データベース・システム 37－2
（１９８３．９．２９）

次世代のデータベース・システム・アーキテクチャ
－ＴＩＳ－

坂本博雄
（シンコム・システムズ・ジャパン株式会社）

1. はじめに

データベース管理システムは、ソフトウェアの発展に大きく貢献してきた。1960年代に設計され、1970年代には広範囲にわたって使用開始された。これは、DBMSが次のような利益をもたらしたからである。

(1) データ重複の減少。

(2) データ間の関連付けの維持。

(3) アプリケーションのデータ・フィールドからの独立。

(4) ファイル変更による影響の減少。

(5) DBMSツールの部分的統合。

これらの利点は、ソフトウェアの生産性面でも多大な恩恵をもたらした。また、即時性を求めて、DBMSは、対話処理のためのオンライン・モニターと組んで、システムの中核を占めるようにもなってきた。

一方、コンピュータの技術応用の先端をいく米国の大手企業では、1970年代半ばからシステムの構築方法を再検討しようという風潮が出はじめた。それは、第一世代のDBMSが設計された1960年から1970年代半ばにかけて、システムの構築環境が大きく変わってきたからである。DBMSが設計された当時、大型機のCPU 能力でさえ、現在のマイコン程度である、ディスク容量にいたっては、今のフロッピー・ディスクと変わらなかったのである。また、ソフトウェア・コストは、比べものにならないくらい少額であった。アプリケーションは、量も少なく単純で、大多数のプログラマは開発に専念し、コンピュータを使用する人も限られていた。

時代が変わるにつれ、企業では、初代のDBMS技術では解決できない問題点があることに気づきはじめた。すなわち、

(1) 多数のファイルがデータベース化されず、そのまま残されると、DBMSはこれらのファイルを処理できない。

(2) アプリケーションをデータ・フィールドから部分的に独立させることには成功したが、以前データ構造に依存しているため、データ構造を変えるとアプリケーションの修正と再コンパイルが必要になる。

(3) DBMS中心のシステムでは、複数のDBMSの統合やDBMS以外のツールとの統合は難しい。

(4) 分散データベース構築のため、DBMS間のデータの共用、排他制御、機密保護、回復機能の設計が難しい。

(5) 大型データベース、とくに多くの更新を伴う処理を行うと、レスポンスが悪い。

(6) システムの運用・維持・管理ツールや、実行

時のユーザ支援機能が欠けている。

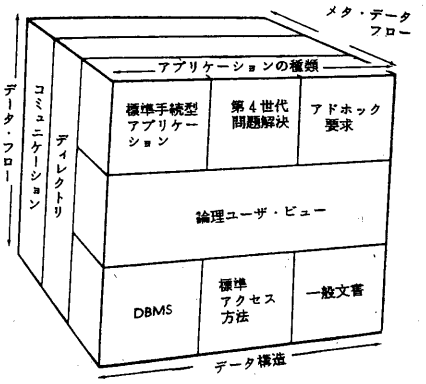
(7) リレーショナルDBMSへの移行が容易でない。

多くの大企業ユーザを持つシンコムは、これらユーザの意見から、従来のDB／DCを中核に据えたシステム・アーキテクチャでは、開発・運用・維持の面で生産性を向上させることに限界があることに気づいた。そして、これらの問題を解決し、長期にわたって生産性を向上させる新しいアーキテクチャの開発が必要不可欠であると考えた。プロダクション用のシステムだけでなく、使いやすさ、効率、柔軟性、統合性、セキュリティ、ユーザの管理等すべての条件を満たし、情報処理部門にとってはまさに画期的と言われる情報センターの構築を可能にする強力なアーキテクチャを備えたＴＩＳを逸早く開発したのである。

2. 機能概要

ＴＩＳは、現在９つのコンポーネントをもつ図—１のようなアーキテクチャとなっている。システムの中核は、開発・運用・維持において生産性向上を図るため、インライン・ディレクトリおよび論理ユーザ・ビューと呼ばれる新型の構成要素から成っている。これにより、DBMSをデータ管理という本来の目的に専念させて、DBMS中心のシステムのひずみ、または生産性の限界を克服した。

