データベースシステム 96－6
（1993．10．29）

データベース･アシスタント構築支援ツール

田中 正邦† 川染 俊樹† 平川 正人†† 市川 忠男††

†広島大学大学院工学研究科

††広島大学工学部

あらまし

利用者のデータベース問い合わせ支援を行うデータベース･アシスタントでは、対象ドメインに依存するドメイン知識が必要である。しかし、このドメイン知識を獲得するには多大な労力と時間を要し、知識ベースが大規模化するにつれて、矛盾が生じる可能性もある。また、知識処理などの専門知識を必要とするため、初心者には記述困難である。

本稿では、グラフィカルなインタフェースを用いて、初心者でも容易に扱えるデータベース･アシスタント構築支援ツールを提案する。ユーザが定義するドメイン上の意味ネットワークから、ツールは知識ベースとデータベースを自動的に作成する。

Tools for Constructing a Portable Database Assistant

Masakuni Tanaka† Toshiki Kawasome† Masahito Hirakawa†† Tadao Ichikawa††

†Graduate School of Hiroshima University

††Faculty of Engineering, Hiroshima University

1-4-1 Kagamiyama Higashi-Hiroshima 724, Japan

Abstract

A database assistant needs knowledge which is specific to an application domain. Acquisition of the domain-specific knowledge, however, requires a considerable amount of labor and time. In addition, as a knowledge base grows in size, it becomes quite probable that improper knowledge input increases causing inconsistency in knowledge.

In this paper, we provide database assistant construction tools, which integrate assistant facilities, on a graphical interface. A Knowledge base and a database are then constructed by these tools automatically based on the description of a semantic network given by the user for a specific domain.

1 はじめに

データベースの重要性が一般に認識されるにつれ、非専門家であるユーザが直接データベースを操作する機会が増えている。しかし、データベースに関す専門知識をもたないユーザが、データベースおよびその検索言語を学習することは負担が大きい。そこで、データベース検索を容易に行わせるために、自然言語インタフェースを用いたデータベース·アシスタントに関する研究が行われている [1]。

自然言語インタフェースを用いたデータベース·アシスタントでは、多様な表現の問い合わせにもシステムが対応できる。これは、データベース検索に要求される知識のいくらかをシステム側で保持しているためである。逆に、システムは応用領域に関する情報、またそれらの相互関係の情報とデータベースの情報の関連を知識として蓄えておかなくてはならない。ところが、これらの知識は応用領域が変わる度に入れ替えを必要とするため、この獲得方法が重要となる。

TEAM [2] は、非専門家であるユーザが新しいデータベースに対する知識ベースを容易に構築できるような、メニュー形式の知識獲得ツールを提案している。CO-OP [3] は、応用領域に依存するドメイン知識を制限することにより、知識獲得に費やす労力を軽減するためのデザイン方法を提供している。System X [4] には、自然言語による問い合わせ処理の過程で、新しい単語やデータベースの値についての知識を新たに学習するための演繹的手法が組み込まれている。

本研究では、知識ベースが容易に作成可能なアシスタント機能を備えたデータベース·アシスタント構築支援ツールを提案する。また、このツールでは一からデータベース·アシスタントを構築したいときに、データベースと知識ベースを同時に作成できる環境も支援する。

データベースデザインについては、ユーザにドメインに対する自然言語の問い合わせを行ってもらい、この後問い合わせ文についてインタビュー形式でドメイン知識を獲得しながら、データベーススキーマ設計を行う ANNAPURNA [6]、I^{2}S-LD [7] などがある。

現在、我々はKDA(Knowledge_based Database Assistant) [8]-[10] の開発を行っている。提案するツールは、このKDAシステムを構築する支援ツールとして定義する。

以下、2章では提案するツール、これにより作成される知識ベース、データベースについての概要を述べる。また、3章ではシステム作成に用いる知識を示し、4章では知識ベース、5章ではデータベースの作成手順を述べる。

