

SQL上の第4世代言語：SL/SQL

池田 秀人 北川 文夫

広島大学総合情報処理センター

データベース言語はSQLが標準化され、商用プロダクトに広く受入れられる事によって、コマンド型および3GL埋めこみ型言語はほぼ完成したと見る事ができるだろう。これによって、現在多くの商用4GL (Fourth-Generation Language: 第4世代言語) もSQLをサポートし始めた。このような動きは今後益々広がっていくと思われる。しかし、殆どの4GLプロダクトが今までサポートしていた言語の構造に引き摺られて、SQL上の4GLの有るべき姿を見失っているようにみえる。

本論文は、SQL上の4GL設計をする上で考えなくてはならない要件について考察し、その観点から現状の4GLの持っている問題点を論じる。更に、この要件を満たす4GLの一つとして画面フローをベースにした新しい4GL: SL/SQLを提案する。SL/SQLは非手続き的な、均一要素に集まりで手続き言語の持つ記述能力と同等な能力を持たせると同時に、各要素の共用性を高めるという特徴を持ち、エンドユーザがプログラムを作成し易い様に外部から見た動作を記述する方法を採用している。最後に、SL/SQLの実現の方法について論じている。

A Fourth-Generation Language on SQL: SL/SQL

Hideto IKEDA and Fumio Kitagawa

Information Processing Center, Hiroshima University
1-1-89 Higashi-Senda, Naka, Hiroshima 730 JAPAN

By the standardization of command SQL and embedded SQL, major trend of study in database languages has been changed into independent procedural languages for SQL databases. Almost 4GLs (Fourth-Generation Languages) have been extended to support the standard SQL. Such movement will spread wider and wider. But these extensions of 4GL are strongly affected by the language structure of previous version and it seems to lost the way to ideal 4GL on SQL.

This paper discusses the design policies of ideal 4GL on SQL and problems of current 4GL products from the point of view. On the basis of the discussion, a new 4GL, called SL/SQL, is proposed. A procedure of SL/SQL is described as a collection of homogenous elements, called screens, and has same power as conventional procedural 4GLs. SL/SQL can also provide friendliness of the language for endusers and make commonality of procedural components higher.

This paper also discusses some possible approaches for implementation of SL/SQL.

1. はじめに

データベース言語SQLの標準化[1]でデータベース言語の研究・開発は新しい局面を迎えた。これからのデータベースはSQLでアクセスするデータベース管理システム(DBMS)に格納されていると仮定してデータベース言語を設計せざるを得なくなったからである。エンドユーザ志向の高生産性データベース言語である第4世代言語(Fourth-Generation Language: 4GL)[4, 5]も続々と標準SQLを基本言語としたデータベース管理システム(SQLデータベースと呼ぶ)をサポートし始めた[2]。

現状の4GLのSQLサポートの方法は様々であるが、大別すると次の3つに分類できる。

- (1) <完全準拠型>: 手続き型4GLでレコード単位の処理を基本にした言語に標準SQLのDDL及びDML規格をその言語の基本命令として加えることにより言語仕様を拡張したもの。埋込型SQLの標準的方法がほぼそのまま使える。
- (2) <DDL準拠型>: SQLのDDL規格合致性のみを実現し、DMLとしては独自の方法を採用したもので、この分類に属する多くのものは、標準SQL-DMLの特長であるカーソルの使用を避けている。例えば Model 204。
- (3) <被覆型>: 従来の言語仕様を変更せず、内部でSQLコマンドを生成して、SQLデータベースにアクセスできる様にしたもの。SQLをアプリケーションがデータベースをアクセスするための中間的言語として位置付け、ユーザにSQLのシンタックスを書かせない。比較的普及している4GLはこの方法でSQLデータベースをサポートしているものが多い。例えば、FOCUS, RAMIS-II 等。

このようなSQLサポートの方向に関して、4GLの立場からどの方向が望ましいのか議論を起こすことは重要な意義があると考えられる。なぜならば現状の4GLはデータベース言語としての基本性格を持っており標準データベース言語であるSQLのサポートは避けて通れない問題であるからである。

