ZESZYTY NAUKOWE POLITECHNIKI ŚLĄSKIEJ 1994

Seria: INFORMATYKA z. 25 Nr kol. 1236

Dariusz AUGUSTYN

MECHANIZMY WIELODOSTĘPU I TRANSAKCYJNEGO PRZETWARZANIA DANYCH W SYSTEMIE PROGRESS 6

Streszczenie. Artykuł prezentuje cechy systemu Progress 6 dotyczące mechanizmu transakcyjnego przetwarzania danych w kontekście blokowej struktury programu. Rozpatrywane jest również zagadnienie związane z pracą systemu w trybie wielodostępnym. Analiza ciągu przykładowych programów zawarta w artykule pozwala na wyjaśnienie pojęć transakcji, subtransakcji, poziomu blokad, związku zakresów transakcji, subtransakcji i blokad.

THE MULTI-USER ENVIRONMENT AND TRANSACTION DATA PROCESSING IN DATABASE MANAGEMENT SYSTEMPROGRESS 6

Summary. This article presents features of multi-user database management system Progress 6 which are related to transaction data processing and a block structure of Progress programs. Considering of sample programs lets to explain ideas of transactions, subtransactions, types of data locking, relation between scopes of transactions and scopes of data locking.

DIE MECHANISMEN DES MEHRBENUTZERZUGRIFFS UND DER TRANSAKTIONSDATENVERARBEITUNG IM SYSTEM PROGRESS 6

Zusammenfassung. Im Artikel wurden die Merkmale des System Progress 6 dargestellt, die den Mechanismus der Transaktionsdatenverarbeitung unter dem Aspekt der Blockstruktur des Programms betreffen. Das Problem der Arbeit des Systems im Mehrbenutzermodus wurde erÖrtert. Die Analyse der im Artikel enthaltenen Beispielprogramme ermÖglicht, die folgenden Begriffe: die Transaktion, die Subtransaktion, die Sperrenart, den Zusammenhang zwischen den Bereichen der Transaktionen und der Sperren zu erlaütern.

1. Wstęp

Progress jest systemem pozwalającym na tworzenie aplikacji zarządzających wielodostępnymi bazami danych oraz zapewniającym transakcyjne przetwarzanie danych.

W uproszczeniu transakcje są wyodrębnionymi fragmentami programu, modyfikującymi bazę danych i realizującymi pewne funkcjonalnie zamknięte zadania. Poprawna realizacja transakcji polega na wykonaniu wszystkich modyfikacji i nazwana jest wypełnieniem transakcji. Jeżeli z pewnych przyczyn typu awaria systemu, kolizja dostępu, wykonanie którejś z modyfikacji nie jest możliwe, wprowadzone wcześniejsze zmiany muszą być anulowane. Proces taki nosi nazwę wycofania transakcji. Nowoczesny system zarządzania bazą danych powinien posiadać wbudowane mechanizmy śledzenia modyfikacji realizowanych w bazie danych przez transakcję i ewentualnego automatycznego ich wycofywania. Implementacja systemu przetwarzania transakcyjnego jest warunkiem koniecznym do zapewnienia spójności bazy danych.

Programista określa zakres obowiązywania transakcji zgodnie z potrzebami zadania. W większości języków przeznaczonych do tworzenia systemów zarządzania bazą danych początek i koniec transakcji wyznaczają wystąpienia specyficznych instrukcji. W Progress-ie blokowa struktura programu i rodzaj operacji dokonywanej na bazie danych domyślnie określa zakres transakcji. Rozwiązanie zastosowane w Progress-ie polega na automatycznym potwierdzeniu modyfikacji przy wykonaniu instrukcji końca bloku, stanowiącego transakcję. (Istnieje możliwość przełączenia Progress-a we wcześniej wspomniany tryb jawnego ustalania zakresu transakcji poprzez jawne wystąpienia instrukcji.)

Wyróżniającą cechą Progress-a jest wielopoziomowa struktura transakcji, polegająca na wystąpieniach zagłębionych transakcji, tzw. subtransakcji, z możliwością wycofania się z dowolnego poziomu zagłębienia. Poprzez wycofanie należy rozumieć anulowanie modyfikacji w bazie danych i dodatkowo anulowanie zmian wartości zmiennych, które wystąpiły w obrębie wycofywanej subtransakcji. Ta ostatnia cecha, dotycząca wycofywania modyfikacji zmiennych w subtransakcji, jest oryginalną własnością Progress-a.

Transakcyjne przetwarzanie jest podstawą działania programu w systemie Progress niezależnie od jednoużytkownikowego lub wieloużytkownikowego trybu pracy systemu.

Wieloużytkownikowy tryb pracy systemu wymaga, by zaimplementowane były mechanizmy pozwalające na bezpieczne zrównoleglenie operacji jednocześnie wykonywanych na bazie danych przez różne zadania. Przykładem może być blokada dla innych zadań dostępu do wcześniej zmodyfikowanych danych i danych aktualnie modyfikowanych aż do momentu zakończenia bieżącej transakcji.

Progress jest systemem przystosowanym do pracy w środowisku wielodostępnym ze względu na zaimplementowane mechanizmy blokad różnych poziomów: bez blokady, blokada w trybie dzielonym, blokada w trybie wyłącznym. Podobnie jak zakres transakcji i subtransakcji, automatyzm zakładania blokad, poziom blokad, zakres ich obowiązywania związany jest z blokową strukturą programu.

Zawarta w artykule analiza ciągu przykładów pozwala wyjaśnić mechanizmy wielodostępu i przetwarzania transakcyjnego.

2. Transakcje

Transakcje są fragmentami programów modyfikujących pliki danych, które po rozpoczęciu wykonania powinny zostać zrealizowane w całości. W sytuacji uniemożliwiającej dokończenie transakcji (np. w przypadku awarii sprzętowej, programowej, naciśnięcia klawisza przerywającego) modyfikacje dokonane przez niewypełnioną transakcję powinny zostać anulowane (wycofane). Blok, który zawiera instrukcje modyfikacji danych, instrukcje odczytu z blokowaniem w trybie wyłącznym lub słowo kluczowe TRANSACTION stanowi transakcję, pod warunkiem, że jest najbardziej zewnętrznym blokiem programu posiadającym którąś z wymienionych cech.

