知能と複雑系 109－3
（1997．8．30）

スキーマ変換システムTRICSにおける関係型データベースの概念構造構築について

梶山民人 伊藤秀昭 飯田三郎 福村晃夫

中京大学大学院情報科学研究科

本論文では，スキーマ変換システムTRICSにおける関係型データベースの概念構造の構築手法について述べる．データベースの概念構造は，関係で表現された概念対象と二つの概念対象間を関連付ける二項関係の集合からなる．既存の関係型データベースから概念構造を求めるために，本システムは関係スキーマと関係インスタンスを解析して概念対象と二項関係を求める．自動的に決定できない概念対象や二項関係が求まった場合にはユーザへの問い合わせを行い，概念構造を対話的に構築する．また，関係スキーマをフレームに変換して，データベースの概念構造を記述したフレーム型知識ベースを生成する．本システムは，平坦なデータ構造により定義される関係型データベースの概念構造を階層構造で表されるフレーム型知識ベースにより明示的に記述することで，データベースの概念構造を直感的に把握することを可能にし，データベースと知識システムの統合に必要なデータベースのデータ構造記述を支援する．

Constructing Conceptual Structures of Relational Databases in a Schema Transformation System TRICS

Tamito KAJIYAMA　Hideaki ITO　Saburo IIDA　Teruo FUKUMURA

Graduate School of Computer and Cognitive Sciences, Chukyo University

This paper describes mechanisms for constructing conceptual structures of relational databases in a schema transformation system, named TRICS. A conceptual structure of a relational database consists of a set of conceptual objects represented by relations, and binary relationships connecting two conceptual objects. To elicit the conceptual structure from the relational database, the system captures conceptual objects and binary relationships among them by analyzing relational schemas and their instances. When the system cannot automatically determine suitable conceptual objects and their relationships, the system tries to construct the conceptual structure interactively. The system also builds a frame-based knowledgebase which describes the captured conceptual structure. The database has a flat data structure, while the frame-based knowledgebase is organized as a hierarchical structure. So, the system improves the understandability of the database, and provides a description of the data structure of the database, which is needed for integrating the database into a knowledge system.

1　はじめに

関係型データベース [1] は様々な分野で応用されている．関係型データベースは関係の集合であり，関係は平坦なデータ構造である．関係間のリンクは属性の値を参照することにより表される．また，データベースの概念的な構造 (概念構造) は明示的な形式でデータベース中に保持されていない．そのため，データベースの概念構造を直感的に把握することは困難であり，既存の関係型データベースを利用する際には以下に示す二つの問題が生じる．

データベース管理者が交代することによって，データベースの概念構造の記述が失われる場合がある．また，データベースの設計の変更に伴ってデータベースの概念構造が混乱している場合がある．そこで，既存のデータベースを解析してデータベースの概念構造を明らかにし，混乱した概念構造を整理する必要がある．しかし，この処理をソフトウェアツールによる支援なしにマニュアルで行うことは困難である．

また，我々は関係型データベースシステムが保持するデータを知識システムで利用するために二つのシステム間のインターフェイスとして機能するシス

テム IKD を開発してきた [2]. IKD はフレームを用いて実装されている. IKD を用いて関係型データベースを知識システムに統合するためには, データベースのデータ構造の詳細な記述を知識システム側に保持しなければならない. しかし, データ構造の詳細な記述はユーザにとって負担である. さらに, 二つのシステム間で記述の一貫性を保つことが困難である. そのため, データベースのデータ構造の記述を支援するためのソフトウェアツールが要求される.

上記の問題点を解決するために, 我々は既存の関係型データベースを解析してデータベースの概念構造を再構成し, 得られた概念構造をフレームで記述するスキーマ変換システム TRICS (Transformation system of Relations Into Conceptual Structure) の開発を進めている [3, 4, 5].

