Zanurzony SQL (Embedded SQL)

Język SQL jest przeznaczony dla dostępu do baz danych. Dostęp może być urzeczywistni się w dwóch następnych trybach:

- Interaktywny dostęp przez ISQL
- Bezpośredni dostęp z aplikacji użytkownika.

Taka dualność tworzy następnie zalety:

- Wszystkie możliwości instrukcji interaktywnych SQL są dostępnie przy oprogramowaniu aplikacji.
- W trybie interaktywnym jest możliwe usuwać błędy głównych algorytmów obrabiania informacji, które potem można wstawiać do aplikacji użytkownika.

Standard SQL-2 jest językiem wszystkich baz danych, ale on nie jest językiem programowania. On nie zawiera taki instrukcji sterowania programem jako „IF …THEN", „DO… WHILE" i in. Oprócz tego język SQL nie może definiować wewnętrznie zmienni, analizować warunki logiczne oraz zmieniać przebieg programu w zależności od tych warunków. W ogóle SQL jest językiem, który wykorzysta się tylko dla sterowania bazami danych.

Dla tworzenia aplikacji wykorzystają się inne języki (np. C++, Pascal, ADA) oraz systemy oprogramowania (np. PowerBuilder, Delphi). W tych językach oraz systemach konstrukcje języka SQL są używane wewnątrz konstrukcji języka głównego. Język główny nazywa się językiem – gospodarzem (host language), a wewnętrzna konstrukcja SQL nazywa się zanurzonym SQL (embedded SQL).

Technologia przetwarzania programu w tych wypadkach zakłada istnienie prekompilatora SQL, który zamieni instrukcje SQL na sekwencje instrukcji języka głównego, odwołujących do interfejsu bazy danych. Interfejs bazy danych ma nazwisko: „sterownik" (driver).

Zanurzony SQL wykorzysta dla zapytań oraz dla manipulacji danymi mechanizm transakcji.

Standard ISO zawiera następujące operatory zanurzonego SQL:

- CONNECT – ten operator połączy aplikację do bazy danych. Ten operator odpowiada opcji CONNECT serwera SQL.
- DISCONNECT – wyłączy aplikację użytkownika od bazy danych. Ten operator odpowiada opcji DISCONNECT serwera SQL.
- EXECUTE – ten operator uruchomi zapamiętaną procedurę, w niektórych realizacjach skrót „EXEC" jest niezbędny dla uruchomienia wszystkich operatorów zanurzonego SQL (np. C++) .
- INSERT – wstawia do tablicy bazy danych rekordy.
- DELETE – usuwa rekordy z tabeli bazy danych.
- UPDATE – modyfikuje rekordy tabel bazy danych.
- SELECT – czyta jeden rekord z tabel(i) bazy danych.
- COMMIT – ten operator pozwala urzeczywistni wszystkie zmiany w bazie danych.
- ROLLBACK - ten operator spędza wycofywanie wszystkich zmian w bazie danych.
- DECLARE – ten operator definiuje **kursor** bazy danych. **Kursor to jest wskaźnik w zbiorze ostatecznym zapytania SQL do bazy danych.** Ten

zbiór jest otrzymany za dopomogą instrukcji SQL „SELECT”, która jest opisana w operatorze DECLARE. Kursor pozwala przesunięcie po rekordach bazy danych.

- OPEN – ten operator uruchomi zapytanie SELECT, które jest opisane w operatorze DECLARE. Wskaźnik kursora po operatorze OPEN jest ustalony przed pierwszym rekordem tabeli zbioru ostatecznego.
- FETCH – ten operator czyta dani w zmieni aplikacji. Po operatorze FETCH wskaźnik kursora jest ustalony do następnego rekordu zbioru ostatecznego.
- CLOSE – ten operator zamyka kursor.