Przerwanie transakcji dla celów testowych jest przeprowadzone poprzez symulację zdarzenia przerywającego tzn. naciśnięcie klawisza Ctrl-C. (Oryginalnym klawiszem przerywającym dla Unixa jest Del, przedefiniowany na Ctrl-C ze względu na edytor Progress-a.)

Oczywiście, transakcja przerwana poprzez naciśnięcie Ctrl-C wycofywana jest od razu, natomiast w przypadku awarii sprzętowej wycofanie nastąpi po ponownym uruchomieniu systemu.

Baza danych w systemie Progress tworzona jest w formie na ogół jednego pliku fizycznego w sensie systemu operacyjnego (tzw. cooked file) o nazwie <nazwa_bazy>.db. Z fizycznym plikiem bazy danych skojarzony jest fizyczny plik o nazwie <nazwa_bazy>.bi (tzw. Before Image File - plik obrazu transakcji), wykorzystywany przez system do rejestrowania transakcji i ewentualnego ich anulowania.

Przykładowe programy służące do ilustracji mechanizmów przetwarzania transakcyjnego realizują operacje na prostej bazie danych, której struktura zamieszczona jest w rozdziale "Dodatek".

W przykładowym programie p1.p każda iteracja pętli REPEAT stanowi niezależną transakcję. Blok wnętrza pętli staje się transakcją, ponieważ zawiera instrukcję modyfikacji danych INSERT oraz bardziej zewnętrzny blok (w tym wypadku blok programu) nie ma charakteru transakcyjnego (np. brak jawnych wystąpień instrukcji modyfikacji). Przerwanie wykonania programu w trakcie wprowadzania danych o kolejnym pracowniku powoduje anulowanie ostatnio wprowadzanego rekordu pracownika. Wcześniej wprowadzone dane o pracownikach nie zostają wycofane.

```
/* przyklad  p1.p */
REPEAT :
  INSERT PRAC WITH 2 COLUMNS.
END.
```

Przykład p2.p dotyczy rozszerzenia zakresu transakcji. Instrukcja DO TRANSACTION wyznacza blok transakcji ze względu na frazę TRANSACTION oraz z powodu tego, że bardziej zewnętrzny blok (blok programu) nie ma charakteru transakcyjnego. Wszystkie iteracje pętli REPEAT (wnętrze bloku DO) stanowią pojedynczą transakcję. Przerwanie transakcji powoduje anulowanie wszystkich wprowadzonych rekordów do pliku PRAC.

```
/* przyklad  p2.p */
DO TRANSACTION:
  REPEAT :
    INSERT PRAC .
  END.
END.
```

W przykładzie p3.p zewnętrzna pętla REPEAT wyznacza transakcję, ponieważ stanowi najmniej zagłębiony blok zawierający instrukcję modyfikacji INSERT. Transakcją jest każda pojedyncza iteracja zewnętrznej pętli REPEAT. Przerwanie transakcji w momencie wprowadzania kolejnego pracownika powoduje anulowanie ostatnio wprowadzonego zespołu i wszystkich wprowadzonych pracowników tego zespołu.

```
/* przyklad  p3.p */
REPEAT :
  INSERT ZESP.
  REPEAT :
    CREATE PRAC.
    PRAC.NRZ = ZESP.NRZ.
    DISPLAY PRAC.NRZ.
    UPDATE PRAC.NRP PRAC.NAZWISKO.
  END.
END.
```

Przykład p4.p dotyczy zawężenia domyślnego zakresu transakcji. Zewnętrzna pętla REPEAT nie stanowi transakcji. W ramach każdej iteracji zewnętrznej petli REPEAT występują dwie transakcje do wprowadzania pojedynczego zespołu i do wprowadzania kilku pracowników. Przy wprowadzaniu pracowników transakcję stanowi każda pojedyncza iteracja wewnętrznej pętli REPEAT. Przerwanie programu w sytuacji wprowadzania pracownika spowoduje anulowanie danych ostatnio wprowadzanego pracownika (bez anulowania danych ostatnio wprowadzonego zespołu i wcześniej wprowadzonych pracowników tego zespołu).

```
/* przyklad  p4.p */
REPEAT :
  DO TRANSACTION:
   INSERT ZESP.
  END.
  REPEAT :
    CREATE PRAC.
    PRAC.NRZ = ZESP.NRZ.
    DISPLAY PRAC.NRZ.
    UPDATE PRAC.NRP PRAC.NAZWISKO.
  END.
END.
```

Przykład p5.p dotyczy zmiany domyślnego zakresu transakcji. Niezależnymi transakcjami są:

- blok DO TRANSACTION z instrukcją INSERT

oraz

- blok DO TRANSACTION z pętlą REPEAT, tzn. wszystkimi iteracjami pętli.

W sytuacji przerwania operacji wprowadzania pracownika następuje anulowanie wprowadzenia wszystkich rekordów pracowników aktualnego zespołu, bez anulowania rekordu ostatnio wprowadzanego zespołu.

```
/* przyklad p5.p */
REPEAT :
  DO TRANSACTION:
    INSERT ZESP.
  END.
  DO TRANSACTION:
    REPEAT :
      CREATE PRAC.
      PRAC.NRZ = ZESP.NRZ.
      DISPLAY PRAC.NRZ.
      UPDATE PRAC.NRP PRAC.NAZWISKO.
    END.
  END.
END.
```

3. Subtransakcje

Subtransakcją jest blok zagłębiony w bloku transakcji. Definicja zakresu obowiązywania subtransakcji zawarta jest w tabeli 1. Mechanizm subtransakcji umożliwia wycofywanie modyfikacji zmian bazy danych i wartości zmiennych, z przeniesieniem sterowania w obrębie bloku wewnątrz transakcji.

Przerwanie transakcji poprzez naciśnięcie klawisza przerywającego Ctrl-C powoduje anulowanie bieżącej transakcji i oddanie sterowania z aplikacji do systemu operacyjnego, wywołującego program. W sytuacji awarii systemu faktyczne wycofanie transakcji następuje w momencie ponownego uruchomienia systemu (podczas tzw. automatycznego Crash Recovery), na podstawie zawartości plików z opisem transakcji (Before Image Files).