2 本研究の概要

2.1 システムの知識

KDAで用いられる知識ベースは、G-Net [9] と辞書から構成される。G-Netは、以下に定義されるようにユーザ概念 G_u およびデータベース概念 G_dの二層から構成される意味ネットワークモデルである。また、それらの間の変換ルールTをもつ。

$G := < C_u, C_d, L_u, L_d, R, T >$
C_u: ユーザ概念の集合
C_d: データベース概念の集合
L_u: C_uに属する概念間の関係の集合
L_d: C_dに属する概念間の関係の集合
R : L_uに属する関係、つまりリンク間の関係の集合
T : ネットワーク変換ルールの集合

1. ユーザ概念 G-Net

   ユーザ概念レベルの問い合わせを表現するユーザ概念 G-NetG_uは、

   $$G_u := < C_u, L_u, R >$$

   で表される。

2. データベース概念 G-Net

   関係スキーマの意味構造は、データベース概念 G-NetG_dを用いて表現される。これは、

   $$G_d := < C_d, L_d >$$

   で表される。

3. 検索言語SQLへのネットワーク変換知識 [9]

KDAでは、ユーザが入力する検索要求文を一旦ユーザ概念G-Netに変換する。そして、これをデータベース概念G-Netへと変換し、データベース検索言語SQLへと変換するが、この際に用いられる知識がネットワーク変換知識Tである。これは、ユーザ概念G-Netのノードとリンクを変換するためのノード変換知識ならびにリンク変換知識の二種類から成る。

2.2 アプローチ

自然言語インタフェースを用いたデータベース・アシスタントの知識ベースを構築する際には、対象ドメインに対する容易な知識ベース構築環境が求められる。このためには、ユーザが思い描くドメイン概念を素直に表現できる環境を提供することが最も適切であると考えられる。

KDAシステムでは、知識ベースデザイナー(以後ユーザ)は、データベースのスキーマを意味ネットワーク(ユーザ概念G-Net)で考えながら、それに対する知識を逐次コーディングしなければならない。しかしながら、この作業は逐次意味ネットワーク構造を思い描きながら進められるため時間と労力がかかり、煩わしい。また、入力ミスによる矛盾も生じやすくなる。

そこで提案するツールでは、ユーザがデータベースドメインに対するユーザ概念を、ノードとリンクを用いて、意味ネットワークの形式で描いていく。そして、必要に応じてユーザと対話を行いながら、システムが自動的に内部的な知識に変換していく。このため、自然言語、知識処理などの専門知識をもたないユーザでも容易に知識ベースが構築できる。

また、知識ベース構築と併せてデータベース構築が進められるように、データベース作成環境も提供する。このときも、ユーザに詳しいデータベース設計(正規形、関数従属性など)についての細かい点を意識させずにデータベースが構築できる。

このように、提案するツールは、データベースが存在するか否かによって、次の二通りの支援方法を提供する。

Type1: 既存のデータベースに対する知識ベースを構築する

ユーザはエンティティとそのインスタンス(データベースでの具体値)を入力してノードを作成する。これを受けて、4章で述べる手順で知識ベースが生成される。

Type2: これからデータベースと知識ベースを同時に構築する

まず、ユーザはデータベースで関係表にしたいエンティティから作成する。このときには、関係表に付加する属性も同時に入力する。4、5章で述べる手順でそれぞれ知識ベースとデータベースが作成される。