ＴＩＳの構成要素一つ一つは、ＴＩＳ　DBMSと深い関係を持っているが、ここではスペースの関係上、４つの構成要素の説明にとどめておく。



図—１　ＴＩＳのアーキテクチャ

(1) 論理ユーザ・ビュー

論理ユーザ・ビューは、リレーショナル・ビューとも呼ばれている。システムがリレーショナル・ビューを採用することによって、データのＩ／Ｏ要求手続きを簡素化し、アプリケーションをデータ・フィールド、データ構造、ファイル・アクセス手法、データベース管理システムから独立させ、これらアプリケーションを囲む環境の変更による影響からプログラムを保護する。

正規化されていないファイルは、データ構造が隠された意味をもっているため、新たなアプリケーションが追加された時、その新しい要求に合わせた構造に変更することは簡単にできない。新しい要求にデータ構造を無理やり合わせようとすれば、既存のプログラムの中に隠された意味を維持し、解釈するための特別なプログラム論理を追加するという修正作業が発生することになる。

正規化された二次元のファイルは、このような

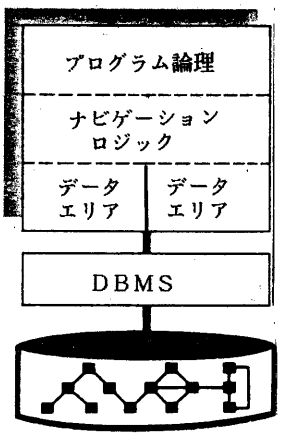
隠された意味を持たないので、より安定性があり、より簡単に、複数アプリケーションで長期にわたるデータの共有が可能となる。また、データベース・システムがデータ構造を要求された形式に再結合することも容易になる。

アプリケーションのリレーショナル化

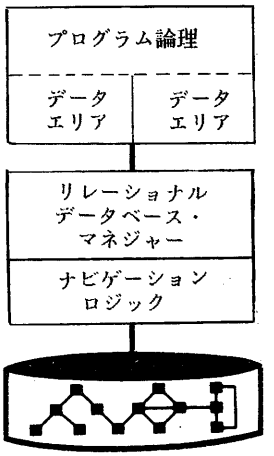
しかし、ハードウエアが正規化されたデータを元の形式のままで記憶でき、要求に応じてダイナミックに論理レコードを再構成できる十分な力を備えるようになるまで、本当のリレーショナル処理が実現されることは難しい。大量の処理に耐えうる効率の良いアクセスは、ポインタや外部テーブルでなくデータ値を基礎にしていなければならないからである。これは、「データ値に基づいた」記憶システム、あるいは「データ内容でアクセスできる」記憶システムが必要なことを意味している。

そのようなハードウエア技術が安価に使用可能になるまでは、プログラマやエンド・ユーザにとっては、使い易さ、維持の容易さ、新しい技術の導入のしやすさなどの利点を持つ、データのリレーショナル・ビューが既存ファイルに対し与えられていることが望ましい。

データベース・ファイルが、データ構造を持っている場合、アプリケーションは、必要なデータを取り出すために、データ構造にそってファイルを追いかけなければならない。そのため、アプリケーションの処理ロジックの中にDBMSを操作するナビゲーション・ロジックが必要となる（図―2）。また、アプリケーションをデータ構造から隔離させるためには、このナビゲーション・ロジックをプログラムから分離することが必要となる（図―3）。