Jeżeli przerwanie następuje z powodu błędu programowego (tzn. takiego, przy którym sterowanie nadal jest realizowane przez system Progress), może nastąpić wycofanie

Tabela 1
Tabela zakresów obowiązywania transakcji i subtransakcji

Instrukcje	Transakcja nieaktywna	Transakcja aktywna
- DO TRANSACTION - FOR EACH TRANSACTION - REPEAT TRANSACTION - pętle FOR EACH,REPEAT blok procedury, DO ON ERROR, DO ON ENDKEY, zawierające instrukcje modyfikacji danych lub odczytu w trybie wyłącznym	start transakcji	start subtransakcji
- pętle FOR EACH,REPEAT, blok procedury, DO ON ERROR, DO ON ENDKEY, nie zawierające instrukcji modyfikacji danych ani odczytu w trybie wyłącznym	-----	start subtransakcji

subtransakcji. Przykłady błędów programowych:

- próba wstawienia rekordu z nieunikalnym atrybutem kluczowym do pliku z aktywnym indeksem unikalnym, zbudowanym według tego atrybutu,
- kolizja dostępu.

Przy wystąpieniu błędu programowego istnieje możliwość wycofania z bieżącej subtransakcji i wznowienie wykonania dowolnego bloku subtransakcji, w którym wycofywana subtransakcja była zagnieżdżona. Wycofywanie z subtransakcji polega na odtworzeniu (na podstawie Local Before Image Files) poprzedniej zawartości plików danych, jak również wartości zmiennych programu.

Mechanizm transakcji służy odtworzeniu stanu bazy danych. Mechanizm subtransakcji służy odtworzeniu stanu programu. Istnienie transakcji jest warunkiem koniecznym wystąpienia subtransakcji. Transakcja jest jednocześnie największą subtransakcją.

Przykład p6.p ilustruje zastosowanie instrukcji anulowania subtransakcji (wykonanie instrukcji UNDO). Transakcję stanowi każda pojedyncza iteracja zewnętrznej pętli REPEAT. Subtransakcję stanowi każda pojedyncza iteracja wewnętrznej pętli REPEAT. W sytuacji, gdy suma kwot (zmienna KW) kolejno wprowadzanych rekordów do pliku DOCH przekroczy wartość 100, nastąpi anulowanie ostatnio wprowadzanego rekordu z pliku DOCH (również odtworzenie wartości zmiennej KW) i ponowne wznowienie subtransakcji (instrukcja RETRY).

```
/* przyklad  p6.p */
DEF VAR KW LIKE DOCH.KWOTA INITIAL 0 LABEL "SUMA".
BLOK1 :
REPEAT :
  INSERT PRAC.
  BLOK2 :
  REPEAT :
    CREATE DOCH.
    DOCH.NRP =  PRAC.NRP .
    DISP DOCH.NRP.
    UPDATE DOCH.NRT KWOTA.
    KW = KW + KWOTA .
    IF KW > 100 THEN DO :
      MESSAGE " ZBYT DUZO ".
      UNDO BLOK2 , RETRY BLOK2.
    END.
    DISPLAY KW.
  END.
END.
```

Przykład p7.p ilustruje możliwość wycofywania subtransakcji i wznawianie programu od subtransakcji na niższym poziomie zagłębienia. W sytuacji przekroczenia kwoty granicznej, wznowiona zostaje subtransakcja o szerszym zakresie niż zakres bloku bieżącej subtransakcji.

```
/* przyklad  p7.p */
DEF VAR KW LIKE DOCH.KWOTA INITIAL 0 LABEL "SUMA".
BLOK1 :
REPEAT :
  INSERT PRAC.
  BLOK2 :
  REPEAT :
    CREATE DOCH.
    DOCH.NRP =  PRAC.NRP .
    DISP DOCH.NRP.
    UFDATE DOCH.NRT KWOTA.
    KW = KW + KWOTA .
    DISPLAY KW.
    IF KW > 100 THEN DO :
      MESSAGE " ZBYT DUZO ".
      UNDO BLOK1 , RETRY BLOK1.
    END.
  END.
```

```
END.
```

Struktura blokowa przykładu p8.p została zaprojektowana tak, by w sytuacji przekroczenia granicznej kwoty następowało anulowanie wszystkich wprowadzonych dochodów, bez anulowania ostatnio wprowadzonego rekordu pracownika.

```
/* przyklad  p8.p */
DEF VAR KW LIKE DOCH.KWOTA INITIAL 0 LABEL "SUMA".
BLOK1 :
REPEAT :
 INSERT PRAC.
 BLOK12 :
 DO ON ERROR UNDO, RETRY:
   BLOK2 :
   REPEAT :
    CREATE DOCH.
    DOCH.NRP =  PRAC.NRP .
    DISP DOCH.NRP.
    UPDATE DOCH.NRT KWOTA.
    KW = KW + KWOTA .
    DISPLAY KW.
    IF KW > 100 THEN DO :
      MESSAGE " ZBYT DUZO ".
      UNDO BLOK12 , RETRY BLOK12.
    END.
   END.
 END. /* BLOK12 */
END.
```

Instrukcja UNDO [<blok>] powoduje anulowanie zmian dokonanych w plikach danych i zmiennych programowych, jeśli <blok> stanowił subtransakcję. W przeciwnym wypadku, przy braku subtransakcji, instrukcja jest ignorowana.

Instrukcja RETRY [<blok>] powoduje wznowienie wykonania instrukcji <bloku>.

Instrukcja LEAVE [<blok>] powoduje opuszczenie <bloku>.

Instrukcja RETURN [<blok>] powoduje wznowienie wykonania od następnej iteracji, jeśli blok jest pętlą.

Brak etykiety <blok> w wyżej wymienionych instrukcjach oznacza domyślne przyjęcie bieżącego bloku.

4. Obsługa błędów programowych

W sytuacji błędu programowego następuje przeniesienie sterowanania do najbliższego, bardziej zewnętrznego bloku, posiadającego własność obsługi błędu tzw. Error Property. Zgodnie z domyślną własnością obsługi błędu danego bloku następuje wycofanie ewentualnej, bieżącej subtransakcji i wznowienie wykonywania instrukcji bieżącego bloku.

Własność obsługi błędu mogą posiadać instrukcje bloku:

DO ... ON ERROR ...