一方、4GLはエンドユーザ志向の高生産性データベース言語としての役割を果たすため3世代言語には必ずしも無かった機能が要求される。エンドユーザでも簡単に使える様にするため、(a)メニュー方式、アイコン方式、自然言語方式、テンプレート方式、手続きチャート方式等の視覚的インタフェースが採用され、(b)これにマルチスクリーン、画面スクロール等の方法で仮想画面の拡大をはかり、(c)多様な文字フォントとカラーの採用、グラフや画像等のマルチメディア表示でより説得力のある画面を可能にすることで、エンドユーザがより少ない予備知識で、自然な発想で、かつ効率良く情報の取得や操作が出来るようにすることを目指してきた。SQLデータベースを前提としながら、このような4GLの目標を実現するためにはいろいろな問題を解決しなければならない。

本論文は、このような問題を解決する新しい4GLとしてSL/SQLを提案したものである。まず(1)SQL上の4GL設計をする上で考えなくてはならない要件について、現状の4GLの持っている問題点を指摘しながら考察する。更に、(2)この要件を満たす4GLの一つとして画面フローをベースにした新しい4GLであるSL/SQLを提案する。つぎに(3)SL/SQLの特徴として、非手続き性、要素の均一性および外部動作記述志向性を解説し、これらの性質が4GLの基本要件であるエンドユーザ志向性、高生産性にどのように関連するかをしめす。またSL/SQLの持つ記述能力について手続き言語の持つ記述能力と比較しながら論じる。最後に、(4)SL/SQLの実現の方法について幾つかの可能性を論じる。

2. SQLデータベース上のSQLの満たすべき要件

ここではまずSQLを標準言語として備えたデータベース管理システムでデータが管理される事を前提としてあるべき4GLの姿を議論する。そのためには4GLの基本要件である(1)エンドユーザ志向、(2)高生産性、(3)データベース操作のそれぞれにとってSQLがどのような役割を果たし得るか、SQL上でどのような工夫が成し得るかを検討して見よう。

(1) <エンドユーザ志向システムを実現する上でのSQLの役割>

SQLデータベースの上でエンドユーザ志向システムを実現するにはいくつかの問題がある。データベースを幾つかの(基本)表の集まりと考え、その表からユーザの欲しい(導出)表を宣言的に構成するという方法はエンドユーザにとっても、SQL側にとっても望ましい方法であり、SQL上の4GLで有る限り、標準SQL-DDLの構造はそのまま踏襲しても4GLの要件を満たす。しかし、基本表の数が多くなったり、列の数の多い表が含まれているとエンドユーザにとって導出表を定義することが困難になるので、データ辞書(またはデータカタログともいう)を使って関係のある表を探させたり、テンプレート方式で表を定義させたりするようなインタフェースは4GLとしては望ましいものである。

..... 標準SQL-DDLへの準拠

問題はSQL-DMLである。標準SQL-DMLはCOBOLやPL/1などのレコード単位の処理を基本にし

た手続き型言語に埋め込んで使うことを基本にして提案された。4GLが標準SQL-DMLのカーソル処理に引き摺られると、必然的にレコード単位の処理を記述するものになり、手続き的性格の強いものにならざるを得ない。手続き型4GLならこの方法で対応できるが非手続き4GLとしては、これは必ずしも望ましいものではない。

..... カーソル処理の放棄

4GLは画面対話型アプリケーションを柔軟に実現できるものが望ましい。SQL上の4GLでこの場合問題になるのは画面に表示される情報（の集まり）の構造と、SQLの表構造の違いである。一般に4GLが画面として表示したい情報の集まりは、SQLの複数枚の表になる。また画面対話型アプリケーションプログラムの外部スペックは画面レイアウトと画面フローとして表現できる。

..... 外部仕様 => 画面フロー

外部スペックがこの方法を採用と、アプリケーションプログラムの内部スペックは表の集まりと画面との写像としての役割を持つことになる。構造の異なる二つの対象の間の写像をどのように表現するかが問題である。現状の多くの4GLは画面側の構造に制約を付けることでこの問題を避けている。