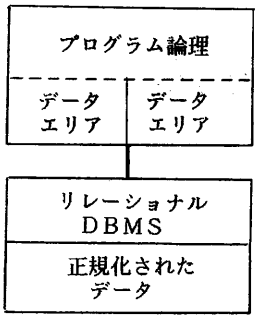


図―2　現在のDBMS



図―3　リレーショナルDBMSへの第一ステップ

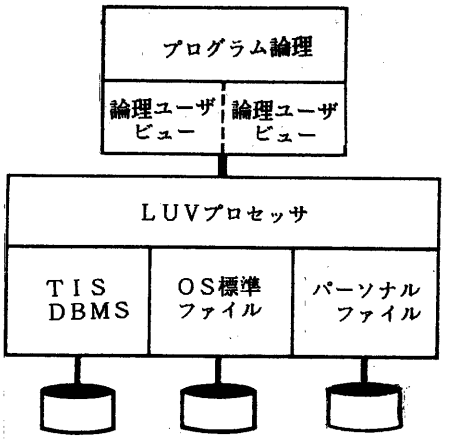
一度、データ構造に依存するナビゲーション・ロジックがプログラムから分離されると、データの入出力要求は、基本的な「追加、削除、更新、読出し」の４つの命令に簡素化される。これによって、DBMSタイプからの独立が可能になり、図―３から図―４のように純粋なリレーショナル技術の出現に伴い、アプリケーションの手直しを強制されずに、リレーショナルDBMSを導入する道が開かれる。



図―４　純粋なリレーショナルDBMSを採用

実用化リレーショナル処理

従来のDBMSのほとんどは、データベース化されたファイルしか処理できず、従って、DBMS中心に統合されたDBMS用ツールもDBファイルにしか適用できなかった。エンド・ユーザにはVSAMのような標準アクセス方法のファイルが、DBファイルとは別に定義されてサポートされる。しかし、これでは、データ記述、保全、保護について、DBMSが解消しようとしていたデータの重複という問題が再び発生することになる。



図―５　LUV プロセッサを用いたＤＢＭＳ

この問題を上手に解決していくには、現行のシステムを拡張したり、他のシステムと結合することが必要である。それには、ユーザは、構造化データベース・システムと標準アクセス・ファイルに対してデータのリレーショナル・ビューを通して、共通のインタフェースを持つことになる。ＴＩＳの論理ユーザ・ビューは、システムの中でこの役割を果たし、データ構造から完全な隔離を図る（図―５）。

論理ビューの定義は、データベース管理者（DBA）が行う。DBA は、第４世代言語を用いることによって、ビューの中で使用される物理ファイルごとにひとつのACCESSステートメントだけで、非常に複雑なデータベースでも、最適なナビゲーション・アルゴリズムを選択することができる。また、DBA は、特定のオペレーションに対するビューの機能を制限し、例えば、更新は許可するが、

追加や削除には使用を禁止することもできる（図—6）。

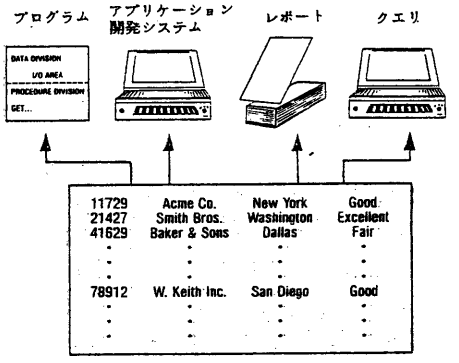
```
LOGICAL __VIEW=EXPENSE
KEY DEPT__NUMBER
KEY ACCT__NUMBER
PROJECT __TITLE
DETAIL__AMOUNT
ACCESS DEPT USING DEPT__NUMBER
ACCESS EXPN VIA DEPTLKEY ALLOW ALL
ACCESS PROJ USING PROJ__NUMBER
```

図—6　ＴＩＳ論理ユーザ・ビュー

論理ユーザ・ビューを用いることによって、エンド・ユーザとプログラマは、二次元テーブルあるいは、「ターブル」形式で、ひとつの同一データ・ビューを共有する（図—7）。データベース管理者（DBA ）は、ビューを一度定義するだけとなり、必要なプログラミングの量を減らすことができる。これは、エンド・ユーザやプログラマが、データがどこにどのような形式で記憶されているかを知る必要がないことを意味している。また、ファイル・ナビゲーションについては、ファイル設計者が指示した効果的な方法を標準化して用いることができる。