既存のデータベースから概念構造を求める手法として, 与えられたデータ従属から求める手法 [6], データベースへの問合せ文から求める手法 [7, 8], 関係インスタンスから求める手法 [9] が提案されている. Chiang ら [10] は既存のデータベースの概念構造を ER モデルで記述する手法を提案している. 本システムはフレームを用いてオブジェクト指向でデータベースの概念構造を記述する.

本論文では, TRICS の設計, およびデータベースの概念構造の構成と記述について述べる.

2 設計方針

本システムの設計方針は以下の通りである.

1. データベースの概念構造をボトムアップで明らかにするために, 既存の関係インスタンスを解析することによって得られる関数従属と包含従属の二つのデータ従属 [1] を利用する. これらはデータベース設計時にトップダウンで与えられる制約条件であり, 計算によって得られたデータ従属は設計時に与えられたものと必ずしも一致しない. しかし, 計算によって得られたデータ従属は既存の関係インスタンスによって満たされているので, 本システムでは計算によって得られたデータ従属を利用する.
2. ある関係が第三正規形 [1] を満たすとき, その関係は単一の固有の概念対象を表現するとする. 第三正規形でない関係は, 第三正規形を満たす複数の関係に分割して, 一つの関係が一つの概念対象を表現するようにする.
3. 既存の関係を分割して整理するだけでなく, いくつかの関係に共通して含まれる概念対象を明示的に表現するために, 必要に応じて新しい関係を定義する.
4. 自動的に選べない複数の選択肢が存在する場合や, 新しく定義される関係に関係名を与える場合にはユーザに問い合わせを行う. ユーザは必要に応じて概念対象や二項関係を修正し, 対話的に概念構造を構築する.
5. データベースの概念構造を記述するフレーム型知識ベースを作成する. フレーム間のリンクの種類は is-a, is-part-of, is-associated-with の三種類であり, それぞれ汎化–特化関係, 全体–部分関係, 連想関係を表す.

PERSON(*SSN*, *Name*, *Address*)
EMPLOYEE(*SSN*, *Name*, *Salary*, *HiredDate*)
MANAGER(*SSN*, *Rank*, *PromotionDate*, *DeptNo*)
CUSTOMER(*SSN*, *CustID*, *Name*, *Credit*)
DEPARTMENT(*DeptNo*, *DeptName*, *Location*)
PRODUCT(*ProdID*, *Description*)
PRICE(*ProdID*, *MinPrice*, *MaxPrice*)
ORDER(*OrderID*, *OrderDate*, *ProdID*, *CustID*)
CARRIER(*CarrierID*, *Name*, *Address*)
PROJECT(*ProjName*, *DeptNo*, *Budget*)
WORK_FOR(*SSN*, *DeptNo*, *StartDate*)
CAN_PRODUCE(*DeptNo*, *ProdID*, *UnitCost*)
SHIPMENT(*PackNo*, *OrderID*, *ShipDate*, *CarrierID*)

図 1: 関係スキーマの例

3 概念構造の構築

概念構造を求める手順を示すために, 図 1 に示す関係スキーマを例として用いる. これらは, 関係型データベースの概念構造を ER モデルで記述する手法を示した文献 [10] で用いられた関係スキーマである. なお, [10] には関係インスタンスが示されていないので, 関係の候補キーが [10] に示されたものと同じになるように適当な関係インスタンスを与えた. また, 本システムの実装に用いた関係型データベース管理システム Informix [11] の制約のため, 関係名と属性名を変更した.

3.1 概念対象の抽出

データベースの概念構造を求めるために, 既存の関係から概念対象を得る.

最初に, 関係の関係インスタンスを解析して, その関係が満たすすべての関数従属を求める. X, Y をある関係の属性の集合とすると, 関数従属は $X \rightarrow Y$ の形式で表される. この関数従属は, 任意の X の値に対して唯一の Y の値が存在することを表す. 例えば, 社員とその配属先の部門を表す関係 *EMP_DEPT* があり, その関係スキーマが *EMP_DEPT*(*SSN*, *Name*, *DeptNo*, *DeptName*) であるとする. *EMP_DEPT* の関数従属を以下に示す.

