リレーショナルデータベース（RDB）と構造化検索言語（SQL）について

八　尋　直　之

Ⅰ．序論

データベースとは，様々な情報を効率良く管理し，運用する為に生まれたデータ処理システムで，次の様に3つに大別される。

1）ツリー型
2）ネットワーク型
3）リレーショナル型

これらの中で，最近，データベースと言えばリレーショナルデータベース（Re－lational Database ；以下RDBと略する）を指す事が多い。それはツリー型，ネットワーク型がコンピュータ専門家の為に作られたシステムであるのに対し，リレーショナル型がユーザーのレベルを問わないと言う視点・観点から作られたシステムだからである。

本論文ではRDBに着目し，RDBの概要に触れると共に，データ操作の方法としてのSQL言語について若干の考察を行う。

Ⅱ．リレーショナルデータベース（ＲＤＢ）について

ＲＤＢは「集合」をもとにしたモデルである。ＲＤＢの考え方では情報の単位を表と言う形式で表現する。つまり，１つのファイルを１つの表とするのである。

表

フィールド

レコード

項目１	項目２	項目３	……

→列方向

↓

行方向

（注）以下の文中・表中に次のものは夫々同意である。

(1) 表…テーブル，ファイル，リレーション

(2) 行…レコード，ロー，タプル

(3) 列…ファールド，カラム，アトリビュート，項目

ＲＤＢとは，この様な表が１つ以上集まって管理されているものを言うが，あくまでも，情報の最少単位は項目であり，１つの表は何時でも最少単位である項目に分解可能であり，また，この最少単位が色々な条件に従って組み合わせ自由である事こそがＲＤＢの最大の特徴なのである。

１つの表から分解された項目を自由に組み合わせて別の表を作り出す事を「表の組み換え」と言うが，即ちこれは，我々がどのような観点からデータベースを眺めるかと言うことである。それぞれの項目は，それぞれが独自の意味領域に属する集合であり，相互独立関係にある。特定の項目が，別の特定の項目と依存関係にあるとか，供存関係にあるということは原則としてあ

り得ない。従って，独立した項目を組み合わせて別の表を構成する事は，或る特定の「ものの見方」を具体化する事なのである。

例えば既存の表を利用するとしても，その内の幾つかの項目を対象とする見方もあるが，その表から必要な項目だけを抜き出して別の新しい表を作ればいいのである。

また行も，項目と同様，それ自体は独立したものであるから，既存の表から或る条件を持った行だけを抜き出して別の表を作り出すこともできる。これらを同時に行って別の表を作り出す事もできる。

1個の表から一部分を抽出する事ができる様に，複数の表の各々から我々が必要とする部分だけを抽出して，それを1つの表にすることができれば，我々の要求はより一層満たされる。ＲＤＢ上では，幾つかの表を一定の規則に従って結合することで，それは実現されるのである。

この様に，1つ以上の表から自由に列と行を取り出し，自由に組み合わせたりする為に「ＳＱＬ」を使用するのである。

Ⅲ．ＳＱＬの概要

「ＳＱＬ」は1970年，E. F. Codd氏が提唱した，関係データベース（ＲＤＢ）の為の言語として開発されたもののひとつである。数多くのモデルが発表された中に，D. D. Chamberlin 氏により開発された「ＳＥＱＵＥＬ」があった。この「ＳＥＱＵＥＬ」が整備・拡張され，ＩＢＭ社のＲＤＢ管理システム「ＳＹＳＴＥＭ－Ｒ」の言語として採用されるようになり，「ＳＱＬ」と呼称されるようになった。ＲＤＢが普及するに伴い，ＲＤＢ言語の標準化が計られるようになり，ＩＳＯ（International Organization for Standard）などでも検討が始められ，1986年にＡＮＳＩ（American National Standard Instituteアメリカ国家規格協会）で承認され，続いて1987年ＩＳＯでも規格が制定された。日本では，ＩＳＯ規格に対応するＪＩＳ規格が同年制定されている。

このようにＳＱＬの標準化は急がれたものの，各社・各所で開発された所

謂方言ＳＱＬが既に数多く存在していたので，この時点では最大公約数的標準化であったに過ぎず，標準ＳＱＬの拡張提案がすぐに提出され，より高機能的な参照・整合性の導入等が計られるようになり，また各国語機能の問題も検討されるようになった。