(2)　インライン・ディレクトリ

新しいアーキテクチャのための、もう一つの中核コンポーネントが、ディレクトリである。前項で述べた通り、論理ユーザ・ビューは、アプリケーションの論理データ入出力をサポートする。これに対し、ディレクトリは、物理ビューを管理しながら、論理ユーザ・ビューを用いた高レベルの論理データ入出力要求を、アーキテクチャを構成しているそれぞれのコンポーネントで処理できるよう、低レベル論理要求（例えば、データベース命令）へ翻訳する働きをする。すなわち、論理ユーザ・ビューと一体になって、各コンポーネントの統合、アプリケーションの独立、システムの制御を支援する。



図—7　共用ビューがアプリケーションで使われる

インライン・ディレクトリとは、システムの中核となるディレクトリが、データ管理コンポーネントとアプリケーションとの間の直列の関係（インライン）に位置していることからこう呼ばれている。ディレクトリが、インラインになったことによって、従来のディクショナリ技術では不可能だったアプリケーション実行時におけるシステムの運用、制御、管理が可能になった。

インライン・ディレクトリとデータ・ディクショナリは、稼働環境、機能、利点等に大きな違い

があるが、ここでは、その特徴や利点の違いについて、まず説明してみよう。

ディクショナリの限界を克服する

インライン・ディレクトリ

元来、情報処理部門では、ディクショナリがあればシステムの統合・管理が可能になると考えていた。すなわち、中央照会ポイントとしてディクショナリを使用すれば、情報処理部門はもっと迅速に変更に対応でき、より優れたレポートを提供できると考えた。

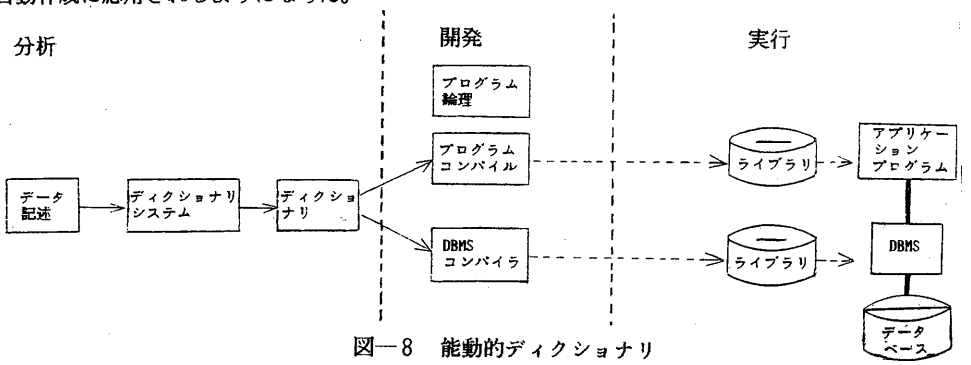
しかし、初期のディクショナリは、受動的なもので、アプリケーション開発の分析段階でしか機能せず、次の開発段階でコーディングされるプログラムとは直接関連していないため、システムの維持という観点からは貧弱なものにすぎなかった。

1970年代半ば、この受動的ディクショナリを一歩進め、コーディングの段階でも活発に働く「能動的ディクショナリ」が出現した（図―8）。この新しい能動的ディクショナリによって、分析段階で収集されたデータは、プログラムの作業変数領域とデータベース生成に必要なファイル記述の自動作成に応用されるようになった。

また、このプログラムやファイル記述に関する情報については、定義された情報を再びコーディングする必要がなく、直接提供できた。しかし、ディクショナリとプログラムとの間に矛盾を起こさせないためには、いずれかのアプリケーション・プログラムが変更される前にディクショナリを更新しておかなければならないシステムもあった。

能動的ディクショナリは、コンパイル段階では役立ったが、アプリケーション開発の実行段階では機能しなかった。すなわち、「インライン」ではなかったのである。そのため、プログラマは、データの定義に変更が加えられるたびに、プログラム全体の保守を行い、再コンパイルしなければならなかった。また、能動的ディクショナリでは、データベース・ナビゲーション（航行）をすべてアプリケーション・プログラムで操作する必要があった。

