図１０に示したトランザクションインスタンスを生成する。

【００６６】

そして、変換後実行部１６は、Ｓ１０１～Ｓ１０４の処理によって変換された変換後のアプリケーションを実行する（Ｓ１０６）。このとき、データアクセッサ１８は、各データストアマシンに対応したアクセッサに、データアクセス部１７から入力された各種処理を振り分け、各アクセッサを用いてデータストアに処理を実行する。また、トランザクション管理部１７ｄは、MetaTransactionクラスに基づいて、実行されたトランザクションが同時に実行されて同時に終了するように制御する。

【００６７】

［実施例２による効果］

実施例２によれば、データストアを跨ぐ参照関係があってもデータ登録が可能になり、ソースコードを修正することなくデータの配置を変更可能になる。各データストアにおけ

るテーブル作成時にリレーションを外部キー（Foreign key）として設定しなくなり、テーブル作成が可能になる。

【００６８】

図１５は、クラス変換後のプログラムの実行によって作成されるテーブルの例を示す図である。図１５の左図に示すように、アプリケーションサーバ１０は、Departmentクラスへのリレーションを有するEmployeeクラスが実行された場合に、Employeeクラスが有するリレーションを永続化対象外として、新たな属性である識別属性「dep__id」を定義する。

【００６９】

アプリケーションサーバ１０は、図１５の左図にように変換したクラスを実行することで、データストアマシン１に「社員ID、役職、名前、dep__id」の社員テーブルを生成する。このとき、アプリケーションサーバ１０は、社員テーブルの「dep__id」に、別のクラスによってデータストアマシン２に生成された部署テーブル「部署ID、部署名」の部署IDを取得して格納する。この結果、アプリケーションサーバ１０は、社員テーブルが部署テーブルを参照する関係にある場合でも、社員テーブルに外部キーを設定することなく、参照関係を維持することができる。

【実施例３】

【００７０】

さて、これまで本発明の実施例について説明したが、本発明は上述した実施例以外にも、種々の異なる形態にて実施されてよいものである。そこで、以下に異なる実施例を説明する。

【００７１】

（データストアの種別）

実施例１や２では、RDBで管理される社員テーブルがKVSで管理される部署テーブルを参照する例について説明したが、これに限定されるものではない。例えば、対象となるDBの種別や参照関係は任意に設定変更できる。また、テーブルの名称等もあくまで例示であり、これに限定されるものではない。

【００７２】

（処理種別）

実施例１や２では、テーブル作成を実行するアプリケーションを例にして説明したが、これに限定されるものではない。例えば、テーブルにデータを登録するアプリケーション、テーブルのデータを更新するアプリケーション、テーブルからデータを削除するアプリケーションなどについても、実施例１や２と同様の処理を実行することができる。また、データアクセス部１７は、任意によって実行させないように設定し、予め定めたデータアクセス部を用いることもできる。

【００７３】

（プログラム言語）

実施例２等では、Java（登録商標）においてJPAを利用するプログラムを例にして説明したが、これに限定されるものではなく、Ｃ言語など任意にプログラムを用いることができる。その場合でも、実施例２等で説明した処理と同様の処理を実行することで、実施例２等と同様の効果を得ることができる。

【００７４】

（変換例）

実施例２等では、実行されたアプリケーションＡに対して、識別属性や識別属性のsetterメソッド等を追加し、元々あるリレーション属性のsetterメソッドを変更することで、変換後にアプリケーションを生成する例を説明したが、これに限定されるものではない。例えば、変換前のアプリケーションＡを複製した複製後のアプリケーションＡに対して実施例２等の変換処理を実行してもよく、実施例２等の変換処理によって別ファイルのアプリケーションを生成するようにしてもよい。

【００７５】

（システム）
また、本実施例において説明した各処理のうち、自動的におこなわれるものとして説明した処理の全部または一部を手動的におこなうこともできる。あるいは、手動的におこなわれるものとして説明した処理の全部または一部を公知の方法で自動的におこなうこともできる。この他、上記文書中や図面中で示した処理手順、制御手順、具体的名称、各種のデータやパラメータを含む情報については、特記する場合を除いて任意に変更することができる。

【００７６】
また、図示した各装置の各構成要素は機能概念的なものであり、必ずしも物理的に図示の如く構成されていることを要しない。すなわち、例えば追加部１５ｂと定義部１５ｃとを統合するなど各装置の分散・統合の具体的形態は図示のものに限られない。つまり、その全部または一部を、各種の負荷や使用状況などに応じて、任意の単位で機能的または物理的に分散・統合して構成することができる。さらに、各装置にて行なわれる各処理機能は、その全部または任意の一部が、ＣＰＵ（Central Processing Unit）および当該ＣＰＵにて解析実行されるプログラムにて実現され、あるいは、ワイヤードロジックによるハードウェアとして実現され得る。

