CONTEXT データモデルに基づく網構造データベース用高水準利用者インタフェース

1982年 3月

静岡大学大学院 電子科学研究科

秋口忠三

内容梗概

網構造データベースに対する高水準利用者インタフェースの構成法は現在模索の段階にある。　本論文では，この問題について原理的な考察から始め，新しいデータモデル——CONTEXTデータモデル——を考案し，このデータモデルに基づく一貫した利用者インタフェースの構成法を示した。

CONTEXTデータモデルは，強力なデータ記述・写像能力を有し，網構造データベースに対して，特定の目的をもつ利用者が要求を記述するのに適した場——CONTEXTの語意——を設定しうる。　CONTEXTはレコード型を節点とする木を基本構造とし，網構造スキーマから二段階に分けて構成される。　すなわち

1. レコード間の論理的関係をアクセス関数で一様に表現する。
2. アクセス関数とレコード型からCONTEXTを構成する。

CONTEXTの構成においては，仮想レコード型，階層レコード型，れ次のレコード間関係の階層化等の概念を導入し，網構造モデルのデータ表現の制約を打開できた。　また再帰的CONTEXTの概念を導入することにより，従来処理がきわめて困難であった再帰構造データベースの処理を容易にした。

CONTEXTデータモデルでは，アクセス関数を利用してレコード型から単純データ型への関数を定義するという形式で，導出データ項目を記述する簡潔で強力な記法を与えている。　再帰的な関数を定義することにより，再帰構造データベースから要約的な情報を抽出することも容易である。

目的に合致したCONTEXTを設定することができれば，その上で要求を記述することに困難はない。　本論文では，CONTEXTを通じて網構造データベースを利用するために，最終利用者用の問い合せ言語QLCと応用プログラマ用のデータ操作言語を考案した。　木構造モデル用の問い合せ言語は，すでに実用化されているものも含め何種類かが提案されている。　これらの問い合せ言語と比較してQLCは次の際立った特長を有する。

1. 処理単位の概念を導入することにより，木構造データビューに対する問い合せを曖昧なく表現できる。　また複雑な量操作を含む問い合せを明快に表現できる。
2. 再帰的 CONTEXT に対して順行子を宣言することにより再帰構造データベースから構造に関する情報を抽出することが可能である。

CONTEXT データモデルでは，網構造スキーマの構造に制限されることなく処理要求に適したデータ構造を定義できるので，応用プログラマは整構造プログラムをかき易い。　さらに，網構造スキーマの変更の多くは写像定義の変更で吸収できるので，応用プログラムに高いデータ独立性をもたせることができる。　このように CONTEXT データモデルによってデータベース応用プログラムの作成・保守の大幅な改善が期待できる。

幅広い利用者層の多様な要求に応じて適切な利用者インタフェースを設定するのはデータベース管理者の仕事である。　二段階構成法をとる CONTEXT データモデルでは，利用者のレベルに応じた利用者インタフェースの構成が可能であり，利用者とデータベース管理者の間で適切に仕事を分担することができる。　この特性はデータベースシステム全体の効率向上につながるもので，CONTEXT データモデルの最大の特長と言えよう。

目次

第 1 章　序論

1.1 緒言

C.W. Bachman らによって最初のデータベース管理システム IDS (Integrated Data Store) が開発されたのが 1963 年のことである。それ以来，商用ベースあるいは研究レベルで数多くのデータベース管理システムが開発され，多くの組織体でデータベースの導入が計画・実行されてきた[1)2)]。また 1970 年には，E.F. Codd によって，データベースの数学的なモデルとして関係モデルが提案され，データベース分野の諸問題に対して理論的なアプローチが可能になった[3)4)5)]。

データベース管理システムの開発・データベースの構築の経験の蓄積，およびデータベース理論の発展によって，「データベース」という概念，データベースの編成法，データベース管理システムの構成法等のデータベース技術を形成するあらゆる要素に対して，さまざまな局面から新しい光が当てられ，多くの検討が加えられてきた。これらの努力によって，データベース技術は，データ変換（計算）技術，データ通信技術と並んで，情報処理技術を形成する一つの柱として認識されるまでに成長した[6)]。

データベース技術の進歩・発展によって，データベースは，単に事務処理の分野だけにとどまらず，各種設計情報の管理，実験・観測データの管理，研究資料の管理等の，相互に関連をもつ大量のデータの管理を必要とする多くの分野で，有用性が認められ，導入が進められている。また個々のデータベースも大規模化の傾向にある[7)]。

このようにデータベースの応用分野の拡大・大規模化が進むにつれて，情報処理の専門家でない利用者が相当複雑なデータベースの利用を望む機会がますます増えてくることが予想される。これからのデータベース管理システムは，この種の利用者を含む幅広い利用者層に対して，データベースを利用するための道具を簡単な手続きで提供できる機能——利用者用機能 (user facility) ——を備えるべきである。

データベース利用者インタフェースは，データベース管理者が利用者用機能を使って構成する。本論文は，利用者用機能に関する筆者の研究の

成果をまとめたものである。

1.2 網構造データベースに対する利用者インタフェース

データベースの編成・管理・運用を組織的に行なうためには，データベースの論理構造を形式的に表現する枠組みが必要である。データモデルはこのような表現の枠組みを与える。代表的なデータモデルとして関係モデル，木構造モデル，および網構造モデルがある[8)]。現在稼動しているデータベース管理システムの多くは，基本的にこれらのデータモデルのどれか一つに基づいて構成されているといってよい。

これらのデータモデルは，データベースの論理構造の表現形式，効率の良いデータベース管理システムの実働化の難易度，高水準利用者インタフェース構成の難易度等の面で一長一短があり，「あらゆる使用状況において最良のデータモデルはどれか」といった問に対して解答を出すことはできない。データベースのそれぞれの応用分野，さらに各応用分野の中の個々の組織体において，管理すべきデータの特性，データの利用形態等の実情に応じて最適なデータモデルが選択されることになろう。

筆者は，データモデルの種々の特徴の中で，網構造モデルのデータ表現能力の豊かさとそれに随伴する欠点であるデータ構造の複雑さに注目したい。従来のように，データ構造の複雑さを直接利用者の目にさらす限り，データ操作が煩雑になることは避けられないと思われる。そのために，特に複雑なデータを取り扱う応用分野において次の問題に行き当るであろう，

> 網構造モデルのデータ表現能力には魅力がある。しかしデータベースの利用が困難である。

今後とも，複雑なデータの効率の良い管理を必要とする応用分野はますます広がるものと予想され，網構造モデルの強力なデータ記述能力の有用性が減ずることは考えられない。それゆえに，網構造モデルに基づいて編成されたデータベースの簡便な利用を可能にする利用者インタフェースの構成法を早急に確立する必要性が痛感されるのである。

1.3 高水準利用者インタフェースの必要性

利用者インタフェースは，利用者から見たデータベース――データビュー（data view）――の記述とこのデータビューに対して要求を記述するための言語から構成される。情報処理の専門家でない利用者にとって，利用者インタフェースにおけるデータビューが利用目的に適した構造をもっていることが本質的に重要である。

一方，実用的な規模のデータベースは多くの利用者によって共同利用される。各々の利用者の要求は多種多様であるから，これらすべての利用者の要求に応えるように編成されたデータベース全体の論理構造は，個々の利用者のデータビューと合致しない場合がしばしば現れると考えられる。

しかるに現実のデータベース管理システムの利用者用機能は低く，利用者のデータビューの記述はデータベース全体の論理構造によって大きく制限される場合がほとんどである。そのために，データベースの利用者は不適当なデータビューに対して複雑な要求記述を強いられるという状況が生まれる。

特に網構造モデルに基づくデータベース管理システムでは，利用者のデータビューの記述がデータベース全体の論理構造の一部を選択するだけにとどまり，データ構造の複雑さがそのまま利用者インタフェースに現れるのが現状である。集合演算を基本とする高水準利用者用言語の構成が困難になるのはそのためである。

以上の問題認識に基づき，筆者は，網構造データベースに対する高水準利用者インタフェースの構成法について研究した。特に，網構造モデルのデータ表現能力をいかし，しかもデータ操作の記述が容易に行なえるインタフェース構成を目指した。また，インタフェース構成そのものが簡単な手続きで容易に行なえるように工夫した。

1.4 論文の構成

第1章では研究の動機、意義について述べた。

第2章では，データベース利用者インタフェースの構成法に関連した諸点を整理し，また関連分野の研究の現状を概観することを通じて，本研究で扱かう問題の範囲および本研究の位置づけを明らかにする。

第3章では，データベース全体の論理構造および意味構造を扱う概念レベルについて詳細に検討する。 特に，データベース全体の論理構造を表現するための二つの代表的なデータモデル——関係モデルと網構造モデル——について構造論的な側面と意味論的な側面から検討し，網構造モデルの最も重要な概念である親子集合構造の有用性と意味論上の問題を指摘する。

第4章では，第3章の検討を踏まえ，利用者指向のデータモデルとして新しく設計したCONTEXTデータモデルについて論ずる。

第5章では，CONTEXTデータモデル用に新しく設計した問い合せ言語QLCについて述べる。

第6章では，CONTEXTデータモデル用データ操作言語の機能の概観を与える。

第7章では，以上の議論に基づいて，利用者のレベルに応じた利用者インタフェースの構成法を示す。

第8章では，結論と今後の課題を与える。

図1.に各章の相互関係を示す。

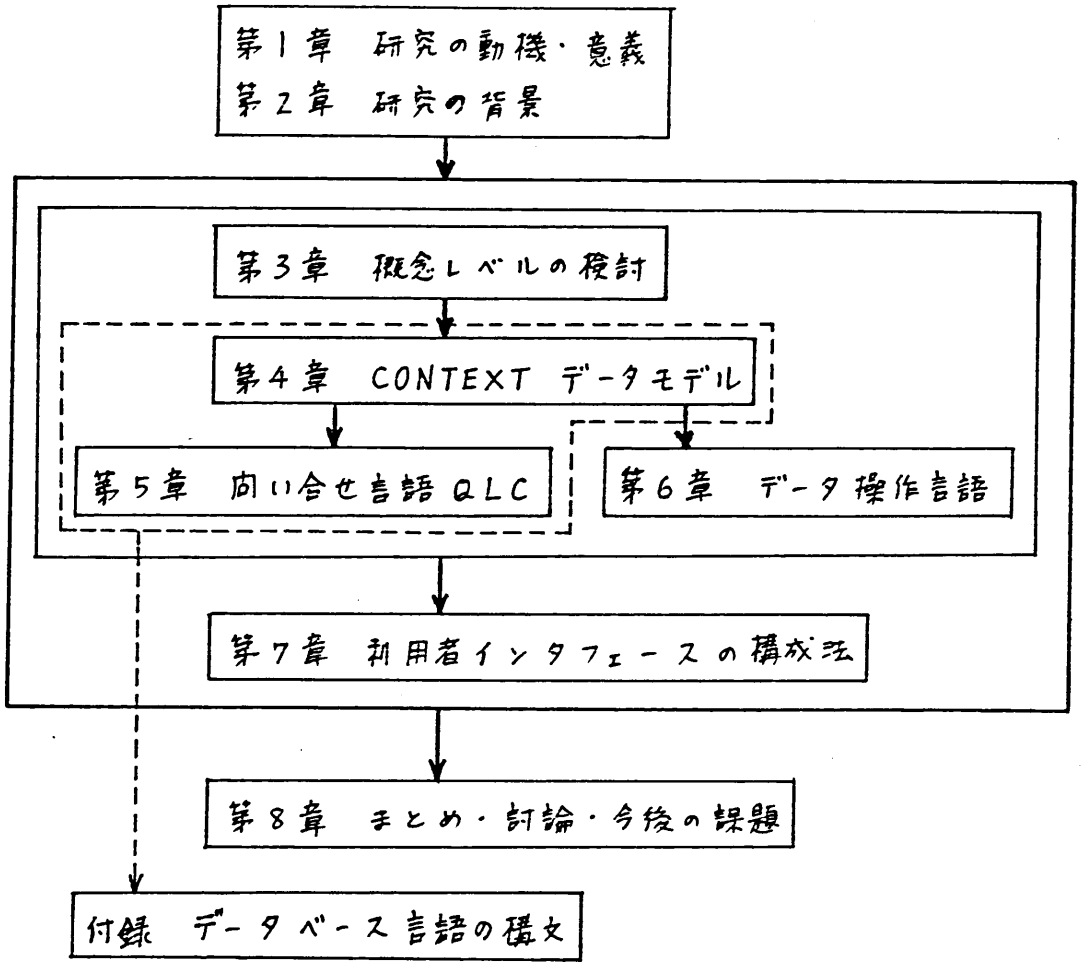

第1章　研究の動機・意義
第2章　研究の背景
第3章　概念レベルの検討
第4章　CONTEXT データモデル
第5章　問い合せ言語 QLC
第6章　データ操作言語
第7章　利用者インタフェースの構成法
第8章　まとめ・討論・今後の課題
付録　データベース言語の構文

図1　論文の構成

第 2 章　データベース利用者インタフェースの構成法に関連した問題

2.1 緒言

本章では，データベース利用者インタフェースを構成する際に考慮しなければならない諸点を整理し，また関連分野の研究の現状を概観することを通じて，本研究の位置づけおよび本研究の主題を明確化する。

2.2では，「データベースを利用する」ということを中心に，データベースの基本概念を整理する。

2.3では，データベースの利用者および利用者の要求を分類し，本研究で対象とする利用者の種類と要求の種類を明らかにする。

2.4では，データ保護・保全の問題について簡単にふれる。

2.5では，高れ準利用者インタフェースの実現へ向けた諸研究を，本研究との関連において述べる。

2.2 データベース利用環境

図2.1にデータベース利用環境の概観を与える。

2.2.1 データベース管理

企業・行政機関・大学・銀行・病院等の一定の目的をもった組織体（enterprise）は，その目的達成のためにさまざまな活動を展開している。組織体が能率良くそれらの活動を進展させるためには，状況に応じて適切な情報を獲得し，これに基づいて正しい意志決定を行なう必要がある。正確な情報を得るためには，データの収集・編成・格納・更新および情報の抽出・分配を行なうための手段が必要である。この手段が情報システム（information system）である[9]。

時間とともに変化する多様な情報要求に，敏速，適切かつ柔軟に対応するために，組織体が関心をもっているあらゆるデータを組織的に編成し，これを統制された制御の下で一元管理するという情報管理の方式が確立されてきた。組織体が関心をもっている現実世界の特定の部分を対象世界

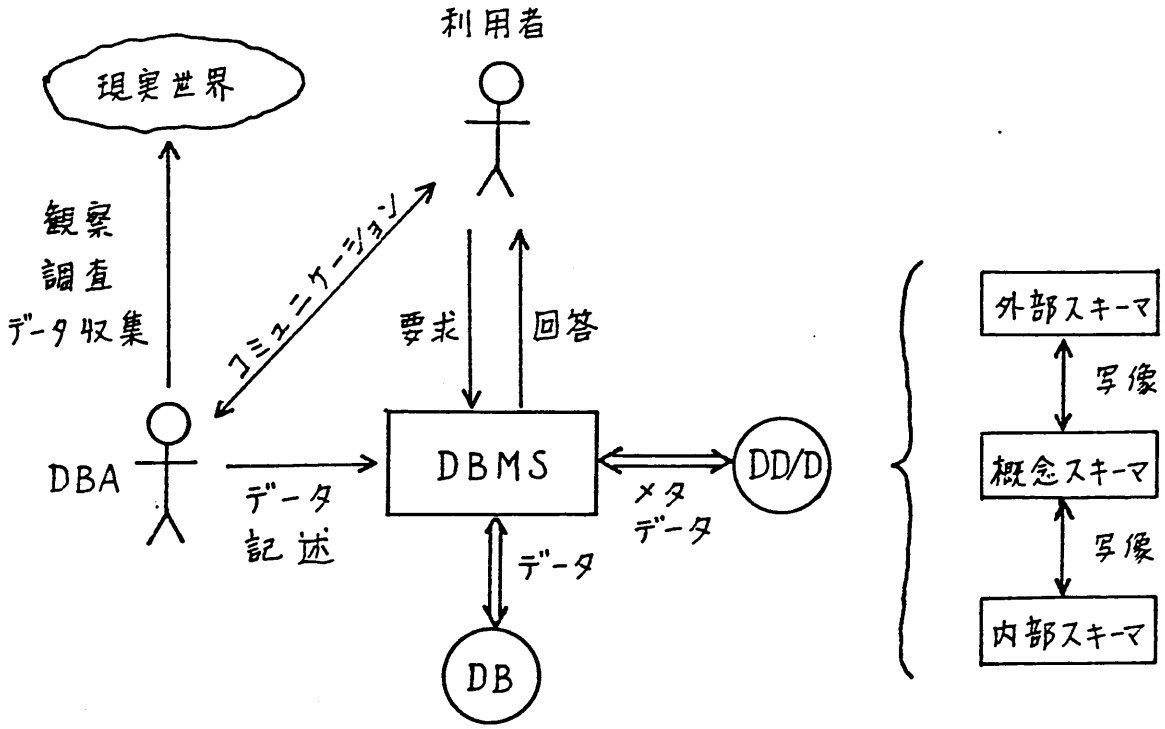


図2.1 データベース 利用環境

(universe of discourse , UoD) と呼ぶ[10]。対象世界におけるあらゆるデータを組織的に編成したデータの集合体がデータベース (database , DB) である。

データベースを管理するために，データベース管理システム (database management system , DBMS) とデータベース管理者 (database administrator , DBA) は中心的な役割を果す。データベース管理システムは，データベースの構築・維持・利用を支援するためのソフトウェアシステムであり，データベース管理者は，データベース管理システムを使って，データベースの編成・管理・運用の仕事を遂行する役割をもつ人あるいは人の集団である。データベースを運用するシステムをデータベースシステム (database system , DBS) と呼ぶ。データベースシステムは，データベース，データベース管理システム，データベース管理者，および計算機システムから構成される。

2.2.2 データベース管理システムの基本構成

ANSI/X3/SPARC データベース管理システム研究班は，データベース管理システムの標準化の可能性を調査するオー段階の作業として，データベース管理システムのもつべき機能を詳細に検討し，その結果として3層スキーマ構成に基づくデータベース管理システムの基本構成（architecture）を示した[11]。

3層スキーマ構成に基づくデータベース管理システムでは，データベースを概念レベル・外部レベル・内部レベルの三つのレベルに分けて記述する。これらのデータベースの記述を，それぞれ概念スキーマ（conceptual schema），外部スキーマ（external schema），内部スキーマ（internal schema）と呼ぶ。

概念スキーマは，データベースの物理的編成法や個々の利用者・応用業務（application）の情報要求とは独立に，データベース全体の論理構造および意味構造を記述したものであることが望まれる。外部スキーマは，データベースの個々の利用者・応用業務から見たデータビューを記述したもので，応用業務ごとに一つの外部スキーマが定義される。内部スキーマはデータベースの物理的編成法を記述したものである。外部スキーマと内部スキーマは，それぞれ概念スキーマに基づいて記述され，常に概念スキーマとの一貫性が保たれていなければならない。

これら3種類のスキーマおよびスキーマ間の写像情報は，厳密に定義された言語を使ってデータベース管理者が記述し，データベース管理システムのデータ辞書（data dictionary / directory，DD/D）機能によって管理される（図2.1参照）。

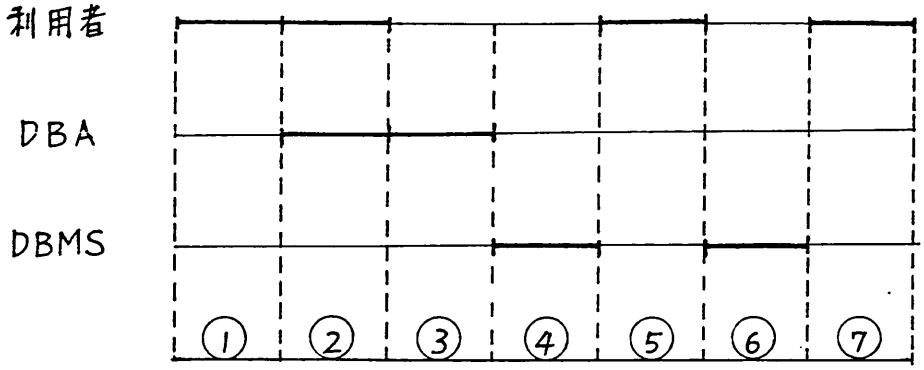
2.2.3 データベースの利用

データベースからのデータの読取り，データベースの内容の更新等のデータベースに対する操作の総称をデータベースアクセスという。データベースを利用するとは，データベースをアクセスして必要とする情報を抽出したり対象世界の状態の変化をデータベース上に反映させることである。前者をデータベースの検索，後者をデータベースの更新と呼ぶ。

データベースの個々の利用者は，データベース全体に関心があるわけではない。利用者は，データベースの必要な部分だけを，自分の利用目的に最も適した形で見ることを望むであろう。利用者から見たこのようなデータベースの見かけの姿を記述したものが外部スキーマであった。適切な外部スキーマを定義するためには，利用者の要求と概念スキーマとを正確に理解していなければならない。大規模なデータベースでは概念スキーマは非常に複雑になり，一般の利用者に概念スキーマを理解してもらうことは不可能である。従って外部スキーマの定義はデータベース管理者の仕事と解すべきである。小規模なデータベースでは，利用者本人がデータベース管理者の仕事を行うことになるかもしれない。

利用者がデータベースを利用する手順は，およそ図2.2に示す通りであろう。まず，利用者は自分の要求をよく整理し（①），それをデータベース管理者に正確に伝える（②）。データベース管理者はこの要求に合った外部スキーマを定義し（③），それをデータベース管理システムに入力する。データベース管理システムのデータ辞書機能はこの外部スキーマを内部形式に変換し，データ辞書に格納する（④）。利用者はこの外部スキーマに基づいてデータベースの検索・更新要求を出す（⑤）。データベース管理システムは，データ辞書に格納されてあるスキーマ情報を参照しつつ，これらの要求に応えるべくデータベースをアクセスし，結果を利用者に返す（⑥）。利用者は結果を解釈し（⑦），必要ならば次の要求を出す。

高水準利用者インタフェースを構成する目的は，利用者の要求記述（⑤）の負担を軽減することにある。しかし，その代償としてデータベース管理者の負担（②と③）やデータベース管理システムの負担（④と⑥）が不当に増大するのでは，その構成法は実用的でない。本論文で提案する利用者インタフェースの構成法は，データベースシステム全体の効率向上を目的としたものである。このような観点から，利用者インタフェースの構成法を具体的に提案した研究はこれまでになかった。



① 要求の整理
② 利用者とDBAの間のコミュニケーション
③ 外部スキーマの定義
④ 外部スキーマをデータ辞書（DD/D）に格納
⑤ 要求の記述
⑥ 要求の実行
⑦ 結果の解釈

図2.2　データベース利用のプロセス

2.3　利用者の種類と利用要求の種類

データベースの利用者はおよそ次のように分類できる[12) 13)]。

1. 応用プログラマ（application programmer）

応用プログラマは，データベース操作機能をもつプログラミング言語を用いて，データベースをアクセスするプログラムをかく。このデータベース応用プログラムは応用プログラマ自身が使用するためのものかもしれないし，最終利用者のためのものかもしれない。

2. 最終利用者（end user）

a）パラメータ利用者（parametric user）

パラメータ利用者は，あらかじめ定められた手順に従ってパラメータを設定することによって，定形的なデータベース処理の業務を遂行する。

b) 不定利用者 (casual user)
データベースを常時利用する利用者に対して，不定利用者がいる。不定利用者は，データベースを利用する積極的な動機をもたず，利用できるならば利用したいという消極的な利用者である。 不定利用者は不定形なデータベースの利用を望む。 しかし，データベースの構造の理解や形式的な問い合せ言語の学習は敬遠するだろう。

c) データベース応用分野の専門家
この種の利用者は，自分の仕事の遂行のために，不定形で複雑なデータベースの利用を要求するが，必ずしも情報処理の専門家とは限らない。 このような利用者は，要求を非手続き的に記述できる強力な問い合せ言語を必要とする。

本論文では，データベース応用分野の専門家のための利用者インタフェースに焦点をあてる。 応用プログラマ用の利用者インタフェースについても簡単にふれる。

データベース応用分野の専門家は，さまざまな形態でデータベースを利用することを望むであろう。 これらの利用形態は次のように分類できる。

1. データベースの検索

a) ひろい読み (browsing)
必要な情報が明確でない場合，データベースの内容をひろい読みしながら必要な情報を探し当てる。

b) 条件検索
検索要求が明確である場合には，検索条件および条件を満足するデータ群から抽出すべき情報の仕様を明確に指定する。

2. データベースの更新

a) データの格納

b) データの消去

c) データの修正

本論文では，条件検索要求を記述する言語機能について詳述する。データベースの更新機能についても簡単にふれる。

2.4 データ保護・保全

統合されたデータベース（integrated database）では，データ保護（security）・保全（integrity）の問題はきわめて重要である。データ保護とはデータのプライバシを守ることであり，データ保全とはデータベースを常に正しい状態に保つことである。

本論文では，この問題の考察は行なわないが，重要性だけは指摘しておきたい。

2.4.1 データ保護

データ保護の問題は，利用者インタフェースを構成する際に，データベース管理者が考慮しなければならない最も重要な問題の一つである。いかなる利用者も自分が参照する権利を有するデータだけしか見ることができない。データベース管理者は，利用者のデータアクセス権を慎重に検討して外部スキーマを定義しなければならない。利用者は，使用が認められた外部スキーマだけからデータベースをアクセスできる。

2.4.2 データ保全

データベースに対する誤った更新からデータベースを守ることはデータ保全の主要な問題の一つである。データベースに対する更新要求はすべて外部スキーマを通して行うべきであろう。誤った更新を防止することは，外部スキーマの重要な役割であると考えられる。

2.5 関連分野の研究の現状

2.5.1 CODASYL の活動

CODASYL (the Conference on Data Systems Languages, データシステム言語協議会) は，一連の活動を通じてデータベース管理の一つの方式を確立した[14]。この方式は CODASYL 方式または網構造方式と呼ばれている。CODASYL 方式は，幾多の議論をへて[15)~26)] しだいに精練され今日に至っている[27)~30)]。

CODASYL 方式によれば，現在のデータベース技術を用いて実用的なデータベースを構築することが可能である。しかし，利用者インタフェースとして応用プログラマ用のデータ操作言語だけしか与えておらず，しかもその水準は低い[28]。特に，現在指示子をたよりに網構造データベースの中を巡航するというデータベースアクセスの方式[31]は，プログラムの作成・保守を困難にする大きな要因である[32]。

最終利用者用のインタフェースとしては，CODASYL EUFTG (End User Facility Task Group) が Form アプローチを提案しているが，これはデータの表示形式として Form が良いという主張だけにとどまり，Form を記述する言語，Form と網構造データベースとの写像の方法，Form に基づくデータ操作等の具体的な内容の決定はこれからの課題としている[33]。