これは、プログラマがDBMSについての知識を持っていなければならなかっただけでなく、データ構造に変更が生じた場合、アプリケーションにどう影響するかを知っていなければならないことを意味した。インライン・ディレクトリは、これら



図―8　能動的ディクショナリ

データ・ディクショナリの構造上の問題点を考慮して、新たな機能として設計された（表―1）。

アプリケーションの分析から実行まで

インライン・ディレクトリ情報は、分析とプログラミングの段階でプログラマのドキュメンテーションに使用される。また、プリ・コンパイル段階を経て、プログラムに作業変数領域を生成し、使用情報を把握して更新を行う。その結果、ディレクトリは、どのプログラムがどのデータを使用しているかを常に正確に把握し、レポートすることができる。

このアーキテクチャの下で稼働するDBMSは、実行段階で物理制御ブロックとよばれる物理ファイルの情報を得るため、インライン・ディレクトリにアクセスする（図―9）。

論理ユーザ・ビューは、インライン・ディレクトリの論理制御ブロックを使って、使用データの外部ビューを記述する。ただし、このような情報は、実行時の最終バインディングとよばれる段階で、はじめて「物理情報」と結合される。

以上のようにデータ構造、作業変数領域、アクセシング・ナビゲーションなどさまざまな面が自動的に提供されるが、通常のプログラムからは分離されている。

新しいアーキテクチャのディレクトリは、アプリケーションのライフサイクル、すなわち分析から開発、実行に致るまでを一貫してインラインに制御する。また、情報処理部門の運用においても、インラインの特徴が有効に働くので、データ・ディクショナリの欠点も克服されている。

インライン・ディレクトリは
実用的な運用ツール

インライン・ディレクトリは、ディレクトリの全データを提供することができるので、ディクショナリの機能を果たすこともできる。レポートには、エンティティ一つ一つのディレクトリ内容と、ディレクトリの中にあるエンティティ間の関係が示される。

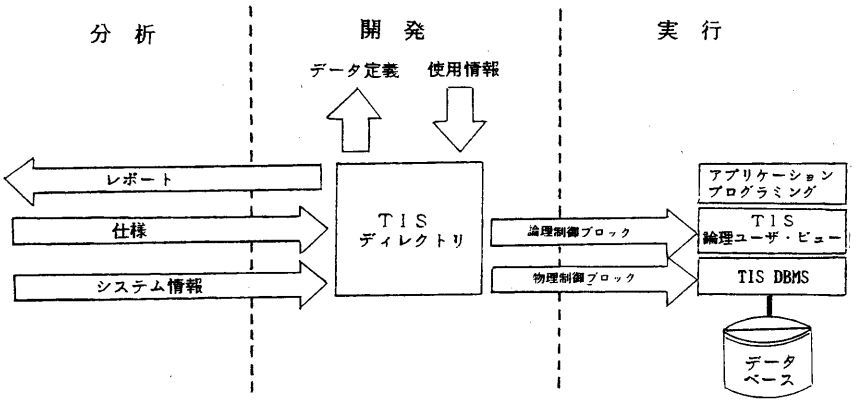
また、物理ファイル、レコード、フィールド、オペレーティング環境および物理データと論理データの関係やデータとユーザの関係についても示される。

データベース管理者（DBA）は、対話形式でシステム記述の作成や保守ができ、ディレクトリ内容をいつでも表示することができる。また、インライン制御方式では、インライン・ディレクトリに行われた変更を本番稼働時に直接伝えることができ、正確な最新実行環境を常に反映できるため、独立ディクショナリ方式に比べ、システムの維持が非常に簡単になる。

データ記述のバージョンをひとつしか持たないため、データベース管理者、アナリスト、プログラマの労力は、最小限に押さえられる。その結果、システム全体にわたっての正確さ、安全性、品質管理の容易さが著しく向上する。

特　　徴	受動的ディクショナリ	能動的ディクショナリ	TIS インライン・ディレクトリ
分析段階での制御	有	有	有
開発段階での制御	無	有	有
実行段階での制御	無	無	有
DBA のレポートが実行次のシステムに矛盾しないことを保証	無	一部有	有
プログラムの作業領域作成の手間を省く	無	有	有
データベース記述を作成する手間を省く	無	有	有
データベース記述に必要な特別なコンパイル段階をなくす（リアル・タイム処理）	無	無	有