【００７７】
（プログラム）
ところで、上記の実施例で説明した各種の処理は、あらかじめ用意されたプログラムをパーソナルコンピュータやワークステーションなどのコンピュータシステムで実行することによって実現することができる。そこで、以下では、上記の実施例と同様の機能を有するプログラムを実行するコンピュータの一例を説明する。

【００７８】
図１６は、データストア制御プログラムを実行するコンピュータのハードウェア構成例を示す図である。図１６に示すように、コンピュータシステム１００は、バス１０１に、ＣＰＵ１０２、入力装置１０３、出力装置１０４、通信インタフェース１０５、ＨＤＤ（Hard Disk Drive）１０６、ＲＡＭ（Random Access Memory）１０７が接続される。

【００７９】
入力装置１０３は、マウスやキーボードであり、出力装置１０４は、ディスプレイなどであり、通信インタフェース１０５は、ＮＩＣ（Network Interface Card）などのインタフェースである。ＨＤＤ１０６には、データストア制御プログラム１０６ａとともに、図３に示した各テーブルに記憶される情報が記憶される。記録媒体の例としてＲＯＭ１０４を例に挙げたが、ＨＤＤ、ＲＡＭ、ＣＤ－ＲＯＭ等の他のコンピュータ読み取り可能な記録媒体に各種プログラムを格納しておき、コンピュータに読み取らせることとしてもよい。なお、記憶媒体を遠隔地に配置し、コンピュータが、その記憶媒体にアクセスすることでプログラムを取得して利用してもよい。また、その際、取得したプログラムをそのコンピュータ自身の記録媒体に格納して用いてもよい。

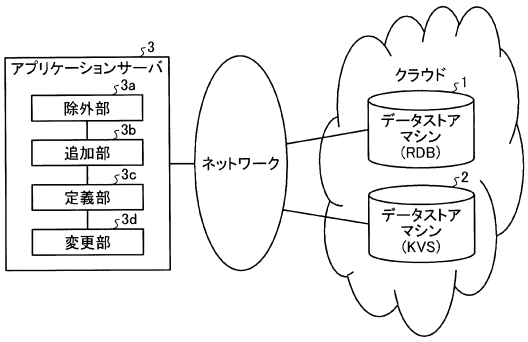
【００８０】
ＣＰＵ１０２は、データストア制御プログラム１０６ａを読み出してＲＡＭ１０７に展開することで、図１や図３で説明した各機能を実行するデータストア制御プロセス１０７ａを動作させる。すなわち、データストア制御プロセス１０７ａは、除外部３ａ、追加部３ｂ、定義部３ｃ、変更部３ｄと同様の機能を実行する。また、データストア制御プロセス１０７ａは、永続化クラス変換部１５、データアクセス部１７と同様の機能も実行することもできる。このようにコンピュータシステム１００は、プログラムを読み出して実行することでアドレス学習方法を実行する情報処理装置として動作する。

【符号の説明】
【００８１】
１　データストア（ＲＤＢ）
２　データストア（ＫＶＳ）
３、１０　アプリケーションサーバ

３ａ、１５ａ　除外部
３ｂ、１５ｂ　追加部
３ｃ、１５ｃ　定義部
３ｄ、１５ｄ　変更部
１１　変換前アプリテーブル
１２　変換後アプリテーブル
１３　設定ファイルテーブル
１４　変換前実行部
１５　永続化クラス変換部
１６　変換後実行部
１７　データアクセス部
１７ａ　作成部
１７ｂ　更新部
１７ｃ　参照部
１７ｄ　トランザクション管理部
１８　データアクセッサ

