データ・ベースシステム 71－6
（１９８９．５．１８）

動的ＳＱＬにおけるインタフェース

附田克晃　（株）日立ソフトウェアエンジニアリング
大里博志，三谷政昭　（株）富士通

ＩＳＯのデータベース言語規格ＳＱＬ２における動的ＳＱＬ機能導入について述べる。規格化に際しての基本的問題は、アプリケーションプログラムとデータベース管理システム間における動的ＳＱＬ文のパラメタ記述情報受け渡しインタフェースにある。このインタフェース方式として、（１）ホスト言語の構造体による構造体インタフェース、（２）一時表利用のリレーショナルインタフェース、（３）パラメタ記述情報受け渡し専用ＳＱＬ文導入による関数インタフェース、の３つの案が出されて検討されてきた。現検討段階では、関数インタフェースに基づき検討を進めているが、広くインプリメントされている構造体インタフェースも併存させる方式としている。関数インタフェース方式は、ホスト言語との独立性と分かり易さにおいて他方式よりも優れている。

INTERFACE TO DYNAMIC SQL

KATSUAKI TSUKUDA
Hitachi Software Engineering Co., Ltd.
6-81, Onoe-machi, Naka-ku, Yokohama, 231 Japan
HIROSHI OHSATO, MASAAKI MITANI
Fujitsu Ltd.
140 Miyamoto, Numazu-shi, Shizuoka, 410-03 Japan

The dynamic SQL in ISO database language standard SQL2 is discussed. A fundamental issue to be solved is a parameter passing interface between an application program and a database management system. Three approaches are presented and discussed; (1) a structure interface using a host language data structure, (2) a relational interface using temporary tables, and (3) a functional interface introducing special SQL statements for passing parameters. Currently, the functional interface is adopted besides the structure interface widely implemented. The functional interface supports a better interface in terms of independence from host languages and easiness of understanding.

１．　まえがき

多くの場合、ＳＱＬ文は実行時点より前に知られており、埋込み構文を用いてアプリケーションプログラムの中に書くことができる。しかし、端末画面から直接、又は間接的にＳＱＬ文を入力することによってデータベースを操作させるようなインタフェースをサポートする場合、ＳＱＬ文を実行時点で動的に生成して実行できる機能が必要である。対話型アプリケーションや４ＧＬは、この代表例である。このための機能を動的ＳＱＬと呼ぶ。本機能は、汎用アプリケーションの移植性を保つために重要であるが、ＩＳＯのＳＱＬ規格には含まれていない。[1,2] そのため、ＩＳＯ規格委員会では、日本と米国が中心となり、本機能をＳＱＬ２に入れる検討を行なってきた。[3] その結果、ＳＱＬ２ ＤＰ(Draft Proposal)では、動的ＳＱＬが規定された。[4] 本論文は、動的ＳＱＬの導入過程で検討したホスト言語とデータベース管理システムとのインタフェースの問題について述べる。

２．　概説

２．１　検討経過

動的ＳＱＬは、１９８７年１０月にＡＮＳＩからＩＳＯへ提案された。[5] ここで提案された動的ＳＱＬのＳＱＬ記述子領域（後述）のインタフェースは、ホスト言語の構造体配列を利用する方式であった。同時期に、ＡＮＳＩでの開発と並行して、日本でも動的ＳＱＬの検討を行なっていた。ＩＳＯでは同年６月の東京会議で、ＳＱＬ標準に繰り返し群（repeating groups)と配列(arrays)が現れるのは望ましくないという理由で、複合データタイプのサポート提案に反対する結論を出していた経緯があり、ＡＮＳＩの方式のままでは、動的ＳＱＬをＩＳＯのＳＱＬ２に入れるのは困難という状況であった。