$\{SSN\} \rightarrow \{Name, DeptNo, DeptName\}$
$\{DeptNo\} \rightarrow \{DeptName\}$
$\{DeptName\} \rightarrow \{DeptNo\}$

次に, 得られた関数従属の集合を調べて関係の候補キー [1] を求める. 候補キーは, 個々の関係インスタンスを識別するための最小の属性集合である. 例

$d\_CAN\_PRODUCE[ProdID] \gg d\_ORDER[ProdID]$
$d\_CAN\_PRODUCE[ProdID] = d\_PRICE[\underline{ProdID}]$
$d\_CAN\_PRODUCE[ProdID] = d\_PRODUCT[\underline{ProdID}]$
$d\_CARRIER[Address] = d\_DEPARTMENT[Location]$
$d\_CARRIER[\underline{CarrierID}] \gg d\_SHIPMENT[CarrierID]$
$d\_CUSTOMER[\underline{CustID}] \gg d\_ORDER[CustID]$
$d\_DEPARTMENT[\underline{DeptNo}] \gg d\_CAN\_PRODUCE[DeptNo]$
$d\_DEPARTMENT[\underline{DeptNo}] \gg d\_PROJECT[DeptNo]$
$d\_DEPARTMENT[\underline{DeptNo}] = d\_MANAGER[DeptNo]$
$d\_DEPARTMENT[\underline{DeptNo}] = d\_WORK\_FOR[DeptNo]$
$d\_EMPLOYEE[\underline{SSN}] \gg d\_MANAGER[SSN]$
$d\_EMPLOYEE[\underline{SSN}] = d\_WORK\_FOR[SSN]$
$d\_MANAGER[DeptNo] \gg d\_CAN\_PRODUCE[DeptNo]$
$d\_MANAGER[DeptNo] \gg d\_PROJECT[DeptNo]$
$d\_MANAGER[DeptNo] = d\_WORK\_FOR[DeptNo]$
$d\_ORDER[\underline{OrderID}] \gg d\_SHIPMENT[\underline{OrderID}]$
$d\_PERSON[Address] \gg d\_CARRIER[Address]$
$d\_PERSON[Address] \gg d\_DEPARTMENT[Location]$
$d\_PERSON[\underline{Name}] \gg d\_CUSTOMER[\underline{Name}]$
$d\_PERSON[\underline{Name}] \gg d\_EMPLOYEE[\underline{Name}]$
$d\_PERSON[\underline{SSN}] \gg d\_CUSTOMER[\underline{SSN}]$
$d\_PERSON[\underline{SSN}] \gg d\_EMPLOYEE[\underline{SSN}]$
$d\_PERSON[\underline{SSN}] \gg d\_MANAGER[\underline{SSN}]$
$d\_PERSON[\underline{SSN}] \gg d\_WORK\_FOR[SSN]$
$d\_PRICE[\underline{ProdID}] \gg d\_ORDER[ProdID]$
$d\_PRICE[\underline{ProdID}] = d\_PRODUCT[\underline{ProdID}]$
$d\_PRODUCT[\underline{ProdID}] \gg d\_ORDER[ProdID]$
$d\_PROJECT[\underline{DeptNo}] \gg d\_CAN\_PRODUCE[DeptNo]$
$d\_WORK\_FOR[\underline{DeptNo}] \gg d\_CAN\_PRODUCE[DeptNo]$
$d\_WORK\_FOR[DeptNo] \gg d\_PROJECT[\underline{DeptNo}]$
$d\_WORK\_FOR[SSN] \gg d\_MANAGER[\underline{SSN}]$

図 2: 包含従属集合の例

えば, $\{SSN\}$ は EMP_DEPT の (唯一の) 候補キーである.