3 システム作成に用いる知識

3.1 準備

データベースの関係、属性の集合をそれぞれ

$$R = \{R_i | 1 \leq i \leq m\}$$

$$A = \{A_i | 1 \leq i \leq n\}$$

と表す。そして、キー属性の集合を

$$K = \{K_i | K_i \text{は } R_i \text{のキー属性}, 1 \leq i \leq m\}$$

とする。

また、ユーザ概念G-Netにおいて、データベースの関係に対応するエンティティの集合を

$$E = \{E_j | E_j \text{は } R_j \text{に対応}, 1 \leq j \leq m\}$$

リンクの集合を

$$L = \{L_j | 1 \leq j \leq l\}$$

とする。

3.2 Join-Net

データベース内の複数の関係表を用いるような問い合わせの場合は、それらの関係表に共通な属性を外部キーとするJoin結合が行われる。

例えば、

KDA > *Get the professor who teaches CS001 ?*

という問い合わせに対して、図1のようなシステムが生成するSQLの問い合わせ形式コマンドが生成されたとする。この場合、関係表*#PROFESSOR*と*#CLASS*の結合がそれぞれの共通属性*PNO*を用いて行われている。これをエンドユーザの問い合わせと対比してみると、teachという概念でprofessorとclassを結合している。

Join-Netは、データベース内の関係表間の関連を表現する無向グラフであり、$\boldsymbol{R}$をノード、その間の関連をリンクで表す。関連については、$R_p, R_q \in \boldsymbol{R}$に対して共通の属性$A_r$が存在する、つまりJoin結合可能

$$R_p \bowtie_{A_r} R_q$$

であれば、そのノード間に何かしらの概念が存在すると予想され、リンクを設定する(図2)。なおデータベースでは、同一概念を表す属性は同名であると仮定する。これはデータベース定義ファイルから自動的に作成される。

このJoin-Netとユーザが作成するユーザ概念G-Netを比較することで、ドメイン概念の過不足を検出する。過不足が見つかった場合は、ユーザに確認を促す。またリンク変換知識作成の際には、リンクで結ばれる関係表をJoin結合するのに必要な属性の記述が必要であるが、これをJoin-Netから獲得することで、リンク変換知識の自動生成が可能となる。

```
SELECT   PROFESSOR.PNAME
FROM     PROFESSOR
WHERE    PROFESSOR.PNO =
  ( SELECT  CLASS.PNO
    FROM    CLASS
    WHERE   CLASS.CLNAME = "CS001" )
```

図 1: システムが生成するSQL

3.3 Dependency-Net

Dependency-Netは、データベース内での概念間の関数従属性を表現する有効グラフであり、エンティティを表すノードと従属関係を表すリンクから成る。

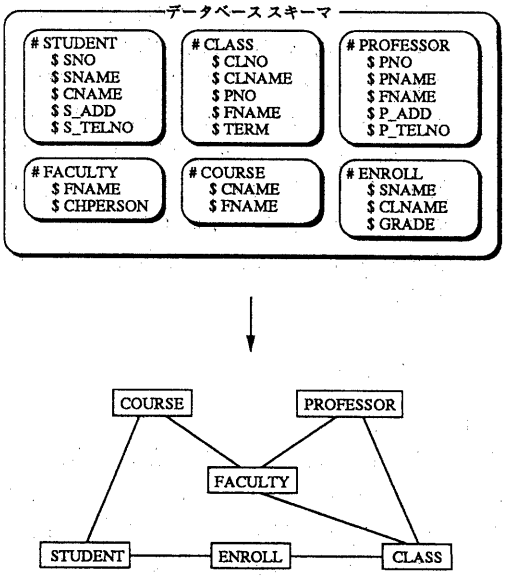

図 2: Join-Net

本来、関数従属性は関係表内の属性の依存関係を表すが、Dependency-Netではユーザ概念(エンティティ間)の従属関係に反映させて表す。

Dependency-Netは、2.2.2節で分類した設計アプローチType1、Type2に応じてそれぞれ作成される過程が異なる。

3.3.1 Type1

この場合のDependency-Netは、データベーススキーマから生成されるJoin-Netよりシステムが自動的に作成し、知識ベース作成の際に用いられる。

$R_p, R_q \in \boldsymbol{R}$間に$L_r \in \boldsymbol{L}$が結ばれているとき、$L_r$を結ぶための共通属性$A_r \in \boldsymbol{A}$が$K_p \in \boldsymbol{K}$であれば、$R_p$から$R_q$への依存関係が検出され、$R_p$から$R_q$へリンクの向きを付加する。

