V Y U Ž I T Í D Y N A M I C K Y Z A V Á D Ě N Ý C H Č Í S E L N Í K O V Ý C H M O D U L Ů , T E C H N O L O G I E J E J I C H U Ž Í V Á N Í A Ú D R Ž B Y

Ing.Jiří Prikner, Ing.Božena Vaňková , RNDr.Pavel Hofman

I. Generační číselníkové moduly a jejich dynamické volání

1. Programová podpora dynamického volání a generačních číselníků vypracovaná ve VS POLDI SONP KLADNO

1.1. Spojovací program

První částí programové podpory je spojovací program. Ten umožňuje dynamicky vyvolat z uživatelova programu pod jedním jménem jeden z množiny číselníků, jehož jméno je v zadané relaci se jménem zadaným ve spojovacím programu. Spojovací program dovoluje dynamicky zavedený číselník také zrušit. Tím je umožněno šetření paměti.

Spojovací program má vstupní body TPETCH (pro dynamické zavedení) a TRELSE (pro zrušení dynamicky zavedeného modulu).
Jeho volání v PL/I je provedeno příkazem ve tvaru:

CALL TPETCH (jmod,parem);

kde: jmod - jméno určující volaný číselník,
parem - seznam parametrů pro volaný číselník.

Chceme-li zavedené moduly zrušit, má volání tvar:

CALL TRELSE (seznam);

kde: seznam - seznam jmen modulů.

Jméno určující volaný modul je generační, tím je myšleno, že je vytvořeno podle syntaxe x g v y, kde:
x, y jsou řetězce znaků pevné délky a konstantní pro všechna jména verzí určitého modulu.

g je pevně určený znak, označující začátek a délku řetězce, označujícího versi jména.

v je řetězec znaků pevné délky, označující určitou versi.

V dalším bude g = 'L' a určuje versi jako řetězec dvou znaků. Např. jméno XULOL++A, kde '++' je libovolná (povolená) dvojice znaků, určuje množinu versí číselníků, jednotlivé verse mají význam datumu, od kdy určitá verse platí. Kódování datumu ve versi má tvar RM, kde R je rok (A - 1981, B - 1982, ...Z - 2006), M je měsíc (A - 1, B - 2, ...L - 12). Spojovací program při volání např. jména XULOLCCA vyhledá v dostupných zdrojích (t.j. operační paměť nebo knihovna s údjménem) číselník se stejným generačním jménem. Jeho verse bude odpovídat datumu, který je nižší než datum kódovaný ve jménu XULOLCCA (t.j. 1.3.1983). Je tedy vyhledán číselník, který je platný k určenému datu. Jméno skutečně vyvolaného číselníku je předáno do proměnné FNAME. Relaci, kterou mají splňovat verse volaného a volajícího jména je možné zadat v proměnné FRELT, jako implicitní hodnota je zvolena relace 'LE' (menší nebo rovna). V jediném příkazu CALL TFETCH je tedy možné dynamicky vyvolávat číselníky se stejnou strukturou parametrů, jejichž jména jsou určena až při běhu programu.

1.2. Generační číselníky

Generační číselník je množina číselníků se stejným generačním jménem. Druhá část systému podpory předpokládá, že každá verse generačního číselníku je určena obdobím její platnosti, kde období je kódováno způsobem popsaným v 1.1.

Jména všech právě platných číselníků jsou uchovávána ve zvláštním souboru - katalogu jmen. Ten je aktualizován vždy při vzniku nového generačního jména, popř. vzniku nové verse. Naplnění knihovny load modulů pro dynamické volání provádí program, který

podle katalogu jmen vybere jména versí číselníků, která platí v obdobích zadaných v parametru. Období je možné zadat absolutně i relativně vzhledem k zadanému datu.
Např. u generačního číselníku XU10L++A chceme pracovat s versemi platnými v minulém pololetí, u generačního číselníku XU15L++B chceme pracovat s versemi platnými minulý měsíc.
V seznamu jmen pro naplnění load knihovny uvedeme:
XU10L++A, .../XU15L++B, ...
V parametru určujícím období zadáme T = '-P,-M'.
Podle katalogu generačních jmen jsou vybrána všechna jména vyhovující zadaným podmínkám a odpovídající load moduly jsou zavedeny do knihovny. Spojovací program pak vybere podle jména zadaného v jeho volání určitý číselníkový modul.