次に, 一つの関係が一つの概念対象を表現するように, 得られた関数従属の集合を解析して関係を第三正規形に分割する. ある関係が満たす任意の関数従属 $X \rightarrow Y$ について, 差集合 $X - Y$ 中のどの要素もその関係のある候補キーに属するとき, 関係は第三正規形である. 例えば, EMP_DEPT の関数従属 $\{DeptNo\} \rightarrow \{DeptName\}$ について, 差集合 $\{DeptNo\} - \{DeptName\}$ の要素 $DeptNo$ は EMP_DEPT の候補キー $\{SSN\}$ に属さないので, EMP_DEPT は第三正規形ではない. そのため, EMP_DEPT は次の社員と部門の概念を表現する二つの関係に分割される.

$d\_EMP(SSN, Name, DeptNo)$

$d\_DEPT(DeptNo, DeptName)$

次に, 既存の関係から求めた概念対象を表現するために, 各概念対象に対応するビューをデータベースに定義する. ビュー名はユーザにより与えられる.

図 1 に示した 13 個の関係はすべて第三正規形であり, 各々固有の概念対象を表現する. そのため, 元の関係と同じ属性を持つ 13 個のビューがデータベースに定義される. ここでは, 元の関係名の前に 'd_' を付けた記号をビュー名として用いている.

3.2 二項関係の計算

二つの概念対象間にある二項関係を求めて概念対象集合を構造化する. 二つの概念対象 R, S の間に設定される二項関係は is-a, is-part-of, is-associated-with の三種類である. R is-a S は R が S の特殊化, あるいは S が R の一般化であることを表す. R is-part-of S は R が S の部分として参照されることを表す. また, R is-associated-with S は R が S と連想関係にあることを表す. これらの二項関係を求めるために, 関係スキーマと関係インスタンスの二つをそれぞれ解析する.

関係スキーマの解析

同じドメインを持つ属性は同じ属性名を持つと仮定し, 任意の異なる二つの関係 R, S について, R のすべての属性名の集合 $\mathbf{R}$ と S のすべての属性名の集合 $\mathbf{S}$ の間に $\mathbf{R} \subseteq \mathbf{S}$ が成り立つとき, S is-a R を設ける. これは, S が R のすべての属性を継承し, さらに S に固有の属性を持つことを表す.

図 1 の関係はそれぞれ固有の属性を持つので, この手法による関連付けは行われない.

関係インスタンスの解析

まず, 関係インスタンスを解析して, 任意の異なる二つの関係 R, S の間に成り立つすべての包含従属を求める. X, Y をそれぞれ R と S の属性の集合をすると, 包含従属は $R[X] \gg S[Y]$ と表される. ここで, $R[X]$ は関係 R の属性集合 X が取るすべての値の集合を表す ($S[Y]$ も同様). この包含従属は $S[Y]$ に現れる任意の値が $R[X]$ にも現れることを意味する. また, $R[X] \gg S[Y]$ かつ $S[Y] \gg R[X]$ であるとき, $R[X] = S[Y]$ と表す. 例えば, 図 1 に示す関係集合に対して, 図 2 に示す包含従属の集合が得られる. 属性集合の下線は, その属性集合が候補キーであることを表す.

次に, 得られた包含従属と候補キーの組合せに基づいて二項関係の種類とリンクの方向を求める. 包含従属 $R[X] \gg S[Y]$ が成り立つとき, X, Y がそれぞれ候補キーか否かによって以下に示す四つの場合がある.

(1) X, Y 共に候補キーである.

(2) X のみ候補キーである.

(3) Y のみ候補キーである.

(4) いずれも候補キーでない.