ＳＱＬの標準化について，次のことが要求される。それは，標準化ということが人為的な基準・制限の設定を意味するのであるから，一部では自由性や創造性を喪失しうる反面，他方では品質の均質化や合理性・効率の向上をもたらすべきであるということである。

コンピュータという技術革新の激しい分野においての標準化では主に次の点に重きが置かれている。

Ⅰ．移植生（Portability）の向上

使用されるＳＱＬが標準に準拠したものであれば，ＳＱＬの実行にハードウェア・ソフトウェアの環境を問わない。

Ⅱ．技術の共有化

ＳＱＬを用いたＲＤＢ関連の蓄積された技術が，例えばエンジニアがそのワーキング・フィールドを変更したとしても，その場所で有効に使用できるようにする。

Ⅲ．異種ＲＤＢ間のインターフェイスとしての存在

異種ＲＤＢ間のコミュニケーション・ツールとしてのＳＱＬが使用されるようになること。

標準化の結果的メリットとして，国際的な規格の制定によりＳＱＬのライフサイクルの長期化が保証され，従ってＳＱＬの普及により一層の拍車がかかるようになったこともあげられる。

ＳＱＬの特徴は次の４点である。

第一に，ＳＱＬがその基本に次の３つの機能を持っていることである。

(1) データ定義機能

(2) データ操作機能

(3) データ制御機能

(1) データ定義機能……表の名前や列名を定義したり，列の属性を定義したりする機能で仮想表（View：ビュー　視点）定義することもできる。仮想表とは実存しない表であるが，既存の表の組み替えの結果生成される表で，ＲＤＢ上には登録されない表のことである。

(2) データ操作機能……表の読み書きを可能にし，実行する機能で，検索（読取り：ＳＥＬＥＣＴ），挿入（追加：ＩＮＳＥＲＴ），更新（ＵＰＤＡＴＥ），削除（ＤＥＬＥＴＥ）の４タイプがある。

(3) データ制御機能……表の操作に種々の制限を付加したり，許可を与えたりする機能で，特定の人や部署にのみ特定の利用許可を与えたり，一般の利用を全面的に許可したりする機能である。これはＲＤＢが誰にでも利用できる様にするとともに，特定の事象に対しては自分を守る，自己防衛的な部分でもある。

第二の特徴はＳＱＬが非手続型言語であるという事である。ＲＤＢが汎用に考えられていることから，表の処理に際して煩雑な指示を必要とするのであれば，そのことだけでＲＤＢの意味は殆ど打ち消されてしまう。標準言語としてのＳＱＬでは非常に簡単な指示で，素早く結果を求め得る様になっている。即ち，ＲＤＢに対して「どの表に対して何をしたいか」という指示だけが必要なのであり，必要に応じて補足の指示（グループ処理や条件付け等）を追加するというのがＳＱＬなのである。行や項目を「どの様な順番（手順）で処理するか」という指示は不要なのである。この様な意味からＳＱＬは非手続型言語と言われるのである。

［ＳＱＬと他の言語との比較］

＜ＳＱＬ＞

どの表の
何という条件のものに
対して何をするか

＜他の言語＞

※プログラムの作成が必要。

1．表を利用可能にする。
2．１行を取り出して条件に合うものか検査する。
3．条件に合うものに対してのみ処理を実施する。
4．結果を表示する。
5．２に戻る。
6．行の終了＝表処理の終了

第三の特徴は集合操作が出来るという事である。１個のＳＱＬ指示のみで複数の行を処理する操作の事を「集合操作（セットオペレーション）」という。表を操作する者から見て，単純に「何をしたいか」だけ指示して，該当する行が１行であれ，数百・数千行であれ，一度に自動的に処理されるということは人間の感覚に近いということである。

第四の特徴は，「メーカー提供のＳＱＬ」という，標準に準拠してはいるものの，ある意味では異物のＳＱＬ（開発メーカーのハード環境やソフト環境に左右される部分を未だに保有しているＳＱＬ，所謂方言ＳＱＬ）が存在するという事である。ＳＱＬには「標準ＳＱＬ」と各メーカーが提供するＳＱＬがあり，各メーカー提供の物の間にもそれぞれ大なり小なり差異がある。各メーカーでは採用しているＯＳにより，ＵＮＩＸ系・ＤＯＳ系の制約に従って各々のオリジナリティをできるだけ前面に打ち出そうとしている。現在小生が使用している「informix-ＳＱＬ」は，メーカー提供のＳＱＬであるが大半は標準ＳＱＬと共通している。