REPEAT ... ON ERROR ...

FOR EACH ... ON ERROR ...

Instrukcje FOR EACH, REPEAT posiadają domyślną własność obsługi błędu: ON ERROR UNDO, RETRY.

W przykładzie p9.p błąd programowy może wystąpić przy nieudanej próbie odnalezienia rekordu podczas realizacji instrukcji FIND. Błąd w trakcie realizacji FIND powoduje przeniesienie sterowania do początku pętli REPEAT i wznowienie wykonania instukcji bloku zgodnie z domyślnym ustawieniem ON ERROR UNDO, RETRY.

```
/* przyklad  p9.p */
REPEAT :
  PROMPT-FOR TEMAT.NRT.
  FIND TEMAT USING TEMAT.NRT.
  DISP TEMAT.
END.
```

Przykład p10.p może ilustrować zdarzenie obsługi błędu wykonania instrukcji FIND, ze wznowieniem wykonania iteracji, bez odtworzenia wartości zmiennej NR. W programie brak wystąpienia bloku transakcji implikuje brak bloku subtransakcji. Ze względu na brak subtransakcji wykonanie domyślnej instrukcji UNDO nie powoduje żadnych skutków.

```
/* przyklad p10.p */
DEF VAR NR AS INTEGER INITIAL 0 LABEL "LP. ".
REPEAT :
  NR = NR + 1.
  PROMPT-FOR TEMAT.NRT.
  FIND TEMAT USING TEMAT.NRT.
  DISP NR.
  DISP TEMAT.
END.
```

Przykład p11.p może ilustrować zdarzenie obsługi błędu wykonania instrukcji FIND ze wznowieniem wykonania bieżącej iteracji i odtworzeniem wartości zmiennej NR (ze względu na wystąpienie subtransakcji).

```
/* przyklad p11.p */
DEF VAR NR AS INTEGER INITIAL 0 LABEL "LP. ".
REPEAT TRANSACTION:
  NR = NR + 1.
  PROMPT-FOR TEMAT.NRT.
  FIND TEMAT USING TEMAT.NRT.
  DISP NR.
  DISP TEMAT.
END.
```

Przykład p12.p może ilustrować zdarzenie obsługi błędu wykonania instrukcji FIND z wycofaniem sterowania poza blok pętli REPEAT (w konsekwencji wyjście z programu). Nie następuje odtworzenie wartości zmiennej NR ze względu na brak subtransakcji.

```
/* przyklad p12.p */
DEF VAR NR AS INTEGER INITIAL 0 LABEL "LP. ".
REPEAT ON ERROR UNDO , LEAVE:
  NR = NR + 1.
  PROMPT-FOR TEMAT.NRT.
  FIND TEMAT USING TEMAT.NRT.
  DISP NR.
  DISP TEMAT.
END.
```

5. Obsługa zdarzeń generowanych poprzez naciśnięcie klawiszy specjalnych

5.1. Obsługa zdarzenia naciśnięcia ERROR KEY

Klawisz ERROR KEY (definiowany instrukcją ON <kombinacja klawiszy> ERROR.) może być wykorzystywany do symulacji zdarzenia błędu programowego dla celów testowych. Obsługa takiego zdarzenia przebiega tak jak dla zdarzenia błędu programowego, tzn. z uwzględnianiem własności obsługi błędu poszczególnych bloków programu.

Przykład p13.p może ilustrować wycofywanie z zagłębionych subtransakcji. Naciśnięcie ERROR KEY w ramach wewnętrznej pętli REPEAT powoduje anulowanie ostatnio wprowadzanego rekordu pracownika i wznowienie następnej iteracji wewnętrznej pętli. Naciśnięcie ERROR KEY w ramach zewnętrznej pętli REPEAT podczas wstawiania rekordu ZESP powoduje anulowanie wprowadzanego rekordu zespołu i wznowienie iteracji zewnętrznej pętli.

```
/* przyklad  p13.p */
ON CTRL-O ERROR.
REPEAT :
  INSERT ZESP.
  REPEAT :
    CREATE PRAC.
    PRAC.NRZ = ZESP.NRZ.
    DISPLAY PRAC.NRZ.
    UPDATE PRAC.NRP PRAC.NAZWISKO.
  END.
END.
```

Przykład p14.p dotyczy zmiany domyślnego zakresu subtransakcji wycofywanej i wznawianej. Naciśnięcie klawisza ERROR KEY w dowolnym momencie realizacji programu powinno powodować anulowanie ostatnio wprowadzonego rekordu zespołu i wszystkich wprowadzonych rekordów pracowników dla tego zespołu oraz wznowienie iteracji pętli zewnętrznej.

```
/* przyklad p14.p */
ON CTRL-O ERROR.
BLOK1:
REPEAT :
  INSERT ZESP.
  BLOK2:
  REPEAT ON ERROR UNDO BLOK1 , RETRY BLOK1:
    CREATE PRAC.
    PRAC.NRZ = ZESP.NRZ.
    DISPLAY PRAC.NRZ.
    UPDATE PRAC.NRP PRAC.NAZWISKO.
  END.
END.
```

W przykładzie p14.p naciśnięcie klawisza ERROR w trakcie realizacji instrukcji bloku BLOK2, powoduje wycofanie i wznowienie bloku BLOK2 (działanie niezgodne z opisem zawartym w dokumentacji).

5.2. Obsługa zdarzenia naciśnięcia END KEY

Klawisz END KEY (definiowany instrukcją ON <kombinacja klawiszy> ENDKEY) może być wykorzystywany do symulacji zdarzenia błędu programowego z ewentualnym wycofaniem subtransakcji i opuszczeniem bloku. Obsługa takiego zdarzenia przebiega podobnie jak zdarzenia naciśnięcia klawisza ERROR KEY. Różnica polega na zmianie domyślnego schematu działania. W uproszczeniu, po naciśnięciu klawisza realizowane jest wycofanie ewentualnej, bieżącej subtransakcji, po czym w przypadku ERROR KEY następuje wznowienie bieżącego bloku subtransakcji, w przypadku END KEY opuszczenie takiego bloku.

Naciśnięcie klawisza END KEY powoduje realizacje instrukcji związanych z własnością obsługi ENDKEY, należącą do najbliższego, mniej zagłębionego bloku.