(1) の場合, 二項関係 S is-a R を設定する. 例えば, 図 2 の包含従属 $d\_EMPLOYEE[SSN] \gg d\_MANAGER[SSN]$ について, $\{SSN\}$ は二つの関

d_CAN_PRODUCE is-associated-with *d_DEPT_TABLE*
d_CAN_PRODUCE is-associated-with *d_DEPT_TABLE2*
d_CAN_PRODUCE is-associated-with *d_PROD_TABLE*
d_CARRIER is-part-of *d_SHIPMENT*
d_CARRIER is-associated-with *d_CITY*
d_CARRIER is-associated-with *d_CITY2*
d_CUSTOMER is-a *d_PERSON*
d_CUSTOMER is-part-of *d_ORDER*
d_DEPARTMENT is-part-of *d_CAN_PRODUCE*
d_DEPARTMENT is-part-of *d_MANAGER*
d_DEPARTMENT is-part-of *d_WORK_FOR*
d_DEPARTMENT is-associated-with *d_CITY*
d_DEPARTMENT is-associated-with *d_CITY2*
d_EMPLOYEE is-a *d_PERSON*
d_EMPLOYEE is-part-of *d_WORK_FOR*
d_MANAGER is-a *d_EMPLOYEE*
d_MANAGER is-a *d_PERSON*
d_MANAGER is-associated-with *d_DEPT_TABLE*
d_MANAGER is-associated-with *d_DEPT_TABLE2*
d_ORDER is-associated-with *d_PROD_TABLE*
d_PERSON is-part-of *d_WORK_FOR*
d_PERSON is-associated-with *d_CITY*
d_PRICE is-a *d_PRODUCT*
d_PRICE is-part-of *d_CAN_PRODUCE*
d_PRICE is-part-of *d_ORDER*
d_PRODUCT is-part-of *d_CAN_PRODUCE*
d_PRODUCT is-part-of *d_ORDER*
d_PROJECT is-a *d_DEPARTMENT*
d_PROJECT is-part-of *d_CAN_PRODUCE*
d_SHIPMENT is-a *d_ORDER*
d_WORK_FOR is-associated-with *d_DEPT_TABLE*

図 3: 二項関係集合

係の候補キーであるので, 二項関係 *d_MANAGER* is-a *d_EMPLOYEE* が設定される.

(2) の場合, R is-part-of S を設定する. 例えば, 図 2 の包含従属 *d_PRODUCT*[*ProdID*] ≫ *d_ORDER*[*ProdID*] について, {*ProdID*} は *d_PRODUCT* の候補キーであり, *d_ORDER* の候補キーではないので, 二項関係 *d_PRODUCT* is-part-of *d_ORDER* が設定される.

(4) の場合は, 次の二つの選択肢がある. 選択肢 (4a) は, 関係 R の属性集合 X の値の集合 $R[X]$ を R と S に共通する新しい概念対象とみなし, ビュー R' を定義する. さらに, R is-associated-with R' と S is-associated-with R' の二つの二項関係を設定する. 一方, 選択肢 (4b) は, $R[X]$ と $S[Y]$ を別個の概念対象とみなし, ビュー R', S' を定義する. さらに, R is-associated-with R' と S is-associated-with S' を設定する. 加えて, R' と S' は上記 (1) の条件を満たすので, S' is-a R' を設定する. 上記の二つの選択肢のどちらが適切かは R と S の性質に依存するので, ユーザに問い合わせる.

例えば, 図 2 の包含従属 *d_PERSON*[*Address*] ≫ *d_DEPARTMENT*[*Location*] について, {*Address*} と {*Location*} は共に候補キーではないので, 二つの選択肢の一方を選ぶようにユーザに問い合わせる. ユーザが (4a) を選んだとすると, *d_PERSON*[*Address*] から新しいビュー *d_CITY* が定義され, *d_PERSON* is-associated-with *d_CITY* と *d_DEPARTMENT* is-associated-with *d_CITY* の二つの二項関係が設定される.

(3) の場合, $R[X]$ を独立した概念対象とみなしたり, R と S を関連付けたりすることは適当ではないと考えられるため, 本システムはこのような包含従属を扱わない.

また, 包含従属 $R[X] = S[Y]$ が成り立つとき, X, Y が候補キーか否かによって上記 (1), (2), (4) の三つの場合があり, それぞれの場合について次のように二項関係の種類とリンクの方向を決める. (1) の場合は R と S の間に is-a を設ける. しかしながら, リンク方向は自動的に決められないため, ユーザに問い合わせる. (2) の場合は R is-part-of S を設定する. (4) の場合は上記 (4a) の通り, $R[X]$ から新しい概念対象 R' を定義し, R is-associated-with R' と S is-associated-with R' を設定する.