しかし、A_rがK_pかつ$K_q \in \boldsymbol{K}$であるときは、R_pとR_qが多対多関係であるため、リンクは削除される。

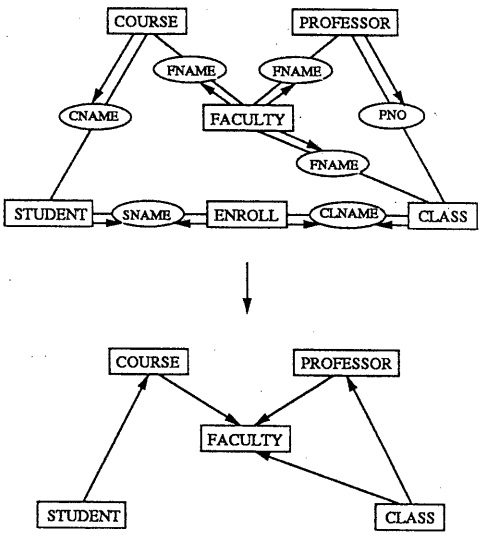

図 3: Dependency-Net (Type1)

3.3.2 Type2

この場合の Dependency-Net はユーザがユーザ概念 G-Net を描きながらシステムが自動的に作成し、データベース作成の際に用いられる。

ユーザ概念 G-Net 上で、$E_x \in \boldsymbol{E}$から $E_y \in \boldsymbol{E}$ へリンクが結ばれたとする。このとき、E_yが E_x と同名の属性 (データベースでは E_xのキー属性となる) をもつと定義された場合、この E_xから E_y へは一対多関係が存在することになる。つまり、E_xが E_yに関数従属

$$E_x \rightarrow E_y$$

であるので、E_xから E_yへリンクを結ぶ。

ところが、この方法で Dependency-Net を作成していく過程で、E_x, E_y間に多対多の関係があるときもリンクが結ばれることになる (図 4)。そこで、これを防ぐために、5.2 節で述べるユーザとの対話によってこの多対多関係を検出し、このときはリンクを結ばないようにする。

関数従属性は関係表内の属性の依存関係を表すため、データベースのデータからのみ関数従属性は検出可能である。ところが、データベーススキーマを設計し、それからデータベースにデータの登録、属性間の関数従属性検出、スキーマの変更という手順を繰り返していくと大変煩わしい。そこで本研究では、データではなく、意味ネットワークから関数従属性を検出する方法をとる。

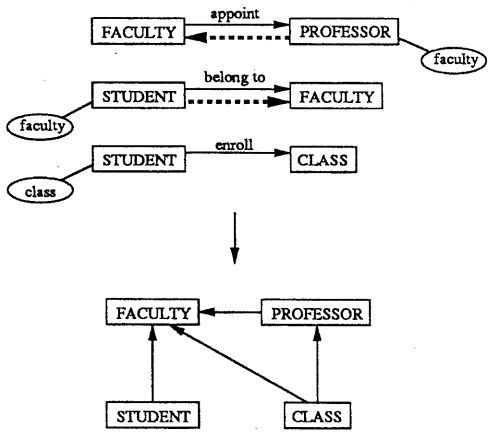

図 4: Dependency-Net (Type2)

4 知識ベース作成

4.1 作成手順

4.1.1 ノードに関する知識

1. ノードとデータベース内の関係表との対応関係をユーザの入力値とデータベースデータファイルから検出する。
2. データベースデータファイルから (関係名、インスタンス) を取り出し、辞書に登録する。
3. ノードについてのユーザ概念とデータベース概念との関係をノード変換知識として作成する。
4. ユーザ概念 G-Net で表される関係などノードについての知識をノード知識として作成する。