そこで、同年１０月のアムステルダム会議で日本から、ＳＱＬ記述子インタフェースに、ＳＱＬ２で導入された一時表を利用するリレーショナルインタフェースを、ＡＮＳＩの動的ＳＱＬ提案を基にして提案した。日本の提案は採用され、ＩＳＯのＳＱＬ２に動的ＳＱＬ機能が入った。

しかし、ＡＮＳＩは、リレーショナルインタフェースは分かりにくいなどの理由で再検討して、関数インタフェースを１９８８年１２月のシドニー会議で提案した。このＡＮＳＩの提案は採用された。

一方、同会議で日本からは、ＳＱＬで扱う制御データ（ＳＱＬ記述子領域のデータや診断情報データ）も関数インタフェースで統一すべきであるとして、診断情報取得に対する関数インタフェースを提案した。この提案も採用された。

２．２　動的ＳＱＬ機能の概要

アプリケーションを書く時点で知られていないＳＱＬ文を処理するために使うのが、動的ＳＱＬ機能である。本機能の中には、動的ＳＱＬ文と動的カーソル宣言がある。以下に、カーソルを使用しない場合と使用する場合に分けて概要を説明する。

（１）カーソルを使用しない場合

①　ＰＲＥＰＡＲＥ　ＳＱＬ文識別子　ＦＲＯＭ　ＳＱＬ文変数

②　ＥＸＥＣＵＴＥ　ＳＱＬ文識別子　［動的ＵＳＩＮＧ句］

③　ＥＸＥＣＵＴＥ　ＩＭＭＥＤＩＡＴＥ　ＳＱＬ文変数

ここで、動的ＵＳＩＮＧ句は

ＵＳＩＮＧ　ホスト変数　［ホスト変数　．．．　］

または

ＵＳＩＮＧ　ＤＥＳＣＲＩＰＴＯＲ　ＳＱＬＤＡ構造体名

の形式である。

動的ＰＲＥＰＡＲＥ文①は、繰り返し実行のため生成されたＳＱＬ文を準備（解釈）するため

に用いられる。ＳＱＬ文識別子は、被準備文（準備されるＳＱＬ文）を識別するために用いる。被準備文はＳＱＬ文変数の内容として与えられる。ＳＱＬ文変数には、例として次のようなＳＱＬ文を、①を実行するまでに設定しておく。

ＳＱＬ文変数 ← 'UPDATE EMPLOYEE SET DEPTNO = ? WHERE EMPNO = ?'

この例のような非ＳＥＬＥＣＴ文の場合、動的ＥＸＥＣＵＴＥ文②は、あたかもそれがプログラムが書かれたときに書かれたかのように、被準備文と動的パラメタ（被準備文中で "?" で表される）を結び付けて実行するのに用いられる。動的パラメタの値は、動的ＵＳＩＮＧ句の指定で、ホスト変数またはＳＱＬ記述子領域と呼ばれる領域（ＳＱＬＤＡ構造体）を用いて与えられる。ＳＱＬ記述子領域は、各々の入力（動的）パラメタ又は出力値を記述するＳＱＬＶＡＲの実現値(occurrences)の数（ＳＱＬＤ）、及びＳＱＬＶＡＲ記述を含む。ＳＱＬＶＡＲ記述は次の内容から成る。

SQLTYPE　：ＳＱＬで規定されるデータ型
SQLLEN　：文字型データのデータ長
SQLSCALE　：真数型データのデータ位取り
SQLPRECISION　：数値型データのデータ精度
SQLNULLABLE　：ナル値の可能性（0:ナル値はとらない,1:ナル値を取りうる）
SQLDATA　：パラメタに対応するアプリケーションのデータ領域アドレス
SQLIND　：SQLNULLABLEが１の時に値がナル値か否かを示すナル標識値（負:ナル値）
SQLNAME　：導出列（検索される列）の名前