Własność obsługi zdarzenia naciśnięcia END KEY mogą posiadać instrukcje bloku:

DO ... ON ENDKEY ...

REPEAT ... ON ENDKEY ...

FOR EACH ... ON ENDKEY ...

Instrukcje FOR EACH, REPEAT posiadają domyślną własność: ON ENDKEY UNDO, LEAVE.

Przykład p15.p może ilustrować wycofywanie z zagłębionych subtransakcji. Naciśnięcie END KEY w ramach wewnętrznej pętli REPEAT powoduje anulowanie ostatnio wprowadzanego rekordu pracownika, wyjście z wewnętrznej pętli oraz wznowienie wykonania iteracji zewnętrznej pętli. Naciśnięcie END KEY w ramach zewnętrznej pętli REPEAT podczas

wstawiania rekordu ZESP powoduje anulowanie wprowadzanego zespołu i opuszczenie zewnętrznej pętli (zakończenie programu).

```
/* przyklad  p15.p */
ON F9 ENDKEY.
REPEAT :
  INSERT ZESP.
  REPEAT :
    CREATE PRAC.
    PRAC.NRZ = ZESP.NRZ.
    DISPLAY PRAC.NRZ.
    UPDATE PRAC.NRP PRAC.NAZWISKO.
  END.
END.
```

Przykład p16.p dotyczy zmiany domyślnego dla END KEY zakresu subtransakcji wycofywanej i wznawianej. Naciśnięcie klawisza END KEY w dowolnym momencie realizacji programu powoduje anulowanie ostatnio wprowadzonego zespołu i wszystkich wprowadzonych pracowników tego zespołu oraz opuszczenie pętli zewnętrznej (zakończenie programu).

```
/* przyklad  p16.p */
ON F9 ENDKEY.
BLOK1:
REPEAT :
  INSERT ZESP.
  BLOK2:
  REPEAT ON ENDKEY UNDO BLOK1 , LEAVE BLOK1:
    CREATE PRAC.
    PRAC.NRZ = ZESP.NRZ.
    DISPLAY PRAC.NRZ.
    UPDATE PRAC.NRP PRAC.NAZWISKO.
  END.
END.
```

5.3. Obsługa zdarzenia naciśnięcia END-ERROR KEY

Klawisz END-ERROR posiada cechy klawiszy END albo ERROR w zależności od etapu realizacji instrukcji interakcyjnych bloku. Klawisz END-ERROR (najczęściej F4) wykorzystywany jest standardowo przy operacjach wprowadzania danych.

Przykład p17.p ilustruje kontekstowy charakter działania klawisza END-ERROR. W sytuacji naciśnięcia klawisza w trakcie instrukcji PROMPT, pierwszej instrukcji interaktywnej w bloku, działanie klawisza END-ERROR jest takie, jak END KEY. Zgodnie z domyślną własnością obsługi klawisza END w instrukcji REPEAT (ON ENDKEY UNDO, LEAVE) następuje unieważnienie subtransakcji i wyjście z bloku pętli (wyjście z programu). Można zinterpretować takie działanie jako świadomą całkowitą rezygnację użytkownika z wprowadzania danych w momencie wyświetlenia formatki.

W sytuacji naciśnięcia klawisza w trakcie instrukcji UPDATE, kolejnej, nie pierwszej instrukcji interaktywnej w bloku, działanie klawisza END-ERROR jest takie, jak ERROR KEY. Zgodnie z domyślną własnością obsługi klawisza ERROR w instrukcji REPEAT (ON ERROR UNDO, RETRY) następuje unieważnienie subtransakcji i wznowienie iteracji (rozpoczęcie subtransakcji). Można zinterpretować takie działanie jako świadomą rezygnację użytkownika z wartości aktualnie wprowadzanych (dotyczących bieżącego wypełnienia formatki) i żądanie ponownego wprowadzania danych (bez całkowitej rezygnacji z operacji wprowadzania danych).

```
/* przyklad  p17.p */
REPEAT :
 PROMPT-FOR PRAC.NRP.
 FIND PRAC USING NRP.
 UPDATE NAZWISKO PRAC.NRZ.
END.
```

Takie kontekstowe działanie klawisza END-ERROR sprawia, że dobrze posługują się nim użytkownicy nie zawsze zdający sobie sprawę z domyślnego, subtransakcyjnego charakteru funkcjonowania systemu.

6. Przykład wyznaczania zakresu transakcji

Prosta programowa metoda określenia zakresu transakcji wynika z reguły: "Zagłębione subtransakcje mogą występować tylko wewnątrz bloku transakcji".

W procedurze istrans.p nastąpi anulowanie wartości zmiennej ISTRANS = NO, jeżeli blok DO ON ERROR ... stanowić będzie subtransakcję, tzn. jeżeli procedura wywołana będzie z bloku zawartego w transakcji. Tylko, gdy blok DO ON ERROR... będzie subtransakcją, instrukcja UNDO dokona odtworzenia poprzedniej wartości zmiennej.

```
/* przyklad  istrans.p */
DEF VAR ISTRANS AS LOGICAL INITIAL YES.
DO ON ERROR UNDO, LEAVE:
  ISTRANS = NO.
  UNDO, LEAVE.
END.
IF ISTRANS
THEN MESSAGE " Transakcja aktywna w {1}".
ELSE MESSAGE " Transakcja NIEaktywna w {1}".
```

Procedura istrans.p wykorzystana została do wyznaczania zakresu transakcji w przykładowym programie p18.p.

```
/* przyklad  p18.p */
RUN istrans.p "zewnetrze".
FOR EACH PRAC:
  DISPLAY PRAC.
  RUN istrans.p "for".
END.
REPEAT :
  INSERT ZESP.
  RUN istrans.p "repeat".
  REPEAT :
    CREATE PRAC.
    PRAC.NRZ = ZESP.NRZ.
    DISPLAY PRAC.NRZ.
    UPDATE PRAC.NRP PRAC.NAZWISKO.
  END.
END.
```

7. Zagadnienie blokad dostępu w trybie pracy wieloużytkownikowej

7.1. Przykłady kontroli dostępu poprzez mechanizmy blokad

W Progress-ie zaimplementowane zostały podstawowe metody kontroli dostępu do rekordów plików danych, tj. dostęp bez nakładania blokady, z nałożeniem blokady w trybie dzielonym, z nałożeniem blokady w trybie wyłącznym. System blokad związany jest z zakresem obowiązywania transakcji w programie. Dla poszczególnych grup instrukcji wprowadzono mechanizmy automatycznego nakładania blokad w zależności od typu instrukcji.