表—1　ディクショナリとディレクトリの比較



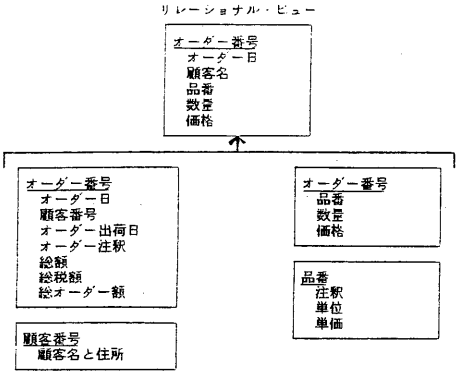
図—9　インライン・ディレクトリの構造

(3) データベース管理システム（DBMS）

ＴＩＳ　DBMSは、正規化されたデータ構造、ハイブリッド・リレーショナルなデータの関連付けおよび仮想アクセス・メソッド処理（VAMP）の採用によって、スループット効率を最適化する。ＴＩＳ　DBMSにおけるデータベースのスキーマ定義は、コンパイルを必要とする定義モジュールではなく、インライン・ディレクトリに対話モードで直接定義されるので、ダイナミックにそれらの定義を変更することができる。これら強力な機能により、ＴＩＳ　DBMSは、効率、柔軟性および拡張性が大幅に広げられ、バッチおよびオンライン処理方式でオペレーションや情報センター要求に応えることが可能になった。

正規化されたデータ構造

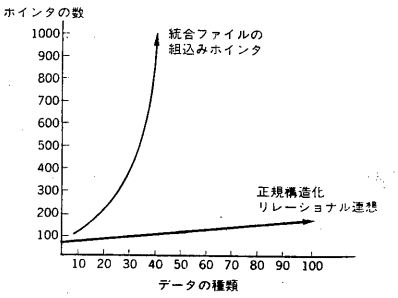
ＴＩＳは、データを正規化した形式で格納または、構成する。正規化されたデータ構造は、データを純粋で柔軟な形式に維持し、さまざまなアプリケーション・ニーズに対応できる利点を持っている（図—10）。



図—10　正規化されたファイル例

ハイブリッド・リレーショナル・データ連想

リレーショナル・データ連想とは、外部インデックス、ポインタ、テーブルを全く使わずに、正規化されたデータ項目間の関連付けをデータの内容に従って論理的に行う方法である。データは、必要に応じて独自のプライム・キーやデータ項目内の各種複合キーによってダイナミックに関連付けられるため、データベースは柔軟性を増す。データの関連付けには、最小限のポインタやインデックスのオーバーヘッドで、複数の論理構造が正規化データベース上に作られる（図—11）。



図—11　統合ファイルと正規化ファイルのポインタ数比較

主要オーバーヘッドとなっているポインタやインデックスが少なくなればなるほど、DBMSの効率は向上する。しかし、今日使用されているディスク装置ハードウエア技術は、リレーショナル技術のデータベース連想が完全にできる状態にまで致っていない。これは、純粋リレーショナルではほとんどが、ランダム・アクセスになるからである。データの内容でアドレスできる新しいハードウエア技術が開発されるまでは、純粋リレーショナル

DBMSが効率良く活動することは難しい。しかし、可能な限り「リレーショナル連想」技術を採用して、ポインタやインデックスによるオーバーヘッドを軽減し、パフォーマンスをできるだけ向上させておくことは可能である。

仮想アクセス・メソッド処理

データベースの効率を左右するディスク・アクセスは、仮想アクセス・メソッド処理（VAMP）を採用することによって、最小限に押さえられる。ＴＩＳ　DBMSは、データがメモリ上に存在していると、ディスクＩ／Ｏを行わない。また、データ・セット（ファイル）が独立したアーキテクチャになっているため、サイズの異なるバッファをいつでも使用することができる。データ構造の操作は、論理ユーザ・ビューとディレクトリが行い、データ構造は「見えない」状態になっているため、プログラマは、データ構造を意識している必要がない。