..... 内部仕様 => 表集合と画面間の写像

(2) <高生産性言語を実現する上でのSQLの役割>

アプリケーション開発の生産性を高めるために多くの4GLで用いられてきたのは、非手続き化、汎用化、共有化高級化の4つの方法である。

非手続き化は画面やレポートのレイアウトの定義など非手続き的に行なえる部分の比率を高めることでアプリケーション開発をエンドユーザでも簡単に行なえるようにし、かつ開発作業の分割化、並列化を行ない、生産性の向上に重要な役割果たしてきた。非手続き化は4GLの基本と呼んでもよいアプローチであり、これを更に進めることが重要である。

..... 非手続き部分の比率拡大

SQLはそれまでのデータベース言語が手続き的に行なってきたデータベースアクセスを導出表として宣言的に定義し、これによりアプリケーションの論理展開の簡素化に大きな役割をはたした。4GLでもこの長所は放棄すべきではない。

..... SELECT文への準拠

汎用化することでプログラムの可用性を高め、これにより生産性を向上させようとするアプローチも4GLにとつて基本的である。データベースエディタ、会話型検索言語などはどのようなデータに対しても同一のプログラムで目的の処理が行なえるようになっており、これにより類似のプログラムをデータの構造に応じて開発することを避けることができた。非手続き型4GLはこのような汎用プログラムを指していることが多い。このようなプログラムを豊富に提供することも有意義であるが、もっと重要なことはこのような汎用プログラムが簡単に開発できる機能を提供することである。

..... 汎用プログラムの開発可能性

共有化は複数のプログラムがいくつかの部分で共有することで3GLではサブプログラムという方法でこれを可能にしてきた。4GLでは画面なども共有化の対象となっている。またではビューが共有でき部分の共有性を高めるのに一役かっている。共有性を高めることも生産性向上に重要な役割をはたすのでより拡大していかなくてはならない。

..... 共有部分の比率の拡大

高級化はプログラム言語の発展の中で一貫して追求されてきた伝統的方法である。4GLでもこれを更に進め、できるだけ少ないステップで目的の処理を表現してやろうという努力はいろいろ見られるが、高級化がややもすると機能の制約や柔軟性の欠如につながり、書けない処理が出てくることは4GLにとっても望ましくない。また無理な高級化で言語の自然性が失われることもある。このような点に注意しながら慎重に高級化を行なわなくてはならない。

..... 無理の無い高級化

(3) <データベース操作を実現する上でのSQLの役割>

標準SQL-DMLのデータベース操作にはカーソルによる方法と、探索による方法が用意されている。この内カーソルによる操作は手続き言語から使用すると便利であるが、4GLのような非手続き言語から使用するのには便利ではない。従って4GLでは探索型の更新文を中心に操作を書かせる方が良く考えられる。

..... 探索型更新文の採用

なお、ここで特に指摘する必要があるのは4GLの記述能力である。いくつかの4GL商品を見るとその言語のサポートしている範囲内の処理なら簡単に書けるが、その許容範囲を越えるとCOBOLなどの3GLで補完せざるを得ないものがある。4GLと銘打ってCOBOLなどの次の世代の言語である言語であると主張するなら3GLの助けを借りずに目的の処理が記述できなければならない。..... 4GLの完全性

3. 画面フローをベースにした新しい4GL：SL/SQL

前節で考察したように4GLは外部仕様としては画面フローであり、内部仕様としてはSQLの表の集まりと画面との間の写像として実現するのが望ましい。これを概念図にすると図1のようになる。写像は2つの方向がある。1つはどの表からどのようにデータを取りだしどのように画面に表示するか（データ変換）であり、他の1つは画面上のデータをデータベース更新にどのように結び付けるか（トランザクション）である。

1つのプログラムをこのような単位（ここでは画面モジュールと呼ぶ）に分割して記述するようにしたのがSL/SQLである。図1からも分かるように画面モジュールは図2のように4つの部分（ここでは節<セクション>と呼ぶ）から成っている。