Instrukcje typu FOR EACH, FIND przed wczytaniem rekordu do bufora próbują automatycznie nałożyć blokadę rekordu w trybie dzielonym. Blokada w trybie dzielonym dopuszcza blokowanie w trybie dzielonym przez inne zadanie (tym samym umożliwia odczyt) natomiast uniemożliwia blokowanie w trybie wyłącznym przez inne zadanie.

Instrukcja CREATE tworzy nowy rekord i nakłada blokadę w trybie wyłącznym. Blokada w trybie wyłącznym uniemożliwia nałożenie blokady tego rekordu w jakimkolwiek innym trybie przez inne zadania (tym samym uniemożliwia wykonanie jakichkolwiek operacji).

Instrukcja UPDATE utrzymuje w czasie wyświetlania i modyfikacji pól formatki blokadę w trybie dzielonym, a w momencie zatwierdzenia zmodyfikowanych danych próbuje nałożyć blokadę w trybie wyłącznym. Jeżeli dane nie zostały zmienione, po zakończeniu UPDATE zostaje utrzymana blokada w trybie dzielonym.

Istnieje możliwość wymuszenia trybu dostępu dla niektórych instrukcji (zmiana domyślnego trybu dostępu) poprzez dodanie fraz EXCLUSIVE-LOCK (tryb wyłączny blokady), NO-LOCK (tryb dostępu bez blokady).

Poniższe przykłady mogą ilustrować zdarzenie kolizji podczas próby uzyskania dostępu w trybie wyłącznym w programie p19.p (po realizacji modyfikacji w UPDATE) przy wcześniejszym uzyskaniu dostępu w trybie dzielonym w programie p20.p (po realizacji FOR EACH). Kolizja może wystąpić również przy próbie dostępu w trybie dzielonym w programie p20.p (przy realizacji FOR EACH) w sytuacji założonej blokady danego rekordu w trybie wyłącznym w p19.p (po zakończeniu UPDATE w trakcie wykonania PAUSE).

```
/* przyklad  p19.p */
FOR EACH TEMAT:    /* - próba blokady w trybie dzielonym */
  UPDATE TEMAT.    /* - próba blokady w trybie wyłącznym */
                   /*  po zakończeniu faktycznej modyfikacji  */
  PAUSE.           /* - oczekiwanie i utrzymywana blokada na wyłączność */
END.               /*  po faktycznej modyfikacji */


/* przyklad  p20.p */
FOR EACH TEMAT:    /* - próba blokady w trybie dzielonym */
 DISPLAY TEMAT WITH 1 DOWN .
END.
```

Nieudana próba nałożenia blokady w którymkolwiek trybie powoduje wejście programu w tryb oczekiwania na zwolnienie rekordu oraz wypisanie komunikatu:

```
<Nazwa pliku> in use by <nazwa użytkownika> on tty <nazwa terminala>. Wait or press Ctrl-C to stop.(121)
```

Naciśnięcie Ctrl-C powoduje przerwanie transakcji i wyjście z programu.

Przykład p21.p ilustruje zmianę trybu dostępu w ramach danego bloku subtransakcji (w każdej pojedynczej iteracji pętli REPEAT). Instrukcja FIND (odczyt do bufora rekordu) realizuje próbę nałożenia blokady w trybie dzielonym. Instrukcja
KWOTA = KWOTA - KW (instrukcja modyfikacji rekordu, odpowiedzialna za wystąpienie transakcji) realizuje próbę nałożenia blokady w trybie wyłącznym.

```
/* przyklad  p21.p */
DEF VAR KW AS INTEGER LABEL "ILE".
REPEAT :
  PROMPT-FOR DOCH.NRT.
  FIND FIRST DOCH USING NRT .
  DISPLAY KWOTA.
  SET KW.
  KWOTA = KWOTA - KW.
  DISPLAY KWOTA.
END.
```

Przykład p22.p dotyczy zmiany domyślnego trybu dostępu dla instrukcji FIND. Intencją programisty jest, by od razu przy odczycie następowało zablokowanie w trybie wyłącznym

wyszukanego rekordu pracownika, ze względu na późniejszą operację modyfikacji (próbującą zakładać blokadę na wyłączność).

```
/* przyklad p22.p */
DEF VAR KW AS INTEGER LABEL "ILE".
REPEAT :
  PROMPT-FOR DOCH.NRT.
  FIND FIRST DOCH USING NRT EXCLUSIVE-LOCK.
  DISPLAY KWOTA.
  SET KW.
  KWOTA = KWOTA - KW.
  DISPLAY KWOTA.
END.
```

Domyślnie, w przypadku kolizji dostępu, program przechodzi do oczekiwania na uzyskanie dostępu. Użycie frazy NO-WAIT powoduje, że ewentualny błąd wynikający z kolizji dostępu jest tak samo obsługiwany przez system, jak inne błędy programowe, tzn. zgodnie z własnością obsługi błędu bieżącego bloku.

W przykładzie p23.p zastosowanie frazy NO-WAIT dla instrukcji FIND powoduje w przypadku błędu kolizji dostępu w trybie dzielonym, automatyczne wycofanie i wznowienie subtransakcji (bieżącej iteracji pętli REPEAT).

```
/* przyklad p23.p */
DEF VAR KW AS INTEGER LABEL "ILE".
REPEAT :
  PROMPT-FOR DOCH.NRT.
  FIND FIRST DOCH USING NRT NO-WAIT.
  DISPLAY KWOTA.
  SET KW.
  KWOTA = KWOTA - KW.
  DISPLAY KWOTA.
END.
```