Ⅳ. 標準ＳＱＬの拡張・成長に対する疑問

標準ＳＱＬはＳＱＬ２，ＳＱＬ３と機能の拡張・成長を続けているが，その方向性について疑問に思うことが２点ある。

第一は，当然の事ながら標準ＳＱＬと同様にメーカーのＳＱＬも拡張・成長を続けているが，両者とも果たしてその方向性がユーザーの要求に対して合方向的なものであるかという点である。以下，標準ＳＱＬと informix ＳＱＬとで若干の対比をしてみる。

標準ＳＱＬには「ユーザーの為のユーティリティ」という発想が認められないので，現在開示しているものが総てであり，ユーザーが意図するものはＳＱＬコマンドの中で発現するか，埋め込みＳＱＬとして「ＣＯＢＯＬ」や「ＦＯＲＴＬＡＮ」等の既存のプログラミング言語に依存するようにする他には手立てがない。それはＳＱＬが，いわゆる非手続型言語であり，繰り返し使用することを前提とした手続型言語ではないからである。それに比べメーカー提供のＳＱＬの場合，エンド・ユーザーの負担を如何に軽減するかという事が念頭にあり，ＳＱＬユーティリティの中に独自のプログラミング機能を持たせることで，既存のプログラミング言語に埋め込むような専門的知識を必要とする事なく，ＳＱＬをより身近に使用できるような工夫が見られる。例えば定義済みの表に，数多くのデータを入力する場合を考えると，

１）ＳＱＬで直接行う場合

INSERT INTO ＜表名＞［（項目名）……］……

を必要な回数だけ，条件があればその条件の数だけ変更しながら繰り返し実行する必要がある。単純明快ではあるが非能率的である。

２）ＣＯＢＯＬに埋め込みＳＱＬとして行う場合

データのやりとりの為に，例えば「ＦＯＲＭＳ２」等の別のユーティリティを組み込んで，オペレーターとのインターフェイスの向上を考慮する手間が増え，ＳＱＬの埋め込み作業と合わせて実行のためのプログラムの

作成にかなりの専門的知識を必要とする。

3）ユーザー・ユーティリティ・プログラムによって行う場合

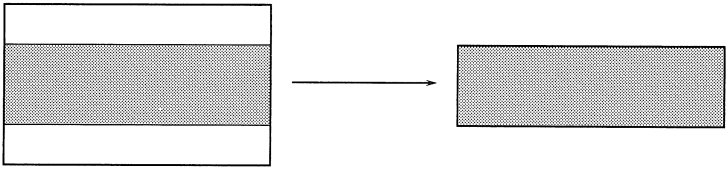
informix-ＳＱＬの場合，このＳＱＬから起動することのできるユーティリティ・プログラムに「ＰＥＲＦＯＲＭ」というスクリーンの上でデータのやりとりを確認しながら実行できるプログラムが準備されている。このプログラムの作成にはＳＱＬの埋め込み作業等は一切なく，且つ作成後のプログラムでは，データの検索・追加・更新・削除が何れも実行可能なのである。ユーザーはプログラム作成の段階でも簡単にＲＤＢに対して意思表示をすることができるのである。

次に、ＲＤＢ上のデータに定義域と呼ばれる制限がある場合を考えてみる。現在の標準ＳＱＬでは「ＣＲＥＡＴＥ　ＤＯＭＡＩＮ」文で，表の定義の際に定義域の設定を効率的に行うことができる。然し，我々の取り扱う各種情報は常に変更と隣り合わせにある。定義域に途中で変更が生じた場合，拡大の方向での変更ならまだしも，縮小の方向の場合，データの保全はどうなるのであろうか。いずれにせよ，一旦入力されたデータは保護されるのであるなら，わざわざ表の定義の段階で「ＣＲＥＡＴＥ　ＤＯＭＡＩＮ」文を使う必要はないし，例えば「ＰＥＲＦＯＲＭ」等のプログラムの中でコーディングすれば十分であると考える。エンド・ユーザーが覚えるコマンドがまた一つ増えただけのことではないか。