各節の役割は次の様になっている。

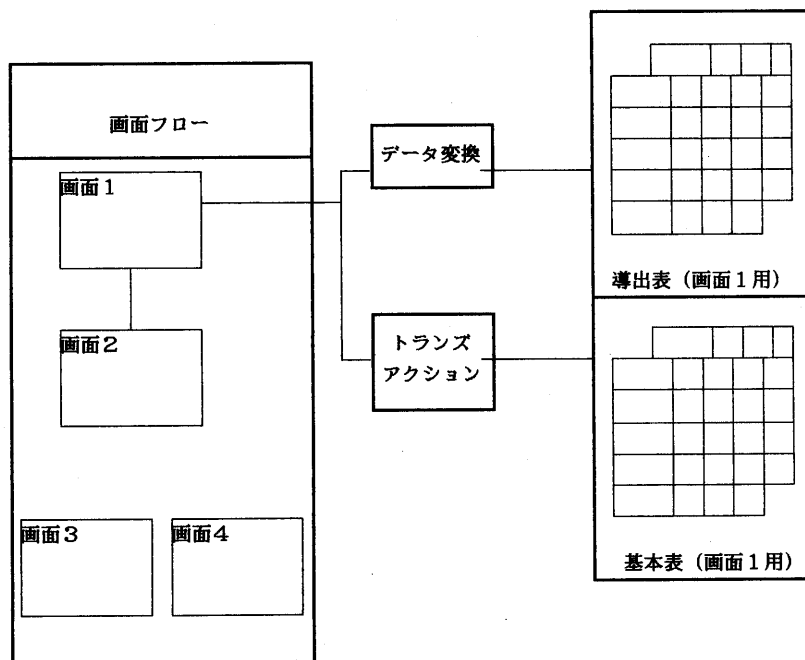


図1 SL/SQLの概念図

画面様式節	・画面の書式すなわちどのようなデータを何処にどのように表示するかを指示したり、画面を採色したり、罫線を引いたり、枠で囲んだりなどの効果表示もこの節で指定する。
データ変換節	・導出表やキーボードからの入力データをもとに、表示データや更新トランザクション用データに変換する節。いろいろな関数を用意しており、これを組み合わせて表現する。また条件記述もできる。
トランザクション節	・データ変換節で作り出されたデータをもとにデータベース更新を指示する節。記述された条件に従って追加、削除、置換、挿入の4つの更新が書ける。
リンク節	・当該モジュールからの飛びだし条件と、次に呼ぶべき画面モジュールが指示される節。この時パラメータも一緒に渡る。

図2 画面モジュールの構造

SL/SQLのシンタックスの詳細は付録に示してあるが、ここでは例を用いて機能の概要を簡単に説明しよう。

```

DEFINE_SCREEN document_ir (document_set) 80*24
("KEYWORD=" + kwd, "DOCUMENT#:" + dno, "TITLE:" + tit, "AUTHOR:" + aut,
"JOURNAL:" + jnl, "ABSTRACT:" + abs, "REMAINED(" + rmd + ")")
    AT (1, 1)-, WITH (HOLLIZONTALLY, BOX(BOLD), H-LINE(SINGLE)) ..... (1)
FUNCTION
kwd=INPUT(KEY) IF INITIAL; ..... (2)
d_set=SELECT((doc#, author, title, journal, abstract),
    FROM(document_db),
    WHERE(keyword=kwd),
    ORDER_BY(author)) IF INITIAL; ..... (3)
d_set=&document_set IF NEXT; ..... (4)
(dno, tit, aut, jnl, abs)=EXTRACT(d_set, 1) IF d_set.NE.NULL; ..... (5)
-rmd=COUNT(d_set); ..... (6)
DATABASE
stat-ir=INSERT(UID, DATE, TIME, kwd) IF INITIAL ..... (7)
CALL
document_ir(d_set) IF d_set.NE.NULL.AND.INPUT(KEY).NE.NULL; ..... (8)
EXIT IF d_set.EQ.NULL.OR.TIMEOUT(5min); ..... (9)

