C言語に変換できるDeviceSQLによるプログラミング ―― テーブルへの挿入と検索

使って覚える組み込みデータベース入門(前編)

近年の組み込みシステムでは，大容量データを扱う需要が増えている．本稿では，組み込みシステム向けのデータベースを使ったプログラミング手法について解説する．ここで取り上げるDeviceSQL言語はC言語への変換を前提とするなど，組み込みシステムとの親和性が高い．DeviceSQLの評価版はhttp://www.encirq.co.jp/からダウンロードできる．（編集部）

高橋 君夫

1．組み込み機器におけるデータの取り扱い

組み込みシステム開発において，製品の世代交代とともにアーキテクチャを一新し，すべてを新規に作り直すことがあります．そのような場合，今後の何世代かの間に陳腐化しない製品アーキテクチャの検討や，開発コストを低く抑え，効率良く開発するための検討が必要になるでしょう．

● カーナビなどではデータベース搭載が標準

最近の組み込みシステムでは，より高度な機能が求められています．一つの例として，ハード・ディスク装置(HDD)内蔵カー・ナビゲーション・システムを考えます．最近のカーナビは，当たり前のようにCDから音楽を録音でき，アルバム名やアーティスト名，曲名などの情報も自動的に保存されます．ユーザは録音された曲を「アルバム」や「アーティスト」，「ジャンル」などのキーワードで選ぶことができます．これらの機能をC言語でゴリゴリ書いていくのは，正直なところかなり大変です．このようなコンシューマ製品で，なおかつ「検索」が必要なアプリケーションでは，確かにデータベースが必要です．

● FA機器などの分野でもデータベースを使用

ここのところ大きく変化しているのが，FA(Factory Automation)など，非常に伝統的な組み込みシステム分野です．近年のFA関連装置では，単にデータをロギングするだけでなく，そのデータを分かりやすく活用できるようにしたいという要望が高まっています．さらに，「このデータが取れるなら，関連してこれも…」というようなユーザ要求の増加により，「さすがに自分で一から構築するのは無理」という話が増えています．

このように，多くの組み込みシステムにおいて「データを効率良く扱う」ことが要求されています．その一つの方法として，データベース・ソフトウェアを活用するユーザが増えています．これまでなかなかデータベースが組み込みシステムの世界で日の目を見なかったのは，「限られた資源と低消費電力」という制約のためだと思います．実際に採用を検討しようと思っても「フットプリントが何Mバイト」というようなスペックを見た瞬間，採用の候補から外れてしまってもおかしくありません．しかし，ここ数年，データベース・ベンダ各社の努力もあって「限られた資源」というハンデの中でも高速に動作する，組み込みアプリケーションに特化したデータベースが登場しています．つまり，数十Kバイトから100数十Kバイトでなおかつ16ビットCPUでも動作するようなデータベースです．

本稿ではそのような組み込み向けデータベースを使用した簡単なデータベース向けプログラムの書き方と，それを応用したサンプル・プログラムを紹介していきたいと思います．

2．手続き型SQLによるプログラミング

さて一口にデータベース・プログラミングといっても，大ざっぱにSQLをベースとするものと，まったく独自APIで記述するものの2種類があります．さらにSQLベースのものでも，SQL文をAPI経由でCアプリケーションからデータベース・サーバに問い合わせるもの，手続き型SQL言語を使用し，データ・アクセスをプロシージャとして処理するものなどに分類できます．そのような中で，今回はよりC言語から利用しやすい手続き型SQL言語であるEncirq社のDeviceSQL言語を利用し，いかにアプリ

KEYWORD ―― DeviceSQL，SQL言語，C言語への変換，データの挿入，データの検索

ケーションからデータベースを使うかを説明します．

● データベース記述言語 DeviceSQL は PL/SQL 由来

DeviceSQL Language（以下，DeviceSQL）は Oracle 社の PL/SQL をベースとしたデータベース記述言語です．SQL の良いところは，高度な検索を一つの SQL 文で処理できるという点ですが，SQL 単体は手続き型の言語ではないため，ある SQL 文の結果を次の SQL 文の入力には使用できません．そのため，必然的に複雑な検索を行おうとすると，SQL 文そのものが複雑になっていきます．

これに対して PL/SQL を基とする DeviceSQL はストアード・プロシージャ，つまり関数を記述する言語として出発しています．つまり手続きを踏むことが可能な SQL 言語なので，簡単な SQL 文の組み合わせで，複雑な検索を行うことが可能です．つまり「これから SQL を学んでみよう」という方にとって敷居の低い言語といえます．

● DeviceSQL コンパイラを通じて C 言語に変換する

今回使用する Encirq 社の DeviceSQL 環境では，DeviceSQL 言語で記述したソース・コードは，DeviceSQL コンパイラを通じて C 言語に変換し，コンパイル・リンクを行って実行形式を作成します．この過程で SQL 文は DeviceSQL Framework が提供するライブラリ・コールに変換されます．このため，SQL 文を実行時に解釈するオーバヘッドがなく，ごく簡単な「SELECT *FROM table WHERE

リスト 1　DoNothing.epl

```
CREATE PROCEDURE    DoNothing
AS
  EXPORT NAME "db_DoNothing";
BEGIN

END;/
```

リスト 2　DoNothing.c

```
void DoNothing(void)
{

}
```

リスト 3　InsertData.epl

```
--**********   TABLE CREATES   *************************

CREATE TABLE UserTable
(
  UserID    LONG,
  Name      VARCHAR,
  Age       SHORT
);/


--**********    IMPORTED FUNCTIONS    *******************

-- c_Print の実態は C 言語側で実装されており，それをインポート
-- する

CREATE PROCEDURE c_Print (msg IN VARCHAR)
AS
  IMPORT
  NAME "c_Print";/

CREATE FUNCTION InsertData ( lUserID IN LONG, vName IN VARCHAR,
sAge IN SHORT)
RETURN LONG
AS
EXPORT NAME "db_InsertData";
BEGIN

-- 年齢が 0 歳未満，150 歳より上であれば，
-- 不正データとしてエラーを返す

   IF (sAge < 0) OR (sAge > 150) THEN

        RETURN -1;

    ELSE

-- テーブルへのレコードのインサート

        INSERT INTO UserTable VALUES (lUserID, vName, sAge);

    END IF;

    RETURN 0;

-- SQL 文でエラーが起きた場合は例外となり，こちらの例外
-- ハンドラで処理を行う．この例では特に SQL エラー・コードに
-- 合わせた処理は行わず，一律にエラーとして -1 を返している

EXCEPTION
      RETURN SQLCODE;
END;/



CREATE FUNCTION GetRecordById (lUserID IN LONG)
RETURN LONG
AS
  EXPORT NAME "db_GetRecordById";

-- ローカル変数は AS-BEGIN 間で定義する
  vName VARCHAR;
BEGIN

-- DeviceSQL ではローカル変数に検索した値を保持することができる

    SELECT Name INTO vName FROM UserTable WHERE UserID =
lUserID;

-- C 言語で記述した関数を呼び出すことが可能なため，プリント文だけで
-- なく，データにフィルタ処理をかけるなど，既に開発
-- 済みの C 言語の資産を利用できる．
-- また，既存のデータで DB にデータがないものでも，インポート
-- 関数を通じてデータの取得や保存が可能であるため，DB だけで
-- なく，システムのデータ全体を DeviceSQL で記述できる

    c_Print(vName);

    RETURN 0;

EXCEPTION
    WHEN OTHERS THEN
      RETURN -1;
END;/
```