次に関係演算について考えてみる。関係演算とは次の6種類である。

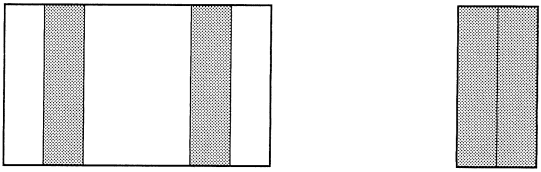
(1) 選択……「ＳＥＬＥＣＴＩＯＮ」または「ＲＥＳＴＲＩＣＴＩＯＮ」

- 表から特定の「行」を取り出す。
- 「ＳＥＬＥＣＴ文」の「ＷＨＥＲＥ節」で特定行を制約する。



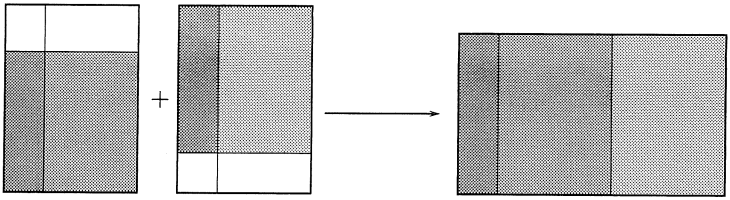
(2) 射影……「ＰＲＯＪＥＣＴＩＯＮ」

- 表から特定の列だけを取り出す。
- 「ＳＥＬＥＣＴ文」に列名を指示する。



(3) 結合……「ＪＯＩＮ」

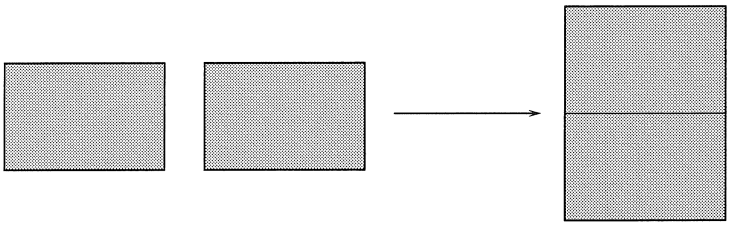
- 特定の列同志照らし合わせて，複数の表を横方向に繋げる。
- 「ＳＥＬＥＣＴ文」の「ＷＨＥＲＥ節」に結合条件を指定する。



(4) 合併……「ＵＮＩＯＮ」

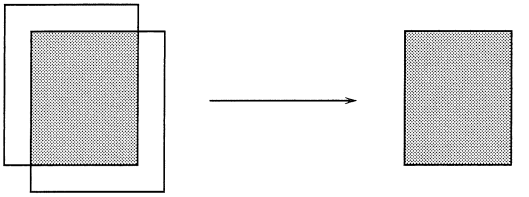
• 複数の表の結果を縦方向に繋げる。

• 「ＳＥＬＥＣＴ文」を「ＵＮＩＯＮ」で結ぶ。



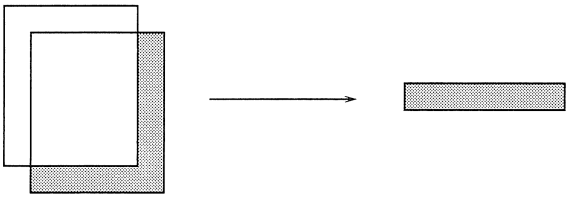
(5) 交差（または共通）……「ＩＮＴＥＲＳＥＣＴＩＯＮ」

• 複数の表の間で，同じ内容を持つものだけを取り出す。



(6) 差……「ＤＩＦＦＥＲＥＮＣＥ」

• 複数の表の間で，最初の表（主表という）に属し，他方（従表という）には属さないものを取り出す。



これら６種類の関係演算の内，３と５については従来のＳＱＬ（標準，メーカー共に）では，空白値（ＮＵＬＬ値）を演算の対象から除外していたので，関係演算を実行する際に制約が多く，かなり不便であった。今般，標準ＳＱ

Ｌの拡張の方向として空白値をも演算の対象内に含むということが打ち出され，この点ではかなりの進展が見られる。

では，最近のＳＱＬ２，ＳＱＬ３に見られる拡張が総て有意義なものであるかと言えば，一部に無駄とでも言うか，自己満足的な技術的発展に重きが置かれ，ユーザーのニーズをしきっていないと思われる部分もある。