II. Využívání generačních číselníkových modulů

2.1. Využití z hlediska návrhu systému a programování

Při výběru problémů vhodných k řešení technologií spolupráce programu ve vyšším jazyce a číselníkových modulů narážíme na jediné technické omezení a to:
Číselníkové moduly zde vystupují jako datové soubory normativní povahy či jako data řídící povahy. Zvláštností těchto datových souborů je to, že musí být cele umístěny v paměti. Tímto způsobem tedy můžeme řešit problémy, kde rozsah programu a rozsah aktuálně používaných číselníkových modulů je menší než maximální rozsah vymezené paměťové oblasti. Toto omezení lze překonat vhodným rozčleněním problému na jednotlivé funkce a k těmto funkcím dynamicky zavádět pouze nezbytně nutné číselníkové moduly, ostatní dynamicky zaváděné moduly musíme vymazat.
Při stavbě struktury programu musíme přihlížet k velikosti reálné paměti a velikosti virtuální paměti, popř. poměru virtuální/reálné paměti. Na velikosti provozovatelné paměti

záleží efektivnost této technologie a struktura stavby programu. Při menší paměti je nutno dělat rozdělení programu na kmen a dynamicky zaváděné a vymazávané větve. Přitom je třeba dbát o to, aby jednotlivé větve nebyly zaváděny často, t.j. rozčlenit řešení tak, aby se na všech datech prováděly operace jednotlivých větví za sebou, či rozčlenit data tak, aby jedna větev programu ošetřovala vždy 1 typ dat. Pro efektivnost musíme též přihlédnout k poměru virtuální/reálné paměti (pokud provádíme výpočet na virtuálním stroji), při vyšším poměru je lépe program rozdělit, aby nerostlo neúměrně stránkování.

Z povahy číselníkových modulů plyne, že jsou velice výhodné pro takové datové základny, kde výstupní normativní údaj je závislý na seznamu klíčových hodnot, či intervalu klíčových hodnot, či průniku seznamů intervalů klíčových hodnot. Navíc členění této datové základny může být velmi proměnlivé a to i v rámci jediné číselníkové tabulky.
Zpracování může být samozřejmě doplněno libovolnými standartními soubory se stálou strukturou a standartním přístupem (VSAM,ISAM,SAM,DAM).

Číselníková data však často potřebujeme v několika časových úrovních najednou a to v okamžiku, kdy pro běžné zpracování platí nový číselníkový modul a pro zpracování závěrek, statistických přehledů a výrobních evidencí musíme používat ty číselníkové moduly, které byly platné v čase pořízení zpracovávaných dat.
Zde s výhodou použijeme generační číselníkový modul, kde datum zadání buď jobu, či přímo volané při TFETCH tabulky zajistí výběr a zavedení správné generace číselníkového modulu.

Programovací technologie masového nasazení číselníkových modulů je tedy vhodná pro záznam normativních dat typu: podnikové kódy a číselníky, kalkulační tabulky, ceníky, technologické

předpisy a výrobní vyjímky, tabulky sloužící jako alternativní indexy normativních souborů (alternativní klíč - tabulka - primární klíč - soubor - věta) a tabulky standartních výběrů pro tisky.

Při použití této technologie zpracování dosahujeme tedy těchto efektů:
- minimalisace spotřeby času, prohledávání číselníkových tabulek je velmi efektivní, nepotřebuje čas na I/O operace (mimo nové zavedení dynamické tabulky),

- dosahujeme lepší přehlednosti problému jednoznačným přiřazením funkce k dané tabulce.