動的ＥＸＥＣＵＴＥ　ＩＭＭＥＤＩＡＴＥ文③は、非ＳＥＬＥＣＴ文の１回のみの準備及び実行に用いられ、動的ＰＲＥＰＡＲＥ文と動的ＥＸＥＣＵＴＥ文を続けて実行するのに等しい。（但し、動的パラメタの使用はできない。）

（２）カーソルを使用する場合

① ＰＲＥＰＡＲＥ　ＳＱＬ文識別子　ＦＲＯＭ　ＳＱＬ文変数
④ ＤＥＳＣＲＩＢＥ　ＳＱＬ文識別子
　　ＵＳＩＮＧ　ＤＥＳＣＲＩＰＴＯＲ　ＳＱＬＤＡ構造体名
⑤ ＤＥＣＬＡＲＥ　カーソル名　ＣＵＲＳＯＲ　ＦＯＲ　ＳＱＬ文識別子
⑥ ＯＰＥＮ　動的カーソル名　［動的ＵＳＩＮＧ句］
　カーソルを開いた後、⑦（と⑧または⑨）を繰り返す。
⑦ ＦＥＴＣＨ　動的カーソル名　動的ＵＳＩＮＧ句
　必要に応じて、⑧または⑨を実行する。
⑧ ＵＰＤＡＴＥ　表名　ＳＥＴ　設定句：位置づけ　．．．
　　ＷＨＥＲＥ　ＣＵＲＲＥＮＴ　ＯＦ　動的カーソル名
⑨ ＤＥＬＥＴＥ　表名　ＷＨＥＲＥ　ＣＵＲＲＥＮＴ　ＯＦ　動的カーソル名
⑩ ＣＬＯＳＥ　動的カーソル名

カーソル使用のＳＥＬＥＣＴ文の場合以下のようになる。①により準備されたＳＥＬＥＣＴ文の導出列の記述（列数、列のデータ型、等（以降、列記述と呼ぶ））は、ＤＥＳＣＲＩＢＥ文④の実行によって、ＳＱＬ記述子領域を経由して得られる。被準備文は動的カーソル宣言⑤によってカーソルと結び付けられる。動的ＯＰＥＮ文⑥によって、カーソルは開かれ、パラメタは被準備文と結び付けられる。動的ＦＥＴＣＨ文⑦は、開かれたカーソルを特定の行に位置付け、その行の列の値をホスト変数へ取り出す。動的ＵＰＤＡＴＥ：位置づけ文⑧と動的ＤＥＬＥＴＥ：位置づけ文⑨は、各々動的ＦＥＴＣＨ文で位置づけられたカーソルを用いた更新と削除をする。動的ＣＬＯＳＥ文⑩は、動的ＯＰＥＮ文によって開かれたカーソルを閉じる。

２．３　インタフェースの問題点と解決案

前述したように、被準備文の動的パラメタ及び動的ＦＥＴＣＨ文から生じる値のためのインタフェースは、ホスト変数のリスト、又はＳＱＬ記述子領域を通じてのいずれかである。しかし、ＤＥＳＣＲＩＢＥ文はＳＱＬ記述子領域によってのみ、列記述を受け取る。

既存のインプリメンテーションでは、このＳＱＬ記述子領域の内容は、ホスト言語での構造体配列による実現方式が多い。しかし、この構造体（ＳＩ：Structured Interface）方式ではホスト言語として構造体配列とポインタを扱えるＰＬ／Ｉ、Ｃ言語だけしかサポートできないという問題などがあり、ＳＱＬ記述子領域としてＳＱＬ２で導入された一時表を使用するリレーショナルインタフェース（ＲＩ：Relational Interface）方式が提案された。

ＲＩ方式では、すべてのホスト言語についてＤＥＳＣＲＩＢＥ文をサポートすることができる。しかし、ＲＩ方式では、まだデータ値やナル標識値をセットするホスト言語の領域へのポインタの扱いをうまく解決できていなかった。これを解決する方式として提案されたのが、関数インタフェース（ＦＩ：Functional Interface）方式である。ＦＩ方式では、構造体配列やポインタの問題点を解決し、他の提案に比べても理解し易くなっている。