(1) 「結合」の場合の拡張の例

★ＩＮＮＥＲ　ＪＯＩＮ：同じ値のものだけを取り出す。

SELECT ＊ FROM （KOKYAKU NATURAL INNER JOIN JUCHU）

又は

SELECT ＊ FROM （KOKYAKU INNER JOIN JUCHU ON KOKYAKU.顧客番号＝JUCHU.顧客番号）

★ＬＥＦＴ　ＪＯＩＮ：左に指定する表は総て取り出し，右の表に該当のものがない場合は空白値の行を作る。

SELECT ＊ FROM （NATURAL LEFT JOIN JUCHU）

★ＲＩＧＨＴ　ＪＯＩＮ：右に指定する表は総て取り出し，左の表に該当のものがない場合は空白値の行を作る。

SELECT ＊ FROM （NATURAL RIGHT JOIN JUCHU）

★ＦＵＬＬ　ＪＯＩＮ：左右の表の総ての行を取り出し，相互に該当するものがない場合は空白値の行を作る。

表明:KOKYAKU

顧客番号	顧客名
F010	山田商会
K101	伊藤販売
S025	㈲ 鈴 木
M007	田中興業
T556	川村商店
A243	中山商事

表明:JUCHU

顧客番号	伝票番号	受注日付
F010	00001	17 / 09 / 1991
M007	00001	13 / 10 / 1991
S025	00001	04 / 09 / 1991
F010	00002	08 / 11 / 1991
V623	00001	25 / 11 / 1991

INNER JOIN

顧客番号	顧客名	伝票番号	受注日付
F010	山田商会	00001	17 / 09 / 1991
F010	山田商会	00002	08 / 11 / 1991
M007	田中興業	00001	13 / 10 / 1991
S025	㈲ 鈴 木	00001	04 / 09 / 1991

LEFT JOIN

顧客番号	顧客名	伝票番号	受注日付
A243	中山商事	……	……
F010	山田商会	00001	17 / 09 / 1991
F010	山田商会	00002	08 / 11 / 1991
K101	伊藤販売	……	……
M007	田中興業	00001	13 / 10 / 1991
S025	㈲ 鈴 木	00001	04 / 09 / 1991
T556	川村商店	……	……

RIGHT JOIN

顧客番号	顧客名	伝票番号	受注日付
F010	山田商会	00001	17 / 09 / 1991
F010	山田商会	00002	08 / 11 / 1991
M007	田中興業	00001	13 / 10 / 1991
S025	㈲ 鈴 木	00001	04 / 09 / 1991
V623	……	00001	25 / 11 / 1991

FULL JOIN

顧客番号	顧客名	伝票番号	受注日付
A243	中山商事	……	……
F010	山田商会	00001	17 / 09 / 1991
F010	山田商会	00002	08 / 11 / 1991
K101	伊藤販売	……	……
M007	田中興業	00001	13 / 10 / 1991
S025	㈲ 鈴 木	00001	04 / 09 / 1991
T556	川村商店	……	……
V623	……	00001	25 / 11 / 1991

上記の４つのものについては確かに機動性が増して好ましいものである。メーカーサイドも早期にこの機能の提供を開始して欲しいものである。但し，informix-ＳＱＬでは別のスタイルで「ＦＵＬＬ　ＪＯＩＮ」以外を提供している。

例：ＩＮＮＥＲ　ＪＯＩＮ

```
SELECT　＊　FROM KOKYAKU,JUCHU
        WHERE KOKYAKU.顧客番号　＝　JUCHU.顧客番号
```

又は

```
SELECT　＊　FROM KOKYAKU,JUCHU
        WHERE KOKYAKU.顧客番号　IN
                        (SELECT　顧客番号　FROM　JUCHU)
```

{因に「差」の場合は“NOT IN”,“ ◇ ”で十分}

例：ＬＥＦＴ　ＪＯＩＮ

```
SELECT KOKYAKU.顧客番号,KOKYAKU.顧客名,JUCHU.伝票番
号,JUCHU.受注日付
        FROM KOKYAKU.OUTER JUCHU
        WHERE KOKYAKU.顧客番号　＝　JUCHU.顧客番号
```

{ＲＩＧＨＴ　ＪＯＩＮは表の左右を逆転すれば十分}

従って全く新しいものと言えば，informix-ＳＱＬに存在しないのは「ＦＵＬＬ　ＪＯＩＮ」ということになる。コーディングの都合から言えば他の３者は有意義な拡張であると言える。