本論文で提案する利用者インタフェース構成法の簡単な拡張によって，網構造データベースに対する Form インタフェースを構成することができる (7.3 参照)。

2.5.2 関係モデルインタフェースの構成

データベースに対する高水準利用者インタフェースの理論的基礎は E.F. Codd によって確立された[5]。彼は，関係データベースに対する問い合せ表現の数学的記法として一階の述語論理に基づく関係計算 (relational calculus) と集合論に基づく関係代数 (relational algebra) を与え，これらの言語のデータ選択能力が等価であることを証明した。

さらに，これらの言語と同等以上のデータ選択能力を有する言語を関係完備である（relationally complete）と定義し，関係モデル用問い合せ言語の記述能力の規準を与えた。以来，これらの言語を基礎にして，使い易さを工夫した多くの問い合せ言語が提案されてきた[34)~40)]。

これらの関係モデル用問い合せ言語を網構造モデルの上で実働化しようという発想はごく自然である。多くの研究者が，この考え方に基づく利用者インタフェースの構成法を論じている[41)~44)]。しかしこの方式には次の難点がある。

現実世界には情報保有型親子集合を用いれば自然に表現できるが，関係モデルでは表現しにくい性質をもつデータ構造も存在する（3.5参照）。関係モデルインタフェースを設定する場合，情報保有型親子集合によって自然に表現されていたデータ間の関係を関係スキーマ上で切り離し，問い合せ要求を記述する際に再度結合するという不自然さがあり，利用者・データベース管理者・データベース管理システムの負担を不必要に重くしている。

E.F. Codd は，網構造データベースが情報保有型親子集合を含む限り非手続き的な高水準問い合せ言語の実現は困難であると指摘している[17)]。しかしこの指摘は正当なものではない。本論文では情報保有型親子集合を有効に利用した利用者インタフェースの構成法を提案する。

2.5.3 網構造データベース用高水準利用者インタフェース

W.C. McGee は，報告書作成・データベースのローディング等の処理を容易に行なえるように，網構造データベースに対するファイルレベルのデータ操作機能を導入した[45)]。これは CODASYL のデータ操作言語にレコードの順次処理を行なう制御構造（for ループ）を追加しただけの簡単な機能の拡張である。

E.K. Clemons は，CODASYL のサブスキーマ機能があまりにスキーマに依存しすぎているとして，より強力な外部スキーマ機能を提案している[46)47)]。彼は，網構造データベースからプログラム開発時に用いた認識

構造に近いレコード記述を与えることによって，応用プログラマがデータベースを利用していることを意識しなくてすむようなインタフェースの構成法を示した。しかし，この方法は完全なものではなく，しかも一般性に欠けるという強い批判がある[48]。

彼の外部スキーマ機能に関する考え方には同意できる部分もあるが，概念スキーマとあまりにかけ離れた外部スキーマは，記述の困難さ，処理系の複雑さ・効率等を考慮に入れると，むしろ有害であるとさえいえる。

B. Shneiderman らは，網構造データベースの巡航経路を簡単な式（path expression）で表現し，この式を引数とするデータ操作言語を与え，応用プログラマ用のかなり強力なインタフェースの構成法を示した[49]。この方式では，線形の巡航経路の表現は容易であるが，木状に分枝する経路の表現は煩雑になる。

J. Bradley は，関係モデルにおける関係計算型言語の結合項を親子集合の引用で置きかえることにより，網構造データベースに対する非手続き的問い合せ言語を与えた[50]。この言語による要求の記述・解釈はともに非常に困難である。

CONTEXT データモデルは木構造を基本構造としている。同じように，網構造データベースに対する利用者のデータビューを木構造で表現する方法は，C. Deheneffe が提案した NUL（a Navigational Users' Language）という問い合せ言語でも採用している[51]。NUL では，網構造データベースを巡航する経路を端末装置からインタラクティブに指定するという形態で木状のデータビューを形成する。しかし表現できる木構造は概念スキーマの構造によって強く制限される。「利用者の要求に合致したデータビューを構成する」という積極的な姿勢はみられない。たとえば，概念スキーマで陽に定義されていないレコード間関係をたどることはできない，3個以上のレコード間関係の取り扱いは不可能である，特定のデータ項目に注目したデータビューの記述も許されていな

い。　また，木構造データビューに対して許される操作は，それを木状に出力することだけである。　これでは利用者の多様な要求に応えることは難しい。

CONTEXT データモデルでは，これらの問題をごく自然な方法で解決している。　CONTEXT データモデルの特徴は，木構造を基本構造としていることに以上に，網構造データベースの上で利用者の要求に合致した木状のデータビューを編成しうる強力なデータ記述・写像能力にあるといえる。

二項関係を基礎にした網構造モデルに対する向い合せ言語としては，M.E. Senko の FORAL[52)~55)] や R. Munz の WELL[56)] が知られている。　これらの言語は，それぞれ興味深い特徴をもっているが，CODASYL 方式の網構造モデルとの対応は明らかでない。

2.5.4　再帰構造データベースの処理

再帰構造データベースを取り扱う問題は物品明細表問題（Bill of Materials' problem）と呼ばれ，取り扱いがきわめて困難であることが良く知られている[57)]。　たとえば，部品の構成関係，プログラムの呼び出し関係，文献の引用関係などを扱うデータベースが再帰構造データベースの例である。

再帰構造データベースの取り扱いが困難な理由は，同じ型に属する対象を節点とする網構造の性質として，ある節点から別のある節点に至る経路長を推定できないことにある。　このような再帰構造の本質的な性質である不定回の繰り返し構造の処理は，再帰的なアルゴリズムによって最も簡潔に表現できるのであるが，手続き的処理，非手続き的処理の別を問わずデータベースの再帰的な処理を許すデータベース管理システムは少ない。しかしその必要性は認識されている[58) 59)]。

関係モデルに基づいて再帰構造データベースを処理するための言語機能については，すでにいくつかの方式が提案されている[60)~62)]。　また，関数プログラミングの概念[63)]に基づき，再帰構造処理の記述に優れた向い

合せ言語も提案されている[4]。　本論文では，これらの方式を参考にして，CONTEXT データモデルの枠組みの中で，再帰構造データベースの処理を行なうための言語機能について述べる。

第 3 章　概念レベルの検討

3.1 緒言

ANSI/X3/SPARC によるデータベース管理システムの3層スキーマ構成の提案[11]は，データベース研究の多くの分野に大きな影響を与えた。特に注目を集めたのが概念レベルの導入である。

概念レベルでは，データベースは現実世界あるいは組織体のモデルである，と解釈される。現実世界の表現は，より現実世界に近いレベルから計算機に密着したレベルまで種々のレベルで考えることができる。概念レベルでは，より現実世界に密着した記述を指向している。

木構造モデル，網構造モデル，あるいは関係モデル等の初期（1970年ごろまで）のデータモデルは，現実世界を忠実に表現することよりもむしろデータベースの編成法を形式化・標準化することに重点が置かれていたように思われる。これらのデータモデルによって，物理的編成法の詳細をほとんど含まないデータベースの論理構造の記述は可能になったが，この論理構造からデータベースの情報内容を読取ることは容易ではない。現実世界との間に，いわゆる意味論的な溝（semantic gap）が存在するからである。

ANSI/X3/SPARC 提案に前後して，現実世界の忠実な表現を指向した，いわゆる意味論的データモデル（semantic data model）が数多く提案された[65)～73)]。意味論的データモデルに対して，初期のデータモデルを構造論的データモデル（syntactic data model）と呼ぶことにする。

意味論的データモデルの研究は，構造論的データモデル——特に関係モデル——と現実世界との間の意味論的な溝を埋めるために，数多くの有用な概念・手段を提供した。たとえば H.A. Schmid ら[67)]や P.P. Chen[68)]による関係の意味論的分類，J.M. Smith らによるデータ抽象化の概念に基づく関係の意味論的編成法[74)75)]，P. Hall らによる実体代用物（surrogate）の概念[69)]，C.W. Bachman らによる役割（role）の概念[76)]，等をあげることができる。E.F. Codd はこれらの概念を関係モデルに

とり入れ，関係モデルの意味論的拡張を行なった[77]。

現在のデータベース管理システムは，ほとんどが構造論的データモデルに基づいており，意味論的情報を扱う機構をもっていない。またその基本構成も明確な3層スキーマ構成をもっているとはいえない。しかしながら，3層スキーマ構成への改良は着実に進んでいるように思われる。その過度期においては，図3.1に示すような4スキーマ構成が適当であろうと思われる*。

図3.1において，論理スキーマは特定の構造論的データモデルに基づいて記述され，外部スキーマと内部スキーマの中継点の働きをする。一方，意味論スキーマは特定の意味論的データモデルに基づいて記述され，現実世界と論理スキーマの間の橋渡し（semantic bridge）の働きをする。すなわち，論理スキーマの設計を助け，同時に意味論情報を補足することによって論理スキーマの理解を助ける。現在は，意味論スキーマはデータベース管理者の手で管理されなければならないが，将来においては，意

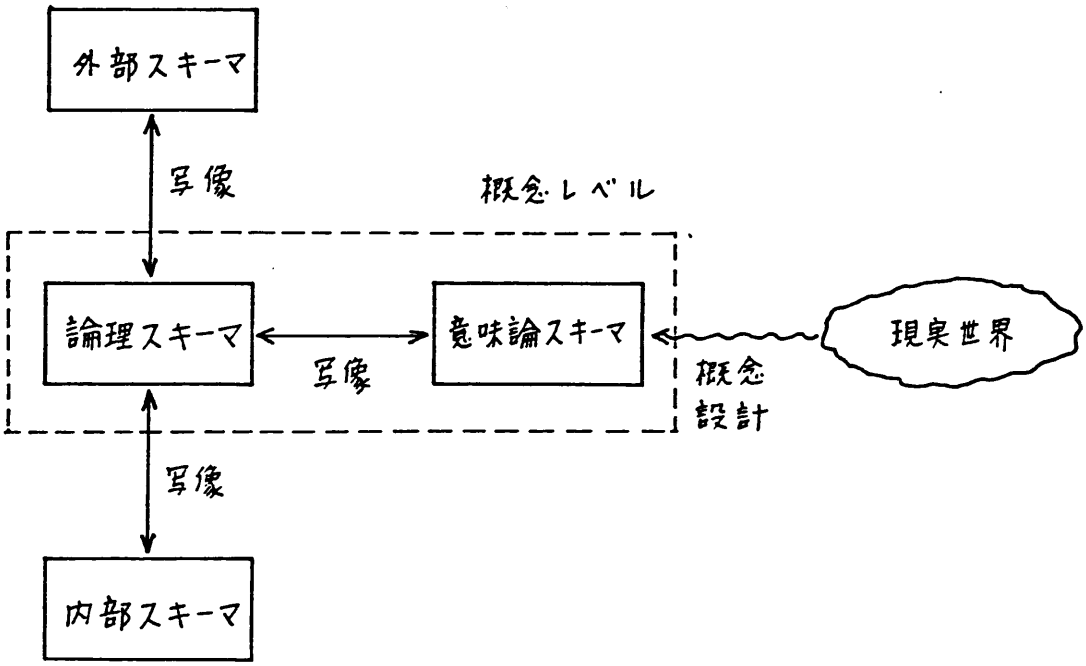


図3.1　データベース管理システムの4スキーマ構成

---

* 3層スキーマ構成を基礎にして概念スキーマを機能別に細分化する試みは[75),79)]にもみられる。

味論スキーマの管理もデータベース管理システムが行なうことになろう。そうなれば，意味論スキーマの上で外部スキーマを構成するという利用者インタフェースの構成法も検討の対象となろう。しかし現状では，構造論的データモデルの意味論的性質——現実世界の情報構造の表現のしかた——を検討し，この性質をうまく生かした利用者インタフェースの構成法を確立することの方が，より現実の要求に即していると思われる。

本章では，このような観点から網構造モデルの意味論的性質を検討する。まず3.2で概念レベルの取扱い方を論ずる。3.3では，データベースの論理構造を表現するモデルとしての観点からCODASYL DDLC 78提案[27]による網構造モデルの諸概念を再整理する。3.4では，E.F. Coddの関係モデルについて述べる。3.5では，関係モデルと網構造モデルの論理データベースの表現能力を比較することを通じて，網構造モデルの最も重要な要素である親子集合構造の有用性と意味論上の問題点を明らかにする。

3.2 概念レベルの取扱い

図3.2に，本節の議論のスケッチを与える。

3.2.1 対象世界の記述

組織体が関心をもっている現実世界の特定の部分を対象世界と呼んだ。一般に，対象世界は種々の性質（property，characteristic）をもつ実体（entity）およびいくつかの実体を結びつける関連（relationship）の集まりとして解釈されることが多い[11) 68)]。

実体の概念は，現実世界を認識し対象世界を定める上で最も重要な概念である。しかしこのことばを厳密に定義することは容易でない。一般に，実体という概念は，ものを，その質や他との関係においてではなく，そのもの自体の存在性においてとらえた概念と考えることができる。しかし，「存在する」という概念は，ものそのものではなく，抽象化された概念を扱う情報の世界では，きわめて取扱いが困難である。結局，実体という概念は，性質あるいは関連という概念との相対的な関係のなかでと

らえる以外にないと思われる。　実体，関連，性質等の概念を，これらの相互の関係において形式的に定義することによって，一つの意味論的データモデルが構成されることになる。

対象世界における実体・関連は，自然法則，組織体内外の諸制度等によってとりうる状態が制限され，この制限の下で，種々の事象の生起によって発生・消滅・変化する。　データベースの分野で扱う対象世界は，

a) 個々の利用者・応用業務の情報要求の総体に基づいて決定される，

b) 個々の実体・関連の個数は膨大であるが，これらの実体・関連は比較的少数の型に分類できる，

という特性がある。

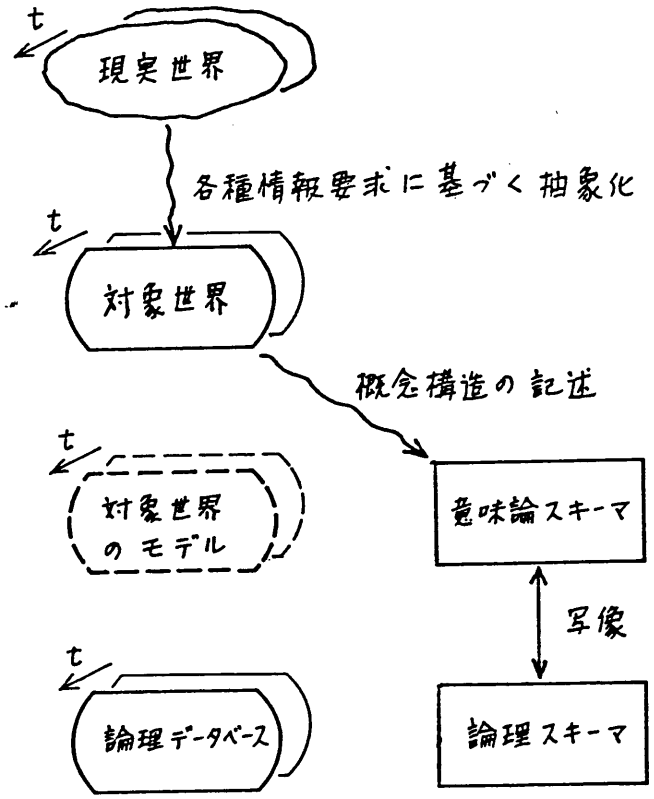


図3.2　概念レベルの取扱い

現実世界，対象世界は時間（t）とともに変化する。　意味論スキーマは，対象世界の時間に不変な概念構造を記述したもので，対象世界のモデルのとりうる状態を規定する。論理データベースは，対象世界のモデルを計算機で処理可能な形態で記号表現したもので，論理スキーマに従って編成される。

特定の意味論的データモデルに基づいて，個々の情報要求から対象世界を編成し，これを表記する過程は，データベースの概念設計と呼ばれる。概念設計では，実体型，実体間関連の型，これらの性質，および意味論的制約条件（semantic constraint）を決定し，対象世界の概念構造を定める。　これを適当な形式的言語で記述したものが意味論スキーマである。

対象世界のモデルの状態は，意味論スキーマの枠組みの中で変化する。

3.2.2 データベースの記述

対象世界における実体・関連に関する諸事実を，一定の記法に従って記号表現したデータの集合体がデータベースである。 レコード構造は，このように一定の書式をもつデータを表現するのに適している[90]。

レコード構造およびレコード間関係構造の表現法を定め，データベースの論理的編成法を与えることによって構造論的データモデルが構成される。特定の構造論的データモデルの枠組みに従ってデータベースを記述するための言語がデータ記述言語である。 対象世界を特定のデータ記述言語で記述することによって一つの論理スキーマができる。

論理データベースは論理スキーマに従って編成される。

3.2.3 データベースの意味論

構造論的データモデルは論理データベースを編成するための道具となり得るが，この道具を使ってデータベースの論理スキーマを作成する方法，およびこの論理スキーマからデータベースの内容を解釈する——データベース中の個々のレコード実現値が対象世界のどの実体のどういう性質を記述しているのかを理解する——方法については，ほとんど示唆を与えてくれない[67]。

データベースの意味論では，対象世界における実体や実体間関連と，データベースにおけるレコードやレコード間関係との間のかかわりのしくみを問題にする。 特に重要な問題として，実体・実体間関連の諸性質をレコード構造やレコード間関係構造を用いて表現する方法論，各々のレコードやレコード間関係の解釈を助けるためのメタ情報の与え方，意味論的制約条件に基づいてデータベースの更新を制御する方法論，などをあげることができる。

本論文では，次の問題だけを考察する。

3.3 網構造モデル

3.3.1 レコード構造

レコード実現値（record occurrence）あるいは単にレコードは，レコード記述に従って構成された一定系列のデータ項目値の集まりである。データ項目（data item）は意味をもつデータの最小単位である。レコード記述は，一意なレコード名と，いくつかのデータ項目記述を与え，一つのレコード型（record type）を定める。各々のデータ項目記述は，データ項目名，データ型，および空値（null value）の許可指定を与える。データ型はデータ項目値の表現形式やとりうる値の範囲を規定する。一つのレコード型の中でデータ項目名は一意でなければならない。本論文ではデータ項目記述の詳細には立ち入らない。

レコード名R，データ項目名 $I_1, I_2, \cdots, I_3$ をもつレコード型を，

$$R(I_1, I_2, \cdots, I_m)$$

で定義する。

データベース中のレコード実現値は一つかつただ一つのレコード型に属し，一意な内部識別子をもつ。レコード型Rに属するレコード実現値を

$$r_i(v_1, v_2, \cdots, v_m)$$

で表わす。ここで r_i は内部識別子，v_j はデータ項目 I_j のデータ項目値（$1 \leqq j \leqq n$）である。以後レコード実現値をその内部識別子で代表させる。

レコード実現値 r_i のデータ項目 I_j のデータ項目値を $r_i.I_j$ で参照する。

どの時点においても，一つのレコード型に属するレコード実現値の集合は一意に定まる。レコード型R に属するレコード集合を

$$Rec(R) \triangleq \{ r_i \mid 1 \leqq i \leqq n_R \}$$

で表わす。ここで n_R はデータベース中でレコード型Rに属するレコード実現値の個数である。

Rec(R) および個々のレコード実現値のデータ項目値は時間とともに変化する。

3.3.2 親子集合構造

親子集合記述は，一意な親子集合名，一つの親レコード型，一つの子レコード型，および親子結合属性を指定し，一つの親子集合型（set type, data structure set type）を定める。親レコード型と子レコード型は定義済でなければならない。親と子が同一のレコード型である親子集合型を再帰的親子集合型（recursive set type）と呼ぶ。

親子結合属性は強結合と弱結合の2種類がある。親子集合名S，親レコード型R，子レコード型Mの親子集合型を，親子結合属性が強結合の場合，

$$S : R \twoheadrightarrow M$$

で，弱結合の場合，

$$S : R \twoheadrightarrow\!\!\!\!\circ\, M$$

で定義する。

親子集合型Sに属する一つの親子集合実現値あるいは単に親子集合は，親レコード型Rに属する一つのレコード実現値 r_i と子レコード型Mに属する0個以上のレコード実現値 $m_1, m_2, \cdots, m_k$ $(k \geq 0)$ の集まりである。この親子集合実現値を

$$< r_i \mid m_1, m_2, \cdots, m_k >$$

で表わす。

親レコード型Rに属する各々のレコード実現値が一つの親子集合実現値を作る。子レコード型Mに属するレコード実現値は，親子集合型Sの高々一つの親子集合実現値に属し得る。従って，親子集合構造は親レコード型のレコード集合と子レコード型のレコード集合との間の1対多関係を表わしているといえる。

親子集合型Sの親子結合属性を強結合であると定義することは，子レコード型Mに属するすべてのレコード実現値がSの一つかつただ一つ

の親子集合実現値に属さなければならない，と宣言することを意味する。S の結合属性を弱結合であると定義すると，S のどの親子集合実現値にも属さない M のレコード実現値の存在を許したことになる。親子結合属性が強結合のとき，S は強い結合性をもつあるいは M は S に関して R に存在従属するという。弱結合のとき，S は弱い結合性をもつあるいは M は S に関して R に存在独立であるという。

再帰的親子集合は，同じ種類の実体を節点とする木——たとえば従業員の上司・部下の関係が作る木——を表現するために利用される。木の根は親をもたないので，再帰的親子集合型の親子結合属性は弱結合でなければならない。

親子集合型を，子レコード型のレコード集合から親レコード型のレコード集合への関数と解釈することもできる[20]。このような解釈の下で，強い結合性をもつ親子集合型は全域関数（total function），弱い結合性をもつ親子集合型は部分関数（partial function）となる。

親レコード型を R，子レコード型を M とする親子集合型 S に，親子集合実現値

$$< r \mid m_1, m_2, \cdots, m_k >$$

が存在しているとして，二つの関数

$$mems : Rec(R) \rightarrow 2^{Rec(M)}$$

$$Owns : Rec(M) \rightarrow Rec(R)$$

を次のように定義する。

$$mems(r) = \{ m_1, m_2, \cdots, m_k \}$$

$$Owns(m_i) = r \quad 1 \leqq i \leqq k$$

親子集合実現値の中の子レコード実現値の順序を子レコード順序と呼ぶ子レコード順序は子レコード実現値の属性である。そこで，親子集合型 S の子レコード型 M の中に，S の下での子レコード順序を表わす仮想的なデータ項目が暗黙裏に宣言されているものとし，これを #S で参照

する。 #S のデータ型は正の整数である。 S の子レコードでない M のレコード実現値に対して #S は空値をとる。 子レコード順序は，親子集合実現値の中への子レコード実現値の挿入の履歴などの情報を記録するのに有用である。

3.3.3 主キーと親子集合キー

レコード型が，それに属するレコード実現値を一意に識別できるデータ項目（あるいはデータ項目の組）をもっているならば，それを主キー（primary key）と宣言することができる。 強い結合性をもつ親子集合型 S の子レコード型のあるデータ項目（あるいはデータ項目の組）が，レコード集合 Rec（M）の中では一意性をもたないが，S の親子集合実現値の中では一意性をもつとき，このデータ項目を S の下での親子集合キー（set key）と宣言できる。

レコード型の定義において，主キー項目は下線で，親子集合キーは上線と親子集合名で示す。

3.3.4 情報保有型親子集合

親子集合型 S の親レコード型 R が主キーをもち，しかも子レコード型 M が R の主キー項目をもっているとき，S は非情報保有型（non-information bearing）であるという。 非情報保有型でない親子集合型は情報保有型（information bearing）であるという[25]。 S が非情報保有型ならば，S の定義を取り消しても S が表わしていたレコード間の一対多関係を保存できる。 ただし S の下での子レコード順序を情報表現の手段として利用している場合には，この情報を表わすデータ項目を M に追加する必要がある。

3.3.5 網構造モデル

網構造モデルでは，レコード構造と親子集合構造を用いてデータベースを編成する。 網構造スキーマは，一つのスキーマ名，一つ以上のレコード型，および 0 個以上の親子集合型から成り，データベースの論理構造を

規定する。網構造データベースは網構造スキーマに従って構成されたレコード実現値と親子集合実現値の集まりである。

網構造スキーマは，次の条件を満たしていなければならない。

1. 主キーをもたないレコード型は，少なくとも一つの親子集合キーをもたなければならない。
2. 輪構造を構成する親子集合型の少なくとも一つは弱い結合性をもたなければならない。輪構造は，あるレコード型から出発して，親子集合型を親レコード型から子レコード型へと順次たどっていったときに，もとのレコード型に達するような親子集合型の一つ以上の集まりから構成される。

この二つの条件は，データ項目値により特定のレコード実現値を一意に識別することを可能にするための条件である。

3.4 関係モデル

3.4.1 データベースの関係モデル

関係モデル（relational model）では，対象世界の各時点の状態を関係の集まりとして表現し，この時間とともに変化する関係の集りをデータベースであると定義する[3)]。

ここで「関係」とは，日常我々になじみのある大小関係，親子関係などの関係の概念を数学の集合論の立場から形式化した概念で，次のように定義される。

n個の空でない集合 $d_1, d_2, \cdots, d_n$（この中には同じ集合があってもよい）の直積のある部分集合 r，すなわち

$$r \subseteq d_1 \times d_2 \times \cdots \times d_n$$
$$= \{ \langle v_1, v_2, \cdots, v_n \rangle \mid v_i \in d_i,\ 1 \leqq i \leqq n \}$$

をn項関係（n-ary relation）あるいは単に関係と呼ぶ。ここでn個の集合 $d_1, d_2, \cdots, d_n$ のそれぞれを定義域（domain），nを関係 r の次数（degree），関係 r の一つの要素 $\langle v_1, v_2, \cdots, v_n \rangle$

をn個組（n-tuple）あるいは単に組と呼ぶ。

E.F. Codd の提案した関係モデルでは，定義域は構造をもたない——すなわちそれ自身がn個組や集合でない——要素の集合でなければならない。この条件を満たす関係を第1正規形（first normal form, 1NF）の関係と呼ぶ。

3.4.2 関係スキーマ

一つの関係の構造を記述したものを関係スキーム（relational scheme）と呼ぶ。データベースを構成するすべての関係の関係スキームの集まりを関係スキーマ（relational schema）と呼ぶ[81]。

一つの関係スキームは，一つの関係名，いくつかの属性名の集まり，および主キーを構成する属性の指定から成る。関係名 R，n個の属性名 $A_1, A_2, \cdots, A_n$，このうち最初のk個（$k \leq n$）の属性を主キーとする関係スキームを

$$R(\underline{A_1, A_2, \cdots, A_k}, A_{k+1}, \cdots, A_n)$$

と表記する。一つの関係スキームの中に二つの同じ属性名が存在していてはいけない。

主キーは関係の特定の組を一意に定める。すなわち，関係スキーム R に従って構成された関係 r に対して，あらゆる時点で次の条件が成立しなければならない，

r の任意のn個組 t_1, t_2 に対して，$t_1.K = t_2.K$ ならば $t_1 = t_2$ である。

ここで $t_i.K$ はn個組 t_i の主キー属性の値あるいは組の組である。

関係スキーム R のある属性（あるいは属性の組）X に対して，X の定義域（あるいは定義域の組）が関係スキーム R' の主キーの定義域（あるいは定義域の組）と同じであるとき，X を R の外来キー（foreign key）と呼ぶ。R' は R と同じであってもよい*。