Przykład realizacji zanurzonego SQL w języku C++.
Ten program modyfikuje rekord tabeli EMPLOYEE bazy danych BAZA_FIRMY.

```
/*The following is a very simple example of an Embedded SQL program.*/
#include <stdio.h>
EXEC SQL INCLUDE SQLCA;
main()
{
  db_init( &sqlca );
  EXEC SQL WHENEVER SQLERROR GOTO error;
  EXEC SQL CONNECT DATABASE “baza_firmy” USER "dba"
       IDENTIFIED BY "sql";
  EXEC SQL UPDATE employee
    SET emp_lname =    'Plankton'
    WHERE emp_id = 195;
  EXEC SQL COMMIT;
  EXEC SQL DISCONNECT;
  db_fini( &sqlca );
  return( 0 );
  error:
  printf( "update unsuccessful -- sqlcode = %ld.n",
  sqlca.sqlcode );
  return( -1 );
}
```

W tym przykładzie klauzula

```
EXEC SQL INCLUDE SQLCA;
```

połączy definicje wszystkich zmiennych oraz struktur SQLCA (SQL Communication Area) do programu. SQLCA to jest obszar pamięci , który jest wykorzystywany dla :

- informacji pro połączenia z bazą danych;
- otrzymania statystyki opracowania transakcji;
- otrzymania komunikatów pro błędy z bazy danych.

Funkcja

```
db_init( &sqlca );
```

inicjalizuje SQLCA dla roboty z bazą danych.
Klauzula

```
EXEC SQL WHENEVER SQLERROR GOTO error;
```
tworzy pułapkę dla błędów, które mogą być przy realizacji transakcji.

Przykład programu z jednorekordowym zapytaniem SQL.

Ten program zaprasza tabelny numer pracownika oraz wyświetli ego dani.

```
#include <stdio.h>
#include <stdlib.h>
EXEC SQL INCLUDE SQLA;
main ()
{
EXEC SQL BEGIN DECLARE SECTION;
        char   staff_no[6];          /* tabelny numer pracownika
        char   first_name[16];       /* nazwisko pracownika*/
        char   last_name[16];        /*imię pracownika*/
        char   address[51];          /* adres firmy */
        char   branch_no[4]          /* numer filii firmy */
EXEC SQL END DECLARE SECTION;

/* Połączenie do bazy danych*/
EXEC SQL CONNECT DATABASE "baza_firmy" USER "dba"
        IDENTIFIED BY "sql";
If (sqlca.sqlcode <0)       exit (-1);

/* zapytania tabelnego numeru pracownika */
printf ("Enter staff number: ");
scanf("%s", staff_no);
EXEC SQL SELECT fname, lname, address, bno
        INTO :first_name, :last_name, :address, branch_no
        FROM staff
        WHERE sno = :staff_no;

/* analiza zakończenia programu oraz wyprowadzania rezultatów */
if (sqlca.sqlcode = = 0) {
        printf (" First name: %s\n",              first_name);
        printf (" Last name: %s\n",               last_name);
        printf (" Address: %s\n",                 address);
        printf (" Branch number: %s\n",                 branch_no);
}
else    if (sqlca.sqlcode = = 100)
        printf ("No staff member with specified number\n");
else
        printf ("SQL error %d\n, sqlca.sqlcode);

/* wyłączenie od bazy danych */
EXEC SQL DISCONNECT;
```

}

Procedury zapamiętane (Stored Procedure) oraz funkcje

Procedura zapamiętana (procedura przechowywana) to: zbiór instrukcji SQL zapamiętany w bazie danych w postaci skompilowanej, przeznaczony do wielokrotnego wykorzystania. Dzięki prekompilacji, procedury zapisane wykonywane są znaczne szybciej niż inne polecenia SQL-owe. Dodatkowo kod procedury znajduje się wewnątrz bazy danych i nie trzeba przesyłać go przez sieć. W związku z tym, w przypadku dużych procedur(a taki zwykle spotykamy w praktyce), następuje zmniejszenie generowanego ruchu w sieci.

Procedury zapamiętane mogą być wywoływane przez użytkowników łączących się z bazą danych jako parametryzowane zapytanie SQL'owe.

Pisanie procedur zapamiętanych ściśle związane jest z możliwościami samej bazy danych. Standardowy zbiór instrukcji SQL nie pozwala na proceduralne programowanie. Dlatego język SQL został rozszerzony o wyrażenia pozwalające na tworzenie warunków, pętli, deklarację zmiennych. Niestety, różni producenci baz danych wprowadzili różne rozszerzenia języka SQL. Mamy więc T-SQL, PL/SQL, embedded SQL, itd.