```

図3 SL/SQLの画面モジュールの例

この例は文献検索のプログラムをSL/SQLで書いたものである。キーボードからキーワード(kwd)を入力し、それを含む文献集合(d_set)をSQLデータベースから検索し、画面に1件ずつ文献番号(doc#)、標題(title)、著者名(author)、収録雑誌名(journal)、抄録(abstract)と残りの文献数(rmd)を表示するもので、キーボードから何か入力があれば次の文献を表示し、もし全ての文献を表示し終わるか、キーボードから5分間何も入力が無い場合は終了(EXIT)する。

この例のSELECT関数(3)のパラメータはSQL-DMLのSELECT文の句と同じであるが、関数型に書くために少しシンタックスを変更した。また、カーソルの使用を避けるためORDER_BY句をパラメータとして加えた。SELECT関数の左辺に来るのは構造を持った変数である。この場合は5列の表である。データ変換部がレコード単位の処理をしないためにはこの様な構造を持ったデータを直接処理することが重要になる。

データの構造化は集合化、リスト化、組合せの3つの方法でおこなわれる。例えばSELECT関数(3)から作成されるのは5列の組合せ(文献番号、著者名、標題、収録雑誌名、抄録)のリスト(ORDER句を用いたため集合がリスト化された)である。この様な構造を持ったデータを直接操作する必要があるため、構造化データ同志の演算を行う関数が豊富に用意されている必要がある。(5)のEXTRACT関数は集合やリストの中から1つの要素を取り出す関数でSQL/DMLのカーソルとFETCH文に似ており、カーソルと言う形態は採っていないがカーソルと実質的に同じ記述能力がある。

FUNCTION節の(2-5)の文の末尾にはIF句がついている。これはデータ変換を条件付で行うためで、木目細かいデータ変換を記述するためにはこれは避けて通れない。INITIALという条件はこの画面を始めて呼び出した時に真となる(CALL節から同じ画面が呼ばれていることに注意)。この節の文は原則として記述順序に従属しない。(但し、順序に意味を付けたいときは(6)の文の様に文の前に継続記号(-)を付けて書く。)例えば、(2)と(3)の文は逆の順序に書いても(3)のSELECT関数を実行するためには変数kwdが必要となるため、(2)の文の方が先に評価される。

画面様式を定義しているのがIMAGE節である。この節の(1)の文の特徴は、構造化データを一つの文で画面に表示している点である。値ごとに表示位置を指示していた画面定義の方法から構造を持ったデータを1文で表示する方法に切り換えることでドキュメント性も良くなり、ステップ数も短くなる。

次にDATABASE節である。(7)の文では検索のログを統計表(stat-ir)に追加している。ここでは1件のレコードを追加しているだけだが、一般に複数のレコードを更新の対象にすることになる。標準SQL-DMLの探索

型更新文はこれに耐えられる機能をもっているのにこれに準拠している。

最後のCALL節は、IF句の条件を満足すれば指定の画面にパラメータと共に制御が渡される。この節の条件はこの節のIF句と異なり割込的に働く。この例でも分かる通り画面の呼び出しは再帰的に行うこともできる。またEXIT画面はプログラムの終了を意味しシステムで予め用意されている。

この画面の実行に当たっては、まずFUNCTION句が評価される。こうしてすべてのデータが作成された後、IMAGE節の定義に従って表示される。但し、INPUT関数でキーボードからデータを入力する必要がある場合は、表示後に行われ、更にその入力に応じた検索や変換が行なわれるものが評価される。表示がすべて終了した段階でDATABASE節のデータベース更新が行なわれる。CALL節で割込条件の無いものは、その後に制御が渡される。IF句のあるものはシステムで常にその条件が満たされるかどうかチェックされ、満足されればその画面が呼びだされる。

4 SL/SQLの特徴

この節ではSL/SQLの特徴を4GLという観点から検討して見よう。

(1) <非手続き性> SL/SQLではいまだ非手続き的に表示されていた画面やデータベース定義、SQLで可能になったデータ検索ばかりでなく、いまだ非手続き的には表現しにくかったデータ変換も非手続き的に表現しようとした。このため構造化変数とその変換関数の導入が重要な役割を果たす。データ変換を関数群で表現すると手続き的要素の一部は関数の評価順序と評価条件として表現される。理論的にはあらゆるデータ変換は関数の集合として表現されるが一部手続き的に表現した方が分かり易いこともありSL/SQLでは順序に意味のある関数列も許している。SL/SQLでは非手続き的要素の比率を増すことを目標として設計されているが完全に手続き的表現を取り除くことは4GLの完全性からできない。SL/SQLでは画面フローという方法で手続き処理を表現する様になっている。