---

* 例えば，R として従業員，X として上司を考えてみよ。

関係スキーム R に従って構成された関係 r のあらゆる組に対して，その外来キーの値と同じ主キー値をもつ組が，関係スキーム R' に従って構成された関係 r' の中に含まれていなければならない。

3.4.3 関係スキーマと関係データベース

関係データベースは関係スキーマに従って構成され維持される。各々の関係スキーム R に対して，その属性名を定義域に対応づける関数 Dom_R とその関係スキームの意味を表わす論理関数

$$Sem_R : Dom_R(A_1) \times \cdots \times Dom_R(A_n) \longrightarrow \{true, false\}$$

が定義されてあるものとする。

Sem_R は関係スキーム R に対して人間が与えた意味であり，$Sem_R(\langle v_1, v_2, \cdots, v_n \rangle)$ が真であるとは，n個組 $\langle v_1, \cdots, v_n \rangle$ が R に関して対象世界における正しい事実を表していることを意味する。

対象世界の状態は時間とともに変化し，それにつれて Sem_R も変化する。データベースを維持するとは，関係スキーマの中の各々の関係スキーム R に対応する関係 r に Sem_R の変化を反映させて，次式が常に成立するように r を更新することである。

$$r = \{ \langle v_1, \cdots, v_n \rangle \mid v_1 \in Dom_R(A_1) \wedge \cdots \wedge v_n \in Dom_R(A_n) \wedge Sem_R(\langle v_1, \cdots, v_n \rangle) \}$$

3.5 関係モデルと網構造モデルの比較

本節では，まず関係モデルと網構造モデルの基本的な類似点と相異点を示し，次に具体例をあげて両データモデルの論理データベースの表現能力を比較する。

3.5.1 基本的な類似点と相異点

主キーをもつレコード型のレコード集合は，関係モデルの関係の概念と等価である。関係モデルと網構造モデルでほぼ同じ概念を表わすと思われる用語の対応を表3.1に示す*。なお，以下の議論では，この用語の対応に基づいて網構造モデルの用語を主に用いる。

両データモデルは，ともにレコード構造を基本にしているという点で大きな類似性があるといえる。両データモデルの相異点はレコード間関係の表現方法にある。関係モデルでは，レコード間関係を表現する唯一の手段はデータ項目値の一致条件による結合である。網構造モデルでは，これに加えてレコード内部識別子によってレコード間の一対多関係を表現する親子集合構造を利用できる。

表3.1 関係モデルと網構造モデルの用語の対応

関係モデルの用語	網構造モデルの用語
関係スキーマ	網構造スキーマ
関係スキーム	レコード型
関係名	レコード名
関係	レコード集合
n個組	レコード実現値
属性	データ項目
主キー	主キー・親子集合キー
外来キー	親子集合型
定義域	データ型

* ほぼ同じ対応表は[1]にもみられる。

3.5.2 論理スキーマの例

ある製造会社で，部品の購入状況と従業員の管理を行なうために設計したデータベースの網構造スキーマ COMPANYDB を図3.3 に示す。

COMPANYDB

DEPT (DNO, DNAME, LOC)

EMP (ENO, ENAME, JOB, SAL)

SCH-CAR (E-S / DEG, SCHOOL)

PART (PNO, PNAME, COST)

SUPPLIER (SNO, SNAME, LOC)

DPS (QTY, D-DPS / #D-DPS)

USAGE (QTY, OUT / #OUT)

D-E : DEPT →→ EMP

E-S : EMP →→ SCH-CAR

D-DPS : DEPT →→ DPS

P-DPS : PART →→ DPS

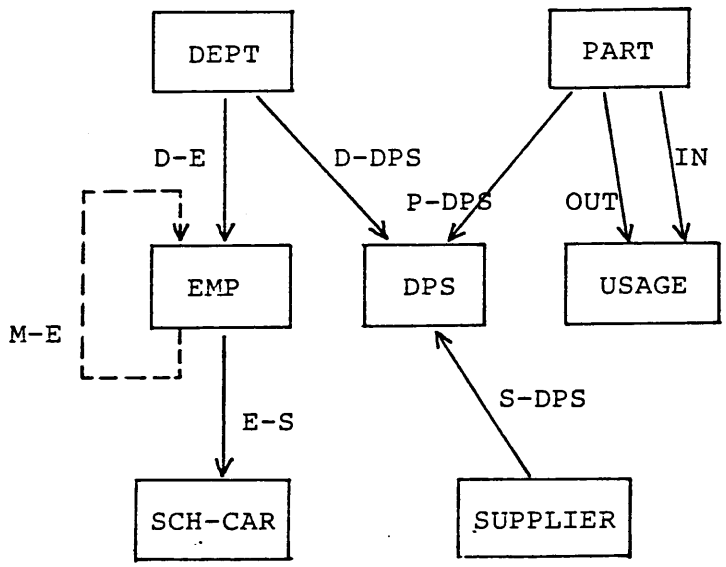
S-DPS : SUPPLIER →→ DPS

OUT : PART →→ USAGE

IN : PART →→ USAGE

M-E : EMP -→→ EMP

a) Network schema.



b) Data structure diagram.

Notes.

DEPT : Department record
EMP : Employee record
SCH-CAR : School carrier record
DPS : Link record

LOC : Location item
SAL : Salary item
QTY : Quantity item
DEG : Degree item
#D-DPS : Member order of DPS record in D-DPS set

Fig. 3.3 A network schema of a manufacturing company database ---- COMPANYDB.

図3.3 ある製造会社のデータベースの網構造スキーマ——COMPANYDB

部門（DEPT）は部品供給者（SUPPLIER）から部品（PART）を購入してある製品を作る。レコード型DPSは，どの部門がどの部品供給者からどの部品を何個（QTY）購入しているかを表すレコード型で，一般に結合レコード型（link record type）と呼ばれている。従業員（EMP）は一つかつただ一つの部門に所属し，いくつかの学歴レコード（SCH-CAR）をもっている。再帰的親子集合型M-Eは従業員間の上司・部下の関係を表わす。ここで各従業員は同時に2人以上の上司をもたないとした。レコード型USAGEは部品間の構成関係を表わす結合レコード型であり，どの部品がどの部品を何個使って構成されているかを表わす。USAGEの親子集合型OUTの下での子レコード順序#OUTは，OUTの下での親子集合キーとして宣言されている。

網構造スキーマからそれと等価な関係スキーマを構成するアルゴリズムはすでに知られている[41]。基本的な方針は，すべての親子集合型を非情報保有型に変換し，その後で親子集合型の定義を削除すればよい。

図3.3の網構造スキーマと等価な関係スキーマを図3.4に示す。

DEPT (DNO, DNAME, LOC)

EMP (ENO, ENAME, JOB, SAL, DNO, MGR)

SCH-CAR (ENO, DEG, SCHOOL)

PART (PNO, PNAME, COST)

SUPPLIER (SNO, SNAME, LOC)

DPS (DNO, PNO, SNO, QTY)

USAGE (MAJOR, MINOR, QTY)

Fig. 3.4 A relational schema equivalent to the network schema in Fig. 3.3.

図3.4　図3.3の網構造スキーマと等価な関係スキーマ

3.5.3 実体間関連の表現法

一般に，意味論的データモデルで実体として認識された概念は，構造論的データモデルではレコード構造として表現される[16) 68)]。しかし，実体間の関連として認識された概念の表現方法は多様である。従って，実体間関連の表現方法を検討することは，構造論的データモデルの特徴を知る最良の方法といえる。特に，親子集合構造を重要な構成要素とする網構造モデルの特徴は，関係モデルとの比較によって明確化する。

以下では，3種類の実体間関連について，両データモデルによる表現方法を詳細に検討する。

(a) 2種類の実体間の一対多関連あるいは階層構造の表現

3.5.2で与えた例では，部門，部品供給者，部品のそれぞれの実体がそれを一意に識別するための単純属性をもっているとしたので，関係モデルによる表現（図3.4）は簡素で分り易く更新の際の問題を生じない。しかし実体を一意に識別するための単純属性が存在しない場合には，関係スキーマの理解は困難になり，また冗長な表現が入り込むことによりデータベースの更新の際に大きな問題が生ずる。

たとえば，従業員の従業員番号が部門内では一意性をもつが，会社全体では一意性をもたないとしよう。図3.3の網構造スキーマに対する変更は，レコード型EMPの主キーENOを親子集合型D-Eの下での親子集合キーとするだけで良い。ところが図3.4の関係スキーマでは，関係スキームEMPとSCH-CARを次のように変更しなければならない。

EMP (<u>DNO, ENO</u>, ENAME, JOB, SAL, MDNO, MGR)

SCH-CAR (<u>DNO, ENO, DEG, SCHOOL</u>)

ここで，EMPのMDNOとMGRの組，およびSCH-CARのDNOとENOの組が外来キーとなる。この新しい関係スキーマでは，情報構造が不明確になり，さらに悪いことに冗長データが導入される。従業員の部門への所属関係が重複して表現されるので，従業員の所属部門が変更されるときに，この従業員の主キーの変更に加えて，この従業員のすべての

部下の属性 MDNO の値，および二の従業員のすべての学歴レコードの属性 DNO の値も変更しなければならない。 一貫性の維持が非常に困難になることが分かるであろう。 この例から階層構造を表現するために親子集合構造は非常に有用であることが理解できる。

関係モデルによるこのような階層構造の表現の不自然さは，情報保有型親子集合の必要性が主張される理由であり[25]，関係モデル第3正規形批判の根拠でもある[82]。

(b) 2種類の実体間の多対多関連あるいは3種類以上の実体間の関連

網構造モデルでは，階層構造は親子集合構造を使って明快に表現できるが，2種類の実体間の多対多関連あるいは3種類以上の実体間の関連の表現には工夫を要する。

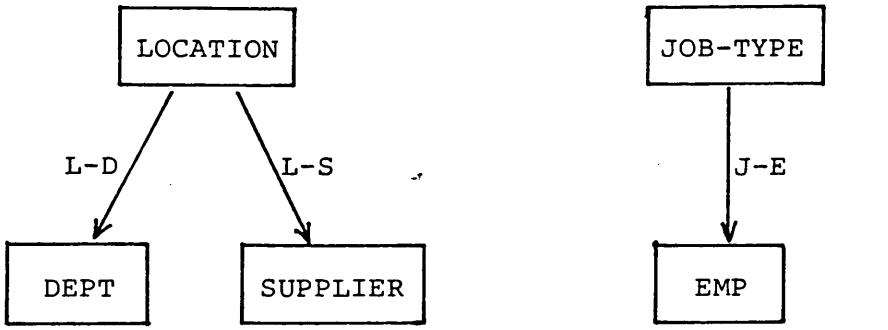
たとえば，3種類の実体——部門・部品供給者・部品——の間の部品購入という関連を表現するために，網構造モデルでは結合レコード型 DPS と三つの親子集合型 D-DPS，S-DPS，P-DPS を定義する。 これらは一つにまとまってはじめて意味をもつことができ，個々のレコード型，親子集合型がもつ意味は希薄である。 このように，一つの意味単位の記述を分散することに対しては，不自然さを感ぜざるを得ない。 また，このような結合レコードを介したレコード間関係の処理は非常に煩雑になる[8]。

一方関係モデルでは，部門と部品供給者と部品の間の部品購入関連は，一つの関係スキーム DPS によって表現できる。

(C) 共通の性質をもつ実体間の関連

ある共通の性質をもつ実体を関連づけることは情報検索の主要な機能要素である。 データベース処理における二の種の実体間関連の表現法としては，データ項目値の一致条件による方法が最も基本的で重要である。 たとえば，同じ所在地をとる部門と部品供給者の間の関連，同じ職種をもつ従業員間の関連などがこの例である。

もちろんこの種の実体間関連を親子集合構造で明示することも可能である（図3.5参照）。 しかしこのような表現は，「特定の利用目的・応用



LOCATION (LOC)　　JOB-TYPE (JOB)

a)　　b)

Fig. 3.5 Set representations of relationships among entities with common property.

図3.5 共通の性質をもつ実体間関係の親子集合による表現

業務に影響されない対象世界の公平・忠実な記述」を趣旨とする概念レベルの表現としては，不適当であると言わざるを得ない。

3.6 結 言

利用者インタフェースの構成法を考える上で，その土台となる概念レベルの検討は不可欠である。本章では，概念レベルの取扱い方を論じ，利用者インタフェース構成法を議論するための準備を整えた。

概念レベルの取扱いは，データベース管理システムの3層スキーマ構成が提案された当初から多くの論議を呼び，現在に至っても概念スキーマがどうあるべきかについての共通の理解が得られているとは言えない。議論を混乱させた原因の一つは，概念スキーマの二つの役割，すなわち

(1) 外部スキーマと内部スキーマの中継点としての役割

(2) 現実世界の忠実な記述を与えるという役割

を混同させていたことにある。

本章では，この二つの役割を分離し，それぞれを論理スキーマと意味論

スキーマに分担させることにより，概念レベルを明快に表現できるようにした。

次章より，論理スキーマを網構造モデルで表現した場合の利用者インタフェースの構成法について論ずる。その前準備として，3.3ではCODASYL DDLC 78提案による網構造モデルの諸概念に対して，意味論の取扱いを容易にするために新しい解釈を試み，いくつかの記法を導入した。親子結合属性，子レコード順序等の取扱いには特に工夫をこらした。

3.4では関係モデルについて述べ，3.5では関係モデルと網構造モデルの実体間関連の表現法を検討することを通じて，網構造モデルの次の特徴を明らかにした。

(1) 親子集合構造は現実世界で頻出する階層構造を表現するのに非常に有用である。

(2) 実体間関連を表現する方法は多様であり，実体間の多対多関連，3種類以上の実体間の関連の表現は不自然で複雑である。

次章では，以上の網構造モデルの特徴をいかし——すなわち長所をいかし，短所を隠す——，利用者の要求に合致したデータビューの記述を可能するデータモデル——CONTEXT データモデル——について述べる。

第 4 章　CONTEXT データモデル

4.1 緒言

本章では，網構造データベースに対する利用者のデータビューを表現するために新しく設計したCONTEXTデータモデルについて述べる。

利用者のデータビューを表現するための外部レベルのデータモデルを設計あるいは選定することは，データベース利用者インタフェースを考える上で，まず最初に検討しなければならない最も重要な問題である。高水準利用者インタフェースを構成するために，外部レベルのデータモデルはどのような性質をもつべきであろうか。本章では，まず最初に4.2で，「CONTEXT」ということばで筆者が表現しようと思っている概念を説明することによって，外部レベルのデータモデルに対する見解を明らかにする。

次に4.3でCONTEXTデータモデル設計の方針を提示し，後続する節でこのデータモデルの詳細な議論を展開する。

なお，本章における諸概念の具体的な記述例は，図3.3の網構造スキーマに基づいている。

4.2 「CONTEXT」の概念

筆者は，「CONTEXT」ということばを「特定の観点からながめた対象世界の特定の部分の状況・様子」と解したい。

対象世界に対して，いくつかの注目すべき特徴を定めることによって一つの観点が形成される。一つの観点を定めることによって一つのCONTEXTが形成される。CONTEXTを定めることにより，対象世界の個々の実体・実体間の相互関係・およびこれらの諸性質に対して明瞭な意味をもたせることができ，議論・考察・要求表現の環境が整備されることになる。目的に合致したCONTEXTを設定することができれば，実りの多い議論・順調な考察・簡潔な要求表現が可能になるであろう。

ところで，データベースの利用者は，特定の情報要求を満たすために対象世界に直接働きかけるのではなく，対象世界のモデルであるデータベー

スに対して要求を出す。　対象世界に対するCONTEXTの設定は認識を自由に働かせることによって可能である。　それでは，データベースに対するCONTEXTはどのように設定できるであろうか。　データベースに対するCONTEXTは，対象世界に対するCONTEXTを忠実にモデル化したものであることが望ましい。　すなわち，利用者の利用目的を良く反映し，要求に合致したデータビューを表現できることが望まれるのである。同時にそれは，2.2.3で述べた理由により，概念レベルのデータベースに無理なく写像可能でなければならない（図4.1参照）。

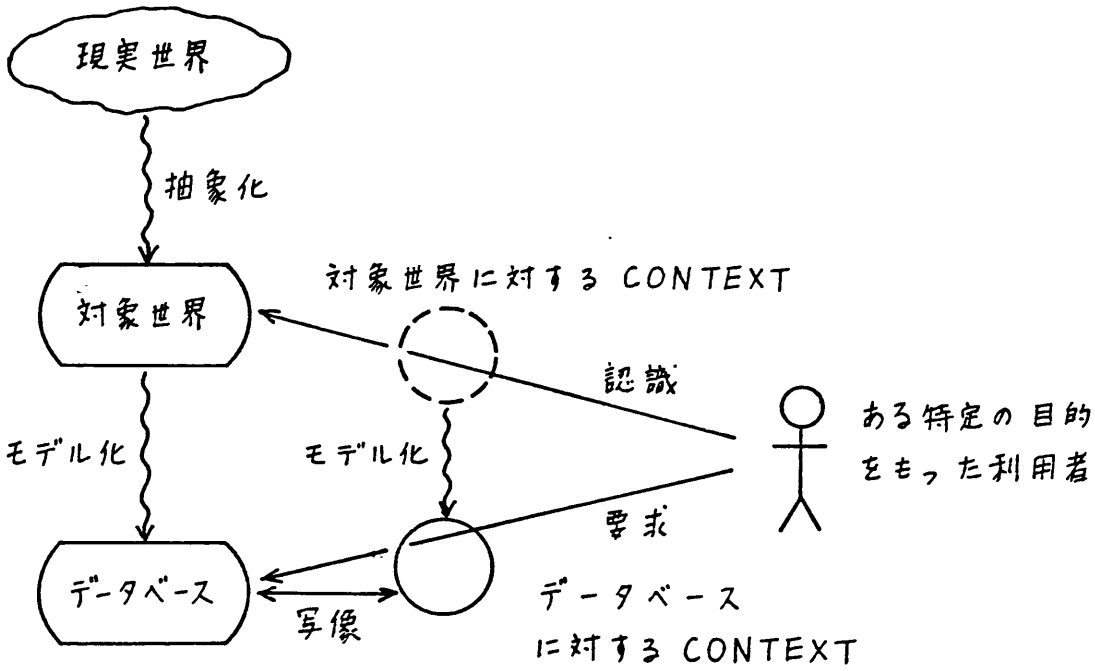


図4.1　「CONTEXT」の概念

ある特定の目的をもった利用者は，対象世界を一定の観点に基づいて認識し，対象世界に対して一定のCONTEXTを作る。これを忠実にモデル化・形式化したものがデータベースに対するCONTEXTである。

以上の議論から，「CONTEXT」の概念は外部スキーマのあるべき姿を示唆しているように思われる。すなわち，外部スキーマは概念スキーマの特定の部分を選択するというだけでなく，より積極的に，各々の利用目的に応じて要求表現のための適切な場——CONTEXTの語意——を提供しうることが望ましい，と筆者は考える。

このように要求記述に先立って，要求記述に適したCONTEXTをあらかじめ宣言的に与えてやることにより，

1. 複雑な要求でも簡潔に表現可能となり，
2. 最終利用者用インタフェースだけでなく応用プログラマ用インタフェースの高水準化を同時に達成することができる。

ANSI / X3 / SPARC 提案では，外部スキーマを「個々の応用業務から見たデータベースの記述」と定義している。「特定の目的をもつ利用者の要求に合致したデータビューの記述」という積極的な解釈はこれまでにほとんど見られない（2.5参照）。

関係モデルのビューの概念[83]はCONTEXTの概念に最も近いといえる。しかし第1正規形の平坦な表の構造は，複雑な要求の記述には向いていない*。平坦な構造に限定するよりも利用者の要求の意図を反映した構造を記述しうることが望ましい。次節ではこの問題を扱う。

---

* 関係モデルにおける，関係代数の割算[5]，関係計算の全称限定[5]，SEQUELのGROUP BY[36]等の機能は，平坦な表を分割あるいは階層化することに本質的な意味がある。一方この機能は，要求記述に際して最も誤りが入り込みやすく利用が困難な機能であるという報告がある[84]。A.L. Furtado らは，この問題の解決のために，関係モデルを拡張した二段の階層構造を基本構造とする分割関係とその上での代数を提案している[85]。

4.3 CONTEXT データモデル設計の方針

CONTEXT データモデル設計の目的は，網構造データベースに対して，利用者の利用目的に合致したデータビューを容易に表現できるような記述の枠組みを与えること，換言すれば網構造スキーマの上で「CONTEXT」を定義するためのデータ記述・写像言語の枠組みを与えることである。

この目的を実現するために，

1. CONTEXT の基本構造
2. CONTEXT を網構造スキーマに対応づける規則

を決定する必要がある。

4.3.1 CONTEXT の基本構造の決定

レコード構造は，対象世界における各々の実体がもつ諸性質を一つにまとめて表現する道具として，その単純さと表現力の面で最も優れていると思われる。 このような特性をもつレコード構造は，利用者のデータビューを表現する基本構成要素としても有用である。

レコード構造を基本構成要素とした場合，CONTEXT の基本構造としては，次の3種類が考えられる。

(1) ただのレコード構造（第1正規形の関係による平坦な表の構造）
(2) レコードを節点とする木構造
(3) レコードを節点とする網構造*

(1)の関係モデルの平坦な表の構造は，前節および 2.5.2 で述べた理由により，網構造データベースに対する利用者のデータビューとしては適当であるとはいえない。

また(3)の網構造は，利用者のデータビューとしては複雑すぎ，要求の記述が困難になる。 たとえば 3.5.2 で与えたデータベース COMPANYDB に対して，次の情報要求が出されたとする。

---

* CODASYL 方式の網構造モデルに限定しない。

ある部門が使用している各々の部品 P1 について，P1 を供給している部品供給者が供給している部品 P2，P1 の親部品 P3，P1 の子部品 P4 に関する情報を検索したい。

網構造を基本構造とする外部スキーマは，およそ図4.2のようになるであろう。この外部スキーマでは，部品という実体の各々の役割 P1, P2, P3, P4 が明示されていないために，要求記述の中で，どの経路をたどるのかの指示と同時に，これらの役割を区別する工夫が必要になる[52)~55)]。

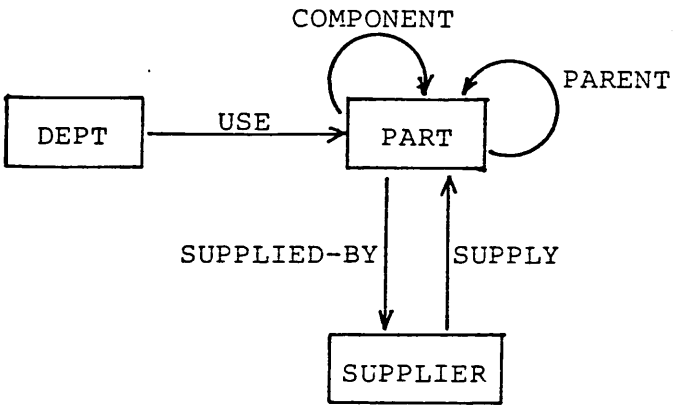


Fig. 4.2 A external schema in network model.

図4.2 網構造モデルによる外部スキーマ

同じ情報要求に対して，(2)の木構造を基本構造とする外部スキーマは図4.3のようになるであろう。この木構造スキーマでは，前述の網構造スキーマでの問題は，自然に解決されている。

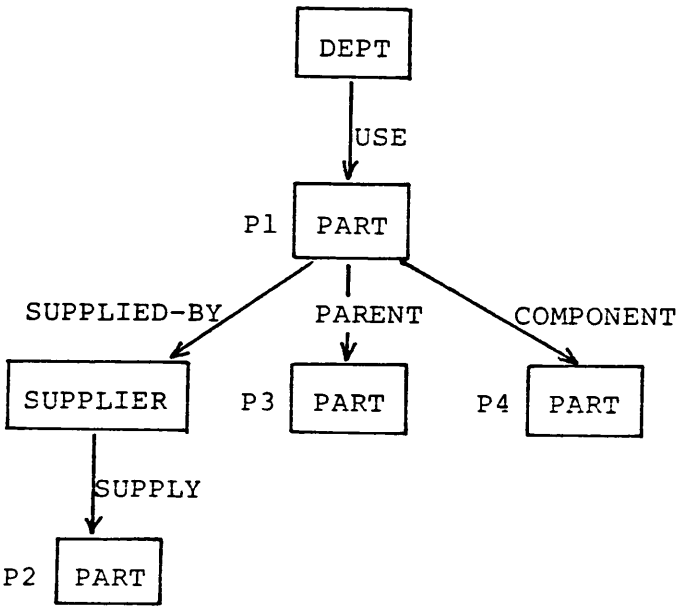


Fig. 4.3 A external schema in tree model.