Tworzenie procedury zapamiętanej:

CREATE PROCEDURE [*owner.*] *procedure-name* ([*parameter*,...]) ... { ... *compound-statement* }
gdzie:
owner – nazwa właściciela procedury;
procedure-name – nazwa procedury;
parameter – parametr procedury;
compound-statement – ciało procedury.

Każdy parametr procedury musi mieć następny format:

parameter_mode parameter-name data-type
gdzie:
parameter_mode – rodzaj parametru procedury, który może być IN|OUT|INOUT
IN – wejściowy parametr;
OUT – wyjściowy parametr;
INOUT – wejściowy/wyjściowy parametr;
parameter-name – nazwa parametru,
data-type – typ danych parametru.

Przykład: Procedura zliczająca ilość pracowników z tabeli ***Staff***, których pensja mieści się w wyznaczonych widełkach (danych przez parametry ***n1, w2*** typu integer). Rezultat przekazany jest do parametru ***tek_count***.

```
CREATE procedure "DBA".PROBA (in n1 integer,in w2 integer, out tek_count
integer) /* parameters,... */
on exception resume /*umożliwia wykonanie w wypadku błędu*/
begin
  select(select "count"(Cno)
     from Staff
```

```
    where Salary not between n1 and w2) into tek_count
end
```

Wywołanie tej procedury:

```
%
% % Ensure our test variables do not already exist
 SET OPTION On_error = 'continue';
CREATE VARIABLE "tek_count" integer;
% % Execute the procedure
CALL "DBA"."PROBA"(6000, 12000, "tek_count" );
% % View the output variables
SELECT "tek_count" FROM DUMMY;
%
```

Rezultat procedury zapisze się do systemowej tabeli DUMMY, która ma tylko jedną kolumnę. Ostatnia instrukcja SELECT czyta wyjściowy parametr ***tek_count.*** Procedury mogą bez ograniczeń wywoływać inne procedury oraz funkcje.

Format instrukcji dla stworzenia funkcji ma postać:

```
CREATE FUNCTION [creator.]"func_name" (parameters,... )
RETURNS /* return type */
BEGIN
        DECLARE /*return name */ /* return type */;
         ...
        compound-statement
        RETURN (return name);
END
```

gdzie
creator – imię użytkownika, właściciela funkcji;
func_name – imię funkcji;
parameter – parametr funkcji w formacie: IN parameter-name parametr-type
compound-statement - tekst z kodami funkcji;
return type – typ parametru, który funkcja musi wrócić;
/*return name */ /* return type */ - imię funkcji oraz typ tego imię.

Przykład: Funkcję, która dokonuje połączenia ciągów znaków przekazanych jej jako parametry

```
CREATE function fullname(in firstname char(30),in lastname char(30))
returns char(61)
begin
  declare "name" char(61);
  set "name"=firstname||' '||lastname;
  return("name")
end
```

Skonstruujemy zapytanie SQL, które będzie zawierać imię i nazwisko klientów tablicy CUSTOMER oraz wykorzystać funkcję ***fullname***:

```
SELECT fullname( "firstname", "lastname" )
FROM Customer;
```

Odtwarzanie bazy danych

Odtwarzanie bazy danych (Recovery Data Base) to jest proces przywrócenia je do poprawnego stanu zgodnie z postulatem ASOT.
Baza danych wykorzystuje się dwa typy pamięci:

- ulotną
- trwałą.

Wyróżniają się następny typy awarii, które powodują konieczność odtwarzania danych:

- uszkodzenia pamięci ulotnej;
- uszkodzenia pamięci trwałej.

Odtwarzanie bazy danych przy uszkodzeniach pamięci ulotnej

Pamięć ulotna jest pamięć operacyjna, a pamięć trwała jest pamięć dyskowa. Pomiędzy tymi dwoma typami pamięci musi być gwarantowane odwzorowanie. Pamięć ulotna zawiera specjalne bufory I/O mające na celu zmniejszenie liczby kosztownych operacji dostępu do wolnej pamięci dyskowej. Bloki dyskowe najczęściej są sprowadzone do bufora i pamiętane tam jako strony w buforze. Informacje o tym, jakie bloki dyskowe znajdują się w buforze , są przechowywane w tablicy bufora (lookaside table). Te bloki mogą zawierać informacje o danych BD. Kiedy w buforze zabraknie miejsca jakaś strona będzie zrzucona na dysk zgodnie ze strategią LRU, według której na dysk zostaje strona najmniej używana.
Wszystkie strony w buforze zawierają rezultaty aktualizacji transakcji współbieżnych w bazie danych oraz zostają w buforze aż:

- zostaną usunięte w wyniku zastosowania strategii LRU,
- zostaną zapisane na dysku po zadanym czasie.