Členění symbolického modulu PL1 programu je pak následující:

```
příprava vstupních parametrů;
CALL tabulka (Y, X1, .....);
IF Y = prvek není v tabulce
THEN opatření pro tento případ;
ELSE další výpočet;
       :
```

- z programu vymizí všechny podmíněné příkazy a logickými výrazy, kde jsou uváděny konstantní hodnoty dané číselníkové proměnné. Výskyt těchto příkazů při jakékoliv změně číselníkové hodnoty silně znepříjemňuje život. Tyto výrazy jsou nahrazeny volením číselníkového modulu, který vrací hodnoty ano/ne, případně číslo alternativy.

- stejný program a job může být beze změny provozován pro normativní tabulková data různých údobí užijeme-li generační dynamickou formu číselníkového modulu.

Z hlediska projektování tedy dosahujeme výrazných kladných efektů.

2.2. Nutné provozní zajištění této programovací technologie

Pro správnou funkci takto navržených programů musíme zajistit i provozní předpoklady a to:

2.2.1. Zajistit standartní provozní údržbu číselníkových modulů.
Protože číselníkový modul je datový soubor i programová přístupová procedura, musíme provést:

- proškolení uživatele v programování číselníkových modulů, aby byl schopen je správně aktualisovat, zapsat změny do formuláře a seznámit ho s náležitostmi nutnými ke standartní údržbě jeho tabulek. Musíme ho naučit zadávat zkušební data a kontrolovat obdržené výsledky.

- zavést v provoze počítače standartní údržbu číselníkových modulů, t.j. naučit skupinu pracovníků provozu zpracovávat zaslané formuláře změn a zkušebních dat číselníkových modulů, zadávat je do příslušných provozních jobů pro jejich údržbu a kontrolu, výsledky rozesílat a opravit následky případného výpadku počítače při zpracování aktualisačních jobů.
Provozní skupina zajistí převod správné tabulky do provozních load knihoven na pokyn uživatele.

2.2.2. Stanovit jednoznačně odpovědnost a to:

- uživatele za obsah tabulky, uživatel má právo vytvořit si novou generaci, ale nemá právo měnit počet, pořadí, zobrazení a měřítko parametrů tabulky.

- projektanta za stanovení parametrů tabulky.

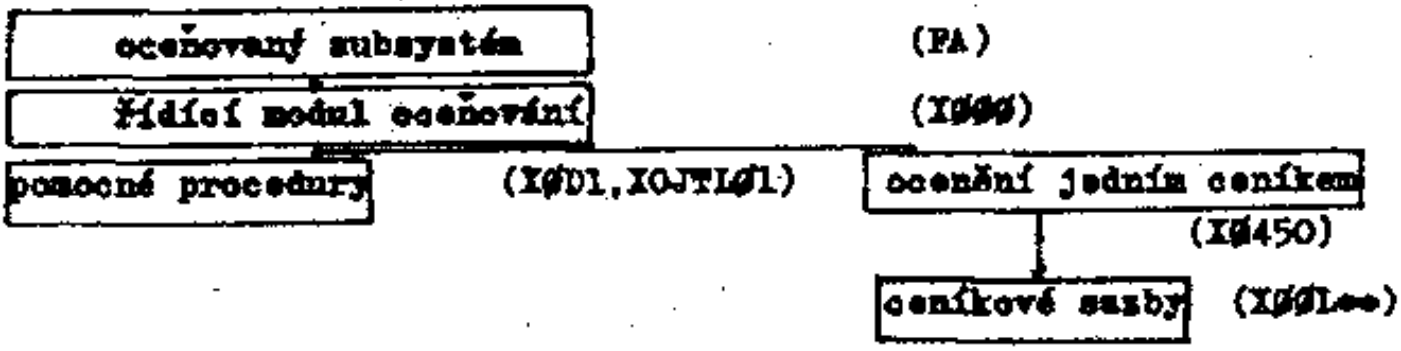
- provozu za údržbu dat v tabulce dle datových formulářů.