図4.3 木構造モデルによる外部スキーマ

データベースに対して要求を表現する場合，一回の処理の対象となるレコード実現値群を関連づけ，一つの処理単位を構成するための機能が必要である。　処理単位の構造を木構造として宣言的に与えてやることにより，非常に明快なデータビューを得ることができる（4.6参照）。　要求表現の場として見ると，木構造モデルは関係モデルあるいは網構造モデルと比べて次の利点をもっている。

1. 関係モデルと比較して，レコード間の基本的な関係は構造的に表現できるので，処理要求を表現する際にデータ値の一致条件によるレコード結合を行なう必要がほとんどない。
2. 網構造モデルと比較して，二つのレコードの間には一つの関係しか存在しないので，要求表現の際にどの経路をたどるのかを指定する必要がない。

従って，木構造モデルでは，要求記述に際してデータ操作部での記述を最小限におさえることができる。

以上の議論より，CONTEXTの基本構造として，レコードを節点とする木構造が最も適当であると結論づけることができる。　この決定の妥当性は，本論文の以下の議論でしだいに明らかになるであろう。

4.3.2　網構造スキーマからCONTEXTを編成する方法

CONTEXTと網構造スキーマとを対応づける方法として，網構造スキーマから二段階に分けてCONTEXTを構成するという方針をとった。

まず，利用者にとって興味のあるレコード間の論理的関係を一様に表現するための手段としてアクセス関数の概念を導入した。　次にレコード構造とアクセス関数からCONTEXTを構成する規則を与えた。　ここで仮想レコード型，階層レコード型，n次のレコード間関係の階層化等の概念を導入し，レコード構造と親子集合構造によるデータ表現の制約を打開する具体的な手段を与えている。

このようにCONTEXTの記述を二段階に分けることによって，利用者のレベルに応じた利用者インタフェースの構成が可能になり，データベー

ス管理者と利用者の間で仕事を適切に分担できるようになる。

4.4 外部レコード型

データベースの特定の利用者は，データベース全体ではなくその特定の部分だけに興味がある。 従って，外部スキーマでは，まず最初に利用者にとって関心のあるレコード型の仕様を記述する。

外部スキーマでは，論理スキーマで定義されてあるレコード型の中から必要なものだけを選択し，それぞれに対して必要なデータ項目を指定する。 さらに望むならば名前の変更を行なう。 外部スキーマで定義されたレコード型を特に外部レコード型（external record type）と呼ぶことがある。

外部レコード型の記述は次の形式をとる。

RECORD CR => ER (CI1 => EI1, CI2, ...)*

ここで CR は論理スキーマにおけるレコード名，ER は外部スキーマで新しく定義しなおしたレコード名，CI1, CI2, … は論理スキーマにおけるレコード型 CR のデータ項目名，EI1 は CI1 の新しいデータ項目名である。 CI2 は外部スキーマでも同じ名前で参照される。

論理スキーマのレコード型 CR と全く同じ外部レコード型を定義したい場合，次の略記法がある。

RECORD CR (ALL)

以下の節では，図3.3 の網構造スキーマのすべてのレコード型，すべてのデータ項目が，名前の変更なしに外部スキーマで定義されてあると仮定する。

---

*予約語は下線で示す。

4.5 アクセス関数

二項関係を基礎にした網構造モデルでは，網構造のある節点を他の節点に対応づける機能をモデルの主要な構成要素とする場合が多い。この機能は，アクセス関数（access function）[66]，関連（association）[52] などと呼ばれている。本論文では，あるレコードを一定の論理的関係で結びつけられた別のレコードへ対応づける機能としてこの概念を導入する。

4.5.1 アクセス関数の概念

アクセス関数は二つのレコード型の間で定義される。一方を定義域レコード型（domain record type），他方を値域レコード型（range record type）と呼ぶ。両者は同じレコード型であっても良い。アクセス関数名 F，定義域レコード型 D，値域レコード型 R，のアクセス関数を次のように記述する。

ACCESS FUNC F : D => R : <af-spec>

ここで <af-spec> は，アクセス関数 F の実動化の仕様（後述）である。F を D から R へのアクセス関数と呼ぶことがある。

アクセス関数は，定義域レコード型の各々のレコード実現値を値域レコード型の 0 個以上のレコード実現値群へ対応づける。従って，定義域レコード型のレコード集合から値域レコード型のレコード集合の巾集合への関数と考えることができる。すなわち

$$F : Rec(D) \rightarrow 2^{Rec(R)} .$$

F によって $d \in Rec(D)$ から対応づけられるレコード型 R のレコード集合を $F(d)$ で参照する。

あらゆる $d \in Rec(D)$ に対して $F(d)$ が高々一つのレコード実現値から成るレコード集合であるとき，F は単一値アクセス関数，そうでないとき複数値アクセス関数と呼ぶ。単一値アクセス関数は，Rec(D)

から Rec (R) への部分関数と考えることもできる。

4.5.2 アクセス関数の実働化の仕様

アクセス関数は，実働化の仕様によって次のように分類できる。

1. 基本アクセス関数
   a) 順親子集合アクセス関数
   b) 逆親子集合アクセス関数
   c) 記号結合アクセス関数
2. 合成アクセス関数

記号結合アクセス関数は定義を明示し，その名前によって参照しなければならないが，その他のアクセス関数は実働化の仕様を参照すべき箇所に直接かくことができる。

a) 順親子集合アクセス関数

順親子集合アクセス関数は，定義域レコード型 D を親，値域レコード型 R を子とする親子集合型 S によって実働化されたアクセス関数であり，次のように記述される。

ACCESS FUNC F : D => R : S

F は，D に属する各々のレコード実現値 d を，R のレコード実現値群 Mems (d) へ対応づける。

b) 逆親子集合アクセス関数

逆親子集合アクセス関数は，定義域レコード型 D を子，値域レコード型 R を親とする親子集合型 S によって実働化されたアクセス関数であり，次のように記述される。

ACCESS FUNC F : D => R : *S

F は，D に属する各々のレコード実現値 d を，R のレコード実現値

Owns (α) へ対応づける。

c) 記号結合アクセス関数

記号結合アクセス関数は，定義域レコード型 D のデータ項目 ID と値域レコード型 R のデータ項目 IR を使って，次のように記述される。

ACCESS FUNC F : D => R : (ID = IR)

ここで，データ項目 ID と IR のデータ型は同じでなければならない。F は，D に属する各々のレコード実現値 α を R のレコード実現値群 $\{ r \mid r \in Rec(R) \wedge r.IR = \alpha.ID \}$ へ対応づける。比較すべきデータ項目対が複数の場合に拡張することは容易であろう。

d) 合成アクセス関数

合成アクセス関数は，定義済のアクセス関数から，アクセス関数合成規則を適用することによって構成できる。レコード型 $D = R_1, R_2, \cdots, R_{k+1} = R$ と，R_i から R_{i+1} へのアクセス関数 F_i ($1 \leqq i \leqq k$) が定義されてあるとする。このとき $F_1, F_2, \cdots, F_k$ を合成して得られる D から R へのアクセス関数 F は，次のように記述される。

ACCESS FUNC F : D => R : $F_1.F_2.\cdots.F_k$

k が 2 の場合，F は D に属する各々のレコード実現値 α を R のレコード実現値群 $\bigcup_{t \in F_1(\alpha)} F_2(t)$ へ対応づける。

$k \geqq 3$ の場合 $F_1.F_2.\cdots.F_k$ は $(\cdots((F_1.F_2).F_3).\cdots).F_k$ と解釈される。ここで $(F_i.F_j)$ は F_i と F_j の合成アクセス関数を意味する。

4.5.3 アクセス関数記述の例

(AF1) ACCESS FUNC SUPPLY : SUPPLIER => PART
: S-DPS.*P-DPS

SUPPLY は，レコード型 SUPPLIER から PART へ向うアクセス関数で，順親子集合アクセス関数 S-DPS と逆親子集合アクセス関数 *P-DPS の合成アクセス関数である。 SUPPLY は，SUPPLIER の各々のレコード実現値を，その部品供給者が供給している部品 PART のレコード実現値群へ対応づける。

(AF2) ACCESS FUNC EQLOCSUP : DEPT => SUPPLIER
: (LOC = LOC)

EQLOCSUP は記号結合アクセス関数の例である。 EQLOCSUP は，レコード型 DEPT の各々のレコード実現値を，その部門と所在地が同じ部品供給者 SUPPLIER のレコード実現値群へ対応づける。

(AF3) ACCESS FUNC QUICK-SUPPLIED-PART : DEPT => PART
: EQLOCSUP.SUPPLY

QUICK-SUPPLIED-PART は，定義済のアクセス関数，EQLOCSUP と SUPPLY の合成アクセス関数であり，レコード型 DEPT に属する各々のレコード実現値を，その部門と同じ所在地の部品供給者が供給している部品 PART のレコード実現値群へと対応づける

4.5.4 アクセス関数の概念の有用性

以下にアクセス関数の有用性を列挙する。

1. アクセス関数によって，二つのレコード型の間の論理的関係を一様に表現できる。 従ってレコード構造に基づくデータベースの統一的な表現を与えることができる。
2. アクセス関数は，何らかの物理的アクセス法によって実働化されるが，その詳細を利用者は一切気にする必要がない。 またアクセス関数の実働化の仕様の記述・変更は容易である。 従って高度のデータ

独立性を実現しやすい。

3. アクセス関数は論理的・意味論的概念であるから，利用者がこの参照を明示することには何の障害もないと思われる。一方データベース管理システムの側では，アクセス経路が明示されることによって実行時の最適化の処理をかなり容易にできることが期待できる。

4. 利用者は，あるレコードに関連するレコードをもとめるのにアクセス関数を用いる。これ以外の方法ではレコード向のアクセスは許されない。従って，アクセス関数の使用権を調整できる機構を設けることによって，データベースへのアクセスをきめ細かく制御することが可能である。

4.6 CONTEXT

本節では，レコード構造とアクセス関数からCONTEXTを構成する規則と記法について述べる。まず簡単な例から始め，順次複雑なCONTEXTを記述するための概念を導入していく。

4.6.1 単純CONTEXT

単純CONTEXTは，外部レコード型を節点，アクセス関数を矢とする有向木（これを木形アクセスパスグラフ，略してT-APGと呼ぶ）の各節点に対して，必要に応じてレコード選択式を付け加えて構成される。一つのCONTEXTの中に同じレコード型の節点が複数個存在しうる。この場合には，それらの節点を識別するために，適当な識別名を付けなければならない。この識別名を役割名と呼ぶ。

T-APGを通して見たデータベースのビューは，このT-APGの根レコード型に属する各々のレコード実現値を根とするレコード実現値の木の集まり，すなわち林である。網構造データベースはレコード実現値を節点とする網構造をもつが，T-APGを通すと，網構造をレコード実現値の重複をもたせて木状に展開した形で見える。

レコード選択式は，T-APGの各節点に対して，そのレコード型に属

するレコード実現値の中でCONTEXTから見ることのできるレコード実現値の条件を与える。 CONTEXTのある節点に付随するレコード選択式は，そのレコード型のデータ項目，その節点の祖先レコード型のデータ項目，および定数項等で作られる論理式である。

まず最初に，T-APGの根レコード型のレコード集合に対して，対応するレコード選択式が適用される。 選択条件を満足しないレコード実現値を根とする木は，このCONTEXTのデータビューから取り除かれる。条件を満たすレコード実現値は残され，その各々の子レコード実現値について同様の選択が繰り返される。 このようにして得られたデータビューをCONTEXTビュー（CONTEXT view）と呼ぶ。

CONTEXTを構成するレコード型は，論理スキーマにおけるレコード型あるいは外部レコード型と，意味論的に同一の概念を表わしているとはいえない。 普通は，論理スキーマにおけるレコード型は対象世界における実体の公平・忠実な記述を与える。 外部レコード型はその特定の属性に注目しているとはいえ，やはり実体の概念が中心にあると思われる。それに対して特定のCONTEXTの節点となるレコード型は，特定の観点からながめた実体の一側面の記述，一定の「CONTEXT」の下で色づけされた実体の記述を与える。 このように，CONTEXTを構成するレコード型は，実体の記述というよりもむしろ，そのCONTEXTにおける実体の役割を記述したものであると考える方が適当であろうと思われる。

このような理由から，CONTEXTの各節点を役割節点と呼ぶことにする。 CONTEXT記述の際に，その節点に新しく付ける名前を「役割名」としたのはこのような考えからである。 この考えをさらに進めて，CONTEXTの節点に新しく名前が付けられない場合には，その節点のレコード型のレコード名を役割名と考えることにする。

単純CONTEXTの記述例を以下に示す。

```
(C1)  CONTEXT  POOR-EMP-SPEC :
          EMP == E-S ==> SCH-CAR ,
          EMP == *D-E ==> DEPT ,
          EMP == *M-E ==> EMP / EMGR
          WHERE  EMP.SAL <= 50000
```

POOR-EMP-SPEC は，レコード型 EMP を根とし，その子としてレコード型 SCH-CAR，DEPT，EMP をそれぞれアクセス関数 E-S，*D-E，*M-E で結合して構成された T-APG に対して，根に当るレコード型 EMP にレコード選択式 EMP. SAL <= 50000 を付け加えたものである。この CONTEXT では従業員レコード型 EMP が2箇所に現れる。一方はこの CONTEXT の根であり，他方は，根の従業員レコード型からアクセス関数 *M-E で関連づけられたその従業員の上司を表わすレコード型である。両者を識別するために，後者には役割名 EMGR（Employee's MamaGeR）を付けてある。

CONTEXT は図式表現すると直感的理解が得やすい。そこで CONTEXT ダイアグラム（CONTEXT diagram）と呼ぶ図式表現法を導入する。CONTEXT ダイアグラムでは，役割節点を長方形，アクセス関数を矢で表わす。レコード名は長方形の中に，アクセス関数名あるいはその仕様は矢のそばにかく。レコード選択式は対応する役割節点の右側に，役割名は左側にかく。POOR-EMP-SPEC の CONTEXT ダイアグラムを図4.4に示す。

POOR-EMP-SPEC は，給与が5万円以下の従業員を根とし，その従業員の学歴レコード群，所属部門，および上司である従業員を子とする木状のデータビューを与える。POOR-EMP-SPEC の CONTEXT ビューを図4.5に示す。

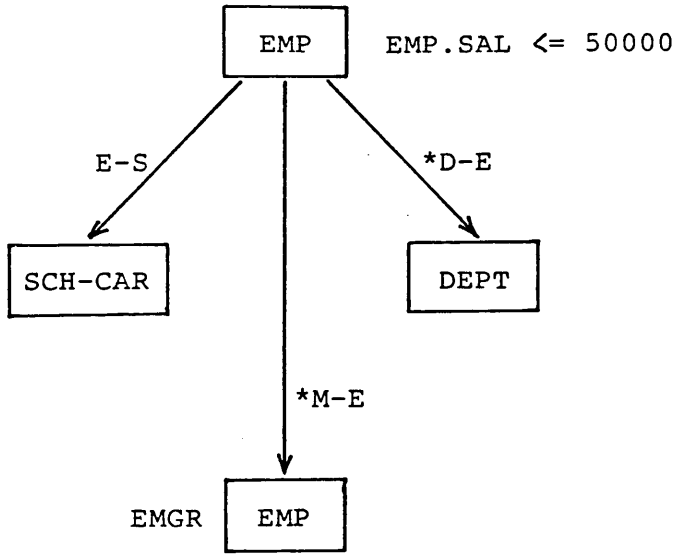
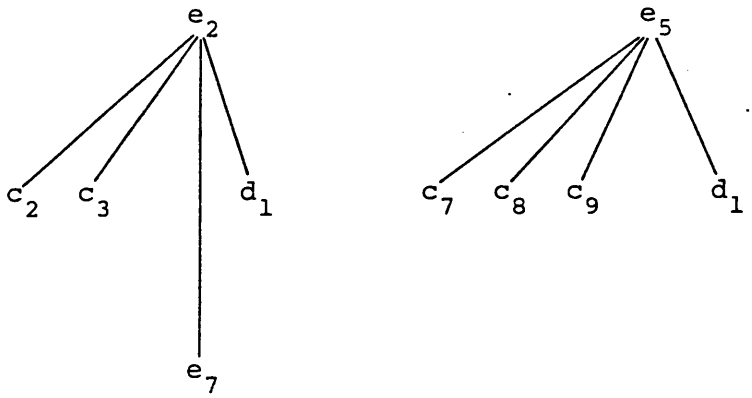


Fig. 4.4 A CONTEXT diagram of POOR-EMP-SPEC.

図4.4 POOR-EMP-SPEC の CONTEXT ダイアグラム



Notes. e_2.SAL <= 50000, e_5.SAL <= 50000

$\{e_2, e_5, e_7\} \subseteq$ Rec(EMP)

$\{c_2, c_3, c_7, c_8, c_9\} \subseteq$ Rec(SCH-CAR)

$\{d_1\} \subseteq$ Rec(DEPT)

E-S (e_2) = $\{c_2, c_3\}$

E-S (e_5) = $\{c_7, c_8, c_9\}$

*M-E (e_2) = $\{e_7\}$

*M-E (e_5) = $\{\ \}$

*D-E (e_2) = $\{d_1\}$

*D-E (e_5) = $\{d_1\}$

Fig. 4.5 A CONTEXT view of POOR-EMP-SPEC.

図4.5 POOR-EMP-SPEC の CONTEXT ビュー

(C2) CONTEXT DEPT-SUP-PART-1 :
DEPT == D-DPS.*S-DPS ==> UNIQUE SUPPLIER
== S-DPS.*P-DPS ==> UNIQUE PART

DEPT-SUP-PART-1 は，レコード型 DEPT を根とし，DEPT の子としてレコード型 SUPPLIER，SUPPLIER の子としてレコード型 PART をもつ。アクセス関数 D-DPS.*S-DPS によって，各々の DEPT レコード実現値は，その部門が部品を購入している部品供給者 SUPPLIER のレコード実現値群に関連づけられる。ここで SUPPLIER に対する UNIQUE の指定は，同一の DEPT レコード実現値の下で，同じ SUPPLIER レコード実現値の重複を取り除くことを意味する。アクセス関数 S-DPS.*P-DPS は，各々の SUPPLIER レコード実現値を，その部品供給者が少なくともーつの部門に供給している部品 PART のレコード実現値群に関連づける。

図4.6に網構造データベースの一部を，図4.7に DEPT-SUP-PART-1 の CONTEXT ダイアグラムとこの CONTEXT から見た図4.6のデータベースの CONTEXT ビューを示す。

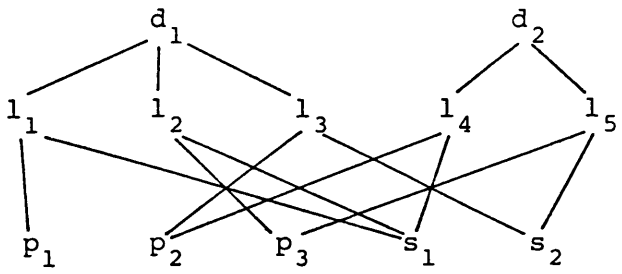
DEPT-SUP-PART-1 では，部門 d が部品供給者 s を子にもち，かつ s が部品 p を子にもつということから，部門 d が部品供給者 s から部品 p を購入しているという事実を導き出すことはできない点に注意したい。

4.6.2 仮想レコード型と階層レコード型

仮想レコード型は，あるレコード型のあるデータ項目を実体として解釈したいという場合などに有用である。

レコード型 $R(I_1, I_2, \cdots, I_n)$ のデータ項目 I_i に関する仮想レコード型は，R のレコード集合あるいはその部分集合の各レコード実現値から I_i のデータ値を取り出し，重複を取り除いて構成されるデータ項目値の集合を規定する。この仮想レコード型の記述を

$R(I_i) / X$



Rec(DEPT) = $\{d_1, d_2\}$
Rec(PART) = $\{P_1, P_2, P_3\}$
Rec(SUPPLIER) = $\{s_1, s_2\}$
Rec(DPS) = $\{l_1, l_2, l_3, l_4, l_5\}$

Fig. 4.6 A part of the COMPANYDB database.

図4.6 COMPANYDBデータベースの一部.

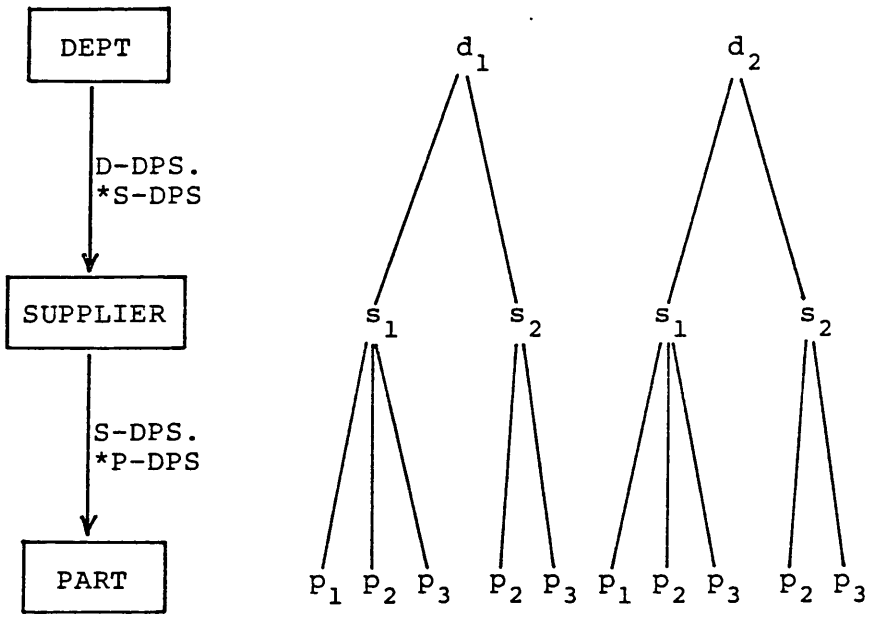


Fig. 4.7 A CONTEXT diagram of DEPT-SUP-PART-1 and a CONTEXT view for the network database in Fig.4.6.

図4.7 DEPT-SUP-PART-1 の CONTEXT ダイアグラムと図4.6 の網構造データベースに対する CONTEXT ビュー

で与える。　Xは仮想レコード型 $R(I_i)$ の参照名，すなわち役割名である。

あるレコード型に属するレコード集合あるいはその特定の部分集合を，あるデータ項目の値の同一性に基づいて類別し，各々のデータ項目値の同値類に対して集計的な処理を施すことが頻繁に行なわれる。　たとえば，「ハードウェア部に所属する従業員に対して，各職種ごとに平均給与を計算せよ。」というような要求がこの代表的な例である。　このような要求の表現に適したデータビューを与えるために，階層レコード型の概念を導入する。

レコード型 R を，データ項目 I_i について階層化した2段の階層レコード型は，仮想レコード型 $R(I_i)$ を上位，レコード型 R を下位とする2階層のレコード型であり，R のレコード集合あるいはその部分集合を，I_i の値の同一性に基づいて類別したデータビューを与える。　この階層レコード型を

$$R(I_i) / X \Longrightarrow R$$

で表わす。　$R(I_i)$ の各々のデータ値の下に，データ項目 I_i に同じ値をもつ R レコード群が集められる。

同様にして多段の階層レコード型を定義することもできる。　たとえば3階層の階層レコード型

$$R(I_i) / X_i \Longrightarrow R(I_j) / X_j \Longrightarrow R$$

は，R のレコード集合を，まずデータ項目 I_i によって分割し，その各々の類をデータ項目 I_j で分割したデータビューを与える。

階層レコード型の概念は，通常のファイル処理における整列化（sorting）に対応する。　また，関係モデル用の問い合せ言語 SEQUEL の GROUP BY 句に対応する[36]。

仮想レコード型と階層レコード型の概念は，レコードにではなくデータ

項目に注目したデータビューの記述を可能にする。　この二つの概念によって，レコード構造と親子集合構造から成る網構造モデルの表現力の制約から解放されることができた。

次に階層レコード型を用いたCONTEXTの記述例を示す。

(C3)　CONTEXT　DEPT-JOB-EMP :
　　DEPT == D-E ==> EMP (JOB) / EJOB ==> EMP

DEPT-JOB-EMPは，各部門ごとに，そこに所属する従業員を職種について分類したデータビューを与える。　DEPT-JOB-EMPのCONTEXTダイアグラムとCONTEXTビューの例を図4.8に示す。

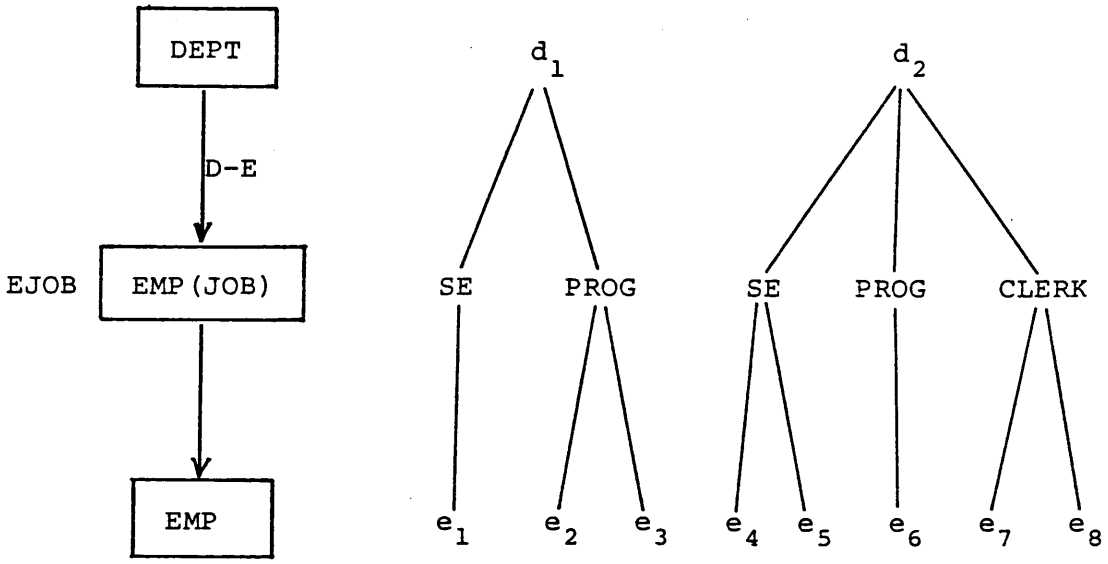


Fig. 4.8　A CONTEXT diagram and A CONTEXT view of DEPT-JOB-EMP
---- an example using hierarchical record type.

図4.8　DEPT-JOB-EMPのCONTEXTダイアグラムとCONTEXTビュー —— 階層レコード型を使用した例.

4.6.3 n次のレコード間関係の階層化

n個（$n \geq 3$）のレコード型の間の関係（n次のレコード間関係）を表現するために導入した結合レコードが作る網構造を木構造に展開するためには，後もどり処理（backtracking）が必要である。後もどり処理の指定をもつCONTEXT記述の例を次にあげる。

```
(C4)  CONTEXT  DEPT-SUP-PART-2 :
          DEPT == D-DPS.*S-DPS ==> UNIQUE SUPPLIER
               == @S-DPS.*P-DPS ==> PART
```

DEPT-SUP-PART-2は，各部門が各々の部品供給者からどの部品を購入しているか，を表わすCONTEXTである。図4.9に，図4.6の網構造データベースをDEPT-SUP-PART-2を通して見たときのCONTEXTビューを与える。アットマーク，@で後もどり処理の指定を行なう。レコード型SUPPLIERに対するUNIQUEの指定は，同一のDEPTレコー

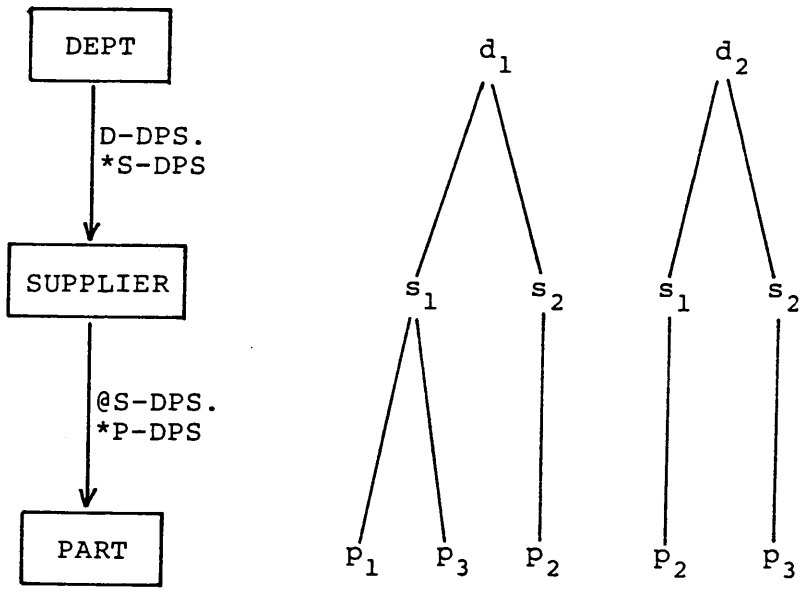


Fig. 4.9 A CONTEXT diagram of DEPT-SUP-PART-2 and a CONTEXT view for the network database in Fig.4.6 ---- an example using backtracking.