上記のノードに関する知識作成手順を図 5に整理する。

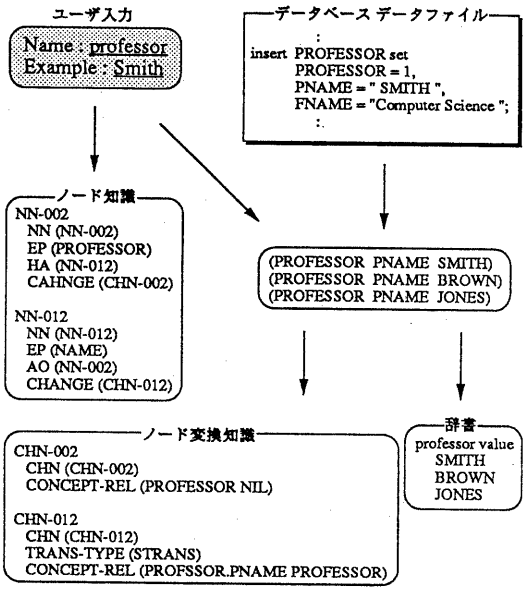

図 5: ノードに関する知識作成手順

4.1.2 リンクに関する知識

1. リンクについてのユーザ概念とデータベース概念との関係をリンク変換知識として作成する。
2. ユーザ概念 G-Net で表される関係などリンクについての知識をリンク知識として作成する。

上記のリンクに関する知識作成手順を図 6に整理する。

4.2 高次正規形時の対処

既存のデータベースから知識ベースを作成する場合 (Type1)、基本的に、関係表はノードに対応するものとして知識の作成を行う。ところが、第二正規形を考慮したデータベースの場合、必ずしも関係表がノードに対応せず、リンクに対応することがある。

$E_x \in \boldsymbol{E}$から $E_y \in \boldsymbol{E}$ へリンクが結ばれたとする。このとき、E_x, E_yにそれぞれ対応する Join-

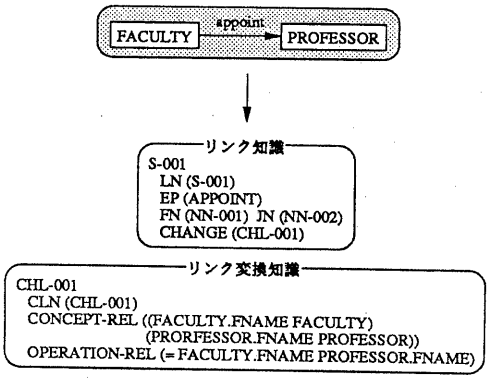

図 6: リンクに関する知識作成手順

Net の $R_x, R_y \in \boldsymbol{R}$間に $R_z \in \boldsymbol{R}$が存在した場合、この可能性が考えられる。

そこで、Join-Net と Dependency-Net を重ね合わせたグラフを用いる。R_zは複数の存在が考えられるが、このうち R_x, R_y共に R_zに従属関係をもたない R_zがリンクに対応する関係表となる。

そして、Join-Net と Dependency-Net の重ね合わせを用いて第二正規形を満たすデータベースであることを検出した場合、リンクを関係表に対応させた知識を作成する。