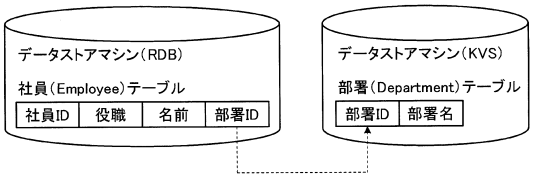
【図１】

実施例1に係るアプリケーションサーバを含むシステムの全体構成を示す図



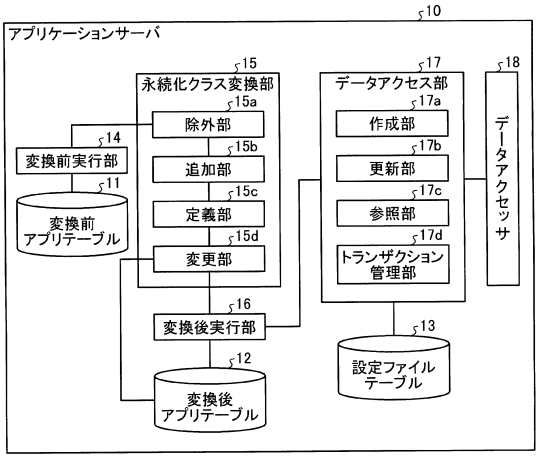
【図２】

テーブルの参照関係を示す図



【図３】

実施例2に係るアプリケーションサーバの構成を示すブロック図



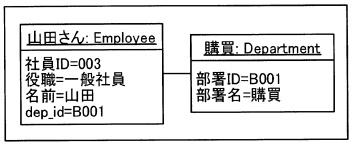
【図４】

設定ファイルテーブルに記憶される設定ファイルの例を示す図

```
<datastores>
 <datastore type="RDB" name="RDB4Update"
          URL="http://localdomain/RDB">
  <Entities>
   <Entity class="Department"/>
  </Entities>
 </datastore>
 <datastore type="KVS" name="KVS"
          URL="http://localdomain/KVS">
  <Entities>
   <Entity class="Employee"/>
  </Entities>
 </datastore>
</datastores>
```

【図５】

識別属性およびリレーションが共に設定されている例を示す図

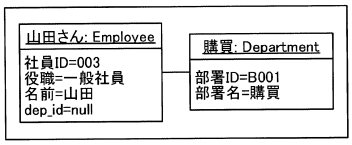


【図６】

識別属性には値が設定されているがリレーションが設定されていない例を示す図

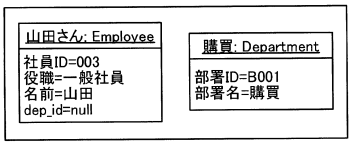
【図７】

識別属性には値が設定されていないがリレーションが設定されている例を示す図



【図８】

識別属性およびリレーションが共に設定されていない例を示す図



【図９】

データアクセス部が実行するプログラムの例を示す図

```
public MetaEntityManager implements EntityManager {

   //永続化クラスと配置先データストアに対応する
   //アクセッサ(EntityManager)をマップで保持
   Map<Class, EntityManager> mgrMap;

   public void MetaEntityManager(String path){
      //データストアと配置されるクラスの一覧ファイルを取り込み
            //mgrMapにクラスをキー配置先データストアのデータストアアク
      //セッサ(EntityManager)を設定する
   }

      //更新用メソッド
   public void persist(Object entity) {
      EntityManager em = map.get(entity.getClass());
      em.persist(entity);
   }
       //参照用メソッド
   public <T> T find(Class<T> cls, Object id) {
      final EntityManager mgr = mgrMap.get(cls);
      return mgr.find(cls, id);
        }
   //トランザクションインスタンスの取得
   public EntityTransaction getTransaction(){
     final Collection<EntityTransaction> trans =
              new ArrayList<EntityTransaction>();
     for (final EntityManager em : this.unitToEm.values())
         trans.add(em.getTransaction());
     return new MetaEntityTransaction(trans);
   }
}
```

【図１０】

図9に示したトランザクションインスタンスの取得の実装例を示す図

```
public class MetaEntityTransaction implements EntityTransaction {

    public MetaEntityTransaction(final Collection<EntityTransaction> trans) {
        this.trans = trans;
    }

    private Collection<EntityTransaction> trans;

    public void rollback() {
        for(final EntityTransaction t : this.trans)
            t.rollback();
    }

    public void commit() {
        for(final EntityTransaction t : this.trans)
            t.commit();
    }

    public void begin() {
        for(final EntityTransaction t : this.trans)
            t.begin();
        }
    }
}
```

【図１１】

変換前のEmployeeクラスの例を示す図

```
package personnel.domain;

import javax.persistence.Entity;
import javax.persistence.Id;
import javax.persistence.OneToOne;

@Entity
public class Employee {
@Id
private String id;

private String name;                                    A

@OneToOne
private Department dep;

public String getId() {
return id;
}
public void setId(String id) {
this.id = id;
}
public String getName() {
return name;
}
public void setName(String name) {
this.name = name;
}
public Department getDep() {
return dep;
}
public void setDep(Department dep) {
this.dep = dep;
}
}
```

【図１２】

リレーションを永続化対象から除外して識別属性を記述した例を示す図

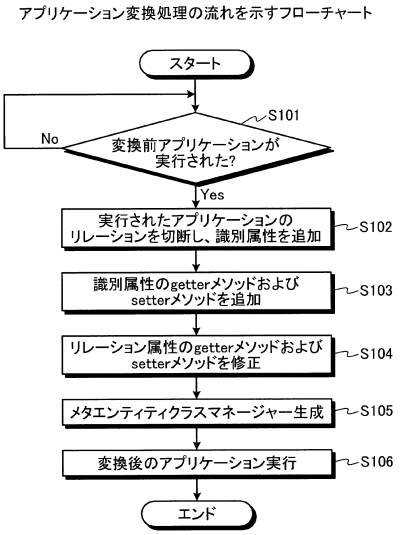
```
/**package, import**/
@Entity
public class Employee {
  @Id
  private String id;

  private String name;

  @OneToOne
  private transient Department dep;

  private String dep_id;

  /**各フィールドのsetter, getter**/
}
```