図4.9 DEPT-SUP-PART-2のCONTEXTダイアグラムと図4.6の網構造データベースに対するCONTEXTビュー ——後もどり処理を使用した例

ドの下で同じ SUPPLIER レコードを合体し重複を取り除く指定である。

(C2) の単純 CONTEXT DEPT-SUP-PART-1 の CONTEXT ビュー (図4.7) との比較をされたい。

4.6.4 再帰的 CONTEXT

再帰構造データの例として部品構成関係を考えてみよう。図4.10に部品構成関係の具体例を示す。この図で，各節点はそれぞれ異なる種類の部品を，矢は部品間の使用関係を表わす。$P_i \xrightarrow{k} P_j$ は，部品 P_i が部品 P_j を k個使用していることを表わす。

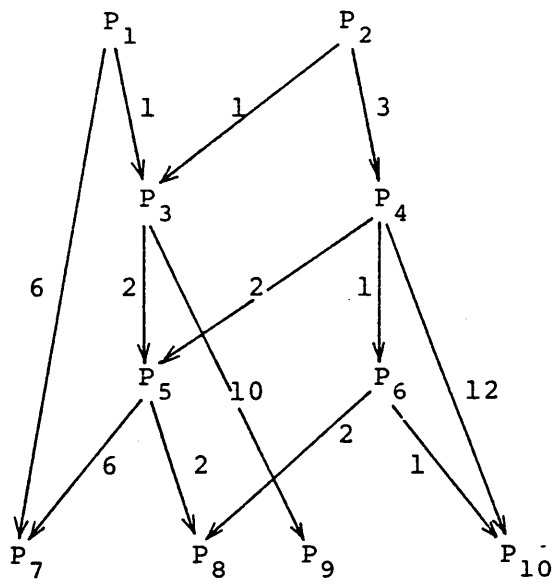


Fig. 4.10 A sample of recursive data ---- part usage relationship. $P_i \xrightarrow{k} P_j$ indicates that a part P_i uses k parts of P_j.

図4.10 再帰構造データの例——部品構成関係

この図から，節点の集合と矢の集合の間には，二種類の一対多関係が存在することが分かる。一方は節点とその節点から出て行く矢との関係 (OUT)，もう一方は節点とその節点に入り込む矢との関係 (IN) である。従って図4.10の部品構成関係の網構造スキーマは，節点すなわち部品を表わすレコード型 PART，矢すなわち部品の使用関係を表わすレコード型 USAGE，および PART を親，USAGE を子とする二つの親子集合型 OUT と IN から構成できることが分かる。

このような同質の節点から成る網構造 (homogemeous network) では，二つの節点の間の経路長を推定することが一般に不可能である。従って，同質節点から成る網構造を単純 CONTEXT のように固定された CONTEXT で表現するのは適当でない。再帰構造データを表現するた

めには再帰的な CONTEXT の記述が必要である。 再帰的 CONTEXT の記述例を次に示す。

```
(AF4) ACCESS  FUNC  COMP : PART => PART : OUT.*IN
(AF5) ACCESS  FUNC  PARENT : PART => PART : IN.*OUT

(C5)  CONTEXT  PART-TREE-1 :
          PART / P ==> *P
          WHERE NOT EXIST  P(1).PARENT
```

PART-TREE-1 の根である役割節点 P は，下位の役割節点の記述の中で，星印，* に続いて続いて現われており，見かけ上は閉路を形成しているが，役割名 P はレベル番号によって修飾され，たとえばレベル i の役割節点は P(i) で参照できる。 ここで根の役割節点のレベルを 1，

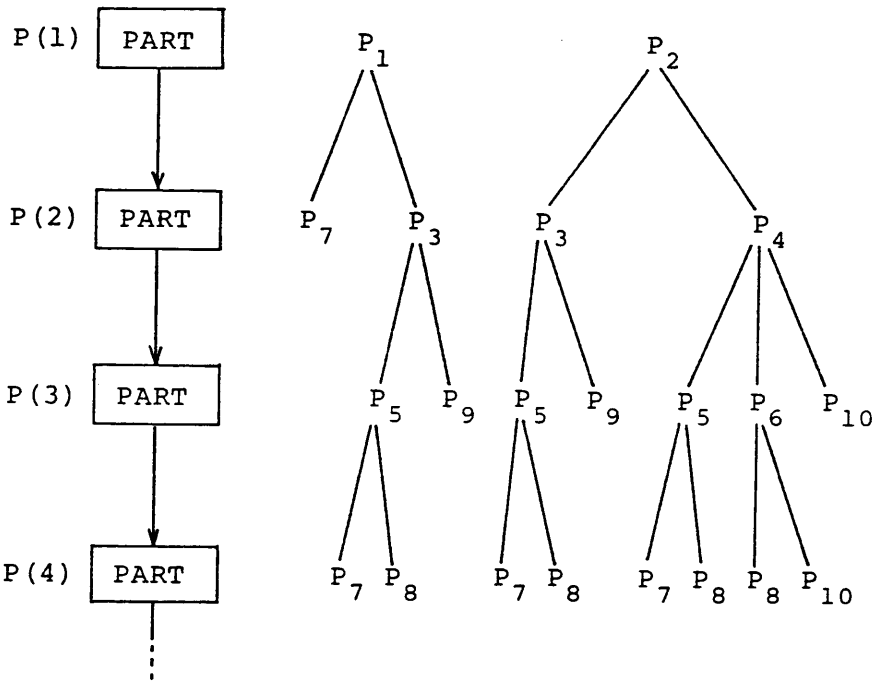


Fig. 4.11 A CONTEXT diagram and a CONTEXT view of PART-TREE-1 ---- an example of recursive CONTEXT.

図4.11 PART-TREE-1 の CONTEXT ダイアグラムと CONTEXT ビュー
——再帰的 CONTEXT の例

レベル i の節点の子のレベルを $i+1$ とする。 WHERE 句のレコード選択式では，P(1).PARENT すなわち根の役割節点 P(1) の親部品は存在しない，という条件を表わしている。

PART-TREE-1 の CONTEXT ダイアグラムと CONTEXT ビュー を図 4.11 に示す。

図4.11 から分るように，CONTEXT データモデルでは，網構造を展開して木状のデータビューを得るために，レコード実現値を重複させている。 しかしこのような重複節点に対する処理の重複を避けたい場合も考えられる。 たとえば，部品展開表を作成する場合，多くの親部品に含まれる子部品は各親部品に対して展開表が作られることになり，表の見易さが損なわれる。 そこで重複節点の出現を制御するために，次の2種類の制約文を導入する。

(a) DUPLICATE <role-name> IS LEAF

(b) DUPLICATE <role-name> IS NOT INCLUDED

(a) は重複節点の下位節点（子孫）の出現を抑制し，(b) は重複節点それ自身の出現も抑制する。 たとえば，次に定義する再帰的 CONTEXT PART-TREE-2 の CONTEXT ダイアグラムと CONTEXT ビュー は，図 4.12 のようになる。

```
(C6)  CONTEXT  PART-TREE-2 :
          PART / P == OUT ==> USAGE / U
                   == *IN ==> *P
          WHERE  NOT  EXIST  P(1).PARENT
          DUPLICATE  P  IS  LEAF
```

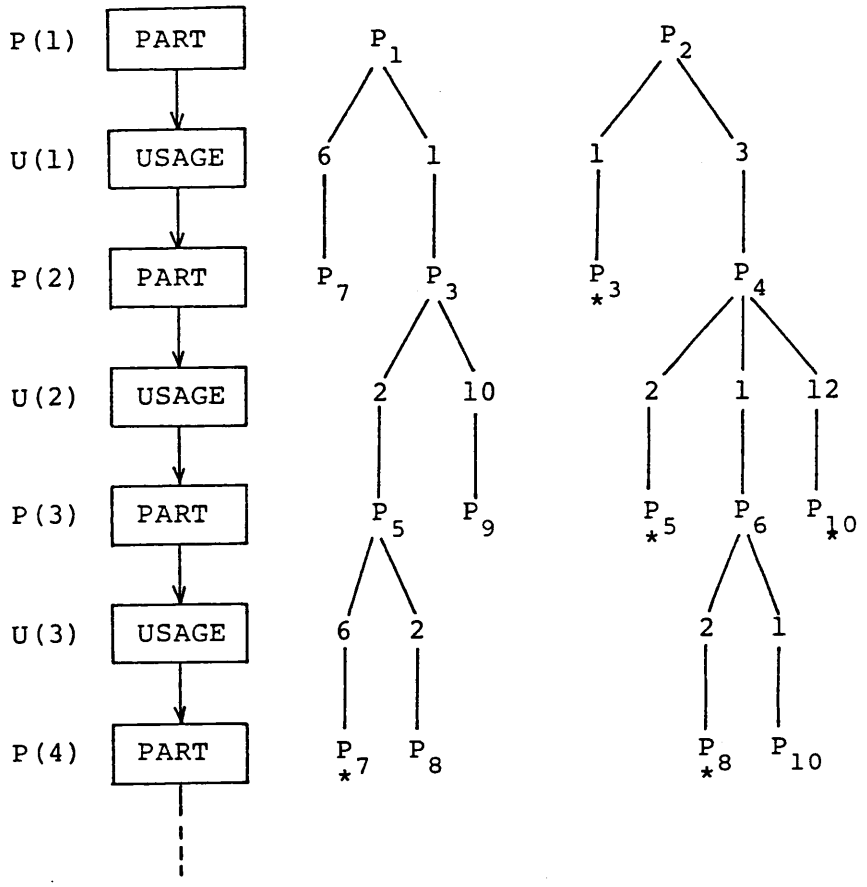


Fig. 4.12 A CONTEXT diagram and a CONTEXT view of PART-TREE-2 ---- an example of recursive CONTEXT with a control clause for duplicate nodes.

図4.12 PART-TREE-2 の CONTEXT ダイアグラムと CONTEXT ビュー ——重複節点の制御節をもつ再帰的 CONTEXT の例

4.7 関数定義機能

データベースから複雑な内容をもつ情報を容易に抽出するために，強力な導出データ項目の記述能力は有用である。

あるレコード型のデータ項目は，形式的にはそのレコード型のレコード集合から特定の単純データ型のデータ値集合への関数であると考えることができる。 従って，導出データ項目もあるレコード型（のレコード集合）からある単純データ型（のデータ値集合）への関数として定義することができる。

レコード型 R からデータ型 T への関数 F を次の形式で定義する。

FUNCTION F (X : R) : T = < expression >

ここで X はレコード型 R に対する仮引数，< expression > は，X あるいは R を定義域とするアクセス関数等から構成される式である。

次に関数定義の例を示す。

(F1) FUNCTION MGR-SAL (E : EMP) : INTEGER
= E.*M-E.SAL

MGR-SAL は，従業員レコード型 EMP から整数型 INTEGER への関数であり，引数として与えられた従業員に対してその上司である従業員の給与を対応づける。 *M-E は従業員からその上司である従業員への単一値アクセス関数であり，E.*M-E は従業員 E の上司を，E.*M-E.SAL はその上司の給与を表わす式である。 アクセス関数への引数の適用は，レコードのデータ項目参照の形式に従った。

再帰的 関数定義を許すことによって，再帰構造データベースから要約的情報 (summary information) を抽出することが可能となる。次に再帰的な関数定義の例を示す*。

(F2) FUNCTION TOTAL (P : PART) : INTEGER
= P.COST + SUM(U = P.OUT) (U.QTY * U.*IN.TOTAL)

TOTAL は，部品レコード型 PART から整数型 INTEGER への関数であり，引数として与えられた部品に対してその総価格を対応づける。 ここで，PART のデータ項目 COST は，素部品についてはその価格，組立て部品については組立てに要する費用を表わすものとする。

SUM (U = P.OUT) (…) は，PART から USAGE への複数値アクセス関数 OUT によってレコード型 PART の仮引数 P から関連づけられた

---

* この例は文献[64]から引用したものである。

それぞれのUSAGEレコード実現値Uについて，2番目の丸カッコ内の式を評価し，その結果の総和（SUM）をとった値を示す。　U.*INは，Uから単一値アクセス関数*INによって関連づけられた，PARTレコード実現値Pの一つの子部品を表す。　U.*IN.TOTALはその子部品の全価格を表わす。　ここでUをレコード変数と呼ぶ。

関数TOTALは再帰的に定義されている。　この再帰的な処理の終結はレコード集合P.OUTが空集合となったとき。　すなわち部品Pが素部品となる場合である。　このときSUMの値は0と解釈される。

関数TOTALを数学的な記法で定義すれば次のようになるであろう。

$$TOTAL(\ p\ ) = COST(\ p\ ) + \sum_{u \in OUT(p)} QTY(\ u\ ) \times TOTAL(\ *IN(\ u\ )\ )$$

第 5 章　問い合せ言語 QLC

5.1 緒言

前章では，網構造データベースに対して要求表現のための適切な場を設定するために，CONTEXT データモデルを提案し，CONTEXT 記述のための言語機能について述べた。本章では，CONTEXT を通じて網構造データベースを非手続き的に利用するために設計した問い合せ言語 QLC（Query Language for CONTEXT data model）について論述する。

木構造モデル用の問い合せ言語は，すでに実用化されているものを含め何種類かが提案されている[9) 86) 87)]。これらの問い合せ言語と比較して QLC は次の際立った特長を有する

1. 処理単位の概念を導入することにより，木構造データビューに対する問い合せを曖昧なく表現できる。また複雑な量操作を含む問い合せを明快に表現できる*。
2. 再帰的 CONTEXT に対して順行子を宣言することにより再帰構造データベースから構造に関する情報を抽出することが可能である。

---

* W.T. Hardgrave の報告によれば，従来の木構造モデル用問い合せ言語では，妥当と思われる検索条件の解釈のしかたが少なくとも 4 通り存在する[88)]。そのために，特に複雑な検索条件をもつ問い合せについて，利用者の要求の意図と問い合せ処理システムの解釈との間に食い違いが生ずる危険が高い。このような検索条件の曖昧さ・多義性は処理単位を明示することによって解消できる。

5.2 データベース処理の基本概念

5.2.1 データ選択

データベースに対する操作は，検索と更新に大別できる。 どちらの操作も，操作の種類の指定と同時に，操作すべき対象の種類と範囲を明示しなければならない。 データベースの中から関心のある特定の部分を他と区別し，操作すべき対象を限定していく過程をデータ選択（data selection）と呼ぶ。 データ選択はデータベース処理の根幹をなす。

データ選択の方法は，採用するデータモデルによって大きく異なり，さらに同じデータモデルに対しても種々の異なる方式が考えられる[40]。 しかしどのような選択の方式をみても，次の二つの基本的なパターンを認めることができる[9]。

(a) 内容に基づく選択（content addressibility）

データ値を指定し，一定の条件を満たす一定の単位のデータ実現値を選択する。 たとえば，所在地（LOC）が東京の部門（DEPT）のレコード実現値の選択。

(b) 関連構造に基づく選択（data relatability）

あるデータ実現値からデータ間関係によって結びつけられた別のデータ実現値を選択する。 たとえば，部門レコード実現値からその部門に所属する従業員（EMP）のレコード実現値群の選択。

データ選択は，以上2種類の基本パターンの組合せによって実現される。

CONTEXTデータモデルでは，データ選択の大部分をCONTEXT記述でうけもっている。 従ってデータ操作部での記述を最小限におさえることができる。 この性質は，特に複雑な構造をもつデータベースに対する複雑な要求を記述する場合に，きわめて好ましい性質であるといえる。

5.2.2 処理単位

データベース処理は，本質的にデータ実現値の集合に対する処理である。ある型に属し，ある条件を満足する一定の単位のデータ実現値のそれぞれ

に対して，ある共通の処理を施す。　これが最も基本的な処理パターンである。　ここで一回の処理の対象となる一定の単位のデータ実現値を処理単位（a unit to be processed）と呼ぶ。

単純な処理要求では，処理単位は単純な構造をもつ。　たとえば一つのレコード実現値が一つの処理単位となる。　しかし量限定（quantification）や集計演算（aggregate operation）を含む複雑な処理要求では，処理単位はより複雑な構造をもつ。　その最も典型的な構造は，レコード実現値を節点とする木構造である。

要求表現において処理単位を明示することは，要求の表現そ本自身を容易にし，また要求表現の意味の曖昧さを排除する上で重要であると考えられる。　しかしデータモデルの種類を問わず，処理単位を明示している問い合せ言語はみられない。

量限定や集計演算を含む問い合せの簡単な例を考えてみよう。

(a)　部門に所属する従業員の平均給与をもとむ。

(b)　すべての従業員の給与が10万円以上ならば，各部門の部門番号をもとむ。

(a)は，「平均をもとめる」という集計演算の演算対象となる従業員集合のとらえ方によって，いくつかの解釈が可能である。　たとえば

(a.1)　各部門ごとにその部門に所属する従業員の平均給与をもとめる。

(a.2)　会社で働いている全従業員の平均給与をもとめる

問い合せ言語は，両者の区別を明示できる言語機能を備えているべきである。　処理単位を指定することによって，このような問い合せの意味は自ずと明らかになる。　(a.1)の解釈に対しては，一つの部門とその部門に所属する従業員全体を一つの処理単位とし，(a.2)の解釈に対しては会社で働いている全従業員を一つの処理単位とすればよい。

(b)についても同様のことがいえる。　すなわち，全称限定子「すべての」の作用する範囲が会社全体なのか，それとも特定の部門内なのか曖昧である。　この場合も，処理単位を明示することによって曖昧さを除去できることは明らかであろう。

このように，処理単位の概念は，集計演算や量限定の範囲をはっきりさせ，複雑な要求を簡明に曖昧なく表現するために，きわめて有用な概念であるといえる。 CONTEXT データモデルでは，利用者の要求に合致したデータビューの記述が可能であるので，処理単位の指定は，CONTEXT の役割節点を指示するだけで十分である。

5.2.3 木の表記法

次節より問い合せ言語 QLC の説明に入るが，その前に木の表記法を約束しておこう。

レコード実現値を節点とする木を，次の二つの規則に従って表現する。

(a) レコード実現値 r は，r を根とする木である。

(b) n 個の木 $t_1, t_2, \cdots, t_n$ の根レコード実現値を子とし，レコード実現値 r を根とする木を $r(t_1, t_2, \cdots, t_n)$

たとえば，図4.9 の CONTEXT ビューを形成する二つの木は，それぞれ次のように表現できる

$$d_1(s_1(p_1, p_3), s_2(p_2))$$
$$d_2(s_1(p_2), s_2(p_3))$$

5.3 検索要求の表現

データベースから必要とする情報を抽出することをデータベースの検索と呼んだ。 データベース管理システムに対して，データベースの検索を依頼することを問い合せ（query）と呼ぶ。 問い合せを表現するための言語を問い合せ言語（query language）と呼ぶ。 問い合せ言語は，データベースの検索要求だけでなく，更新要求を記述する機能もっているのが普通である。

本節では，QLC の問い合せ機能について述べる。 更新機能については次節で論述する。

5.3.1 問い合せ文の基本構造

QLC では，キーワードによって処理の種類，CONTEXT，処理単位，および抽出すべきデータの仕様を指定する。問い合せ文は GET というキーワードで始まる。問い合せ表現の基本的枠組みを次に示す。

GET <target-list>
FROM <context>
FOR EACH <role-name> (<qualification>)

FROM 句で CONTEXT を指定する。この CONTEXT（C とする）の中で，興味の中心にある役割節点（N とする）を FOR EACH 句で指定する。N を注目役割（remarkable role）と呼ぶ。C の CONTEXT ビューの中で N のレコード型に属するレコード実現値を注目レコード（remarkable record）と呼ぶ。注目レコードを根とする部分木と，もしあったらその祖先レコード実現値が一つの処理単位となる。FOR EACH 句の代わりに FOR SYSTEM とかくと，C の CONTEXT ビュー全体が一つの処理単位となる。

例として（C4）で定義した CONTEXT DEPT-SUP-PART-2 を考えてみよう（図4.9参照）。FOR EACH 句で DEPT を指定すると，図4.9 の CONTEXT ビューで，d_1 と d_2 のそれぞれが注目レコードとなり，d_1 を根とする木と d_2 を根とする木のそれぞれが一つの処理単位となる。FOR EACH 句で SUPPLIER を指定すると，d_1 の子の s_1 と s_2 および d_2 の子の s_1 と s_2 のそれぞれが注目レコードとなり，次の四つの処理単位ができる。

$d_1 (s_1 (p_1 , p_3))$
$d_1 (s_2 (p_2))$
$d_2 (s_1 (p_2))$
$d_3 (s_2 (p_3))$

レコード限定式（qualification）は，注目レコードを子孫レコード実現値群の内容に基づいて限定する。一般に，同一レコード型の子レコード実現値は0個以上任意個存在するので，量限定（quantification）による限定が必要である。QLCにおける量操作の考え方は，注目レコードの子レコード実現値群に対する量操作の結果が注目レコードの属性になるということである。すなわち存在限定（SOME）や全称限定(ALL)の結果は注目レコードの真理値データとなり，集計演算の結果は注目レコードの数値データとなる。

GET句では，各々の処理単位から抽出すべきデータを表わす式をかく。この式は集計演算を含んでいても良い。

次に，図3.3の網構造スキーマに対する問い合せの例を示しながら，QLCの問い合せ機能を明らかにしていく。

5.3.2 単純な問い合せ

まず，単純な問い合せの例をあげよう

(Q1) 部門番号がD-25かD-47のいずれかの部門に所属し，給与が24万円以上の従業員の従業員番号，職種，給与，および所属部門番号をだせ。

```
(G1)  GET  DNO, ENO, JOB, SAL
      FROM DEPT == D-E ==> EMP
           WHERE  DNO = 'D-25' OR DNO = 'D-47' , SAL >= 240000
      FOR EACH  EMP
```

この問い合せの結果は，たとえば次の形式で表示されるであろう。

(T1)

DNO	ENO	JOB	SAL
D-25	E-10	SYSTEM ENG.	263100
D-25	E-15	PROGRAMMER	240500
D-47	E-32	TRUCKER	314000

この場合，処理単位ごとに一行の結果が出力される。

FOR EACH 句で DEPT を指定するか，または FOR SYSTEM とかくと，次の形式の表示が期待できる。

(T2)

DNO	ENO	JOB	SAL
D-25			
	E-10	SYSTEM ENG.	263100
	E-15	PROGRAMMER	240500
D-47			
	E-32	TRUCKER	314000

FOR EACH 句で DEPT と指定した場合，最初の3行が一つの処理単位 d_{25} (e_{10} , e_{15}) に対する結果である。ここで d_i は DNO が D-i の部門レコード実現値の内部識別子を，e_j は ENO が E-j の従業員レコード実現値の内部識別子を表わす。

同じ問い合せを，異なる CONTEXT に対して表現することもできる。

```
(G1') GET  DNO, ENO, JOB, SAL
      FROM  EMP == *D-E ==> DEPT
            WHERE  DNO = 'D-25' OR DNO = 'D-47' , SAL >= 240000
      FOR EACH  DEPT
```

この問い合せ結果は (T1) と同じになる。

(G1') の FOR EACH 句で EMP を指定すると，(T1) とも (T2) とも異なる結果が出力されることになる。たとえば次のように

(T3)

DNO	ENO	JOB	SAL
*	E-5	MANAGER	456000
D-25	E-10	SYSTEM ENG.	263100
D-25	E-15	PROGRAMMER	240500
*	E-24	ERECTRIC ENG.	275000
D-47	E-32	TRUCKER	314000

この場合，給与が 24 万円以上の従業員のそれぞれに対して一つの処理単位が形成される。その従業員の所属部門の部門番号が D-25 でも D-47 でもない場合には，その処理単位の中に部門レコード実現値は含まれない。

そのために部門番号の参照に対して空値（星印で表示）が返されることになる。　ただし役割節点 EMP に対するレコード選択式に条件式 EXIST EMP.＊D-E を追加すれば，(T1) と同じ結果が得られるであろう。　この条件式は，EMP.＊D-E すなわち従業員の所属部門が存在するという条件を表わす。

(Q2)　各部門における従業員の最低の給与に対して，会社全体での平均をもむ。

この要求に応えるためには，部門レコード型に対して；所属する従業員の最低給与を与える導出データ項目 MIN-SAL を次のように定義しておくと良い。

```
(F3)  FUNCTION  MIN-SAL ( D : DEPT ) : INTEGER
          = MIN( E = D.D-E ) ( E.SAL )
```

この導出データ項目を使って (Q2) は次のように簡単に表現できる。

```
(G2)  GET  AVG ( DEPT.MIN-SAL )
      FROM  DEPT
      FOR SYSTEM
```

FOR SYSTEM の指定により，レコード型 DEPT に属するすべてのレコード実現値の全体が一つの処理単位になる。　AVG は，この処理単位に含まれる部門レコードの MIN-SAL 項目に対する平均値をもめる集計演算子である。

5.3.3 量操作を伴う複雑な問い合せ

(Q3) 各部門について，職種別に従業員の平均給与をもとむ。
この問い合せは (C3) で定義した DEPT-JOB-EMP (図4.8参照) を使って，次のように表現できる。

```
(G3)  GET  DNO, EJOB.JOB, AVG( SAL )
      FROM  DEPT-JOB-EMP
      FOR EACH  EJOB
```

FROM 句では，CONTEXT を新たに定義してもよいし，この問い合せのように定義済の CONTEXT を参照することもできる。

(Q4) すべての部品供給者から2種類以上の部品を購入している部門の部門番号と部門名をもとむ。
この問い合せは (C4) で定義した DEPT-SUP-PART-2 (図4.9参照) を使って，次のように表現できる。

```
(G4)  GET  DNO, DNAME
      FROM  DEPT-SUP-PART-2
      FOR EACH  DEPT ( ALL SUPPLIER ( COUNT( PART ) >= 2 ) )
```

FOR EACH 句で指定された役割節点 DEPT に属する各々の部門レコード実現値が注目レコードになる。この注目レコードは，レコード限定式 (ALL SUPPLIER (···)) によって限定される。この限定式は，注目レコードが次の条件を満足しなければならないということを主張している。すなわち，注目レコードの子である SUPPLIER レコード実現値のすべてが，2番目の丸カッコ内の条件を満足する，という条件である。2番目の丸カッコ内の条件 COUNT (PART) >= 2 は，SUPPLIER レコード実現値の子となる PART レコード実現値の個数が2以上であることを意味する。

(Q5) 部門番号が D-50 の部門が使用している部品をすべて供給している部品供給者のすべてのデータ項目をおむ.