全く無駄に開発したと思われるのが「合併：ＯＵＴＥＲ　ＵＮＩＯＮ」である。

ＯＵＴＥＲ　ＵＮＩＯＮの例

顧客番号	顧客名	顧客番号	伝票番号	受注日付
F010	山田商会	……	……	……
K101	伊藤販売	……	……	……
S025	㈲ 鈴 木	……	……	……
M007	田中興業	……	……	……
T556	川村商店	……	……	……
A243	中山商事	……	……	……
……	……	F010	00001	17 / 09 / 1991
……	……	M007	00001	13 / 10 / 1991
……	……	S025	00001	04 / 09 / 1991
……	……	F010	00002	08 / 11 / 1991
……	……	V623	00001	25 / 11 / 1991

レコード数が少ないのであれば，これでも何かの足しにはなるかもしれないが，ＲＤＢ上のデータであるという事実上，左右の表に「無関係」以外の関係が無いのは唯の羅列に過ぎず，ＲＤＢの特性を生かしているとは言えない。テクニックに走りすぎた結果ではないかと思う。

Ⅴ．メーカー・ユーティリティ・プログラムの真価

では何故，ＳＱＬ２，ＳＱＬ３という進歩型のＳＱＬの拡張・成長にこのような疑問をもつか，述べてみたい。

⑴　ＳＱＬは非手続型言語で１回限りの利用に際しては非常に有能な言語であるが，我々の日常のコンピュータ操作は繰り返し作業が多く，そのために埋め込みＳＱＬを使用したＣＯＢＯＬ等の手続型言語プログラムを開発するとなると，ＲＤＢとＳＱＬがユーザーのレベルを問わないという大前提と矛盾してくる。

⑵　我々ユーザーサイドは，常にＣＲＴ上でＳＱＬによる実行結果を得るだけでは要求は満たされない。合目的的なレポートとして出力することを要

求されるが，ＳＱＬの延長上に，既存のものとは全く別のＳＱＬの特性を生かしたプログラミング言語があってしかるべきである。

(1)について

先にも少し触れたが，データの入力について考えてみる。ＲＤＢ上の表にデータを入力する場合，常に全項目に入力するとは限らないし，入力できるとも限らない。１回限りのコマンドを利用して直接入力するなどと言うことは到底考えられない。

では繰り返し作業のために，埋め込み型ＳＱＬを使用した手続型言語，例えば高級言語と呼ばれるＣＯＢＯＬやＦＯＲＴＬＡＮ，Ｃ等を使用するとなると一般ユーザーにはすぐには実現不可能な話で，どうしても専門のプログラマーが必要となる。設計変更の度に或いは新規の必要性の度にとなると，管理・運用の面ばかりでなく，費用の面でも問題がある。少量の変更でも専門家の手に委ねるとなるとコンピュータを利用して小回りの効く運営など，画餅と同じである。

もしＲＤＢに直結したユーティリティ・プログラムがあれば話は違ってくる。そもそもユーティリティ・プログラムというものは，エンドユーザーの為に開発されるものであるが，各社それぞれに工夫を凝らしている。「容易な利用」について，合目的的な機能が集約されており，現在小生が利用している「ＰＥＲＦＯＲＭ」はユーザー提供の informix-ＳＱＬから起動することができ，またコマンドで直接起動することもできるが，ＣＯＢＯＬ等よりも遥かに使いやすい。ＲＤＢに直接，書き込み・検索・更新・削除が実行できるし，プログラムの生成が非常に楽だからである。このようなユーティリティ・プログラムがあってはじめてのＲＤＢではないだろうか。ＳＱＬの特性を生かした，より簡便な入力方法が確立されれば，少しの訓練で大半のユーザーは自分でデータの管理をするプログラムを組みことが可能になり，量の多寡にかかわらず，データの保全・管理が容易に実行することができるようになるのである。とすれば，現時点におけるＳＱＬの拡張・成長は一体何を

目指しているのであろうか。恐らく，高級言語（ＣＯＢＯＬ，ＦＯＲＴＬＡＮ等）の中に埋め込んだ際のプログラミング手続きの利便性の向上が目的にあるのではないだろうか。理解しやすく使いやすいはずのＲＤＢとＳＱＬに矛盾が生じてくる様に思われてくる。

(2)について

レポート出力は繰り返し作業が基本であるから，そのプログラムはどうしても手続型言語になる。現在，各種の手続言語において構造化（Structurizing）が盛んに進められているが，本来構造化言語であるＳＱＬを報告書作成用言語へ発展させて，もっと有為に使用する方法はないものであろうか。