【図１３】

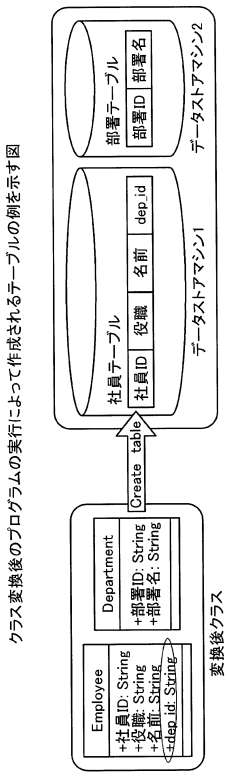
変換後Employeeクラスの例を示す図

```
/**package, import**/
@Entity
public class Employee {
  @Id
  private String id;

  private String name;

  @OneToOne
  private transient Department dep;

  private String dep_id;                                B

  public String getDep_id() {
    if(null != this.dep)
      return this.dep.getId();
    return dep_id;
  }

  public void setDep_id(String dep_id) {              C
    this.dep_id = dep_id;
  }

  public Department getDep() {                        D
    if(null == this.dep && null!=this.dep_id) {
      setDep(MetaEntityManager.getInstance().
                find(Department.class,
getDep_id()));
    }
    return dep;
  }

  public void setDep(Department dep) {                E
    this.dep = dep;
    if(null == this.dep) {
      dep_id = null;
    }
    else {
      setDep_id(dep.getId());
    }
  }

  /**各フィールドのsetter, getter**/
}
```

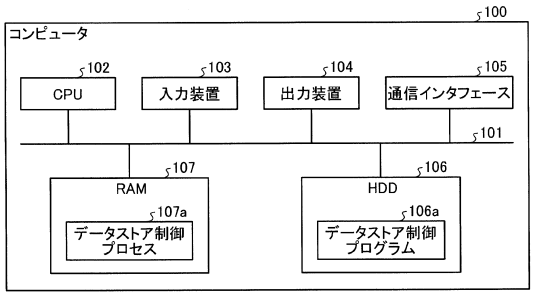
【図１４】



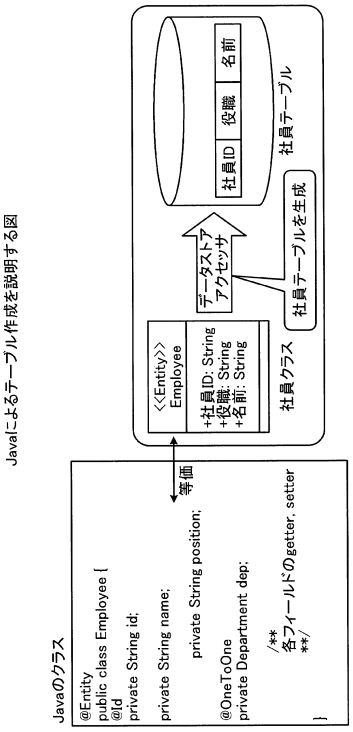
【図１５】



【図１６】

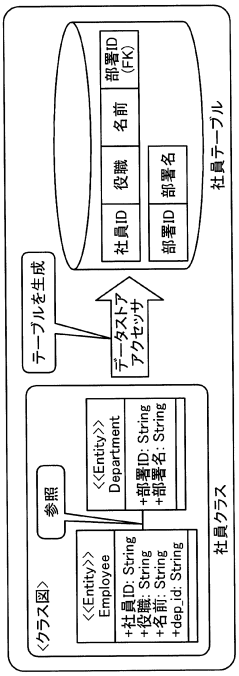


【図１７】



【図１８】

参照関係を有するテーブル作成例を示す図



フロントページの続き

(72)発明者　大嶽　智裕
神奈川県川崎市中原区上小田中４丁目１番１号　富士通株式会社内

審査官　原　秀人

(56)参考文献　特開平１０－０９１４９４（ＪＰ，Ａ）
特開２０１０－１２３０６０（ＪＰ，Ａ）
特開平０７－２９５８６８（ＪＰ，Ａ）
特表２００９－５４４０６４（ＪＰ，Ａ）
米国特許出願公開第２０１０／０２１１９３９（ＵＳ，Ａ１）
溝渕　裕司，クラウド時代のデータ自動最適配置技術の開発・評価，情報処理学会研究報告，日本，一般社団法人情報処理学会，２０１１年　1月　4日，平成２２年度　４　［ＣＤ－ＲＯＭ］，Vol.2010-SE-170 No.14

(58)調査した分野(Int.Cl.，ＤＢ名)
Ｇ０６Ｆ　　１２／００