この問い合せを表現するためには，部門が部品を使用しているという事柄を表わす CONTEXT と，部品供給者が部品を供給しているという事柄を表わす CONTEXT の二つが必要である。

```
(G5)  GET  SUPPLIER
      FROM  SUPPLIER == SUPPLY ==> PART / PS ;
            DEPT == D-DPS.*P-DPS ==> PART / PD
      FOR EACH  SUPPLIER  SOME  DEPT ( DNO = 'D-50' )
      HAVING  SET( PS ) >= SET( PD )
```

ここで SUPPLY は (AF1) で定義したアクセス関数である。

複数の CONTEXT にまたがる条件式は HAVING 句で指定する。一般に，各々の CONTEXT に対して処理単位は複数個存在し得るので，HAVING 句の条件式に対して量限定を行なう必要がある。

(G5) は次のように読むことができる。

各々の部品供給者に対して，部門番号 D-50 のある部門が存在し，その部品供給者の供給している部品の集合 (SET (PS)) がその部門の使用している部品の集合 (SET (PD)) を含む (>=) ならば，その部品供給者の情報を取り出せ。

この例で示したように，GET 句で役割名を指定すると，そのレコード実現値のすべてのデータ項目が出力される。

(Q6) 取得したすべての種類の学位を同一の学校から受けている従業員の名前をおむ。

```
(G6)  GET  ENAME
      FROM  EMP == E-S ==> SCH-CAR ( SCHOOL ) / SCX
                       ==> SCH-CAR ( DEG ) / DX ,
            EMP == E-S ==> SCH-CAR ( DEG ) / DY
      FOR EACH  EMP ( SOME  SCX ( SET( DX ) = SET( DY ) ) )
```

この問い合せ表現はCONTEXTデータモデルおよびQLCの特徴を最も良くだしている。これは次のように読むことができる。

各々の従業員に対して，「ある出身校があって，そこで取得した学位の集合が，その従業員が取得した全学位の集合に等しい」ならば，その従業員の名前を取り出せ。

「」内がレコード限定式の解釈である。要求が非常に明快に表現できることが理解できるであろう。

5.3.4 再帰構造データベースに対する問い合せ

再帰構造データベースに対する利用者のデータビューを表現するために，4.6.3において再帰的CONTEXTの概念を導入した。ここでは，再帰的CONTEXTを通じて再帰構造データベースから情報を抽出する方法について述べる。

再帰構造を取り扱うために，QLCでは順行子（traverser）の概念を導入した。順行子の宣言はTRAVERSER句で行なう。TRAVERSER句は次の形式をとる。

TRAVERSER <role-name> (<level-var>)
(<qualification>)

順行子は，再帰的CONTEXTの繰り返し役割節点（<role-name>）とレベル変数（<level-var>）を指定することによって定まる。順行子は，再帰的CONTEXTのレベル番号付き役割節点を自由に上下しながら，再帰的CONTEXTビューのレコード実現値の木の中を上から下へ左から右へと順行する（traverse）かのようにふるまう。順行子が訪問するレコード実現値の満たすべき条件を条件式（<qualification>）で指定する。条件式では，レベル変数に対する条件およびレコード実現値のデータ項目値に対する条件を記述できる。特に再帰構造データベースの構造に関する情報を得るために，レベル変数に関する次の二つの論理関数が有用である。

LEAF(L) ≡ Lは葉レベルである。

LOWEST(L) ≡ Lは最下位レベルである。

図4.11 のCONTEXT ビューにおいて，順行子がレコード実現値 P_7，P_8，P_9，P_{10} のいずれかに滞在しているとき LEAF(L) は真となり，第4レベルのレコード実現値のどれかに滞在しているとき LOWEST(L) は真となる。

GET 句で順行子・レベル変数を参照すると，処理単位ごとに，指定された条件式を満足するレコード実現値・そのレベル番号の値の集まりが返される。

以下に，再帰構造データベースに対する問い合せ例を示す。これらは(C5)で定義した PART-TREE-1 (図4.11 参照) を使用する。

(Q7) 第1レベル* の各々の部品に対して，全子孫部品のレベル番号と部品番号をだす。

```
(G7)  GET  P(1).PNO, L, P(L).PNO
      FROM  PART-TREE-1
      FOR EACH  P(1)
      TRAVERSER  P(L)( L >= 2 )
```

これは2項関係の推移閉包 (transitive closure) をもとめる操作に対応している。

(Q8) 第1レベルの各々の部品に対して，直接的あるいは間接的に使用しているすべての素部品のレベル番号と部品番号をだす。

---

*親部品をもたない部品を第1レベルの部品とし，第iレベルの部品の子部品のレベルを i+1 とする。

(G8) GET P(1).PNO, L, P(L).PNO
FROM PART-TREE-1
FOR EACH P(1)
TRAVERSER P(L) (LEAF(L))

(Q9) 第1レベルの各々の部品に対して，それが部品名 GEAR の部品を含んでいるならば，そのレベル番号と部品番号をもむ。

(G9) GET P(1).PNO, L, P(L).PNO
FROM PART-TREE-1
FOR EACH P(1)
TRAVERSER P(L) (PNAME = 'GEAR')

再帰構造データすなわち同質の節点から成る網構造は，図4.11 あるいは図4.12 に示すように，木状に展開した表現をとることによって，取り扱いが容易になるばかりでなく，この表現構造を保存したまま出力すれば有用な文書情報が得られる。そこで，再帰的 CONTEXT の CONTEXT ビューをそのままの形で出力するための言語機能を設けることにしよう。次の問い合せはこの言語機能の使用例である。

(Q10) 第1レベルの部品を根とする5レベルまでの部品展開図をもむ。

(G10) GET TREE(P(L).PNO, U(L-1).QTY)
FROM PART-TREE-2
FOR EACH P(1)
TRAVERSER P(L) (L <= 5)

L=1 のとき U(L-1) = U(0) となり，この式の評価の結果は空値となる。

5.4 更新要求の表現

対象世界の状態の変化をデータベース中のデータに反映させることをデータベースの更新と呼んだ。データベースの更新は，データの格納，修正，および消去の3種類に分類できる。本節では，CONTEXT を通じてデータベースの更新を行なうためのQLCの言語機能について述べる。

5.4.1 データの格納

データベースへのデータの格納の単位はレコード実現値の木である。QLCでは，レコード実現値の格納と同時に，親子集合実現値の生成および既存の親子集合実現値への子レコード実現値の挿入を一度に行なう。

(U1) 部門番号 D-55 の部門に，新しく採用した従業員を，従業員番号 E-10 の従業員の部下として配属する。この従業員の従業員番号，名前，および学歴は分っているが，職種，給与は未定である。

```
(S1) STORE INTO DEPT == D-E ==> EMP == E-S ==> SCH-CAR ,
                                EMP == *M-E ==> EMP / EMGR
     : *DEPT( DNO = 'D-55' )
       EMP( ENO = 'E-20', ENAME = 'TANAKA' )
       SCH-CAR( 'SHIZUOKA', 'BE' )
       SCH-CAR( 'SHIZUOKA', 'ME' )
       *EMGR( ENO = 'E-10' )
```

データ格納要求の記述では，格納要求を意味する STORE INTO というキーワードに続けてCONTEXTを指定し，次に格納すべきレコード実現値の木の仕様を記述する。レコード実現値の木の仕様において，役割名の次の丸カッコ内をレコード仕様と呼ぶ。役割名の前に星印をもつレコード仕様を位置決め節点（locating node），もたないレコード仕様を格納節点（storing node）と呼ぶ。位置決め節点では主キーの値あるいは親子集合キーの値を指定する。位置決め節点は，データベースに既に格納されてあるレコード実現値を探索するために用いられる。格納節点は，データベースに格納され，探索によって位置決めされたレコー

ド実現値に接続される。　外部レコード型の中で宣言されたすべてのデータ項目についてデータ値を指定する場合には，上の例のSCH-CARのレコード仕様のようにデータ項目名を示さなくても良い。　この場合，データ項目値の順序は，外部レコード型のデータ項目の順序に従う。

5.4.2　データの消去

データベースからのデータの消去の単位は処理単位の注目レコードおよびその子孫レコード実現値である。　消去すべきレコード実現値に対応する役割節点をすべて指定しなければならない。　消去の指定のないレコード実現値は，親子の接続が切断されるがデータベースから消去されない。

(U2)　部門番号 D-25 の部門に所属するプログラマおよびそのプログラマの学歴レコードを消去せよ。

```
(D1)  DELETE  EMP, SCH-CAR
      FROM  DEPT == D-E ==> EMP == E-S ==> SCH-CAR
            WHERE  DNO = 'D-25', JOB = 'PROGRAMMER"
      FOR EACH  EMP
```

消去される従業員レコードが，親子集合型 M-E に対して親レコード（この従業員の上司の従業員レコード）あるいは子レコード（この従業員の部下の従業員レコード）をもっている場合，その親子の接続は自動的に切断される。

5.4.3　データの修正

データベースの修正の単位は注目レコードのデータ項目値である。

(U3)　部門番号 D-55 の部門に所属する従業員の給与を一律 10 %昇給せよ。

```
(M1)  MODIFY  SAL = 1.1 * SAL
      FROM  DEPT == D-E ==> EMP
            WHERE  DNO = 'D-55'
      FOR EACH  EMP
```

網構造データベースでは，たとえば従業員の所属部門の変更のような，親子集合構造の子レコード実現値の所属親を変更するという要求は記述が困難である。　その理由は親子集合選択基準の指定の複雑さによるもので，この複雑さは親子集合構造の批判の対象とされてきた

情報表現の手段として結合子（link）を用いるデータモデルでは，結合子の生成・消去・変更の記述が煩雑になることは避けられないと思われる。　一方関係モデルでは，外来キー（3.4.2参照）を構成する属性（データ項目）の修正によって，結合子の生成・消去・変更に相当する要求を簡潔に表現できる。　結合子の更新要求は関係モデル風のデータビューに対して記述するのが最適であると思われる。

4.7で導入した関数定義機能を用いて，網構造モデルの上に関係モデル風のインタフェースを構成することは容易である。　たとえば次の関数定義を考えてみよう。

```
(F4)  FUNCTION  DEPT-NO ( E : EMP ) : CHAR( 5 )
          = E.*D-E.DNO
```

DEPT-NO はレコード型 EMP の導出データ項目と解釈できるので，レコード型 EMP は次のようになる

```
EMP ( ENO, ENAME, JOB, SAL, DEPT-NO )
```

このレコード型を通して，従業員の所属部門の変更要求は，次の例が示すように簡潔に表現できる。

(U4)　部門番号 D-55 の部門に所属する従業員を，部門番号 D-40 の部門へ移せ。

```
(M2)  MODIFY  DEPT-NO = 'D-40'
      FROM  DEPT == D-E ==> EMP
            WHERE  DNO = 'D-55'
      FOR EACH  EMP
```

5.3.4 更新要求に対する制約

更新要求の記述は，どの CONTEXT に対しても許されるというわけにはいかない。 たとえば，次の格納要求は許されない。

```
(S2)  STORE  INTO  SUPPLIER == SUPPLY ==> PART
          : *SUPPLIER( SNO = 'S-25' )
            PART( 'P-15', 'BOLT', 75 )
```

部品供給者 S-25 が部品 P-15 をどの部門に何個納入しているのかが不明だからである。 このようなデータの格納は図5.3のデータベース COMPANY では許されない。

「どのような CONTEXT に対してどのような更新要求の記述を許すか」についての検討は，今後の課題である。

本論文では，更新要求の記述に対して，一応次の制限を設ける。

更新要求の記述は，合成アクセス関数をもたない単純 CONTEXT だけに許される。

第 6 章　CONTEXT に基づく網構造データベースの手続き的処理

6.1　緒　言

データベースを手軽に利用できるようにするために非手続き的に要求を記述できる問い合せ言語は欠かせないが，日常的な応用業務を効率良く実行するため，あるいは相当量の計算を伴うデータベース処理を行なうためには，プログラミング言語の力を借りなければならない。

データベースの応用プログラマは，COBOL，FORTRAN，PL/I 等のプログラミング言語，あるいはアセンブリ言語からデータ操作言語を用いてデータベースの処理を行なう。データ操作言語はいくつかのデータ操作命令から成る。応用プログラマは，データ操作命令を使ってデータベースをアクセスし，データを編集して報告書を作成する仕事，データに対する算術演算，実行の制御等はプログラミング言語によって行なう。

応用プログラマから見たデータビューが処理要求に適した構造をもっていることは，プログラムの制御構造を分り易くし，作成・保守を容易にするために本質的に重要であると考えられる。CONTEXT データモデルの柔軟で強力なデータ記述能力は，応用プログラマ用のインタフェース構成においても非常に有用であるといえる。

本章では，6.2 において CONTEXT データモデル用のデータ操作言語の機能の概観を与える。6.3 では，再帰構造データベース処理の問題について考察する。

6.2　データ操作言語

CONTEXT データモデルによれば，データベース中の必要なデータだけをそれぞれのデータ処理要求に適した木構造として表現できるので，プログラマに対して適切なデータビューと簡潔なデータ操作言語を提供できる。

CONTEXTビューを操作するデータ操作命令として7種類の命令を用意する。これらのデータ操作命令の機能を説明するために，以下の状況を仮定する。

操作の対象となるCONTEXTをC，Cの役割節点をR，CのなかでRの親となる役割節点をW，Rのデータ項目をI1，I2，…，Inとする。CONTEXTの根となる役割節点を子にもつSYSTEMというレコード型の存在を仮定する。SYSTEMレコード型は，ただ一つのレコード実現値 —— SYSTEMレコードと呼ぶ —— をもち，CONTEXTビューを構成するレコード実現値木の根は，SYSTEMレコードの子であるとする。RがCの根である場合，SYSTEMがWとなる。

いずれのデータ操作命令も特定のCONTEXTの特定の役割節点を指定する。CONTEXT Cの役割節点RをC.Rで参照する。曖昧さがなければC.を省略できる。

6.2.1 データ検索命令

CONTEXTを通じてデータベースを検索するために，次の2種類の検索命令を与える。

(a) FINDF (C.R)*

(b) FINDN (C.R)†

これら二つのデータ検索命令は，CONTEXTビューを構成するレコード実現値木を順行する（traverse）ために使用される。検索命令によって，指定された役割節点に対応する一つのレコード実現値rが見つけ出される。このときrはRに固定されたという。rがRに固定されたならば，rのデータ項目値を次の形式で参照できる。

---

* FIND First

† FIND Next

C.R.Ii　　(1≦i≦n)

曖昧さがなければ C.，さらに R. を省略できる。

FINDF(C.R)は，Wに固定されたレコード実現値 w が存在する場合に限り正常に実行され，Rに属する w の最初のレコード実現値*をRに固定する。SYSTEMレコードはプログラムの実行開始時に固定されるとする。

FINDN(C.R)は，Rに固定されたレコード実現値 r が存在する場合に限り正常に実行され，r の親レコード実現値の下で r の次にあるRのレコード実現値*をRに固定する。

検索命令の実行の結果は，次の三つの状態変数に設定される。

(i) ERROR

異常終了した場合 真 に，正常終了した場合 偽 に設定される。異常終了の場合，次の二つの状態変数は定義されない。

(ii) FOUND

レコード実現値が固定された場合 真 に，それ以外の場合 偽 に設定される。

(iii) LAST

レコード実現値が固定されしかもそれが同一の親の下で指定された役割節点の最後のレコード実現値である場合に 真 に，それ以外の場合 偽 に設定される。

木の順行(tree traversal)は，CONTEXT の根の役割節点に対するFINDF命令で始まり，続いて同一の役割節点に対するFINDN命令あるいは下位の役割節点に対するFINDF命令が繰り返し実行される。

---

*レコード実現値の順序の詳細はここでは問題にしない。とにかく何らかの順序が定められているとする。

```
begin
  print ( '  DNO      DNAME' );
  FINDF ( DEPT-SUP-PART-2.DEPT );
  while  FOUND  do
    FINDF ( DEPT-SUP-PART-2.SUPPLIER );
    while  FOUND  do
      C ←- 0;
      FINDF ( DEPT-SUP-PART-2.PART );
      while  FOUND  do
        C ←- C + 1;
        FINDN ( DEPT-SUP-PART-2.PART )
      end;
      if  not  C ≧ 2  then  goto  L1;
      FINDN ( DEPT-SUP-PART-2.SUPPLIER )
    end;
    print ( '  ', DEPT.DNO:5, '    ', DEPT.DNAME:10 );
L1:   FINDN ( DEPT-SUP-PART-2.DEPT )
  end
end
```

Fig. 6.1 A program to execute the query (Q4).

図 6.1 問い合せ (Q4) を実行するプログラム

検索プログラムの例として 5.3.3 の (Q4) を実行するプログラムを図 6.1 に示す。分り易い制御構造をもつプログラム——整構造プログラム (structured program) ——の作成が可能であることが理解できるであろう。同じプログラムを網構造スキーマ上でかくことは，かなり困難であると思われる。

6.2.2 データ更新命令

CONTEXT ゼロ - を更新するデータ操作命令として STORE (格納), CONNECT (接続), DISCONNECT (切断), DELETE (消去), および MODIFY (修正) の 5 種類の命令を与える。

以下の更新命令の説明では，役割節点 W にレコード実現値 w_i が固定されているとする。

(c) STORE (C. R)

この命令を実行する場合には，Rのデータ項目値を前もって設定しておかなければならない。 データ項目 I_i への値 v の設定を次のように表す。

$$C.R.I_i \leftarrow v$$

STORE 命令は，前もって設定されたデータ項目値をもつレコード実現値を w の子としてデータベースに格納する。

(d) CONNECT (C. R)

この命令を実行する場合には，Rの主キー項目を前もって設定しておかなければならない。 CONNECT 命令は，指定された主キー値をもつ R のレコード実現値を見つけ出し，それを w の子とする。

(e) DISCONNECT (C. R)

この命令は，前もってRに固定されてあるレコード実現値 r を w から切り離す。 r を根とする部分木は，CONTEXT ビュー から取り除かれ，参照不可能となる

(f) DELETE (C. R)

この命令は，前もってRに固定されてあるレコード実現値 r をデータベースから消去し，すべての親子関係を切断する。 r の子孫は，データベースから消去されないが CONTEXT ビュー からは取り除かれる。

最後の MODIFY 命令では，操作の対象がデータ項目ないしデータ項目の組であり，その指定はそれぞれ次の形式をとる。

(i) C. R. I_i

(ii) C. R. (I_{i_1}, I_{i_2}, …, I_{i_k})

以下では単一データ項目の修正だけについて説明する。 複数データ項目の修正も単一データ項目の修正と同様に実行される。

(9) MODIFY (C.R. I_i)

この命令を実行する場合には，データ項目 I_i の新しいデータ項目値 v を前もって設定しておかなければならない。 MODIFY 命令は，前もって R に固定されてあるレコード実現値 r のデータ項目 I_i の値を v に修正する。

MODIFY 命令を除く4つの更新命令は，W から R へのアクセス関数が基本アクセス関数の場合に限り適用可能である*。

すべての更新命令について，実行の結果が論理スキーマの制約条件——主キーや親子集合キーの一意性条件，親子結合属性の強結合の条件等——に反する場合には，その実行は無効にされ，状態変数 ERROR の値が真に設定される。 実際にシステムを作成する場合に，これらの誤りの種類およびそのそれぞれに応じた誤り対策等についての詳細な検討が必要であることは言うまでもない。

6.3 再帰構造データベースの手続き的処理

5.2.2でも述べたように，データベース処理はデータ実現値の集合に対する処理である。手続き的処理では，この集合に属するデータ実現値を一時に一つの割合で順次処理を行なう。

レコード集合の順次処理を行なう場合には，ファイルのオープンに相当する前処理によってファイル制御ブロックに相当する制御情報を準備し，以後この制御情報に基づいてレコードのアクセスが制御される。 レコード集合の順次処理の再帰的な実行を可能とするためには，この制御情報の再帰的な管理が必要である。 ところが多くのデータベース管理システムではこの制御情報の再帰的管理を行わない。 そのために再帰構造の処理に適した再帰的なアルゴリズムを使用できず，プログラムが煩雑になることは避けられない。

---

* 合成アクセス関数だと命令の解釈が一意的でなくなる。

CONTEXTモデルでは，レコード集合の順次処理を制御するための制御情報の管理は，CONTEXTの役割節点の管理の問題に置き換えられる。再帰的CONTEXTの繰り返し役割節点の管理を工夫することで，再帰構造データベースの処理を簡潔に表現可能である。

再帰的CONTEXTを通じて再帰構造データベースを処理するプログラムの例として，部品展開表を作成する問い合せ（Q10）を実行するプログラムを図6.2に，このプログラムの実行結果を図6.3に示す。より表質な文書を作成するためには，プログラミング上の工夫が必要であることは言うまでもない。再帰構造データベースを処理する他のプログラムも基本的にこのプログラムと同じ制御構造をとるであろう。

```
procedure  RPRINT( i ):
  if  i < 5  then
    begin
    FINDF ( PART-TREE-2.U( i ) );
    while  FOUND  do
      FINDF ( PART-TREE-2.P( i + 1 ) );
      print ( ' ':5*i, P( i + 1 ).PNO:5, U( i ).QTY:4 );
      RPRINT( i + 1 );
      FINDN ( PART-TREE-2.U( i ) )
    end
  end

begin
  FINDF ( PART-TREE-2.P( 1 ) );
  while  FOUND  do
    print ( P( 1 ).PNO:5 );
    RPRINT( 1 );
    FINDN ( PART-TREE-2 )
  end
end
```

Fig. 6.2 A program to print out the CONTEXT view of Fig. 4.12.

図6.2 図4.12のCONTEXTビューを表示するプログラム

```
P-1
      P-7      6
      P-3      1
            P-5      2
                  P-7      6
                  P-8      2
            P-9     10
P-2
      P-3      1
      P-4      3
            P-5      2
            P-6      1
                  P-8      2
                  P-10     1
            P-10    12
```

Fig. 6.3 An output of the tree expansion program of Fig. 6.2.

図6.3 図6.2のプログラムの実行結果

再帰的でないCONTEXTでは，プログラムの実行開始時に，CONTEXTの各節点ごとに制御情報が静的に確保される。再帰的CONTEXTの場合には，レベル番号で修飾された役割節点に対してFINDF命令が最初に実行される時に，その役割節点に対する制御情報が動的に確保されることになろう。

レコードの処理順序がリンク構造で表現されている場合には，レコード実現値の本体に次のレコード実現値のアクセスに必要な情報が含まれているので制御情報の管理は容易である。一方レコードの処理順序が物理的順序，インデクス，あるいは特別な制御機構*で表現されている場合には，制御情報の管理に特別な配慮が必要である。制御情報の具体的な管理方法はデータベースの記憶構造やデータベース制御システム——データベース管理システムの実行時モジュール——の構成法に依存する問題であるから，本論文ではこれ以上の議は避ける。

* SEQUEL言語[36]のカーソル（cursor）がこの例である。

第 7 章　利用者のレベルに応じた利用者インタフェースの構成法

7.1 緒言

2.3で述べたように，利用者の種類，要求の種類，および利用形態はさまざまである。これら幅広い利用者層の多様な要求に応じて適切な利用者インタフェースを構成するのはデータベース管理者の仕事である。データベース管理者は，利用者とのコミュニケーションを通じて，利用者の情報処理の技量，利用目的，利用資格等を総合的に判断し，適切な外部スキーマとデータアクセス手段を提供しなければならない。データベース管理者がこの仕事をいかに効率良く遂行できるかは，データベース管理システムの利用者用機能 ―― 利用者インタフェースを構成するための機能要素 ―― の水準に依存する。

本章では，CONTEXT データモデルに基づく利用者インタフェースの構成法について述べる。二段階構成方式をとる CONTEXT データモデルによって，利用者のレベルに応じた利用者インタフェースの構成が可能であることを示す。

7.2 利用者インタフェースの構成

7.2.1 外部スキーマの記述

外部スキーマ記述の枠組みを次に示す。

EXT-SCHEMA <identifier> WITHIN <sch-name>
LOCK <lock-code>
{<record-decl>}$_1^n$
{<af-decl>}$_0^n$
{<func-decl>}$_0^n$
{<cxt-decl>}$_0^n$
END

外部スキーマ記述では，外部スキーマ名（identifier），どの論理スキーマの上で定義するのか，その論理スキーマ名（sch-name），施錠コード（lock-code），1個以上の外部レコード型宣言（record-decl），0個以上のアクセス関数の宣言（af-decl），0個以上の関数宣言（func-decl），および0個以上のCONTEXT宣言（cxt-decl）を与え，一つの外部スキーマを定義する。

7.2.2 データベース操作部の記述

問い合せ言語QLCによってデータベースをアクセスする場合の要求記述の枠組みを次に示す

```
USER <user-id>  EXT-SCHEMA <ext-sch-name>
WITHIN <sch-name>  KEY <key-code>
   { <af-decl> }0m
   { <func-decl> }0m
   { <cxt-decl> }0m
   { <request> }1m
END
```

データベースの利用者は，データベース管理者から与えられた外部スキーマに対して要求を記述する。まず，利用者は自分の登録名（user-id），外部スキーマ名（ext-sch-name），論理スキーマ名（sch-name），および外部スキーマの錠を開けるための鍵コード（key-code）を指定する。外部スキーマは，外部レコード型，アクセス関数，関数，CONTEXTから構成されるが，利用者は必要に応じて，さらにアクセス関数，関数，CONTEXTを宣言することができる。これらの宣言を行なうことで，データベースの利用環境が整うことになる。利用者は，宣言されたアクセス関数，関数，CONTEXTを使い，QLCにより要求（request）を記述する。

7.3 利用者のレベルに応じたインタフェース構成

第5章で示したように，複雑な要求でもそれに合致したCONTEXTを構成することができれば，その上で要求を表現することは容易である。問題は，CONTEXTを誰が定義するかということである。

適切なCONTEXTを設定するためには，要求を正確に理解することと同時に，網構造スキーマの完全な理解が必要である。網構造スキーマの理解は，一般のデータベースの利用者にとって容易なことではない。特にレコード間関係の表現方法の多様性は，網構造スキーマ理解の大きな障害になるものと思われる。従ってあるレベルの利用者にとっては，レコード間関係をアクセス関数によって一様に表現したデータモデルが有用であろう。このレベルの利用者は，外部レコード型とアクセス関数から自分の要求に応じて合成アクセス関数，関数，CONTEXTの定義を行なうことができるであろう。利用者とデータベース管理者のコミュニケーションに要する時間を最小限にとどめることが期待できる。

複雑な要求を表現するためのCONTEXTは，必然的に複雑で特殊なものにならざるを得ない。このようなCONTEXTは，利用者にとっても記述が困難である。しかも複雑な要求はデータベース管理者に伝えることが難しい。このような場合に，コミュニケーションの手段として，CONTEXTダイアグラムの記法が有用であると思われる。利用者はCONTEXTダイアグラムを使って要求に合致したデータビューの構造を記述し，データベース管理者が網構造スキーマの上でこのCONTEXTを編成するのである。

あるレベルの利用者は，CONTEXTの構造を理解することも困難であるかもしれない。このような利用者に対しては，日常の情報処理活動の延長線上に利用者インタフェースを設定すべきであろう。CODASYL EUFTGは，この立場から最終利用者インタフェースの構成法を検討し，Formアプローチを提案している（2.5.1参照）。