例えば図 7では、*enroll* というリンクがデータベースの*#ENROLL*に対応し、それに合う知識が作成される (図 8)。

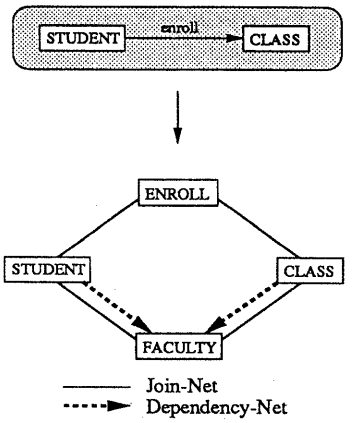

図 7: Join-Net と Dependency-Net の重ね合わせ

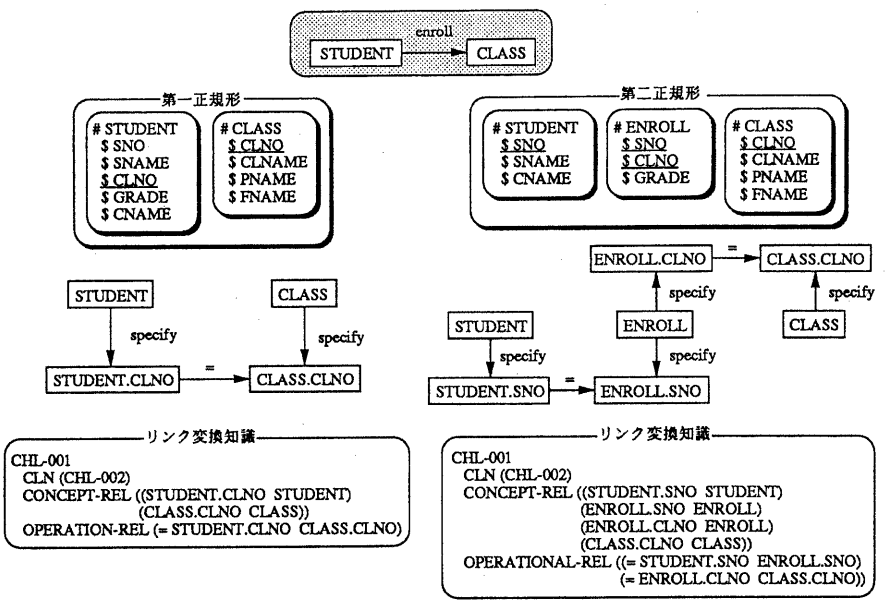

図 8: スキーマ構造による知識の違い

5 データベース作成

データベースへのデータの付加、削除、更新などを無秩序に行うと、思わぬ不都合を招くことが予想される。それを防ぐためにデータベース設計に関する一定の規約、いわゆる正規形と呼ばれる概念がある。データベース設計を行う際には、これを考慮して第三正規形を満たすデータベースをユーザ概念 G-Net から作成するが、ユーザには、この正規形の概念を特に意識させずに設計が行えるようにする。

5.1 作成手順

データベースは次の手順で作成される。

1. ユーザがドメイン概念をユーザ概念 G-Net で描く。
2. 正規形を考慮した場合、スキーマが適切でないときは、第二正規形、第三正規形を満たすデータベーススキーマに変換できるようにユーザ概念 G-Net を変形する。
3. 生成したデータベーススキーマに対するタプル例を一組入力してもらい、これより属性の型 (文字列型、整数型、浮動小数点型) を決定する。
4. スキーマ定義ファイルを作成し、これをロードしてデータベーススキーマが作成される。
5. データをデータベースにロードし、必要なデータは自動的に辞書に登録される。

5.2 第二正規形への変換

ユーザ概念 G-Net において、$E_x, E_y \in \boldsymbol{E}$間にリンクが結ばれ、$E_y$が$E_x$と同名の属性をもつとき、$E_x$から$E_y$への一対多関係は分かるが、$E_y$から$E_x$への関係は検出できない。そこで、$E_y$から$E_x$への関係を検出するための質問をユーザに与える。このユーザとの対話で、E_yからE_xへ一対多関係があることを検出できれば、E_x, E_y間には多対多関係が存在することになる。このときは、このままではデータベースに冗長が生じるので、リンクに対する関係表を自動的に作成し、第二正規形を満たすデータベースへと変換する。また、属性も

関数従属性を考慮して自動的に付加される。

例えば、図9の状態を仮定する。*STUDENT* は *CLASS* とリンクで結ばれているが、同名の属性 *class* をもっている。このとき、

Sys > Does the student enroll any classes ?
Usr > Yes.