Przykład p24.p pokazuje wykorzystanie zmiennych do sterowania liczbą prób uzyskania dostępu do rekordu. Zmienna definiowana instrukcją z frazą NO-UNDO nie podlega procesowi odtworzenia wartości przy wycofywaniu subtransakcji i może być wykorzystana jako licznik zdarzeń kolizji dostępu.

```
/* przyklad  p24.p */
DEF VAR I AS INTEGER NO-UNDO.
DEF VAR KW AS INTEGER LABEL "ILE".
I = 0.
REPEAT :
  I = I + 1.
  IF I > 5 THEN UNDO , LEAVE .
  PROMPT-FOR DOCH.NRT.
  FIND FIRST DOCH USING NRT EXCLUSIVE-LOCK NO-WAIT.
  DISPLAY KWOTA.
  SET KW.
  KWOTA = KWOTA - KW.
  DISPLAY KWOTA.
  I = 0
END.
```

Program p25.p może posłużyć do ilustracji zdarzenia wzajemnej blokady przy próbach uzyskania dostępu dzielonego, a potem wyłącznego. W sytuacji jednoczesnej realizacji programu p25.p przez dwóch użytkowników możliwe jest jednoczesne uzyskanie dostępu do tego samego rekordu w trybie dzielonym po wykonaniu instrukcji FIND. Ze względu na blokadę w trybie dzielonym przez inne zadanie niemożliwe jest późniejsze uzyskanie dostępu w trybie wyłącznym przez żadnego z użytkowników po zakończeniu wykonania instrukcji UPDATE. Wyjście z zakleszczenia zadań (wzajemnej blokady), spowodowanego automatycznymi próbami założenia blokad w trybie wyłącznym, może odbyć się tylko poprzez przerwanie realizacji jednego z zadań.

```
/* przyklad  p25.p */
REPEAT:
  PROMPT-FOR PRAC.NRP.
  FIND PRAC USING NRP.
  UPDATE NAZWISKO NRZ.
END
```

Przykład p26.p pokazuje możliwość wyświetlania zawartości rekordów niezależnie od stanu blokad w pliku. Wyświetlane dane mogą być nieaktualne.

```
/* przyklad  p26.p */
FOR EACH PRAC NO-LOCK:
  DISPLAY PRAC WITH 1 DOWN .
END.
```

7.2. Zakres obowiązywania blokad w kontekście transakcyjnego przetwarzania danych

Blokada w trybie dzielonym może wystąpić w bloku programu niezależnie od tego, czy blok zagłębiony jest w transakcji. W sytuacji, gdy blok nie ma charakteru transakcyjnego, blokada w trybie dzielonym utrzymywana jest do końca zakresu "widzialności" rekordu. W sytuacji, gdy blok zawiera się w transakcji, blokada utrzymywana jest do późniejszego końca transakcji lub końca zakresu rekordu.

Blokada w trybie wyłącznym występuje zawsze w blokach zawartych wewnątrz transakcji. Blokada w trybie wyłącznym utrzymywana jest do końca transakcji. Jeśli zakres rekordu jest szerszy niż zakres transakcji, tryb wyłączny zamieniany jest na tryb dzielony.

Przykład p27.p może posłużyć do ilustracji zakresu obowiązywania blokad. Transakcję stanowi każda pojedyncza iteracja zewnętrznej pętli REPEAT. W ramach iteracji pętli zewnętrznej występują próby nałożenia blokad:

- nałożenie blokady w trybie wyłącznym (INSERT) na nowo utworzony rekord pliku TEMAT,
- w ramach każdej iteracji wewnętrznej pętli REPEAT:
  - nałożenie blokady w trybie wyłącznym (CREATE) na każdy nowo utworzony rekord pliku DOCH,
  - próby nałożenia blokad w trybie dzielonym (FIND) na każdy wyszukany rekord pliku PRAC.

W ramach transakcji, czyli pojedynczej iteracji zewnętrznej pętli REPEAT może nastąpić zablokowanie pojedynczego rekordu pliku TEMAT oraz wielu rekordów plików PRAC i DOCH. (Maksymalna liczba założonych blokad jest parametrem konfiguracyjnym systemu.)

```
/* przyklad  p27.p */
REPEAT:
  INSERT TEMAT.
  REPEAT :
    CREATE DOCH.
    DOCH.NRT = TEMAT.NRT.
    DISPLAY DOCH.NRT.
    UPDATE DOCH.NRP KWOTA.
    FIND PRAC OF DOCH.
    DISPLAY PRAC.
  END.
END.
```

8. Zakończenie

Progress posiada wiele własności świadczących o przeznaczeniu systemu do tworzenia niezawodnej aplikacji pracującej w środowisku wielodostępnym.

Zaimplementowany został system użytkownków, haseł i praw dostępu na poziomie bazy danych, plików, pól rekordów. Obsługa systemu użytkowników odbywa się z poziomu edytora słownika bazy danych.

W skład systemu wchodzi duża liczba programów narzędziowych.

W trybie pracy wieloużytkownikowej program *proshut* pozwala na zakończenie działania procesu serwera bazy danych lub zakończenie wybranych procesów klientów (odłączenia użytkowników od bazy danych).

Program *promon* pozwala na wszechstronne monitorowanie pracy z wybraną bazą danych. Między innymi możliwe jest śledzenie stanu aktywnych procesów zwiazanych z obsługą bazy danych (tzn. wyświetlenie numeru użytkownika bazy danych, nazwy użytkownika, rodzaju procesu, ewentualnej przyczyny oczekiwania procesu, numeru ewentualnej transakcji odpowiadającej procesowi, numeru procesu w sensie systemu operacyjnego, czasu rozpoczęcia procesu itp.). Ilustracją szczegółowości kontroli może być poziom monitorowania tablicy stanu blokowania (np. wyświetlenie dla danego procesu numerów blokowanych rekordów, typów założonych blokad, a nawet flag określających szczegółowe cechy blokad (typu LIMBO LOCK - blokada utrzymana z powodu zakresu transakcji, przy zakończeniu zakresu widzialności rekordu, UPGRADE, QUEUE - oczekiwanie procesu na możliwość podniesienia poziomu blokady z trybu dzielonego na wyłączny itp.).

Program *probkup* umożliwia wykonanie poprawnej kopii bazy danych w trakcie normalnej pracy systemu (tzw. Online Backup). Fakt ten pozwala na tworzenie aplikacji pracujących w cyklu całodobowym.

System Progress wyposażony jest w programy narzędziowe *probkup* i *prorest* umożliwiające wykonanie całościowej albo inkrementacyjnej kopii bazy danych i odtworzenie bazy danych.