これらの３つの方式の提案過程における問題点と解決案をまとめたのが表１である。

表１　各インタフェース方式の比較

項番	問題点	ＳＩ方式	ＲＩ方式	ＦＩ方式
1	ＳＱＬ記述子のインタフェースがホスト言語の構造体に依存する（ＳＩ方式）	X	O (①)	O (②)
2	ＳＱＬＤＡＴＡとＳＱＬＩＮＤの設定にポインタ操作が必要（ＳＩ方式）	X	Δ (③，④)	O (⑤)
3	ＳＱＬＤＡ／ＳＱＬＶＡＲ一時表の操作が複雑（ＲＩ方式）	Δ	X	O (⑥)

○印：問題なし、Δ印：やや問題あり、Ｘ印：問題あり

（　）内の番号は、問題点に対する以下の解決案の番号である。

解決案：①一時表を操作することによって、ＳＱＬ記述子の情報を扱う。（ＲＩ方式）
②関数操作によって、ＳＱＬ記述子の情報を扱う。（ＦＩ方式）
③処理系作成者定義のポインタドメインとして扱う。（日本提案のＲＩ方式）
④”ＲＥＦＥＲＥＮＣＥ　ＴＯ”句の導入によって、ポインタの概念を直接意識させない。（ＡＮＳＩにより改善されたＲＩ方式）
⑤関数操作によって、ホスト変数へ代入する。（ＦＩ方式）
⑥関数コールに相当するＳＱＬ文の導入で、簡潔な規則となる。（ＦＩ方式）

３．　個々のインタフェース

３．１　構造体インタフェース

ＡＮＳＩの最初の動的ＳＱＬ提案は、動的ＳＱＬをサポートしている多数の既存のインプリメンテーションにほぼ共通の基本的な仕様に基づいている。この提案では、動的ＳＱＬを埋込み構文でのみサーポートしており、モジュール言語では直接サポートしていない。その理由の一つには、提案仕様ではホスト言語とのインタフェースとして構造体（ＳＱＬＤＡ）を必要とするが、ＳＱＬ（即ちモジュール言語）では、構造体をサポートしていないためモジュールへＳＱＬＤＡを渡すことが難しいことがある。ここでは、この理由の中の構造体の問題について説明する。（モジュール言語でのサポートに関しては、４章の機能拡張で扱う。）

動的にＳＱＬ文を生成して実行するプログラムでは、検索される行の列記述を一般には知ることができない。この列記述を得るためには、ＤＥＳＣＲＩＢＥ文を実行する。このとき、アプリケーションプログラムは、そのプログラムで処理する可能性のある最大の列数分の列記述を受け取るためのＳＱＬ記述子領域（ホスト変数①図１参照）を確保しておく必要がある。アプリケーションプログラムはこの列記述（②，③）を基に検索行を受け取る領域（⑤）を確保し、そのアドレスをホスト変数（④）に設定して処理系に通知する。

そこで、一般に検索する列の数は複数あるため、列記述及び列受取領域を設定するホスト変数は複数必要となり、一般に構造体の配列による表現が容易であると考えられる。このようにＳＱＬ記述子領域（ＳＱＬＤＡ）に、ホスト言語の構造体配列を前提とする方式が、構造体インタフ