(4) プログラマ論理ビュー・サポート（PLVS）

ＴＩＳのプログラマ論理ビュー・サポート（PLVS）は、COBOL 、COBOL-XT、PL／ＩとＴＩＳ MANTIS（第４世代言語）プログラムで論理ユーザ・ビューを用いたデータのリレーショナル処理を可能にする。このため、プログラマは、簡単な４つの命令、すなわちGET 、INSERT、UPDATE、DELETEを駆使するだけで、データベース・ファイル、OS標準ファイル、パーソナル・ファイルまたはそれを組み合わせたファイルへの非常に複雑な要求にも応えることができる。

プログラミングは、論理ユーザ・ビューとPLVSの使用によって、原点に立ち返った。外部記憶装置が開発されて、ファイル概念が明らかにされる前のプログラミングでは、プログラム内にすべてのデータが存在しており、Ｉ／Ｏ処理ロジックは不要だった。論理ユーザ・ビューは、Ｉ／Ｏナビゲーションロジックがプログラムの中に現れないため、あたかもデータがプログラムの中に存在しているかのような状態で、データ処理を可能にした。

3. ＴＩＳの利点

完全なシステム制御と統合

ＴＩＳは、基本設計時点からすべてのコンポーネントの統合を可能にした。また、これら統合されたコンポーネントは、ＴＩＳ独自のインライン・ディレクトリによって、完全に制御され、維持される。従って、第一世代DBMSを中心にしたシステム統合の場合のように、システムの拡張に伴い整合性、効率、柔軟性、開発時間、保守の手間が大幅に悪化することはない。

インライン・ディレクトリは、データ・ディクショナリ機能を超えて、アプリケーション稼働時の支援も行い、アプリケーション開発の生産性を一段と向上させる。

データ構造からの完全独立

強力なリレーショナル技術が採用されているため、ＴＩＳの論理ユーザ・ビューは、アプリケーションをいかなるデータ構造からも完全に独立させる。従って、プログラム、レポート・ライタ、クエリに悪影響を与えずにデータ構造を再構成することができる。このことは、既存アプリケーションに影響を与えずに、今後開発される新しい技術やアプリケーションを、随時ＴＩＳに導入できることを意味している。

また、論理ユーザ・ビューの採用により、ユーザがデータ構造やアクセス・ロジックを知らなくても、データへのアクセスが可能になる。その結果、ＴＩＳは、エンド・ユーザが必要としている情報を自ら作り出す、「情報センター」の構築に不可欠なソフトウェア・ツールを提供することも容易になる。

強力なデータ操作命令

どんなに複雑なアプリケーションのアクセス要求でも、また、データベース、標準ファイル、パーソナル・ファイルを組み合わせた処理要求でも、簡単な４つの命令を用いるだけでデータ構造は、自動的にナビゲートされ、処理される。そのため、プログラムは簡潔なものとなり、開発期間も短縮される。

新しいDBMS

ＴＩＳ　DBMSは、正規化データ構造をサポートし、可能な限りリレーショナル連想手法でデータ構造を管理する。使用範囲の広い柔軟なデータ構造が構築でき、オーバーヘッドの原因となるポインタの数を大幅に減少させる。また、仮想アクセス・メソッド処理（VAMP）の採用により、大量のトランザクション処理も可能である。

強力なアプリケーション開発ツール

ＴＩＳは、エンド・ユーザ、プログラマやアナリストのために新しい第４世代問題解決ツールを提供している。例えば、MANCALC とＴＩＳ I.Q. は、アドホックな情報センター要求を満たし、MANGRAF や複雑なレポートを作成できるコンプリヘンシブ・リトリーバルは、強力な意志決定支援システムとなる。

ＴＩＳ　MANTISは、対話モードでの開発やプロトタイピング手法を採用しており、アプリケーション開発の生産性をこれまでの10倍以上も向上させることが可能である。