という対話により、*STUDENT*、*CLASS* 間に多対多の関係が検出され、自動的に *#ENROLL* という関係表が生成される。

ところが同様に、*STUDENT* は *FACULTY* ともリンクで結ばれていて、かつ *faculty* という属性をもつので、次のような質問がユーザに発せられる。

Sys > Does the student belong to any faculties ?

これに対してユーザが *No* と答えたと仮定すると、この場合は多対多関係は存在しない。

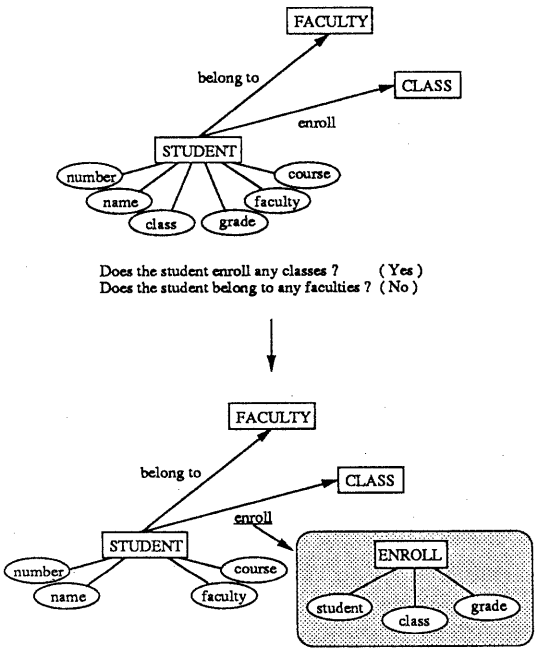

図 9: 第二正規形への変換

5.3 第三正規形への変換

第二正規形を満たすデータベースへと変換できるユーザ概念 G-Net において、$E_p \in \boldsymbol{E}$の非キー属性の集合のうち、二つの属性の組み合わせにすべてついて、関数従属性を検出するための質問 (5.2 節の質問と同形式) をユーザに行う。このとき $A_q, A_r \in \boldsymbol{A}$間に関数従属性 ($A_q \rightarrow A_r$) が検出され、この属性をキーとする $E_q, E_r \in \boldsymbol{E}$が存在しなければ、新しいノード E_q, E_rが作られる。また、E_qに E_pの A_rが付加される。そして Dependency-Net においては、E_pから E_qへ、E_qから E_rへリンクが結ばれる。

そして、データベース作成後の $R_p, R_q, R_r \in \boldsymbol{R}$ において、R_pの K_rと R_p, R_qの Join 結合で得られる K_rの値が常に一致すれば、冗長をなくすために Dependency-Net における E_pから E_rへのリンク E_pの A_rが削除される。

例えば、*STUDENT* の非キー属性の *course* から *faculty* へ関数従属性が検出されたとする (図10参照)。このとき、ノード *COURSE* が作成され、*STUDENT* の *faculty* 属性は *COURSE* へ移される。また Dependency-Net において、*STUDENT* から *COURSE* へ、*COURSE* から *FACULTY* へリンクが結ばれる。

またデータベース作成後、student の faculty は、その student の course に対応する faculty と一致するので、Dependency-Net の *STUDENT* から *FACULTY* へのリンク、*STUDENT* の faculty 属性は削除される。ユーザ概念 G-Net において、同様の関係が *CLASS, PROFESSOR, FACULTY* 間に見られるが、class の faculty がその class を教える professor の faculty と必ずしも一致しない場合 [10] は、前述で削除されたリンク、属性は削除されずに残される。

こうして作成された Dependency-Net は問い合わせの際に利用される。例えば、"ある student の faculty が欲しい" という問い合わせのとき、データベース内では *#STUDENT* には直接 *faculty* 属性は存在しないが、Dependency-Net をたどりながら、*#STUDENT* から *course* 属性を見て、*#COURSE* の *faculty* 属性を検索することで可能となる。