Formは，データベースの内容の一部を二次元的に構成された一定の

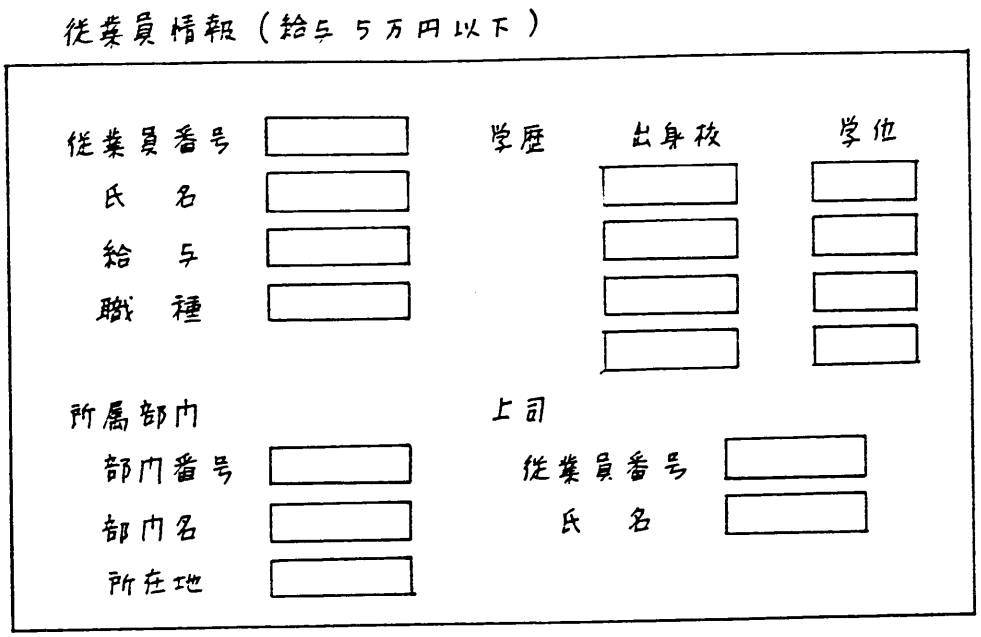


図7.1　POOR-EMP-SPEC の上で定義された Form

書式に従って表現したものであり，CONTEXT の外部表現と考えることができる。従って，CONTEXT モデルの上で Form 表現による利用者インタフェースを構成することに大きな障害はない。図7.1に，(c1) で定義した CONTEXT POOR-EMP-SPEC に対応する Form を示す。

図7.2は，利用者のレベルに応じたインタフェースの構成を示す。最もレベルの高い利用者は，レベル3のインタフェース（UI-L3）を通じてデータベースを利用する。この レベルの利用者は，データベースに関して，データベース管理者と同程度の知識をもっている必要がある。

レベル2のインタフェース（UI-L2）では，データベース管理者が外部レコード型とアクセス関数を定義し，CONTEXT の定義は利用者が行なう。レベル1のインタフェース（UI-L1）では，データベース管理者が CONTEXT の定義を行なう。利用者は CONTEXT の構造と QLC の理解だけが要求される。

レベル0のインタフェース（UI-L0）では，データベース管理者が適当な言語によって CONTEXT の外部表現である Form の枠組みを定義することになろう。利用者に対しては，Form を操作する簡便な言語を用

意しなければならない。　レベル0のインタフェースの構成法の確立は，今後に残された重要な課題である。

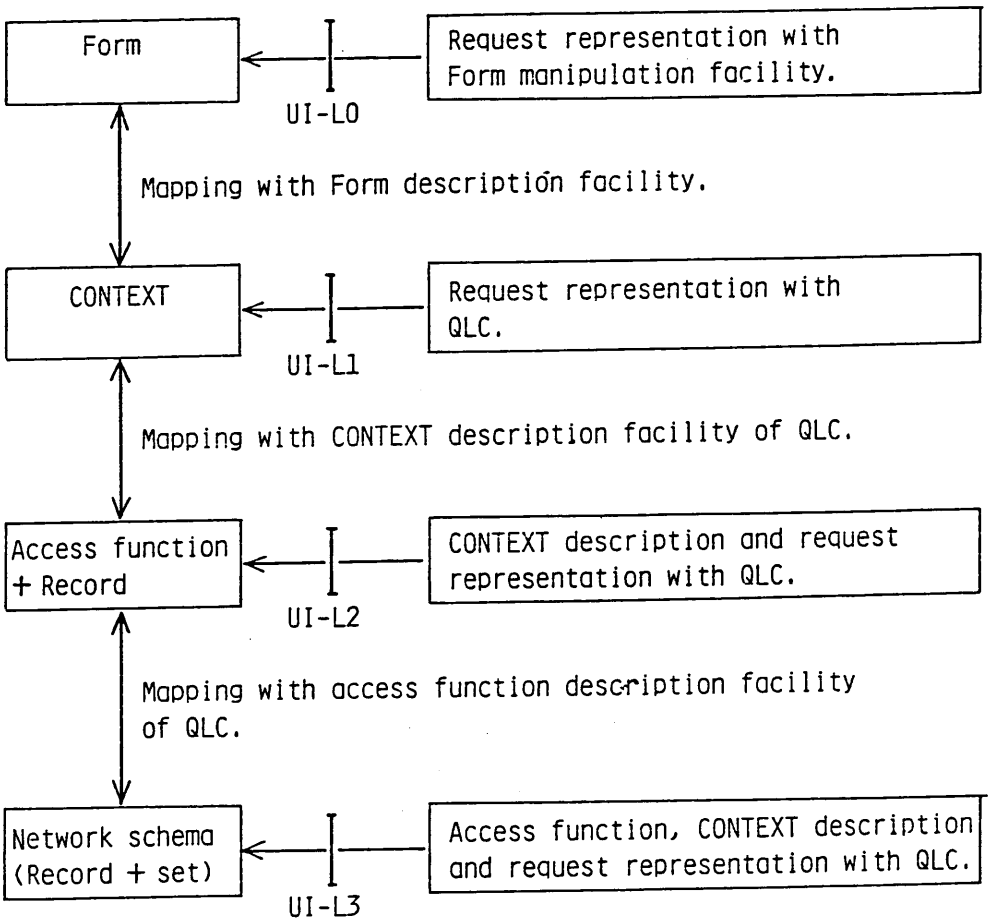


図7.2　利用者のレベルに応じたインタフェースの構成

第 8 章　結論

8.1 内容要約

本論文では，網構造データベースに対する高水準利用者インタフェースの構成法について原理的な考察から始め，新しいデータモデル —— CONTEXT データモデル —— を考案し，このデータモデルに基づく一貫した利用者インタフェースの構成法を示した。

利用者インタフェースは，3層スキーマ構成の枠組みの下で，概念スキーマに対して外部スキーマを記述するデータ記述言語および外部スキーマに基づいて要求を記述するための問い合せ言語・データ操作言語から構成される。利用者インタフェースの構成法を確立する上で，外部レベルのデータモデルの決定は最も重要な問題である。外部レベルのデータモデルは，利用者の多様な要求に柔軟に応えることができ，しかも概念スキーマと無理なく写像可能でなければならない。

そこで，第3章において，概念レベルのデータモデルについて詳細に検討した。まず，概念レベルの取扱いを容易にするために，概念レベルの記述を論理スキーマと意味論スキーマとに分け，3層スキーマ構成を拡張した4スキーマ構成を示した。この枠組みの下で，網構造モデルや関係モデルを論理スキーマを記述するための構造論的データモデルとして位置づけ，両データモデルの実体間関連の表現法を比較することを通じて，網構造モデルの次の特徴を明らかにした。

1. 親子集合構造は現実世界で頻出する階層構造を表現するのに非常に有用である。
2. 実体間関連を表現する方法は多様であり，実体間の多対多関連，3種類以上の実体間の関連の表現は不自然で複雑である。

第4章では，第3章での検討を踏まえ，網構造モデルの特徴を生かし，利用者の要求に合致したデータビューの記述を可能にするデータモデル —— CONTEXT データモデル —— を提案した

CONTEXT データモデルは強力なデータ記述・写像能力を有し，網構

造データベースに対して，特定の目的をもつ利用者が要求を記述するのに適した場——CONTEXTの語意——を設定しうる。　CONTEXTはレコード型を節点とする木を基本構造とし，網構造スキーマから二段階に分けて構成される。　すなわち

1. レコード間の論理的関係をアクセス関数で一様に表現する。
2. アクセス関数とレコード型からCONTEXTを構成する。

CONTEXTの構成においては，仮想レコード型，階層レコード型，n次のレコード間関係の階層化等の概念を導入し，網構造モデルのデータ表現の制約を打開できた。　また再帰的CONTEXTの概念を導入することにより，従来処理がきわめて困難であった再帰構造データベースの処理を容易にした。

CONTEXTデータモデルでは，アクセス関数を利用してレコード型から単純データ型への関数を定義するという形式で，導出データ項目を記述する簡潔で強力な記法を与えている。　再帰的な関数を定義することにより，再帰構造データベースから要約的な情報を抽出することも容易である。

目的に合致したCONTEXTを設定することができれば，その上で要求を記述することに困難はない。　本論文では，CONTEXTを通じて網構造データベースを利用するために，第5章において最終利用者用の問い合せ言語QLCを，第6章において応用プログラマ用のデータ操作言語を，それぞれ提案した。

木構造モデル用の問い合せ言語は，すでに実用化されているものも含め何種類かが提案されている。　これらの問い合せ言語と比較してQLCは次の際立った特長を有する。

1. 処理単位の概念を導入することにより，木構造データビューに対する問い合せを曖昧なく表現できる。　また複雑な量操作を含む問い合せを明快に表現できる。
2. 再帰的CONTEXTに対して順行子を宣言することにより再帰構造データベースから構造に関する情報を抽出することが可能である。

CONTEXT データモデルでは，網構造スキーマの構造に制限されることなく処理要求に適したデータ構造を定義できるので，応用プログラマは整構造プログラムをかき易い。 さらに，網構造スキーマの変更の多くは写像定義の簡単な変更で吸収できるので，応用プログラムに高いデータ独立性をもたせることができる。 このようにCONTEXT データモデルによってデータベース応用プログラムの作成・保守の大幅な改善が期待できる。

第7章ではCONTEXT データモデルに基づく利用者インタフェースの構成法を示した。 幅広い利用者層の多様な要求に応じて適切な利用者インタフェースを設定するのはデータベース管理者の仕事である。 二段階構成法をとるCONTEXT データモデルでは，利用者のレベルに応じた利用者インタフェースの構成が可能であり，利用者とデータベース管理者の間で仕事を適切に分担することができる。 この特性はデータベースシステム全体の効率向上につながるもので，CONTEXT データモデルの最大の特長と言えよう。

なお，CONTEXT データモデルに基づくデータベース言語——外部スキーマ記述言語と問い合せ言語 QLC——の構文は付録で与えてある。

8.2 討論および今後の課題

8.2.1 実働化について

本論文では，データベース管理システムの利用者用機能について論じ，網構造データベースに対する高水準利用者インタフェースを構成するためのデータベース言語を考案した。 このデータベース言語に基づく利用者機能の実働化は今後に残された課題である。 以下に実働化に関する筆者の見解を示す。

実働化の手順としては，まず第6章で与えたデータ操作言語を，次にこのデ

ータ操作言語を使って問い合せ言語 QLC を実働化することになろう。データ操作言語を効率良く実働化するためには，記号結合アクセス関数，仮想レコード型・階層レコード型，後戻り処理を伴うの次のレコード関係の階層化，再帰的 CONTEXT 等の取扱いに工夫が必要である。しかし本質的に困難な問題はないと思われる。

このデータ操作言語を用いて QLC を実働化することにも大きな障害はないように思われる。QLC で記述した要求をデータ操作言語を含むプログラムへ変換する手続きは，比較的容易に実現できそうである（問い合せ記述（G4）と図6.1のプログラムを比較されたい。）。ほとんどの要求に対して，特別の最適化処理を施さなくても効率の良い実行が可能であると思われる。しかしすべてがそうであるとは限らない。さらに次のような問題がある。

ある要求を表現しうる CONTEXT は，一般に一つだけとは限らずいくつも存在し，それぞれに対して，要求記述の容易さ，実行効率（特に実行時間）等にかなりの差があるものと思われる*。要求記述の容易さと実行効率とが両立すれば問題はないが，両者が対立する場合には，どちらか一方を犠牲にしなければならない。

このような問題を解決するために要求実行の適当な最適化処理が必要になろう。「どのような場合にどのような最適化が有効であるか」の検討は，効率の良いシステムを実現するために不可欠である。

8.2.2 データベースの更新について

関係モデルでは，基本関係の上で定義された仮想的な関係——ビュー——に対する更新が大きな問題となっている[83) 89)～92)]。5.4.4でも述べたように，CONTEXT データモデルでも同様の問題に対する検討が必要である。関係モデルに対する考察の結果は，CONTEXT データモデルにも採用できるであろう。

---

*（Q1）の問い合せ要求を記述した（G1）と（G1'）を参照。

8.2.3 Formインタフェースの実現

7.3でも述べたように，CONTEXTデータモデルによって，網構造データベースに対するFormインタフェースを実現することに大きな困難はないと思われる。 Formの枠組みを記述する機能，FormとCONTEXTを対応づける機能，Formエで要求を記述する機能等の言語化は今後の課題である。 文献[87),93)]はこの参考になろう。

8.2.4 異種データベースに対する共通インタフェース

アクセス関数によって，レコード構造に基づくデータベースに対して，統一的なデータビューを与えることができる。 従って，CONTEXTデータモデルによれば，異種データベース——関係データベース・木構造データベース・網構造データベース——に対する共通インタフェースを比較的容易に実現できるものと思われる[94)]。

8.2.5 データベース言語の構文について

筆者は，本論文で与えた言語の構文が最適なものであると主張するつもりはない。 今後積極的に改良していくべきであると考えている。 しかしながら，CONTEXTデータモデルおよびGLCの基本概念について，大きな変更はほとんど必要ないと思う。

謝辞

本研究を進めるに当り，御指導・御鞭撻を頂いた鈴木淳之助教授，有益な御助言・御討論を頂いた阿部圭一助教授ならびに落水浩一郎助教授に，深く感謝致します。

本論文を作成するに当り，審査委員の諸先生方――阿部圭一助教授，落水浩一郎助教授，鈴木淳之助教授，鈴木久喜教授，松井英一教授，松本欣二教授，山田藥教授（50音順）――からは適切かつ有益な御指摘・御助言を頂きました。　とりわけ，鈴木淳之先生，阿部圭一先生，落水浩一郎先生からは，細部に渡って具体的な御指摘・御助言を頂きました。　以上の先生方に対し，心より感謝の意を表します。

本論文の題目の決定に際し，水品静夫教授からは有益な御助言を頂きました。　ここに厚く御礼申しあげます。

筆者の在学中，なにかと御世話になった本研究科事務室の職員の方々，本学工学部情報工学科の職員ならびに学生の方々に，心より感謝致します。

最後に，修士課程で御指導を頂き，本研究科への進学を勧めて下さった山梨大学工学部の伊藤誠助教授に心より感謝致します。

参考文献

1) Fry,J.P. and Sibley,E.H.: Evolution of Data-base Management Systems, Computing Surveys, Vol.8,No.1, pp.7-42 (1976).

2) 植村俊亮：データベースシステムの基礎，オーム社 (1979).

3) Codd,E.F.: A Relational Model of Data for Large Shared Data Banks, Comm.ACM, Vol.13,No.6, pp.377-387 (1970).

4) Codd,E.F.: Futher Normalization of the Data Base Relational Model, in Courant Computer Science Symposium 6, Data Base Systems (Rustin,R., ed.), Prentice-Hall, pp.33-64 (1972).

5) Codd,E.F.: Relational Completeness of Data Base Sublanguages, ibid., pp.65-98.

6) 大須賀節雄：データベース概説，情報処理ハンドブック，情報処理学会，PP. 199-200 (1980).

7) データベース特集号，情報処理，Vol. 17, No.10 (1976).

8) Date,C.J.: An Introduction to Database Systems, Addison-Wesley (1975), 2nd ed.(1977), 3rd ed.(1981).

9) Tsichritzis,D. and Lochovskey,F.H.: Data Base Management Systems, Academic Press (1977).

10) Nijssen,G.M.: On the Gross Architecture for the Next Generation Database Management Systems, Proc. IFIP Congress, pp.327-335 (1977).

11) The ANSI/X3/SPARC DBMS Framework, Report of the Study Group on Database Management Systems (Tsichritzis,D. and Klug,A., eds.), Information Systems, Vol.3,No.3, pp.173-191 (1978).

12) Codd,E.F.: Recent Investigation in Relational Data Base Systems, Proc. IFIP Congress, pp.1017-1021 (1974).

13) Selection and Acquisition of Data Base Management Systems, A Report of the CODASYL Systems Committee (1976), ACM.
邦訳 西村恕彦 監修：CODASYL システムズ委員会報告書，データベースの選び方，bit，1977年4月臨時増刊，共立出版 (1977).

14) Olle,T.W.: The CODASYL Approach to Data Base Management, John Wiley & Sons (1978).
邦訳 西村恕彦，植村俊亮 監訳：CODASYL のデータベース，bit，1979年8月臨時増刊，共立出版（1979）

15) Ashenhurst,R.: A Great Debate, Comm.ACM, Vol.17,No.6 (1974).

16) Bachman,C.W.: The Data Structure Set Model, Proc. ACM SIGMOD Workshop on Data Description, Access and Control. Data Models: Data-Structure-Set versus Relational, pp.1-10 (1974).

17) Codd,E.F. and Date,C.J.: Interactive Support for Non-Programmers: The Relational and Network Approaches, ibid., pp.11-41.

18) Sibley,E.H.: On the Equivalences of Data Based Systems,ibid., pp.43-76.

19) Date,C.J. and Codd,E.F.: The Relational and Network Approaches : Comparison of the Application Programming Interfaces,ibid., pp.83-113.

20) Nijssen,G.M.: Set and CODASYL Set or Coset, Proc. IFIP TC-2 Special Working Conference on Data Base Description, pp.1-72 (1975)

21) Taylor,R.W.: Observations on the Attributes of Database Sets, ibid., pp.73-84.

22) Waghorn,W.J.: The DDL as Industry Standard ibid., pp.121-167.

23) Robinson,K.A.: An Analysis of the Uses of the CODASYL Set Concept, ibid., pp.169-181.

24) Olle,T.W.: An Analysis of the Flows in the Schema DDL and Proposed Implovements, ibid., pp.283-297.

25) Metaxides,A.: "Information Bearing" and "Non-Information Bearing" Sets, ibid., pp.363-367.

26) Douqué,B.C.M. and Nijssen,G.M.: The Wépion Recommendations on the CODASYL DDL 1973, ibid., pp.369-371.

27) Report of the CODASYL Data Description Language Committee, Information Systems, Vol.3,No.4, pp.247-320 (1978).

28) CODASYL FORTRAN Data Base Facility, Version 2.0 of the CODASYL FORTRAN Data Base Committee (January 1980), Information Systems, Vol.6,No.3, pp.161-199 (1981).

29) Manola,F.: A Review of the 1978 CODASYL Database Specification, Proc. 4th VLDB, pp.232-242 (1978).

30) Looms,M.E.S.: The 78 CODASYL Database Model: A Comparison with Proceding Specifications, Proc. ACM SIGMOD, pp.30-44 (1980).

31) Bachman,C.W.: The Programmer as Navigator, Comm.ACM, Vol.16, No.11, pp.653-658 (1973).

32) Lochovsky,F.H. and Tsichritzis,D.C.: User Performance Considerations in DBMS Selection, Proc. ACM SIGMOD, pp.128-134 (1977).

33) A Progress Report on the Activities of the CODASYL End User Facility Task Group (June 1975), ACM FDT, Vol.8,No.1 (1976).

34) Boyce,R.F.,et al.: Specifying Queries as Relational Expressions: the SQUARE Data Sublanguage, Comm.ACM, Vol.18,No.11, pp.621-628 (1975).

35) Held,G.,Stonebraker,M., and Wong,E.: INGRES - A Relational Data Base System, Proc.AFIPS NCC, pp.409-416 (1975).

36) Chamberlin,D.D.,et al.: SEQUEL 2 : A Unified Approach to Data Description, Manipulation, and Control, IBM J.Res.Dev., Vol.20,No.6, pp.560-575 (1976).

37) Zloof,M.M.: Query-by-Example: A Data Base Language, IBM System Journal, Vol.16,No.4, pp.324-343 (1977).

38) Hitchicock,P.: User Extensions to the Peterlee Relational Test Vehicle, Proc. 2nd VLDB, pp.169-180 (1976).

39) Lacroix,M. and Pirotte,A.: Domain-Oriented Relational Languages, Proc. 3rd VLDB, pp.370-378 (1977).

40) Pirotte,A.: Fundamental and Secondary Issues in the Design of Non-Procedural Relational Languages, Proc. 5th VLDB, pp.239-250 (1979).

41) Zaniolo,C.: Design of Relational Views over Network Schemas, Proc. ACM SIGMOD, pp.179-190 (1979).

42) Zaniolo,C.: Multimodel External Schemas for CODASYL Database Management Systems, Proc. IFIP Working Conference on Data Base Architecture, pp.171-190 (1979).

43) Kay,M.H.: An Assessment of the CODASYL DDL for Use with a Relational Subschema, Proc. IFIP TC-2 Special Working Conference on Data Base Description, pp.199-214 (1975).

44) Kalinichenko,L.A.: Relational-Network Data Structure Mapping, Proc. IFIP TC-2 Working Conference on Modelling in Data Base Management Systems, pp.303-309 (1976).

45) McGee,W.C.: File-level Operations on Network Data Structures, Proc. ACM SIGMOD, pp.32-47 (1975).

46) Clemons,E.K.: An External Schema Facility for CODASYL 1978, Proc. 5th VLDB, pp.119-128 (1979).

47) Clemons,E.K.: Rational Data Base Standards: an Examination of the 1978 CODASYL DDLC Report, Information Systems, Vol.4,No.3, pp.235-239 (1979).

48) Metaxides,A.: Comments on a Paper by Erick K. Clemons, Information Systems, Vol.4,No.3, pp.247-249 (1979).

49) Shneiderman,B. and Thomas,G.: Path Expressions for Complex Queries and Automatic Database Program Conversion, Proc. 6th VLDB, pp.33-44 (1980).

50) Bradley,J.: An Extended Owner-Coupled Set Data Model and Predicate Calculus for Database Management, ACM TODS, Vol.3, No.4, pp.385-416 (1978).

51) Deheneffe,C. and Hennebert,H.: NUL: A Navigational User's Language for a Network Structured Data Base, Proc. ACM SIGMOD, pp.135-142 (1976).

52) Senko,M.E.: The DDL in the Context of a Multilevel Structured Description: DIAM II with FORAL, Proc. IFIP TC-2 Special Working Conference on Data Base Description, pp.239-257 (1975).

53) Senko,M.E.: DIAM II : The Binary Infological Level and Its Database Language - FORAL, ACM FDT, Vol.8,No.2, pp.121-140 (1976).

54) Senko,M.E.: FORAL LP : Design and Implementation, Proc. 4th VLDB, pp.255-267 (1978).

55) Senko,M.E.: A Query - Maintenance Language for the Data Independent Accessing Model II, Information Systems,Vol.5, pp.257-272 (1980).

56) Munz,R.: The WELL System : A Multi-User Database System Based on Binary Relationships and Graph-Pattern-Matching, Information Systems, Vol.3,No.2, pp.99-115 (1978).

57) Martin,J.: Computer Data-Base Organization, Prentice-Hall (1975).

58) Stonebraker,M.,Wong,E.,Kreps,P., and Held,G.: The Design and Implementation of INGRES, ACM TODS, Vol.1,No.3, pp.189-222 (1976).

59) Chamberlin,D.D.: A Summary of User Experience with the SQL Data Sublanguage, Proc.Int.Conf.Data Bases, The British Computer Society, Heyden, pp.338-350 (1980).

60) Kambayashi,Y.,Tanaka,K., and Yajima,S.: A Relational Data Language with Simplified Binary Relation Handling Capability, Proc. 3rd VLDB, pp.338-350 (1977).

61) Clemons,E.K.: Design of an External Schema Facility to Define and Process Recursive Structure, ACM TODS, Vol.6,No.2, pp.295-311 (1981).

62) Zloof,M.M.: Query-by-Example : Operations on the Transitive Closure, IBM Research Report No.RC5526 (1976).

63) Backus,J.: Can Programming Be Liberated from the von Neumann Style ? A Functional Style and Its Algebra of Programs, Comm.ACM, Vol.21,No.8, pp.613-641 (1978).

64) Buneman,P. and Frankel,R.E.: FQL : A Functional Query Language (A Preliminary Report), Proc. ACM SIGMOD, pp.52-58 (1979).

65) Senko,M.E.,et al.: Data Structures and Accessing in Data Base Systems, IBM System Journal, Vol.12,No.1, pp.30-93 (1973).

66) Abrial,J.R.: Data Semantics, Proc. IFIP Working Conference on Data Base Management, pp.1-60 (1974).

67) Schmid,H.A. and Swenson,J.R.: On the Semantics of the Relational Database, Proc. ACM SIGMOD, pp.211-223 (1975).

68) Chen,P.P.S.: The Entity-Relationship Model - Toward a Unified View of Data, ACM TODS, Vol.1,No.1, pp.9-36 (1976).

69) Hall,P.A.V.,Owlett,J., and Todd,S.J.P.: Relations and Entities, Proc. IFIP TC-2 Working Conference on Modelling in Data Base Management Systems, pp.201-220 (1976).

70) Falkenberg,E.: Concepts for Modelling Information, ibid., pp.95-109.

71) Nijssen,G.M.: Current Issues in Conceptual Schema Concepts, Proc. IFIP TC-2 Working Conference on Architecture and Models in Data Base Management Systems, pp.31-65 (1977).

72) Biller,H. and Neuhold,E.J.: Semantics of Data Bases: The Semantics of Data Models, Information Systems, Vol.3,No.1, pp.11-30 (1978).

73) Hammer,M. and Mcleod,D.J.: A Modelling Mechanism for Data Base Applications. Proc ACM SIGMOD, pp.26-36 (1978).

74) Smith,J.M. and Smith,D.C.P.: Database Abstractions : Aggregation, Comm.ACM, Vol.20,No.6, pp.405-413 (1977).

75) Smith,J.M. and Smith,D.C.P.: Database Abstractions : Aggregation and Generalization, ACM TODS, Vol.2,No.2, pp.105-133 (1977).

76) Bachman,C.W. and Daya,M.: The Role Concept in Data Models, Proc. 3rd VLDB, pp.464-476 (1977).

77) Codd.E.F.: Extending the Database Relational Model to Capture More Meaning, ACM TODS, Vol.4,No.4, pp.397-434 (1979).

78) Kent,W.: Splitting the Conceptual Schema, Proc. 6th VLDB, pp.10-14 (1980).

79) De,P.,Haseman,W.D., and So,Y.H.: Four-Schema Approach : An Extended Model for Database Architecture, Information Systems, Vol.6,No.2, pp.117-124 (1981).

80) Kent,W.: Limitation of Record-Based Information Models, ACM TODS, Vol.4,No.1, pp.107-131 (1979).