(2) <モジュールの均一性> 従来の3GLでプログラムを書くとはそれは入出力文、データ変換文、流れ制御文などの異なる機能を持つ命令文の列(リスト)として表現された。SL/SQLでは均一の構造を持つ画面モジュールの集合として表現される。こうすることで各モジュール単独のデバッグも容易になり、モジュールの共有性は高まり、各モジュールの機能も理解しやすくなり、ドキュメンテーションや保守も容易になるなどメリットは大きい。

(命令の均一性の特徴はSMALLTALK等でも見かけることができる)

(3) <モジュールの共有性> モジュールを共有化するためにSL/SQLではモジュールの動作をパラメータで制御する方法を採用した。パラメータは構造化変数が入るので、複雑な構造をもつデータの受渡が可能であり、これで検索したデータを順次処理して行くことも可能であり、流れを制御することも可能である。また再帰呼出しにより高い機能を持つモジュールを作成することも出来る。

(4) <モジュールの汎用性> SL/SQLではモジュールの汎用化のためデータ型に依存しない変換や表示機能を含んでいる。殆どのデータ変換関数や表示指示はすべてのデータ型に使えるようにしたり、データ定義の時指定した列名を取り出す関数を用意することで、データベースエディタや会話型検索システムのような汎用プログラムがSL/SQLで作成できるようになっている。

(5) <SQL-DDL, 探索型SQL-DMLへの準拠> SL/SQLではSQL-DMLのカーソル処理を除く全ての機能に準拠している。(規格に準拠していると表現しているのは、SQL-DMLの各文を関数型に書換えているためである)カーソル処理については前の例にも使われている通り、構造化変数(集合カリスト)から指定個数の要素を取り出す関数EXTRACTを用意しており、これで同様の処理が可能になる。

5 おわりに

本論文では画面フローをベースにした新しい4GLとしてSL/SQLを提案した。SQLがデータベース言語として標準化され、多くのDBMSがSQLを標準言語として採用し始めた今日、SQLを主言語とするデータベース管理システムを4GLがサポートすることは避けて通れないものとなって来た。一方SQLはDDLでは非手続き的にデータ構造を定義でき、4GLとしても遜色のないものであるが、SQL-DMLの内のカーソル処理は4GLとしては採用しにくい。このような理由からSL/SQLではデータベース操作をカーソル処理を除く標準SQL-DMLに準拠しながら、より高い生産性を求めて非手続き性、均一性、共有性、汎用性をもつモジュールの集まりとしてプログラムを実現する方法を採った。この時、重要な役割を果たすのが構造化変数であり、構造化変数を処理する関数群である。

このSL/SQLを4GLとして実現する場合、幾つかの注意を要する点がある。まづ、画面の様式を定義するた

めのスプリーンペインタの採用である。SL/SQLは言語としては文字列で記述するシンタックスをもっているが、このシンタックスをより視覚的に作り出す方法は多くの4GLがそうしているように工夫が必要となる。

次にデータ辞書の機能である。SL/SQLで作成した画面や関数の管理を効率良く行うためにはそれらの管理システムが必要となる。またSQLデータベースでも基本表の数が多くなるとデータカタログが必要となる。これらの情報資源を統合して管理する辞書システムが無ければ4GLの役割は十分に果たせない。

最後にSL/SQLのマルチメディアへの拡張を考えて見よう。文字、数値ばかりでなく画像や音声を含んだデータを管理し、操作するためには、画面の定義や関数にもっと豊富な機能が必要となる。またデータベース管理システムも現状のSQLでは十分に対応できない。このためマルチメディア用のDBMSを開発する必要がある。われわれは、論文[3]でこの試みを行っている。まだ研究としては初期の段階であるが実用システムとして実現できるところまで深めたいと思っている。SL/SQLはそのプロトタイプとしての役割も果たすことになる。このシステムを使って、本論文で提案したアイデアの実用性を検証する計画である。