上記の方法に基づいて図 2 の包含従属集合から得られた二項関係の集合を図 3 に示す.

本システムは関係インスタンスが満たす包含従属から二項関係を求めるため, 不適切な二項関係が見つかることがある. 例えば, *d_PERSON* と *d_WORK_FOR* は包含従属 *d_PERSON*[*SSN*] ≫ *d_WORK_FOR*[*SSN*] を満たし, {*SSN*} は *d_PERSON* の候補キーであるので, 二項関係 *d_PERSON* is-part-of *d_WORK_FOR* が設定される. しかしながら, 関係 *d_PERSON* は会社で働いていない人を表す関係インスタンスを含むため, この二項関係を設けることは不適切である.

3.3 冗長な二項関係の削除

二項関係は既存の関係スキーマと関係インスタンスに基づいて計算されるので, 冗長な二項関係が得られる場合がある. 本システムは簡明な概念構造を求めるために以下の三つの冗長性削除ルールを用いて冗長な二項関係を削除する.

R1: 冗長な is-part-of の削除

三つの二項関係 R is-a S, R is-part-of T, S is-part-of T が得られた場合, 二つの is-part-of の一方は冗長なので削除する. 同様に, R is-a S, T is-part-of R, T is-part-of S が得られた場合, 二つの is-part-of の一方は冗長なので削除する. ただし, どちらの is-part-of を削除すべきかは三つの概念対象の性質に依存するので, ユーザに問い合わせて選択する.

例えば, 図 3 に示される二項関係集合は *d_PRICE* is-a *d_PRODUCT*, *d_PRODUCT* is-part-of *d_ORDER*, *d_PRICE* is-part-of *d_ORDER* を含む. ユーザの選択により冗長な *d_PRICE* is-part-of *d_ORDER* が削除される.

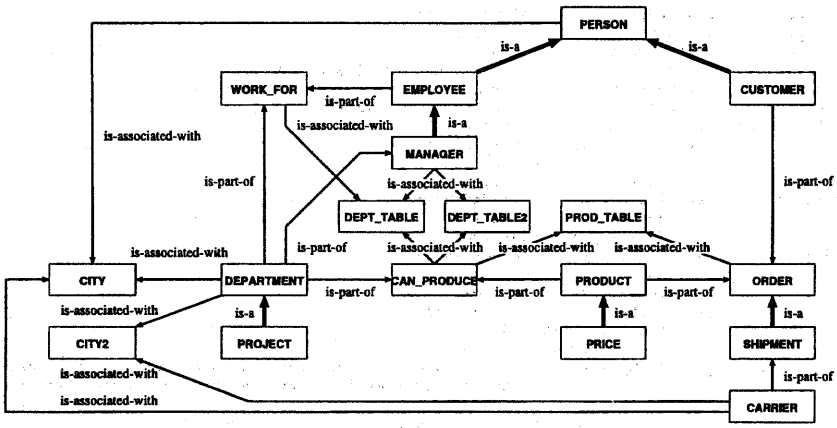

図 4: 概念構造の例

R2: 推移的な is-a の削除

三つの二項関係 R is-a S, S is-a T, R is-a T が得られた場合, 推移的な R is-a T は冗長なので削除する.

例えば, 図 3 の二項関係集合は *d_MANAGER* is-a *d_EMPLOYEE* と *d_EMPLOYEE* is-a *d_PERSON* を含むので, 冗長な *d_MANAGER* is-a *d_PERSON* は削除される.

R3: 主キーの優先による冗長な is-a の削除

二つの二項関係 R is-a S, R is-a T がそれぞれ二つの包含従属 $S[Y] \gg R[X_1], T[Z] \gg R[X_2]$ から得られたとする. このとき, 一方の is-a は冗長であるので, ユーザに問い合わせて R の二つの候補キー X_1, X_2 の一方を主キーとして選択することにより, 主キー間の is-a を優先し, 候補キー間の is-a を削除する.