ェース方式である。図２に、ＳＥＬＥＣＴ文を動的に実行するときのアプリケーションプログラムと処理系とのインタフェースの概要を示す。

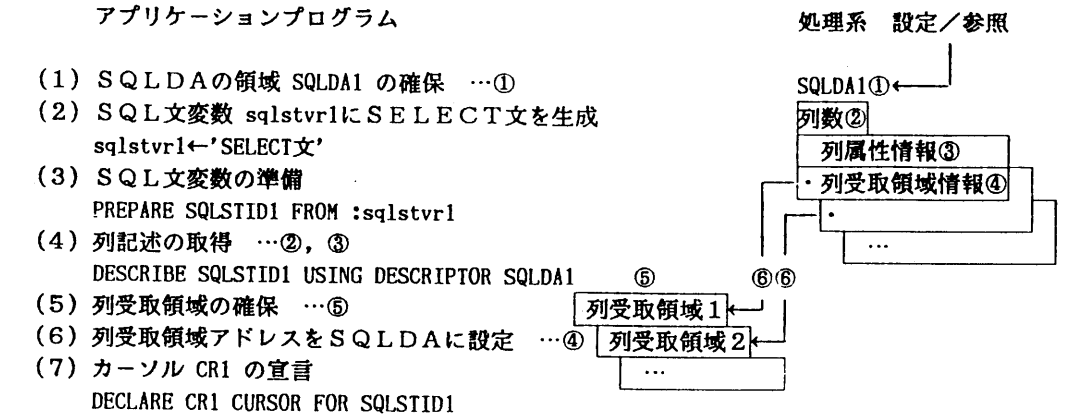

図１　構造体インタフェースの説明図

３．２　リレーショナルインタフェース

ＡＮＳＩの構造体インタフェースを改善するために、日本から、リレーショナルインタフェースを提案した。構造体インタフェースでのＳＱＬＤＡ構造体を、ＳＱＬ２で導入された一時表で定義して、ホスト言語からはこの一時表を操作することによって、構造体インタフェースと同等のインタフェース機能をサポートする方式がリレーショナルインタフェース（ＲＩ）方式である。ＲＩ方式では、ホスト言語に依存せずＳＱＬＤＡを利用した機能を実現できる。

また、ポインタ操作の必要性の問題については、ホスト言語のポインタ属性を直接意識させないように、ポインタドメインという処理系作成者定義として扱った。しかし、ポインタドメイン属性の列に関する挿入、更新、参照の規則の記述があいまいであったため、ＡＮＳＩは、”ＲＥＦＥＲＥＮＣＥ　ＴＯ”句を導入してＳＱＬＶＡＲ一時表のＳＱＬＤＡＴＡ、ＳＱＬＩＮＤ列の挿入、更新、参照の規則を明確にした。

そこでは、”ＲＥＦＥＲＥＮＣＥ　ＴＯ　単純相手指定”句の値は処理系作成者定義の単純相手指定の参照としている。（単純相手指定とは、ナル値をとり得ない値を代入するホスト変数指定のことである。）例えば、Ｃ言語では、”ＲＥＦＥＲＥＮＣＥ　ＴＯ　：ｘ”は”＆ｘ”（ｘのアドレス）として定義することができる。図２に、図１に対応するＲＩ方式の説明を示す。

３．３　関数インタフェース

ＡＮＳＩから、ＳＱＬ記述子領域をデータベース管理システムで管理しアプリケーションとのインタフェースは関数として行う方式が提案されている。アプリケーションがＳＱＬ記述子領域のデータを参照、更新する場合には、引数に値を設定して専用の関数を呼び出す。それに応じてデータベース管理システムでＳＱＬ記述子領域の操作を行う。これにより、ＳＱＬ記述領域の構造をアプリケーションで意識せずにデータの参照、更新ができるようにしている。