ＣＯＢＯＬの場合を例にすると，今やＬ－Ⅱ　ＣＯＢＯＬよりも，ＣＯＢＯＬ－85の時代になりつつある。ＣＯＢＯＬ－85は完全な構造化指向に変身している。フローチャートに忠実に手続を進められるようになっているが，依然として残されている問題がある。

それはグループ処理，ソフト・マージの問題である。ＣＯＢＯＬでプログラムを組む場合は，ソート・プログラムにその都度お世話になるしかない。WORKING-STORAGE の設定や仮想の状態の具現化など，手続が煩雑になればなるほど厄介な存在になる。ところが，ＳＱＬ埋め込み型ＣＯＢＯＬになると事情は急変する（残念ながら，ＣＯＢＯＬ－85にＳＱＬを埋め込んだプログラムを実際に拝見したことがないので，話はＬ－Ⅱ　ＣＯＢＯＬに限定される)。埋め込まれたＳＱＬの“ORDER BY”や“GROUP BY”，“SELECT 節”の表現がＲＤＢにアクセスする段階でこれらの問題の殆どを解消するからである。残る問題は唯一，こういう高度なプログラムはエンド・ユーザーには殆ど作れないということである。

メーカー・サイドは，レポート出力の面でもユーティリティ・プログラムを準備している。しかしどういう訳か，その評価は異常に低いのが現状である。メーカー・サイドからみても自社のユーティリティ・プログラムが広く用いられる様になることは喜ばしい事であるに違いない。しかし，それに伴っ

て，例えばＳＥや専門のプログラマーの出番が減少していって，彼らソフト業者の減収に繋がるとすれば，どの程度のことができるのか余り詳細には触れたくないところであろうが，ソフトも含めて，売り込みの際にユーザーに身近なコンピュータと意識させる為には，抵触せざるをえない部分でもある。もとよりメーカー・サイドはパッケージ・ソフトの販売に熱心であり，増してやソフト屋のメンツは高級言語を駆使してこそ保たれるものであるとする姿勢があることも，ユーティリティ・プログラムの積極的普及にブレーキをかけていると言わざるを得ない。この様な状況から，折角のユーティリティ・プログラムも，メーカー・サイドの手で全体を現わす事なく，或いはその能力の極めて一部だけを表に出していることが多い。

現在，小生はユーティリティ・プログラムの大の愛用者である。小生の利用しているものは「ＡＣＥ」というもので，プログラムの作成から実行までを informix-ＳＱＬから起動でき，また単独で起動・実行できるが，手続のスタイルから言えば，ＳＱＬそのものであると言っても過言ではないと思う。ＲＤＢからのデータの取り出しはＳＱＬ以外の手続は不要であるし，グループ処理やソート・マージの問題は「ＳＥＬＥＣＴ文」の中で考えればよい。通常のリスト出力であれば，他に準備されている「ＦＯＲＭＧ」という簡易帳票出力ユーティリティと組み合わせれば，見た目にも立派なものが出力できる（「ＦＯＲＭＧ」との組合せも実際は非常に簡単である）。予め特別な訓練をせずとも導入後の訓練だけで十分に対処できるし，少しでも他の言語の知識があれば，ユーティリティ・プログラムとはいえども，プログラミングの奥行は深まっていくばかりである。なぜなら，この「ＡＣＥ」は全くの構造化プログラミング言語だからである。極端に言えば，ＣＯＢＯＬ－85からデータの取り込み処理，出力のための転記，グループ処理，ソート・マージを除去した，最小単位の繰り返し処理についてのみ手続をコーディングするだけなのである。但，実際に何にでもできるのかと言えばそうではない。コーディングに際して，文字数や処理ステップ数，同時にオープンできるテーブルの数，宣言可能な変数の数など，若干の制限があるからである。あえて