例えば, 二つの候補キー $\{SSN\}$, $\{DeptNo\}$ を持つ関係 *MANAGER* があり, *MANAGER* is-a *EMPLOYEE* と *MANAGER* is-a *DEPARTMENT* がそれぞれ $EMPLOYEE[SSN] \gg MANAGER[SSN]$ と $DEPARTMENT[DeptNo] = MANAGER[DeptNo]$ から得られたとする. ユーザが $\{SSN\}$ を *MANAGER* の主キーとして選ぶことにより, *MANAGER* is-a *DEPARTMENT* は冗長な二項関係として削除される.

ルールの適用順序

本システムは, 得られている二項関係の集合に対して R1, R2, R3 の順に一つずつルールを適用して冗長な二項関係を削除する. 冗長な is-part-of を削除するために必要な条件となる is-a を is-part-of より先に削除しないように, R1 を最初に適用する. また, R2 の条件に適合するような X is-a Y と X is-a Z は R3 の条件にも適合する場合がある. 一方, R3 はユーザへの問い合わせを必要とするが, R2 では不要である. そのため, 問い合わせの回数が減るように, R2 を R3 より先に適用する.

図 3 の二項関係集合に上記のルールを適用すると, 以下のように冗長性が除かれる. まず, 次の四つの is-part-of の対が R1 の条件に合うので, ユーザに問い合わせて冗長な方を削除する. ここでは, ユーザが各対の後者を冗長なものとして選択したとする.

d_EMPLOYEE is-part-of *d_WORK_FOR*
d_PERSON is-part-of *d_WORK_FOR*

d_PRODUCT is-part-of *d_ORDER*
d_PRICE is-part-of *d_ORDER*

d_PRODUCT is-part-of *d_CAN_PRODUCE*
d_PRICE is-part-of *d_CAN_PRODUCE*

d_DEPARTMENT is-part-of *d_CAN_PRODUCE*
d_PROJECT is-part-of *d_CAN_PRODUCE*

次に, R2 により以下の推移的 is-a が削除される.

d_MANAGER is-a *d_PERSON*

最後に, R3 の適用を試みるが, 図 3 の二項関係集合には R3 の条件に適合する二項関係は含まれていないので, 二項関係は削除されない.

以上の冗長性削除により得られる概念構造を図 4 に示す.

3.4 フレーム型知識ベースの構成

関係スキーマをフレームに変換して, データベースの概念構造を記述したフレーム型知識ベースを構成する.

フレームは複数のスロットからなる. 一つの関係は一つのフレームに変換される. 関係名は relation-name スロットに記入される. 関係の属性はそれぞれフレームのスロットにより記述され, 各スロットにファセットを付加して属性名, 属性のデータ型, お

```
FrameName: frame-d_order        FrameType: class
description-info [text v m]          *undefined*
a-kind-of        [frame vs m]        {root}
created-by       [(text text) os m]  <"kajiyama" "2:19 8/6/1997">
has-a            [frame vs m]        {frame-d_product frame-d_customer}
associated-with [frame vs m]         {frame-d_prod_table}
relation-name    [string v u]        "d_order"
orderid          [attribute v u]     *undefined*
                      name* :   "orderid"
                      data-type* : cinttype
orderdate        [attribute v u]     *undefined*
                      name* :   "orderdate"
                      data-type* : cdatetype
prodid           [attribute v u]     *undefined*
                      name* :   "prodid"
                      data-type* : cinttype
custid           [attribute v u]     *undefined*
                      name* :   "custid"
                      data-type* : cinttype
view-creation-sql [string v u]       "create view d_order (orderid, orderdate, prodid, custid)
                                      as select orderid, orderdate, prodid, custid from order"
```

図 5: 概念構造 *d_ORDER* のフレームによる記述

よびデータ長 (データ型が文字型の場合のみ) を記入する. 各概念対象はビューで表現されるので, そのビューを定義するための SQL 文を view-creation-sql スロットに記入する.