Mówią, że strona w buforze jest „brudna”, jeżeli zastała uaktualniona przez transakcję od czasu ostatniego zapisu na dysk. Brudne strony mogą pozostawać w buforze jeszcze długo po tym, jak uaktualniające je transakcje zostały zaakceptowane.
Przypuścimy, że wystąpił zanik napięcia. Jeżeli aktualizowane strony nie były zapisywane na dysk, to będzie stracona informacja o aktualizacjach.
Z kolei zapisywanie strony na dysk zaraz po jej aktualizacji powoduje małą wydajność SZBD. Wydajność SZBD w tym przypadku byłaby wyznaczona wydajnością dysku magnetycznego. Oprócz tego, natychmiastowe zapisywanie danych może doprowadzić do niespójności, jeśli okazałoby się później, że transakcja, która dokonała modyfikacji, nie została zaakceptowana. Transakcja zostanie więc odrzucona, ale zmiany przez nią wprowadzone pozostaną. SZBD musi zapewnić atomowość i trwałość transakcji oraz dostarczyć mechanizm wycofania transakcji (wycofania zmian przez nie zaakceptowaną transakcję).
Zasadą odtwarzania bazy danych jest przechowywanie zbytecznych danych, które odwzorowują konsekwentność operacji aktualizacji transakcji. Te dani są

w dzienniku transakcji (pliku logu), który zawiera historię przetwarzania wszystkich transakcji od momentu ostatniego zapisywania dziennika na dysk.
Istnieją dwa warianty realizacji dziennika transakcji:

1. Realizacja oddzielnego logu dla każdej transakcji
2. Realizacja ogólnego logu dla wszystkich transakcji.

W pierwszym wariancie każda transakcja chroni historię zmian bazy danych przez operacji tej samej transakcji. Lokalne logi mogą być wykorzystywane dla indywidualnych wycofań oddzielnych transakcji. Oprócz logów lokalnych w tym wariancie jest potrzebny log ogólny dla wyznaczenia kolejności oraz parametrów czasowych wszystkich transakcji. Ten wariant pozwoli realizować szybkie wycofanie oddzielnych transakcji, ale potrzebuje jednoczesnego podtrzymywania wielu logów. Wariant drugi realizuje wykorzystywanie tylko jednego logu. Ten dziennik zawiera informacje pro wszystkie transakcje.
Przykład fragmentu dziennika jest pokazany w tabl.14.
Tabl.14.

Num _Rec	Id_t r	Time	Operacja	Object	Old_ m	New_ m	PPtr	NPtr
1	**T1**	**10:12**	**Start**				**0**	**2**
2	**T1**	**10:13**	**Update**	**Staff SL21**	**20**	**30**	**1**	**8**
3	**T2**	**10:14**	**Start**				**0**	**4**
4	**T2**	**10:16**	**Insert**	**Staff SG37**		**12**	**3**	**5**
5	**T2**	**10:17**	**Delete**	**Staff SA9**	**90**		**4**	**6**
6	**T2**	**10:17**	**Update**	**Dzial Dz16**	**12**	**13**	**5**	**9**
7	**T3**	**10:18**	**Start**				**0**	**11**
8	**T1**	**10:18**	**Commit**				**2**	**0**
9	**T2**	**10:19**	**Commit**					
10	**T3**	**10:20**	**Insert**	**Dzial Dz19**		**40**	**7**	**11**
11	**T3**	**10:21**	**Commit**				**10**	**0**

Przeznaczenie atrybutów kolumn tabl. 14 jest następujące:

- „Num_Rec” - identyfikator rekordu logu;
- „Id_tr” – identyfikator transakcji;
- „Time” – znacznik czasowy operacji transakcji;
- „Operacja” – typ operacji transakcji;
- „ Object” – identyfikator obiektu danych, który był wykorzystywany przez operację transakcji;
- „ Old_ m” – kopia obiektu danych do realizacji operacji transakcji;
- „New_ m” - kopia obiektu danych po realizacji operacji transakcji;
- „PPtr” – wskaźnik na poprzedni rekord w logu, który zawiera operację tej samej transakcji;

- „NPtr” - wskaźnik na następny rekord w logu, który zawiera operację tej samej transakcji.

