

## 5H-3

## 統合プログラミング環境(3)

ソフトウェア設計情報用データベースPDDの開発

湯原 義彦 濱崎 勝久 小林 茂  
(株) 東芝

## 1 はじめに

ソフトウェア開発支援のためのソフトウェア設計情報用データベースPDD(Program Data Dictionary)を、スーパーミニコン、およびEWS上に開発した。PDDは、現在これらのマシン上に構築している統合プログラミング環境、およびCASEツールにおけるソフトウェアオブジェクト(以下、単にオブジェクトという)情報の一元管理を目的としている。

高度な支援機能や作業の自動化の実現には、計算機内部に現実世界を的確に表現し、これを柔軟にアクセスできることが前提となる。PDDは、ソフトウェア開発支援用DBとして十分な汎用性を持つよう、機能/性能をチューニングされた関係DBである<sup>[1]</sup>。PDDを用いた支援機能の具体的なイメージについては<sup>[2]</sup>で紹介することにし、本稿ではPDDの、仕様上の特徴とインプリメントについて説明する。

## 2 仕様上の特徴

ソフトウェア開発支援用DBに、汎用高機能DBを用いた場合には、1)SQL等の専用の構文は、言語処理系や開発者にとって、生産性が高いとはいえない、2)DBマネージャのようなタスクの存在による負荷が高く、多くのツールが起動される環境には向かない、3)多数の開発者が、DBを利用した各種のツールを同時に実行するには処理が遅い、等の問題がある。

よって、PDDの開発では、以下の3点を最重点課題として取り上げた。

- 1)プログラムから柔軟にDBアクセスできること
- 2)ソフトウェアの構造の記述に適した表現力、および開発プロジェクトによるDBのアクセスを想定したデータ管理機能を持つこと
- 3)高速性・コンパクトさを持つこと

## 2.1 データ管理機能上の特徴

**レコード型:** PDDは、ソフトウェアをより正確に表現するため、オブジェクトを明示的にDBに登録する機能を持つ。つまり、DB内部にはオブジェクトを表すレコードと関係を表すレコードが登録され、関係レコードはオブジェクトレコードを参照することができる。この機能の必要性は、例えば同じ綴りで表される変数について関係記述をする際に、変数を名前文字列で表す場合と比較すれば理解できる。なお、オブジェクトや関係のレコード型は、DBを利用するツールの必要に応じて、自由に定義可能である。

**フィールド型:** レコードのフィールド型には、整

数型(半・全語)、実数型(単・倍精度)、名前型、文字列型、バイナリ型、およびオブジェクト型(関係レコードでのみ可能)がある。上記中、名前型と文字列型はともに文字列を値とするが、名前型は、同じ綴りのものはDB全体で一つの実体を共有する点が異なる。名前型により、DBサイズの節約と検索の高速化が実現される。バイナリ型は任意の大きさを持ったバイナリデータであり、設計図面などの保存のために用意した。

**データベースの階層化:** DB内部をサブDBと呼ぶ階層化された領域に分割する機能がある。レコード型の定義は全てのサブDBで共有されるが、検索の際に対象範囲をサブDB単位で指定できるため、開発プロジェクト全体の共通情報と個人の情報の使い分けなど、スコープの管理に利用可能である。また、各々のサブDBについて所有者やアクセス権を設定できる。

**検索:** 検索条件は、典型的にはフィールドに変数(レコードとのマッチングにより値が束縛される)を含むレコードパターンを、AND/OR/NOT結合した形で与える。そのほか、フィールド値を個別に大小比較することも可能である。また、ソフトウェアの開発支援では、モジュール構成やルーチン間の呼出関係のように階層的な構造を扱うケースがしばしば発生すると予想されたため、特定の関係でリンクされたオブジェクトを再帰的にたどる検索手段を用意している。

## 2.2 性能上の特徴

PDDの性能について、重視した項目順に列挙する。

**検索速度:** 標準的な検索(インデックスの有効な、2~3の条件要素からなる検索)では1秒以内に結果が得られる。

**登録・削除速度:** 最もデータ量の多い場合と思われるクロスリファレンス情報では数千件以上のレコードの一括登録・削除が必要となるが、これも開発者の耐えられる待ち時間(数秒間)で処理できる。

**容量:** 開発支援では超大規模なDBは必要でないと思われるため、百万件位までを現実的な性能で動作する許容データ量としている。

**ファイルサイズ:** インデックスなどの制御情報、および空き領域を、DB全体の20~30%程度におさえて、コンパクト化している。

## 2.3 インタフェース上の特徴

プログラムからのDBアクセスインタフェースは、アクセスメソッドと総称する、C言語用ライブラリ関

A Development of Database for Integrated Programming Environment

Yoshihiko Yuhara, Katsuhisa Hamazaki, Shigeru Kobayashi

TOSHIBA Corp.