81) Beeri,C.,Bernstein,P.A. and Goodman,N.: A Sophisticate's Introduction to Database Normalization Theory, Proc. 4th VLDB, pp.113-124 (1978).

82) 植村俊亮：データベースの関係モデル第3正規形批判，情報処理, Vol.18,No.1,pp.58-62 (1977).

83) Chamberlin,D.D.,Gray,J.N., and Traiger,I.L.: Views, Authorization, and Locking in a Relational Data Base System, Proc. AFIPS NCC, pp.425-430 (1975).

84) Reisner,P.: Use of Psychological Experimantation as an Aid to Development of a Query Language, IEEE Trans.Softw.Eng., Vol.SE-3,No.3,pp.218-229 (1977).

85) Furtado,A.L. and Kerschberg,L.: An Algebra of Quotient Relations, Proc. ACM SIGMOD, pp.1-8 (1977).

86) Zloof,M.M.: Query-by-Example --- Operation on Hierachical Data Bases, Proc. AFIPS NCC, pp.845-853 (1976).

87) Luo,D. and Yao,S.B.: FORM OPERATION BY EXAMPLE -- a Language for Office Information Processing, Proc.ACM SIGMOD, pp.212-223 (1981).

88) Hardgrave,W.T.: Ambiguity in Processing Boolean Queries on TDMS Tree Structures: A Study of Four Different Philosophies, IEEE Trans.Softw.Eng., Vol.SE-6,No.4, pp.357-372 (1980).

89) Stonebraker,m.: Implementation of Integrity Constraints and Views by Query Modification, Proc. ACM SIGMOD, pp.65-78 (1975).

90) Dayal,U. and Bernstein,P.A.: On the Updatability of Relational View, Proc. 4th VLDB, pp.368-377 (1978).

91) Furtado,A.L.,Sevcik,K.C. and Santos,C.S.D.: Permitting Updates through Views of Data Bases, Information Systems, Vol.4, pp.269-283 (1979).

92) Bancilhon,F.: Supporting View Updates in Relational Data Bases, Proc. IFIP Working Conference on Data Base Architecture, pp.213-234 (1979).

93) Sorenson,P.G. and Wald,J.A.: PICASSO - An Aid to an End-User Facility,Proc. ACM SIGMOD, pp.30-39 (1977).

94) Takizawa,M. and Hamanaka,E.: The Four-Schema Concept as the Gross Architecture of Distributed Databases and Heterogeneity Problem, Journal of Information Processing, Vol.2,No.3, pp.134-142 (1979).

付 録　CONTEXT データモデルに基づく データベース言語の構文

本付録では，本論で述べた CONTEXT データモデルに基づくデータベース言語 ——外部スキーマ記述言語と問い合せ言語 QLC —— の構文を与える。

A.1 構文表現上の約束

言語の構文を構文図（syntax diagram）で示す。

大文字あるいは特殊記号から成る文字列は終端記号であり，接続記号「－」を含む小文字列は非終端記号である。定義すべき非終端記号には下線が引いてある。この非終端記号は，長方形で囲まれた非終端記号とはだかの終端記号を矢印で連結したダイアグラムによって定義される。紙面の大きさの都合で2個以上に分断されたダイアグラムもある。この場合，出現すべき記号の順序は，ダイアグラムの左から右へ，上から下へを原則とする。

実際の文は，ダイアグラムを矢印の流れにそってなぞっていき，行き当った記号が終端記号ならばその文字列を，非終端記号ならばその非終端記号が示す文を書き出すことによって構成される。

A.2 対象物の定義とその参照に関する一般規則

一般に，対象物は参照される前に定義されていなければならない。

外部スキーマは特定の概念スキーマ（あるいは論理スキーマ）の上で定義される。外部スキーマ定義の中で参照する概念スキーマ，レコード型，データ項目，親子集合型は，適当な概念スキーマ記述言語，たとえば CODASYL DDL を使って定義されてあるものと仮定する。

ある対象物を定義する場合には，識別子（identifier）を指定して名前をつける。この対象物を参照する場合はこの名前を用いる。対象物の参照は，たとえば rec-name というような「name」で終る非終端記号による。CONTEXT データモデルで関心のある対象物として

は，論理スキーマ，レコード型，データ項目，親子集合型，外部スキーマアクセス関数，関数，CONTEXT，役割節点，関数引数，およびレコード変数がある。

A.3 非終端記号の構成に関する一般規則

非終端記号は，英単語の省略形を接続記号「－」で結合して構成する。構文に現われる省略形とそのもとの英単語との対応を表A.1に示す。

表A.1 構文に現われる英単語の省略形

af	access function	id	identifier
agg	aggregate	int	integer
arg	argument	lit	literal
arith	arithmetic	op	operation
attr	attribute	prim	primary
char	character	proc	processing
cond	condition	quant	quantification
const	constant	rec	record
coset	CODASYL set	ref	reference
cxt	CONTEXT	sch	schema
decl	declaration	sel	selection
eq	equal	spec	specification
exp	expression	var	variable
func	function		

A.4 予約語

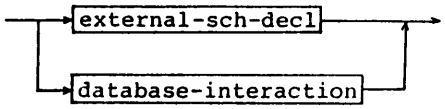
表A.2に，本データベース言語の予約語を挙げる。

表A.2 予約語表

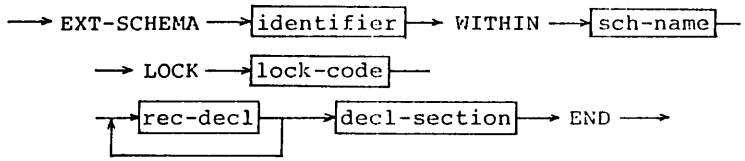
EXT-SCHEMA	UNIQUE	AND
WITHIN	USER	EXIST
LOCK	KEY	SET
END	GET	NULL
RECORD	STORE	COUNT
ALL	INTO	SUM
ACCESS	MODIFY	AVG
FUNC	DELETE	MAX
FUNCTION	FROM	MIN
CONTEXT	HAVING	INTEGER
WHERE	FOR	REAL
DUPLICATE	SYSTEM	CHAR
IS	EACH	DATE
LEAF	SOME	TIME
NOT	TRAVERSER	
INCLUDED	OR	

A.5 Syntax Diagrams for the Database Language based on the CONTEXT Data Model.

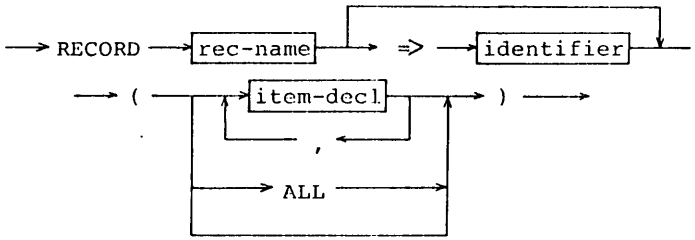
statement



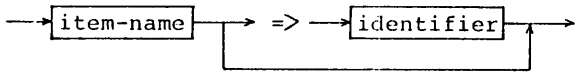
external-sch-decl



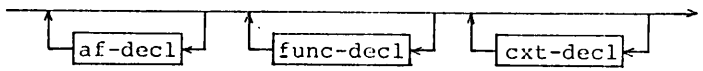
rec-decl



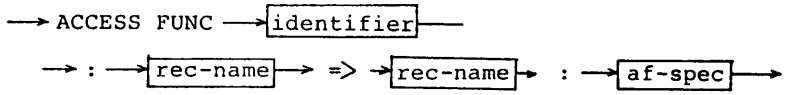
item-decl



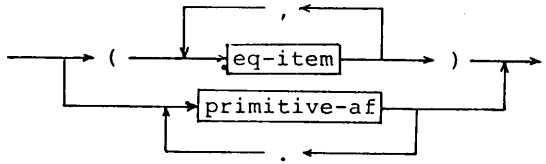
decl-section



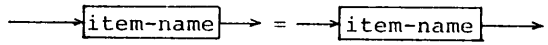
af-decl



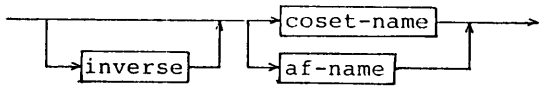
af-spec



eq-item

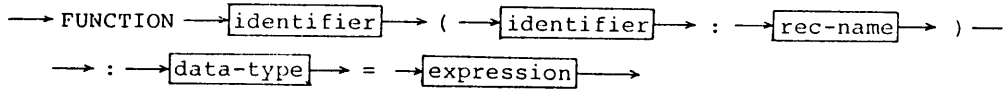


primitive-af

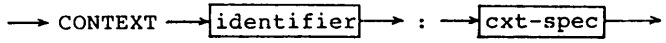


inverse

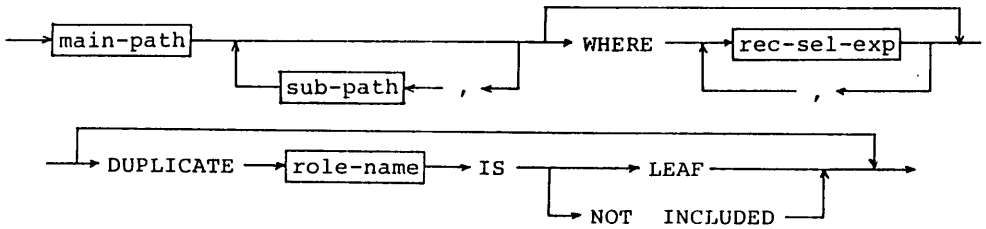
func-decl



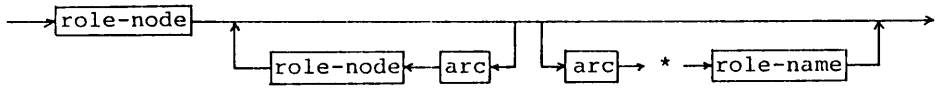
cxt-decl



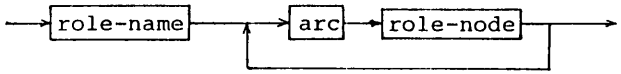
cxt-spec



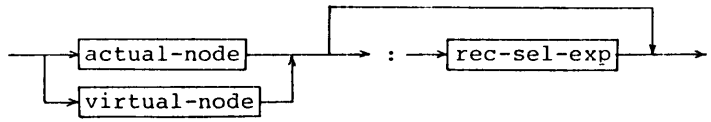
main-path



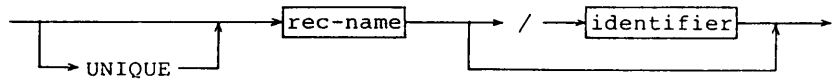
sub-path



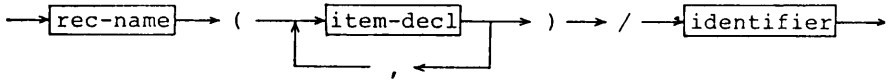
role-node



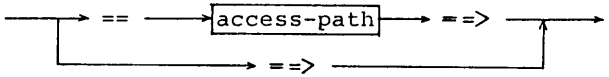
actual-node



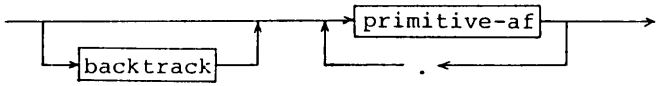
virtual-node



arc

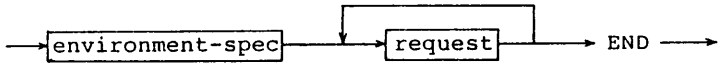


access=path

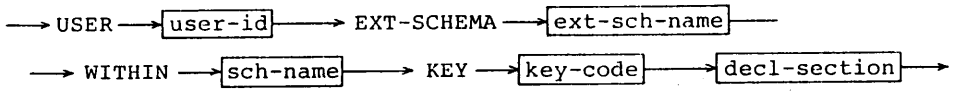


backtrack

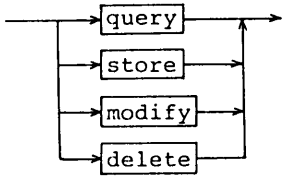
database-interaction



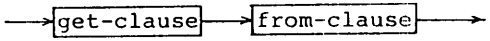
environment-spec



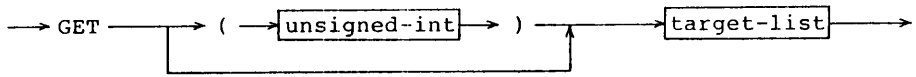
request



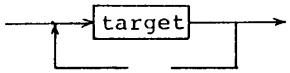
query



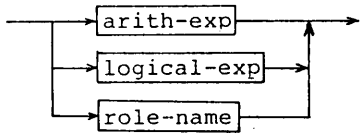
get-clause



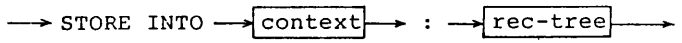
target-list



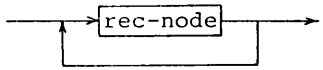
target



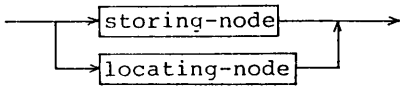
store



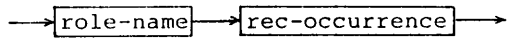
rec-tree



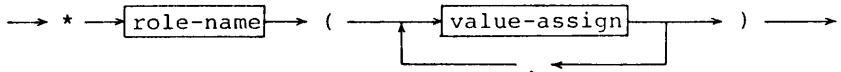
rec-node



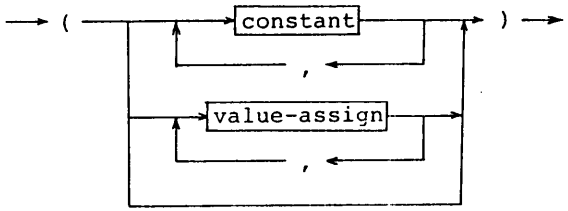
storing-node



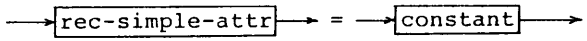
locating-node



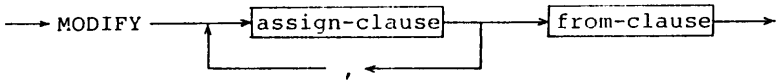
rec-occurrence



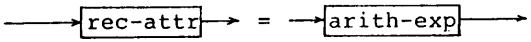
value-assign



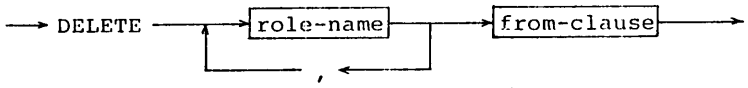
modify



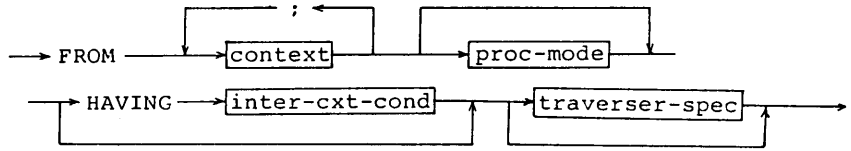
assign-clause



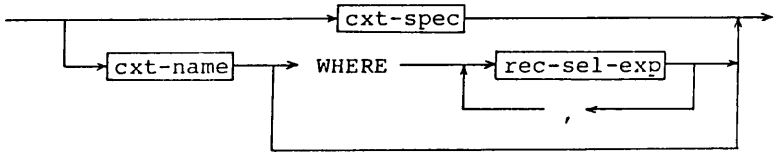
delete



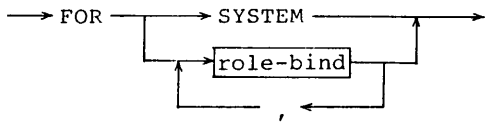
from-clause



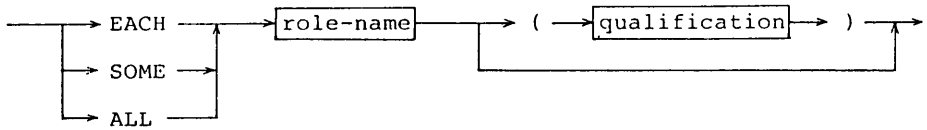
context



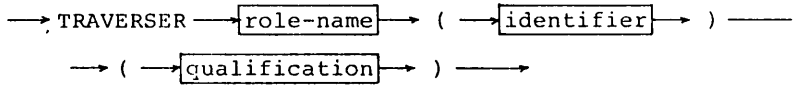
proc-mode



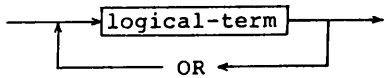
role-bind



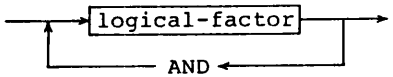
traverser-spec



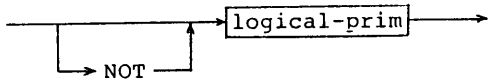
logical-exp



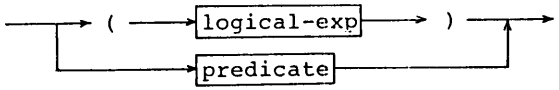
logical-term



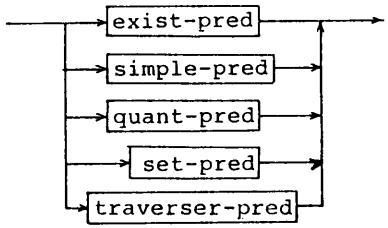
logical-factor



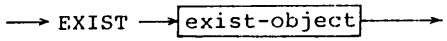
logical-prim



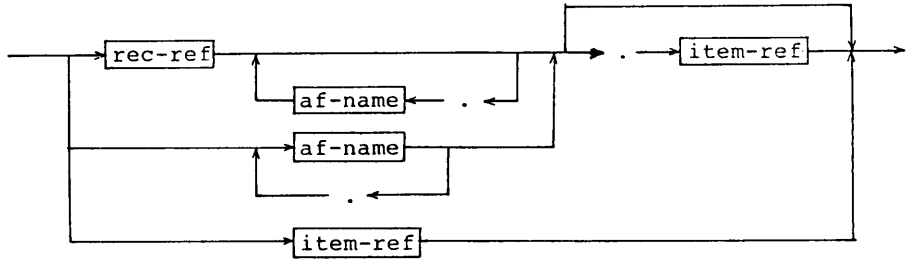
predicate



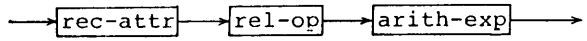
exit-pred



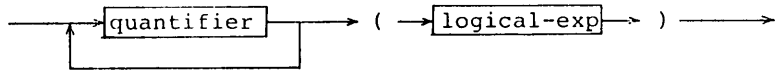
exist-object



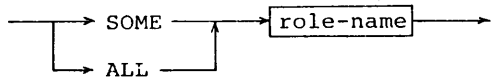
simp-pred



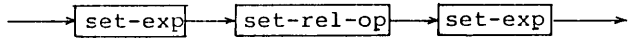
quant-pred



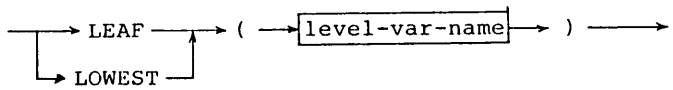
quantifier



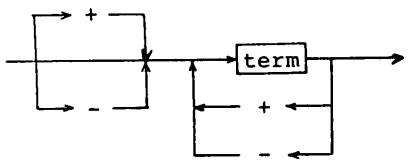
set-pred



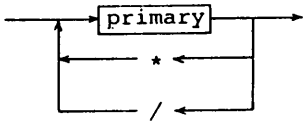
traverser-pred



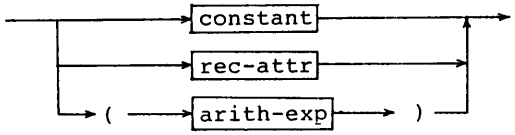
arith-exp



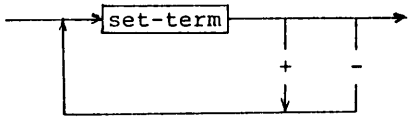
term



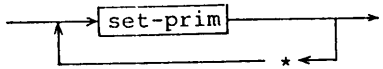
primary



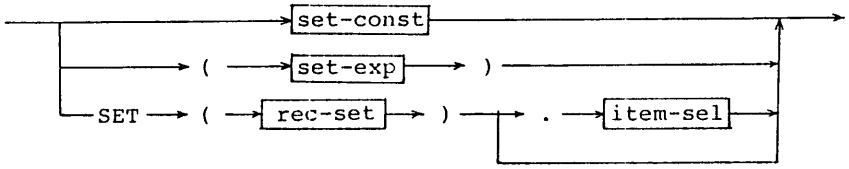
set-exp



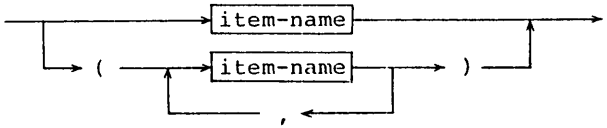
set-term



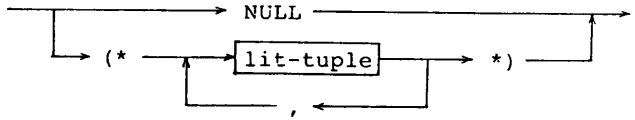
set-prim



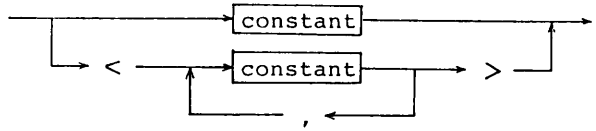
item-sel



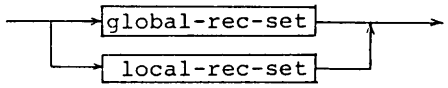
set-const



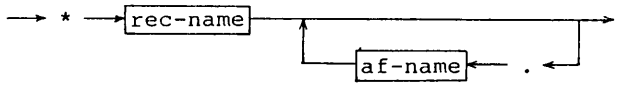
lit-tuple



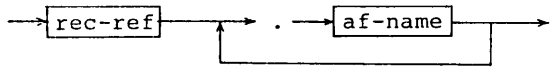
rec-set



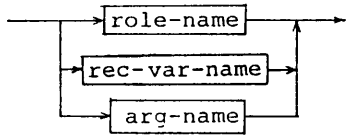
global-rec-set



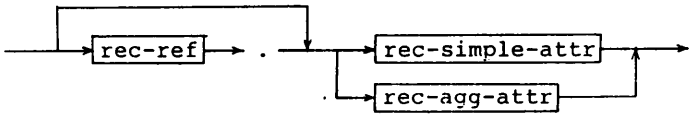
local-rec-set



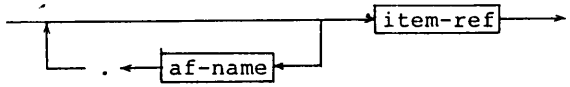
rec-ref



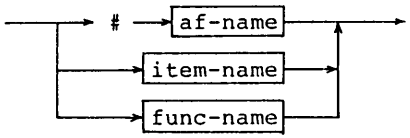
rec-attr



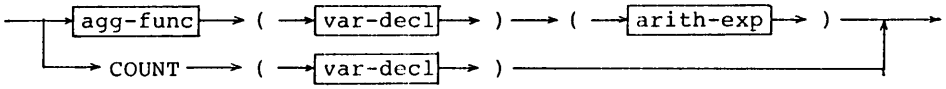
rec-simple-attr



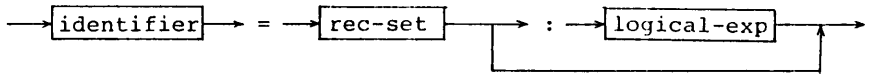
item-ref



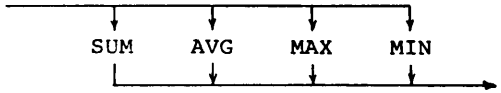
rec-agg-attr



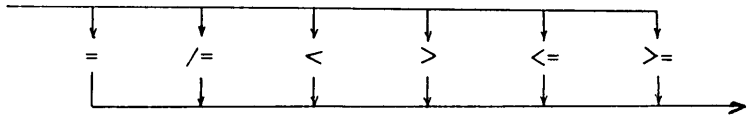
var-decl



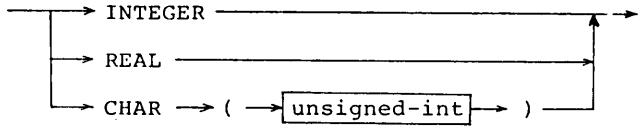
agg-func



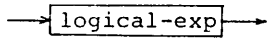
rel-op



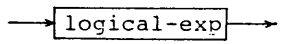
data-type



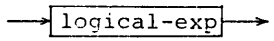
rec-sel-exp



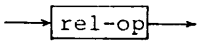
inter-cxt-cond



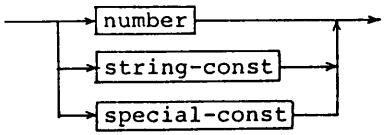
qualification



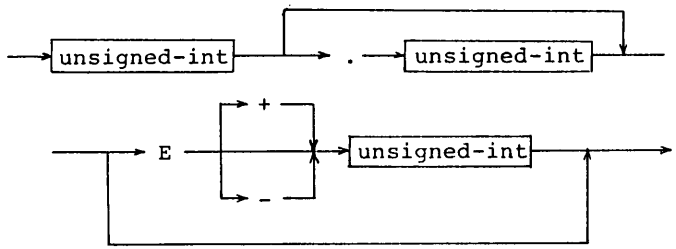
set-rel-op



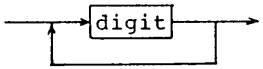
constant



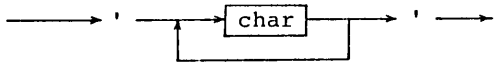
number



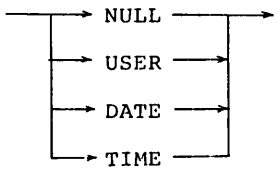
unsigned-int



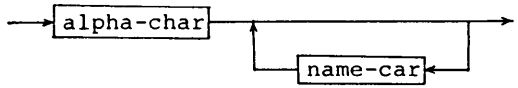
string-const



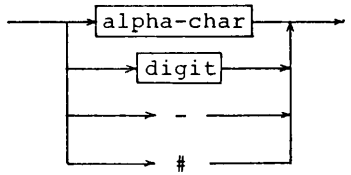
special-const



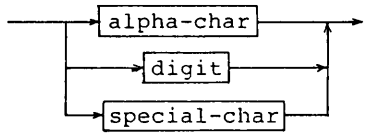
identifier



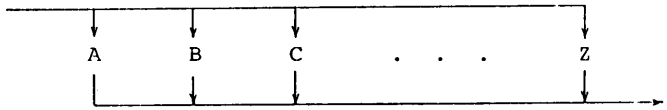
name-char



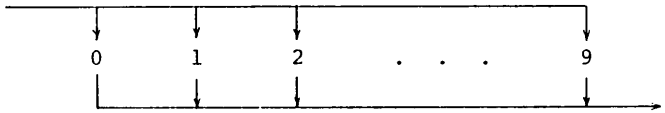
char



alpha-char



digit



special-car