Plik logu jest rozmieszczony w buforu pamięci ulotnej. Rekordy buforu logu mogą być wykorzystywane dla wycofania oddzielnych transakcji przez instrukcje ROLLBACK. Natomiast w przypadkach awarii pamięci ulotnej mogą być stracone wszystkie rekordy logu. W celu odtwarzania bazy danych w tych przypadkach, trzeba mieć kopię logu na dysku oraz ta kopia musi być uzgodniona ze stanem bazy danych. Istnieje protokół WAL(Write Ahead Log), który reglamentuje kolejność zapisywania bufora logu na dysk. W skróci go można określić stwierdzeniem: „zanim zapiszesz coś na dysk najpierw zapisz na dysk bufor logu”. Innymi słowy przed zapisywaniem obiektu bazy danych na dysk, najpierw trzeba zapisać log. Sama baza danych może nie zawierać wszystkich zmian, które odwzorowuje log, ale te zmiany można wprowadzić przez uruchomianie wyznaczonych transakcji w logu. Bufor logu jest zapisywany na dysk w dwóch przypadkach:

- przy akceptacji jakikolwiek transakcji przez operację COMMIT;
- przy zdarzeniu licznika czasu (timer).

W przypadku awarii pamięci ulotnej menedżer odtwarzania bazy danych musi analizować stan transakcji w logu, która zapisywała rekordy do bazy w moment awarii. Jeżeli transakcja wykonała operację akceptacji (commit) , to menedżer odtwarzania bazy danych musi powtórzyć wszystkie je operacje z początku – wykonać operację REDO. REDO polega na wczytaniu do buforu strony z danymi bazy danych, potworzeniu jej aktualizacji i zapisywaniu tej strony na dysk. Nowa wartość zapisywanego rekordu do bazy jest pobierana z rekordu logu.

Kiedy ta transakcja nie była zaakceptowana, to menedżer odtwarzania bazy danych musi wycofać wszystkie je rezultaty – od końca do początku – wykonać operację UNDO. UNDO polega na wczytaniu do buforu danej strony z danymi bazy danych, „odkręceniu” jej aktualizacji i zapisywaniu strony na dysk. Poprzednia wartość danych jest pobrana z rekordu logu.

Na fig.46 jest pokazany przykład wykorzystywania logu dla odtwarzania bazy danych. Stan bazy danych moment tlpc (time of last physical consistency) jest zapisany na dysku. W moment tf (time failure) wystąpiła awaria. Dla odtwarzania bazy danych trzeba odczytać wszystkie strony pamięci trwałej, które odwzorowują stan bazy danych na moment tlpc, do bufora pamięci operacyjnej oraz odczytać plik logu. W logu są historie transakcji :T1 – T5. Dla odtwarzania bazy danych trzeba realizować następne czynności:

- Transakcja T1 była zatwierdzona do momentu tlpc, je rezultaty są w pamięci trwałej bazy danych, dlatego operacji odtwarzania dla tej transakcji nie są potrzebne.
- Część rezultatów transakcji T2 są w pamięci trwałej. Żeby dostarczać postulat atomowości (ASOT) trzeba powtórzyć wszystkie operacji T2 z początku (wykonać operację REDO).

- Część rezultatów transakcji T3 są w pamięci trwałej, ale ta transakcja nie została zatwierdzona. Dla transakcji T3 trzeba odkręcić je operacji od końca do samego początku (wykonać operację UNDO).
- Rezultaty transakcji T4 są nieobecne w pamięci trwałej. Dla transakcji T4 trzeba wykonać operację REDO, bo do momentu awarii została już zatwierdzona.
- Dla transakcji T5 nie trzeba żądnych czynności odtwarzania, bo rezultaty tej transakcji są nieobecne w pamięci trwałej.