Po splnění těchto předpokladů bude tato technologie poskytovat slušné výsledky.

III. Aplikace generačních číselníkových modulů

Příklad se týká oceňování faktur, kdy v jedné dávce budou oceňovány faktury cenových úrovní 1980 i 1984

3.1. Schema oceňovacího subsystému



3.2. Výběr číselníkových modulů

```
//     EXEC PREVDAL
//SYSIN  DD  *
 PA,XØJTLØ1,XØØ3LAA,XØØ3LBA
```

3.3. Programové řešení v PL1/OPT

```
PA: PROC OPTIONS(MAIN);                  /* spracování faktur */ /
    DCL 1 FAKT,                          /* věta faktury      */
          :
          2 IDENT-VYR                    /* identifikační údaje
                                            fakturovaného výrobku */
          2 DATUM                        /* dat.fakt. ve tvaru RRMMDD*/
          2 CENA                         /* velkoobch.cena faktury */
   CALL XØØØ (IDENT-VYR,DATUM,CENA);
          :
   END;

XØØØ: PROC(IDENT-VYR,DATUM,CENA);
      DCL IDENT-VYR,DATUM,CENA,
          XØDEL(5) ENTRY VARIABLE INIT(XØD1,...),
          XØCENIK(5) ENTRY VARIABLE INIT (XØ45Ø,...),
          (XØD1,XØ45Ø,XØDAT,XØJTLØ1) ENTRY,
          CENIK-STARY FIXED BIN(15) STATIC INIT(-1),
          CENIK-NOVY FIXED BIN(15),  DATT CHAR(2);
```

```
      CALL XØDAT(DATT,DATUM);    /* převod datumu    */
      /* XØDAT = PROGRAMOVA TABULKA,NEMENNA NAPLN,
       ZUSTAVA V PAMETI,POUZIVA SE I V JINYCH FAZICH ZPRAC.PA */
      CALL XØJTLØ1 (CENIK_NOVY,IDENT_VYR); /* určení typu ceníku*/
      /* XØJTLØ1 = DYNAMICKA NEGENERACNI TABULKA,RELATIVNE NE-
       MENNA NAPLN,V PAMETI BUDE POUZE PO DOBU OCENOVANI */
      IF CENIK_NOVY¬=CENIK_STARY  /* uvolnění tabulek z paměti */
        THEN CALL XØDEL (CENIK_STARY);
      CALL XØCENIK (CENIK_NOVY) (IDENT_VYR,DATT,CENA);
                                                  /* ocenění */
 END XØØØ;

XØD1: PROC; /* uvolnění generačních modulů ceníku 4/5 */
      CALL TRELSE (XØØ1L**,XØØ2L**,XØØ3L**,...);
 END XØD1;

XØ45Ø: PROC(IDENT_VYR,DATT,CENA); /* VYPOCET VC SUROVE OCELI */
       DCL IDENT_VYR,DATT,CENA;
       DCL 1 VYR DEF IDENT_VYR,
            2 OBOR FIXED(3),2 FORMAT FIXED(3),2 OCEL FIXED(7),
            :
            :;
       DCL 1 PRIR,2 SAZBA FIXED BIN(15),2 ROZM CHAR(5);
       DCL TFETCH ENTRY, JM CHAR(8),(PINTER,PNAME)CHAR(8) EXT;
 ON EROR PUT SKIP EDIT ('ZPRACOVALA SE TABULKA:',PNAME)(A);
 JM = 'XØØ3L'||'DT';    /* VYPOCET PRIRAZKY ZA KRUHOVE INGOTY */
 PINTER = 'ASM';
 CALL TFETCH (JM,PRIR,OBOR,FORMAT,OCEL);
      :
      :
 END XØ45Ø;
```