得られたフレームの集合は一般化を表す階層構造で表される. 各フレームの a-kind-of スロットは二項関係集合中の is-a によって指し示される上位概念の集合を値とする. 上位概念を持たないフレームは root フレームの下位に置かれる. また, is-part-of に基づいて, 全体を表すフレームに部分を表すフレームの集合を値とする has-a スロットが付加される. 同様に, 二項関係 is-associated-with は連想関係にあるフレームの集合を値とする associated-with スロットに変換される.

例として, ビュー *d_ORDER* で表される概念対象を記述したフレームを図 5 に示す.

4 おわりに

本論文では, スキーマ変換システム TRICS における関係型データベースの概念構造の構成とフレームによる記述について述べた. 本システムは, 既存の関係インスタンスと関係スキーマからデータベースの概念構造を求める. さらに, 関係スキーマをフレームに変換し, 得られた概念構造を記述したフレーム型知識ベースを構成する. 本システムを利用することにより, データベースの概念構造を直感的に把握することが容易になり, データベースを知識システムに統合するためのデータベースのデータ構造記述の負担が軽減される.

本システムは, 関係スキーマと関係インスタンスという異なるレベルの情報に基づいて二項関係を求めるため, 同じ概念対象間に異なる種類の複数の二項関係や互いに矛盾する同じ種類の複数の二項関係を設定する場合がある. 今後の課題として, そのような冗長性や矛盾を解消するためのルールを 3.3 節で述べた三つのルールに追加することが挙げられる.

参考文献

[1] Elmasri, R. and Navathe, S. B.: *Fundamentals of Database Systems*, Addison-Wesley, second edition (1994).

[2] 伊藤秀昭, 福村晃夫: 知識ベース-データベース統合化ツール IKD の構造とその利用, 人工知能学会誌, Vol. 8, No. 1, pp. 102–113 (1993).

[3] 梶山民人, 伊藤秀昭, 福村晃夫: スキーマ変換システム TRICS における構造変換規則について, 第 11 回人工知能学会全国大会論文集, pp. 315–316 (1997).

[4] 梶山民人, 伊藤秀昭, 福村晃夫: スキーマ変換システム TRICS における概念対象と二項関係の発見, 平成 8 年度電気関係学会東海支部連合大会講演論文集, p. 281 (1996).

[5] Ito, H., Kajiyama, T. and Fukumura, T.: Discovering Conceptual Structure of Relations, in *Proceedings of the 1996 IEEE International Conference on System, Man, and Cybernetics* (1996).

[6] Johannesson, P.: A Method for Translating Relational Schemas into Conceptual Schemas, in *Proceedings of the 10th International Conference on Data Engineering*, pp. 190–201 (1994).

[7] Anderson, M.: Extracting an Entity Relationship Schema From a Relational Database through Reverse Engineering, in *Proceeding of the International Conference on the Entity-Relationship Approach*, pp. 403–419 (1994).

[8] Petit, J.-M., Toumani, F., Boulicaut, J.-F. and Kouloumdjian, J.: Towards the Reverse Engineering of Denormalized Relational Databases, in *Proceedings of the 12th International Conference on Data Engineering*, pp. 218–227 (1996).

[9] Kantola, M., Mannila, H., Räihä, K.-J. and Siirtola, H.: Discovering Functional and Inclusion Dependencies in Relational Databases, *International Journal of Intelligent Systems*, Vol. 7, No. 7, pp. 591–607 (1992).

[10] Chiang, R. H. L., Barron, T. M. and Storey, V. C.: Reverse Engineering of Relational Databases: Extraction of an EER model from a relational Database, *Data & Knowledge Engineering*, Vol. 12, pp. 107–142 (1994).

[11] アスキー: *informix-SQL User's Manual*, 第 3 版 (1988).