ここで、各ホスト言語毎にＳＱＬ専用の関数を設けることは関数名、データ型等で規格作業上

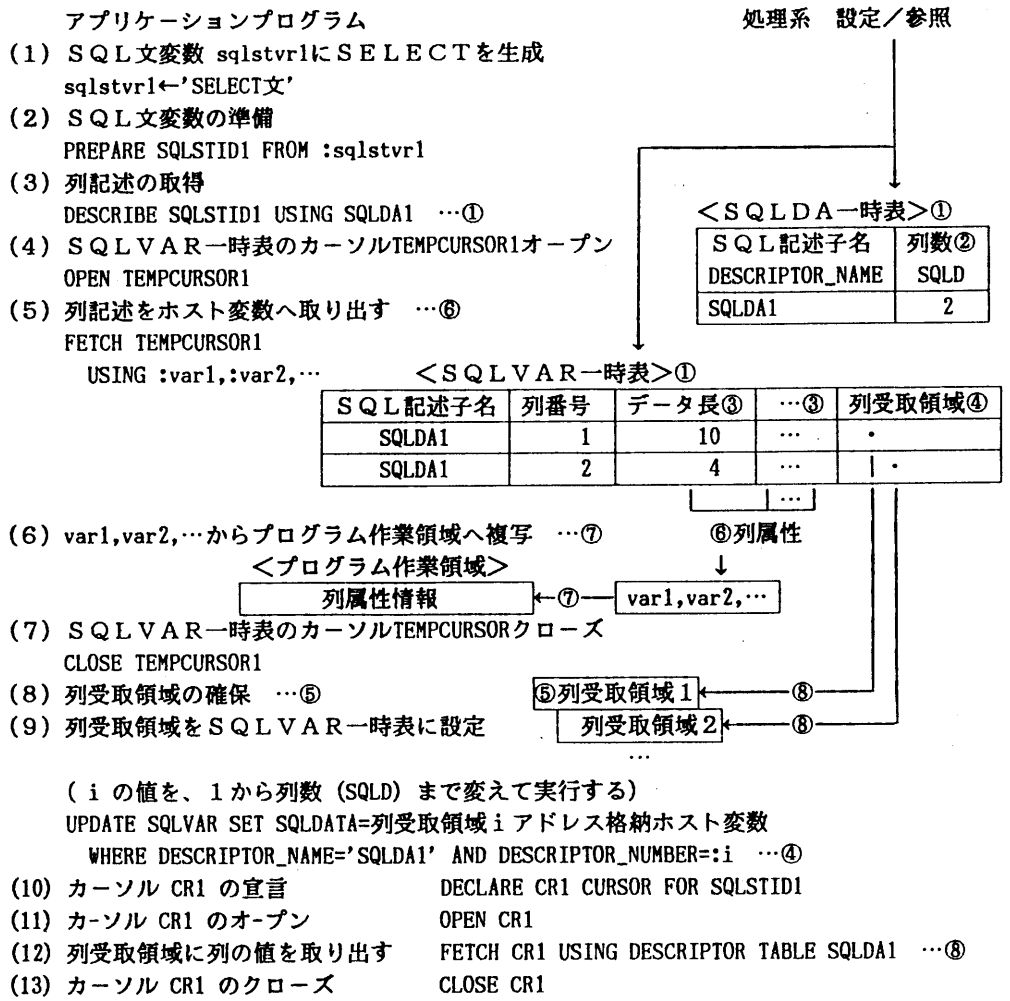

図２　リレーショナルインタフェースの説明図

に困難がでてくるため、ＳＱＬ記述子領域を操作する専用のＳＱＬ文を設けて、それを埋込みＳＱＬ文の形式で関数の代用とする方式になっている。これにより、関数名、データ型の違いを吸収している。

リレーショナルモデルでは、データ操作はリレーショナル演算で指定するのが基本である。ＳＱＬは、リレーショナルモデルのデータ操作言語として、リレーショナル演算に基礎を置いている。しかし、関数インタフェースを導入することで、ＳＱＬで扱うデータの内で、どこまでリレーショナル演算で扱うのかが問題になってくる。これに関して以下の二案があり、現在は後者の案で検討が進められている。

（１）　データベースのデータだけでなく、制御データ（ＳＱＬ記述領域のデータやエラーの詳細情報である診断データ）についても、ＳＱＬで扱うデータは全てリレーショナル演算で扱えるようにテーブル構造にする。