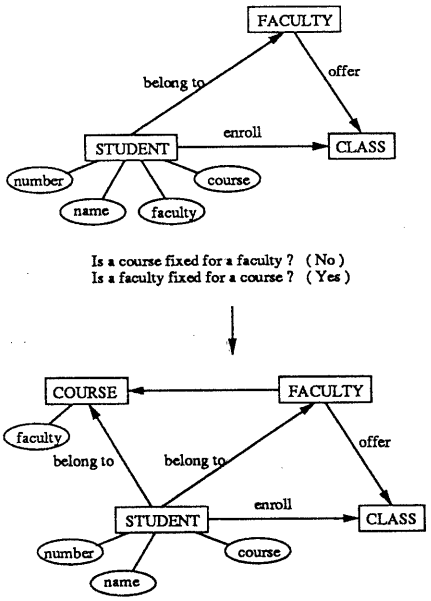

図 10: 第三正規形への変換

6 まとめ

データベース・アシスタントのためのドメイン知識の構築を行うには、かなりの時間と労力を必要とする。そこで、容易にアプリケーション構築を可能とするツールを提案した。

このツール上で、ユーザはユーザ概念を表す意味ネットワークを用いてデータベースドメイン概念を定義することにより、知識ベースとデータベースを同時に作成することができる。

このツールを実現する上で、システムを作成するための知識は、あらかじめ用意せずユーザのアクションから自動的に生成可能なため、汎用性は高いと考えられる。

今後の課題は、構築された知識の妥当性のチェック機能、データベース更新時の知識修正機能が挙げられる。

なお、KDA システムは Sparc Station 上で Common Lisp、X View、G-BASE を用いて構築中である。

参考文献

[1] G. Jakobson, G. Lafond, E. Nyberg, and G. Piatetsky-Shapiro, "An intelligent database assistant," *IEEE EXPERT*, vol.1, no.2, pp.65–78, 1986.

[2] B. J. Grosz, D. E. Appelt, P. A. Martin, and F. C. N. Pereira, "TEAM: An experiment in the design of transportable natural language interfaces," *Artificial Intelligence*, vol.32, no.2, pp.173–243, 1987.

[3] S. J. Kaplan, "Designing a portable natural language database query system," *ACM Trans. Database System*, vol.9, no.1, pp.1–19, 1984.

[4] N. Cercone, G. Hall, S. Joseph, M. Kao, W. S. Luk, P. Mcfetridge, and G. McClla, "Natural language interface: Introducing System X," *Advances Artificial Intelligence Software Engineering*, vol.1, pp.169–250, 1990.

[5] 谷, 飯野, 山口, 市山, "自然言語インタフェース構築キット: IF-Kit," 電子情報通信学会 言語理解とコミュニケーション研究会, NLC 91-62, pp.25–32, 1991.

[6] C. F. Eick, "From natural language requirements to good data base definitions – a data base design methodology," *in Proc. Int. Conf. Data Engineering*, Los Angeles, CA, Apr. 1984, pp324–331.

[7] 川口, 松山, 沼田, 溝口, 野村, 角所, "インタビューに基づくデータベース論理設計支援エキスパートシステム," 人工知能学会誌, vol.4, no.4, pp.421–430, 1989.

[8] X. Wu, and T. Ichikawa, "KDA: A knowledge-based database assistant with a query guiding facility," *IEEE Trans. Knowledge and Data Engineering*, vol.4, no.5, pp.443–453, 1992.

[9] X. Wu, M. Tanaka, and T. Ichikawa, "KDA: A knowledge-based database assistant," in *Proc. Fifth Int. Conf. Data Engineering*, Los Angeles, CA, Feb. 1989, pp.402–409.

[10] 中村(彰), 呉, 平川, 市川, "誤解の検出に基づく協調的応答システム," 電子情報通信学会 人工知能と知識処理研究会, AI 91-74, pp.1–8, 1992.