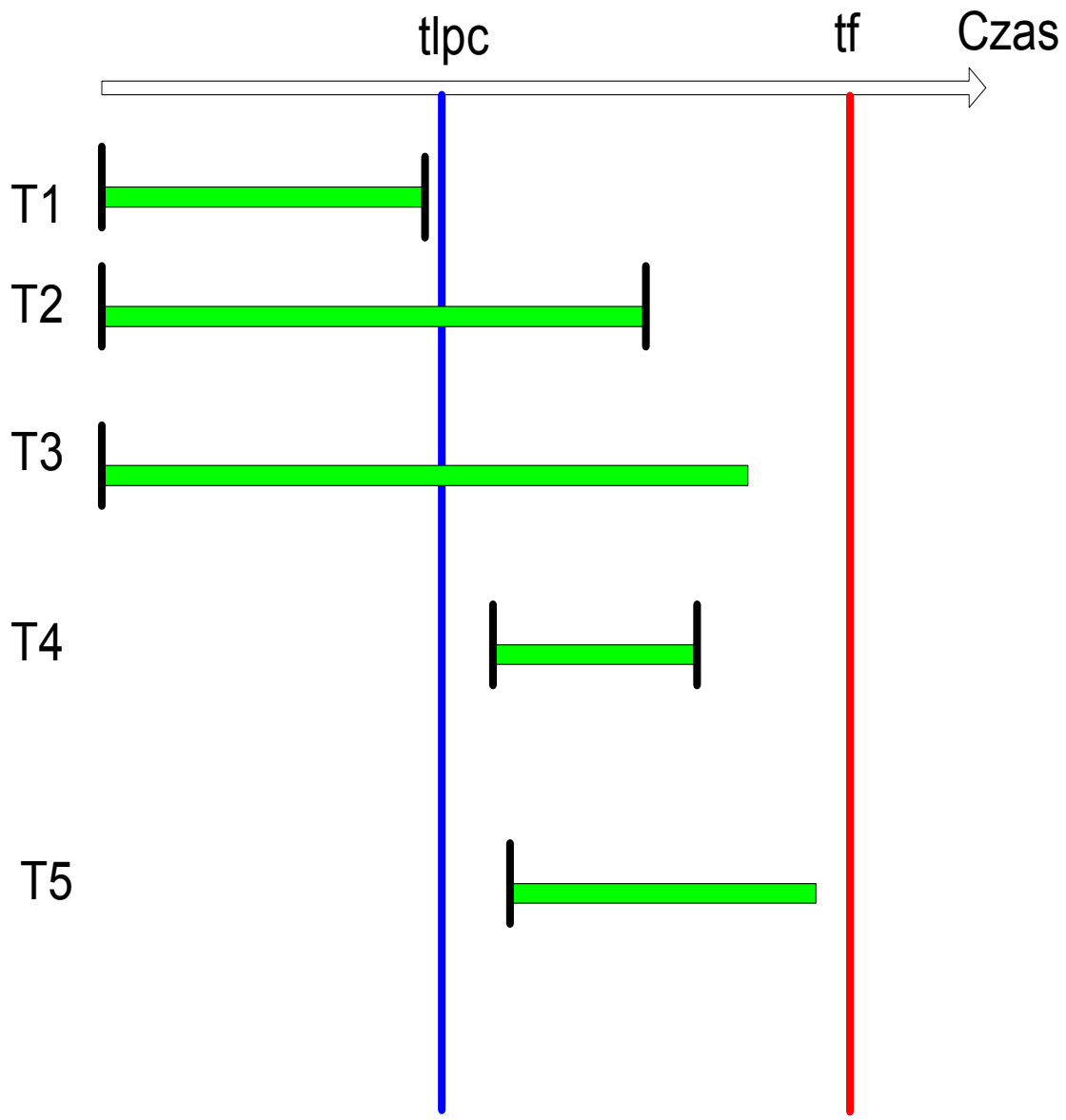

Fig 46

Odtwarzanie bazy danych przy uszkodzeniach pamięci trwałej

Dla odtwarzania bazy danych przy uszkodzeniach dysków trzeba mieć zarchiwowane kopie bazy danych oraz dziennik transakcji od momentu ostatniej archiwacji. Odtwarzanie poczyna się z kopiowania danych z je kopii. Potem dla wszystkich transakcji, które są zaakceptowane realizuje się operacja REDO.