Program *prostrct* umożliwia utworzenie i administrowanie wielowolumenową bazą danych.

Program *proutil* między innymi może służyć do włączenia mechanizmu śledzenia transakcji na rzecz ewentualnego odtworzenia utraconych zmian bazy danych dokonanych przez wypełnione transakcje. Domyślnie system Progress tworzy dla danej bazy danych plik Before Image, służący do odtworzenia zawartości bazy danych w sytuacji wycofywania się z niewypełnionych transakcji (mechanizm Roll Back). Przy użyciu *proutil* można przełączyć system

w stan zapamiętywania obrazu zrealizowanych transakcji w plikach After Image. W przypadku uszkodzenia fizycznego pliku bazy danych, za pomocą programu *rfutil* z użyciem wcześniejszej kopii bazy danych i pliku After Image, można odtworzyć utracone zmiany w bazie (mechanizm Roll Forward).

Program *proutil* może spełniać dodatkowe funkcje np. wybór fizycznej lokacji plików z obrazami transakcji (np. różnej od lokacji plików bazy danych), kontrola rozmiaru plików z obrazami transakcji, przebudowa indeksów, kontrola wewnętrznej spójności fizycznego pliku bazy danych.

W skład systemu Progress-a może opcjonalnie wchodzić pakiet FAST TRACK zawierający generatory: formatek (Screen Painter), menu (Menu Generator), raportów (Report Writer), prostych aplikacji (Query By Form generator).

Progress ma swoje implementacje dla popularnych systemów operacyjnych DOS i Novell Netware, Unix, VMS, BTOS/CTOS. Dla serwera bazy danych przeznaczonego dla środowiska Unix-owego charakterystyczna jest efektywna, wielowątkowa architektura. Serwer bazy danych systemu Progress 6 może wykorzystywać architektury wieloprocesorowe (np. architektury komputerów produkowanych przez firmę Sun Microsystem, opartych na procesorach rodziny SPARC). W przypadku typowych lokalnych sieci komputerowych możliwa jest konfiguracja (z wykorzystaniem NetBios-a) z uruchomionym serwerem bazy danych na dedykowanym komputerze. Dla sieci Novell Netware 3.11 możliwe jest uruchomienie procesu serwera bazy danych na File Serwerze (serwer bazy danych w postaci modułów NLM).

Warto wymienić niektóre charakterystyczne cechy środowiska systemu Progress, w którym aplikacja jest bezpośrednio tworzona:

- Łatwy w obsłudze edytor słownika bazy danych. Instrukcja realizująca dostęp do słownika umożliwia administrowanie bazą danych poprzez ustalanie struktury plików oraz struktury i aktywności indeksów, określanie praw dostępu, więzów integralności dotyczących powiązań między plikami, analizowanych przy wstawianiu rekordu, definiowanie warunków spełnianych przez usuwany rekord, ustalanie domyślnego formatu wyprowadzania pól i ich wartości początkowej itp. Poprzez instrukcję realizującą dostęp do słownika bazy możliwy jest import danych w formatach:
  - ASCII,
  - systemów baz danych ORACLE, Rdb, RMS, dBase,
  - arkuszy kalkulacyjnych DIF, SYLK,
  - niektórych edytorów tekstowych .
- Wygodny edytor tekstowy.
- Łatwy dostęp do krótkiego, bezkontekstowego systemu pomocy.

Język Progress-a ze względu na domyślne działania związane z obsługą ekranu i zakresami transakcji w znacznym stopniu upraszcza proces tworzenia programu.

Progress w wersji 6 umożliwia tworzenie aplikacji znakowych z przeznaczeniem do eksploatacji w instalacjach terminalowych.

Warto zwrócić uwagę na zapowiedź systemu w wersji 7 dla graficznego środowiska Windows i języka opartego na zdarzeniowej koncepcji sterowania.

Wymienione cechy Progress-a pozwalają przewidywać rozwój zainteresowania systemem.

9. Dodatek

Struktura przykładowej bazy danych

Struktura pliku PRAC opisującego pracownika		
Nazwa	Typ	Opis
NRP	INTEGER	numer pracownika, klucz, atrybut o wartościach unikalnych
NAZWISKO	CHARACTER	
NRZ	INTEGER	numer zespołu danego pracownika

Struktura pliku ZESP opisującego zespół		
Nazwa	Typ	Opis
NRZ	INTEGER	numer zespołu, klucz
NAZWAZ	CHARACTER	nazwa zespołu
KIERZ	CHARACTER	nazwisko kierownika zespołu

Struktura pliku TEMAT		
Nazwa	Typ	Opis
NRT	INTEGER	numer tematu, klucz
NAZWAT	CHARACTER	nazwa tematu
KIERT	CHARACTER	nazwisko kierownika tematu

Struktura pliku DOCH opisującego dochody i określającego związek pomiędzy pracownikami i tematami			
Nazwa	Typ	Opis	
NRT	INTEGER	numer tematu	złożenie pól NRT i NRP stanowi klucz
NRP	INTEGER	numer pracownika	
KWOTA	INTEGER	kwota, jaką zarobił pracownik w danym temacie	

LITERATURA

[1] PROGRESS Programming Handbook
[2] PROGRESS Reference
[3] PROGRESS System Administration

Recenzent: Dr hab. inż. Stanisław Wołek

Wpłynęło do redakcji 8 grudnia 1993r.

Abstract

The article presents features of transaction data processing of the database management system Progress 6. The article includes definitions of transaction and subtransaction. Considering of sample programs lets to explain how Progress backs out partially completed transactions with undoing all database changes after a system crash. The transaction system let to keep a database integrity. The relation between scopes of transactions and a block structure of Progress programs is shown in this article.

The article includes the definition of subtransaction and explanations of Progress error handling. A subtransaction (included in a transaction) during backing out, after a program error restores database and variables to values they had at the beginning of this subtransaction. Simply a subtransaction may let to restore program execution to the one of previous states.

The article presents some features of multi-user environment of system Progress such as locking data access (shared, exclusive, no-lock) and relations between scopes of locking data and scopes of transactions. Sample programs lets to explain how Progress handles program error after detecting a data access collision.

The article provides other some information such as system Progress configurations and features of some Progress software tool programs.