（２）　制御データについては関数インタフェースに統一する。

次に、関数インタフェースの操作の概要を説明する。

ＳＱＬ記述子領域の確保、消去はアプリケーション側で操作できるようにしている。ＳＱＬ記

述子領域の確保は、ＡＬＬＯＣＡＴＥ　ＳＱＬＤＥＳＣＲＩＰＴＯＲ文①で行う。このＳＱＬ文で指定する引数は、ＳＱＬ記述子領域の名称及びＳＱＬ記述子領域の大きさ（ＳＱＬＶＡＲの実現値の最大数）である。ＳＱＬ記述子領域の消去は、ＤＥＡＬＬＯＣＡＴＥ　ＳＱＬＤＥＳＣＲＩＰＴＯＲ文④でＳＱＬ記述子領域の名称を指定して行う。ＳＱＬ記述子領域の有効期間は、ＤＥＡＬＬＯＣＡＴＥ　ＳＱＬＤＥＳＣＲＩＰＴＯＲ文で陽に消去するか、ＳＱＬユーザセションが終了するまでである。

ＳＱＬ記述子領域からアプリケーションのデータ領域に値を読む込むのは、ＧＥＴ　ＳＱＬＤＥＳＣＲＩＰＴＯＲ文②で行う。アプリケーションのデータ領域からＳＱＬ記述子領域に値を設定するのは、ＳＥＴ　ＳＱＬＤＥＳＣＲＩＰＴＯＲ文③で行う。なお、ホスト変数のデータ型とデータベースの中のデータ型の間の型判定及び型変換は実行時に行う。

ＳＱＬ記述子領域が、テーブル情報（ＳＱＬＤＡ）と列固有情報（ＳＱＬＶＡＲ）に分かれ、後者は複数実現値からなっているため、ＧＥＴ　ＳＱＬＤＥＳＣＲＩＰＴＯＲ文とＳＥＴ　ＳＱＬＤＥＳＣＲＩＰＴＯＲ文では、テーブル情報と列固有情報を扱うパラメタに分かれている。列固有情報は実現値番号（ｉ）を指定する必要がある（②',③）。また、各列固有情報を参照、更新する際には、該当情報に対応するキーワードを指定する。関数インタフェースで指定するキーワードは２．２節で説明したＳＱＬ記述子領域のＳＱＬＤ、ＳＱＬＴＹＰＥなどと同じである。図３に関数インタフェースの説明を示す。