数群である。通常のファイルアクセスのような手軽さでDBを操作できる。

代表的なアクセスメソッドを以下に示す。

DBの作成・削除	credb("path") remdb("path")
DBのアクセス制御	open("path", mode) close()
レコード型の定義	cretbl("tblname", tblstruct) remtbl("tblname")
レコードの登録	crercd("tblname", fldvals)
DBの検索	select() cond("tblname", fldvals) or() / not() cmp(val1, op, val2) fetch(varlist, bufaddr) remrcd(redlist)

表1. 主なアクセスメソッド

3 インプリメント

3.1 DBのファイル構成

PDDのDBは、各々のテーブル（オブジェクト、関係）に対応するファイルと、名前型と文字列型の値の実体を格納したテーブルファイル、およびDB制御情報ファイル群からなるディレクトリを構成する。

3.2 テーブルの構造

テーブルの構造を簡単にし、かつファイルサイズを小さくするために、インデックスは第1フィールドについてのみ、自動的に作成される。これは、DB設計上かなりきつい制約であるが、ソフトウェア情報の場合には特定のシンボルに注目して検索を行うことが多いため、レコードを代表するシンボルを第1フィールドとしておけば、実用上問題は少ないと考えている。また、特定のフィールドが同じ値を持つレコード群が多数出現する場合には、それらのレコードをグループ化することによって、論理的に2つのフィールドにインデックスを設ける手段がある。

各テーブルは、レコードを格納したブロックとインデックスの領域から構成される。テーブル内の全レコードはソートして格納されるため(図1.)、登録時の処理が幾分重くなるが、検索では、(1)インデックスによるブロックの特定、(2)第1フィールドによるレコードの特定の2段階の2分木探索により所望のレコードを特定できる。オブジェクト型のテーブルでは、このほかにレコードへのエントリ領域があり、テーブル外からのレコード参照に矛盾が生じない構造になっている。インデックスとエントリの領域は、DBをオープンしている間メモリに常駐させ、高速化を図っている。

3.3 検索機構

先にみたように、検索条件の指定では、個々の条件要素に対応してアクセスメソッドを呼出す。検索時には要素間でバックトラックが発生するわけであるから、

各々の呼出では条件を内部に蓄えておき、条件の完結時に一括処理する必要がある。PDDはこのとき検索条件を中間コードにコンパイルし、同時に各種の最適化を行い、これを実行する。

3.4 排他機構

PDDは、1)登録レコードバッファが一杯になったとき、および2)検索用中間コードを実行するときに、メモリ上のデータとDBファイルの内容の同期をとる。いずれも、高々数秒間の処理なので、たまたま競合が発生したとしても実用上問題にならないことと、ソフトウェア開発支援DBは、それほどこまめに更新されないことより、PDDは排他処理は極めて単純化されている。上記の2つの処理の間だけ、他のタスクをブロックする(各々、書き込みモードと読み込みモード)。

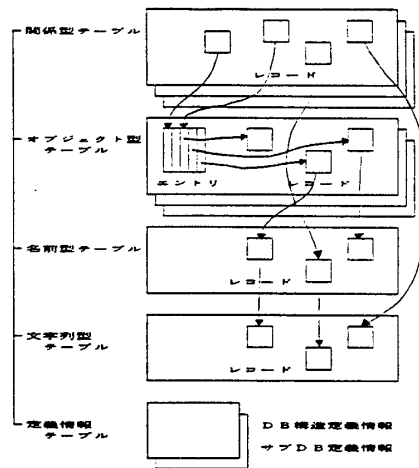


図1. PDDの構造

4 おわりに

ソフトウェア開発支援用DB、PDDについて、特徴とインプリメントの概要を説明した。

当初、この種のDBでは一般のDBで重要視されるような、データの保全性やプログラムとDBの独立性などはあまり問題にならないだろうと考えたが、開発環境構築の過程で結局これらの機能に対する要求が徐々に発生してきている。既にその一部は組み込んでいるが、表現力の強化などと併せて、改良を継続していく予定である。

参考文献

[1]小林他:「統合プログラミング環境(2)開発支援データベース」、情報処理学会第39回全国大会  
 [2]濱崎他:「統合プログラミング環境(4)ソフトウェア設計情報用データベースPDDに基づく開発環境」、情報処理学会第41回全国大会