現時点で「ＡＣＥ」に拡張・成長の注文を出すとすれば，次の２点に絞られる。

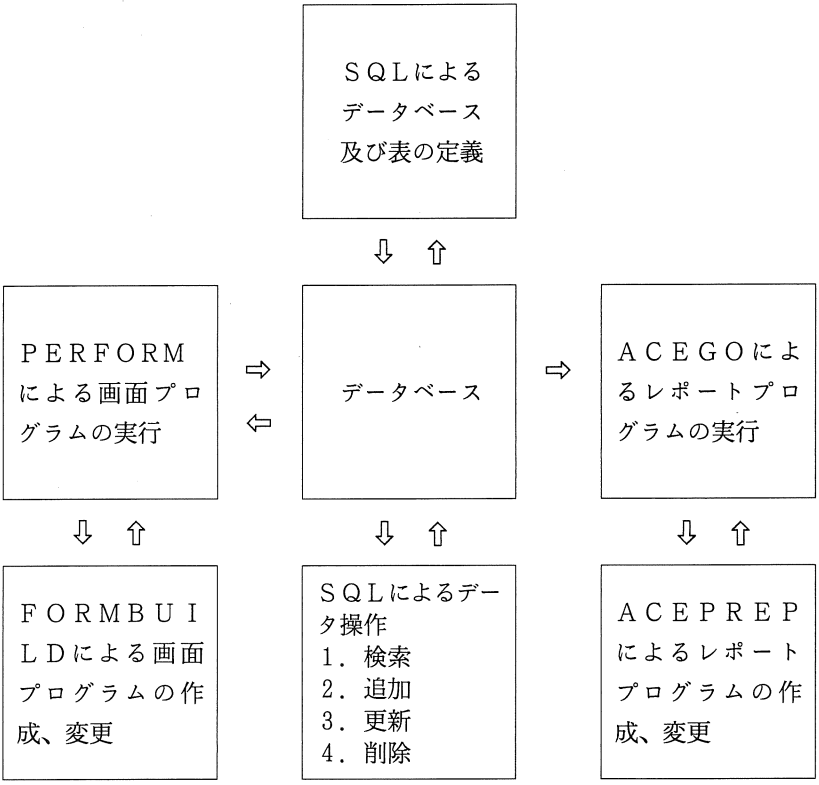
(1)　行単位での文字編集が可能になること。

(2)　「ＯＣＣＵＲ」句や，数字タイプのデータがそのまま添字として使用できるようになること。

この２点が実現される様になれば，この「ＡＣＥ」というユーティリティ・プログラミング言語はＲＤＢと共にあるかぎり，「ユーティリティ」という冠を外すことができるようになると期待するものである。

参考までに informix-ＳＱＬのユーティリティ関連図を紹介しておく。

informix-ＳＱＬ　ユーティリティ関連図



VI. 結論

ＲＤＢが本来目指したものは理解しやすいデータベースであり，使いやすくするために開発されたものがＳＱＬであると小生は考えている。ＳＱＬのコマンドは１回限りの使用を考えて作られている非手続型言語であるが，なぜ，繰り返し使用するコマンドを作る必要がないのであろうか。それはＳＱＬがほかのＣＯＢＯＬやＦＯＲＴＬＡＮ，Ｃ等の高級手続型言語に埋め込まれて，ＲＤＢを操作するために使われている言語として作られたからであって，ＳＱＬそのものが繰り返しを意味するコマンドを持たずともＳＱＬを包含した手続型言語プログラムが現実に「繰り返し実行する」からである。従ってＳＱＬが単独でユーザーの為に働くのは，ＣＲＴ上でのデータのやりとりか，単純な作業で十分なのである。

しかし，ＲＤＢやＳＱＬの「理解しやすく，簡便である」という本来の姿勢を生かすという立場に立った時，ＣＯＢＯＬ等に埋め込まれたＳＱＬや限られた単純作業しか実行できないＳＱＬでは，ＲＤＢもＳＱＬもその特性を十分に発揮しているとは言い難い。ＲＤＢをもっと有意義に使用するために，ＲＤＢのすぐ隣にあって，ＳＱＬの特性を十分に生かした，データの入出力が簡単にできるようなプログラミング手段が，メーカー提供のユーティリティ・プログラムとしてではなく，標準のＳＱＬに派生したものとして存在するべきであると考える。

また，各メーカーも現状を打破し，より多くのユーザーに自作可能なキャパシティを持ったプログラミング言語の開発・研究，そして積極的な開示を行って欲しい。

（参考文献）

『ＳＱＬ入門』平尾隆行・著（オーム社）1990

『標準ＳＱＬ』Ｃ．Ｊ．デイト・著／芝野耕司・監訳／岸本令子・訳（トッパン） 1990

『ＳＱＬ言語活用入門』河村一樹・著（日刊工業新聞社） 1990

『標準ＳＱＬプログラミング』原潔・著（啓学出版）1990