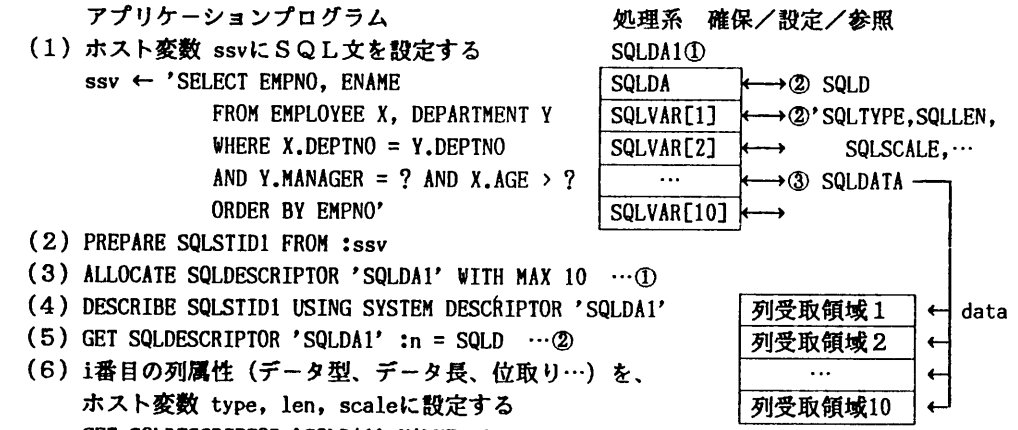

図３　関数インタフェースの説明図

４．　機能拡張

ＳＱＬ２に最初に動的ＳＱＬ機能が追加されてから前述のインタフェースの変更以外に、機能拡張もされてきた。主な拡張機能を次に示す。

（1）　モジュール言語での動的ＳＱＬ文サポート

モジュール言語での動的ＳＱＬ文サポートは、機能拡張というよりは、動的ＳＱＬの埋込み構文でのサポートによって、ＳＱＬで規定しているモジュールの恒久性（persistency）を保証できなくなるという問題点の解決のために行われた。動的ＳＱＬ文の埋込み構文でのサポートは、埋込み構文からそれと等価なモジュールへのマッピングによって規定される。

しかし、動的ＳＱＬ文に指定されるホスト変数の内容をモジュール内のプロシジャにマッピングしたため、プログラム実行時にホスト変数の値が変化するとモジュールの内容を書き換えることになり、モジュールの恒久性を保証できなかった。

正しくは、ホスト変数、及び動的パラメタをプロシジャのパラメタにマッピングしなければならない。このためには動的ＳＱＬ文と類似の機能を持つモジュール内プロシジャが必要である。尚、ＳＱＬＤＡはＳＱＬＳＴＡＴＥと同様に特別なパラメタとして扱う。

（2）　ＤＥＳＣＲＩＢＥ　ＩＮＰＵＴ　文（動的パラメタ記述文）の追加

被準備文中には動的パラメタを含むことができるが、一般に実行時まで、その数やデータ型を知ることはできない。この動的パラメタの記述情報を容易に取得するためのＳＱＬ文としてＤＥＳＣＲＩＢＥ　ＩＮＰＵＴ文が導入された。

（3）　拡張文識別子、及び拡張カーソル名のサポート

拡張文識別子、及び拡張カーソル名のサポートは動的ＳＱＬ文中の文識別子、及びカーソル名にホスト変数の指定を許すものである。これにより、動的に変更されうるホスト変数で動的ＳＱＬ文がパラメタ化されるので、各動的ＳＱＬ文を一つ書けばあとはそれらをパラメタを変えて繰り返し実行すれば済むようになる。

５．　まとめ

動的ＳＱＬのインタフェースの検討過程で、構造体インタフェース、リレーショナルインタフェース、関数インタフェースが提案された。現在は、構造体インタフェースと並行して、関数インタフェースで検討を進めている。関数インタフェースは、ホスト言語の構造体データ型、ポインタ操作の必要性という言語依存の従来からの問題を解消し、操作性の面でも分かり易い、改善された方式である。関数インタフェースの導入に伴い、ＳＱＬ記述子のようなリレーショナル演算機能の必要性のない制御データに関しては、動的ＳＱＬと同様の関数インタフェースで統一する方針で検討し、エラー情報などの診断情報取得のインタフェースにも関数インタフェースを適用した。今後の課題としては、動的ＳＱＬの機能面において、動的カーソルを使用した更新において表名と列名を静的に指定する必要がないようにするなどの動的更新機能の拡張などがある。

６．　参考文献

(1) ISO/IEC Information processing systems-Database Language SQL,
International Standard ISO 9075-1987(E), June 1987

(2) ISO/IEC Database Language SQL with integrity enhancement,
International Standard ISO 9075-1989(E), January 1989

(3) C.J.Date: "A Guide to The SQL Standard" Addison-Wesley, 1987
（芝野，岸本： 邦訳 "標準SQL" トッパン，1988）

(4) ISO/IEC JTC1/SC21: Draft Proposed Database Language SQL2,
Document ISO/IEC JTC1/SC21 N3155, January 1989

(5) ISO-ANSI working draft Database Language SQL2,
Document ISO TC97/SC21 WG3 N382 and ANSI X3H2-87-144, July 1987