参考文献

- [1] American National Standard Institute: Database Language SQL, Document ANSI X3.135, (1986).
- [2] Ikeda, H.: Fourth-Generation Languages: Current Aspects and Future, Advanced Database Symposium (1988). (In Japanese)
- [3] Kitagawa, F. and Ikeda, H.: Multi-Media Database Language SL/B5 based on Screen flow, To be published at International Symposium on Database Systems for Advanced Applications, Seoul, 1989.
- [4] Martin, J.: Application Development Without Programmers, Prentice-Hall, (1982).
- [5] Martin, J.: Fourth-Generation Languages, Prentice-Hall, (1985).

付録A SL/SQLの関数

関数名	ラメータ	機能の概要
EXTRACT APPEND	対象, 要素数, [条件] 対象1, 対象2 [, 位置]	対象から条件を満足する要素を指定された件数だけ取り出す 対象1に対象2を追加する。リスト型の場合は挿入位置を指定する。
ERASE CHANGE	対象, 条件 対象1, 対象2, 条件	指定された条件を満たす要素を対象から削除する 対象1の要素のなかで条件を満足する要素を対象2と置き換える
*	対象1, 対象2	2つの対象の積
+	対象1, 対象2	2つの対象の和
-	対象1, 対象2	2つの対象の差
/	対象1, 対象2	2つの対象の商
COUNT	対象	対象の要素数
INDEX	対象1, 対象2	対象1から対象2の位置を探す
TRANS	対象, 変換指定	対象を指定の変換指定に従い変換する
EDIT	対象, 編集指定	対象を指定の編集指定に従い編集する
INPUT	入力装置	データの入力
TYPE	対象	対象の型

PROGRAM := { MODULE }
MODULE := (モジュール名 (パラメータ) 領域サイズ, 画面様式節、 トランザクション節、リンク節) モジュール名 := 英数字文字列 パラメータ := 変数
画面様式節 := IMAGE 様式記述 様式記述 := 表示要素 表示環境 様式記述, 様式記述 表示要素 := 変数 定数 システム値 { 表示要素, 表示要素, ... } < 表示要素, 表示要素, ... > (表示要素, 表示要素, ...) 関数 (表示要素) 表示環境 := AT 座標 [WITH 表示属性] 座標 := (整数, 整数) [- [(整数, 整数)]] 表示属性 := SIZE (領域サイズ) BASE (背景色) BOX (線種) H-LINE (線種) V-LINE (線種) HOLLOZONETALLY VERTICALLY TABULAR (次元) TREE 領域サイズ := 整数 X 整数 背景色 := WHITE BLACK RED BLUE GREEN YELLOW PURPLE PINK BROWN LIGHT-BLUE LIGHT-GREEN LIGHT-YELLOW THROUGH 線種 := NONE SINGLE DOUBLE BOLD システム値 := DATE TIME UID MODULE-NAME SYSTEM-NAME
データ変換節 := FUNCTION 変換記述 変換記述 := 変換式 変換記述 ; 変換記述 変換式 := 変数 = 関数 (対象) [IF 条件式] 変換式 ; - 変換式 条件式 := 対象 . 関係 . 対象 割込条件 (SOME 変数) 条件式 (ANY 変数) 条件式 条件式 . AND . 条件式 条件式 . OR . 条件式 . NOT . 条件式 関係 := EQ GE GT LE NE INC ELM IN 割込条件式 := ATTEN TIMEOUT (時間) ERROR 時間 := 整数 sec 整数 min 整数 h
トランザクション節 := DATABASE 更新記述 更新記述 := 更新文 IF 更新条件 更新記述 ; 更新記述 更新文 := <DELETE : 探索> <UPDATE : 探索> <INSERT> 標準SQL-DMLのシンタックスに従う
リンク節 := CALL 呼出し記述 呼出し記述 := モジュール名 (変数) [IF 条件式] 呼出し記述 ; 呼出し記述