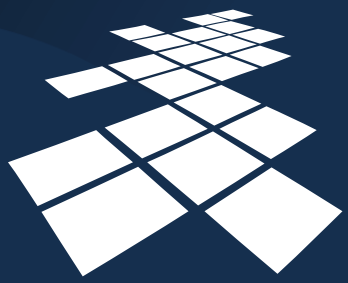
Systemy baz danych

Algorytmy zarządzania współbieżnym wykonywaniem transakcji
część I

Wykład przygotował:
Tadeusz Morzy



UCZELNIA
ONLINE

BD – wykład 9

Celem wykładu jest przedstawienie i omówienie podstawowych algorytmów zarządzania współbieżnym wykonywaniem transakcji. Rozpoczniemy od przedstawienia algorytmów blokowania. Omówimy podstawowy algorytm blokowania – algorytm blokowania dwu-fazowego. Następnie przedstawimy zjawisko zakleszczenia i omówimy podstawowe algorytmy rozwiązywania zakleszczenia. Na zakończenie wykładu, przedstawimy i omówimy problem duchów.

Klasyfikacja algorytmów

Algorytmy zarządzania współbieżnym wykonywaniem transakcji możemy sklasyfikować następująco:

1. **algorytmy blokowania** - uszeregowanie transakcji wynika z kolejności uzyskiwanych blokad (algorytm blokowania dwufazowego – 2PL)
2. **algorytmy znaczników czasowych** - uszeregowanie transakcji wynika z wartości znaczników czasowych związanych z transakcjami
3. **algorytmy optymistyczne** - walidacja poprawności uszeregowania

Algorytmy zarządzania współbieżnym wykonywaniem transakcji możemy sklasyfikować następująco:

algorytmy blokowania - uszeregowanie transakcji wynika z kolejności uzyskiwanych blokad (algorytm blokowania dwufazowego – 2PL);

algorytmy znaczników czasowych - uszeregowanie transakcji wynika z wartości znaczników czasowych związanych z transakcjami;

algorytmy optymistyczne - walidacja poprawności uszeregowania.

Algorytmy blokowania (1)

- Blokada jest zmienną skojarzoną z każdą daną w bazie danych, określającą dostępność danej ze względu na możliwość wykonania na niej określonych operacji
- Ogólnie, z każdą daną mamy skojarzoną jedną blokadę

Ze względu na proces blokowania, dane w bazie danych mogą występować w jednym z trzech stanów:

– dana nie zablokowana (***0)***

– dana zablokowana dla odczytu ***R*** (współdzielona ***S***)

– dana zablokowana dla zapisu ***W*** (wyłączna ***X***)

Podstawową grupą algorytmów zarządzania współbieżnym wykonywaniem transakcji są algorytmy blokowania. Algorytmy te opierają się na mechaniźmie blokad zakładanych przez transakcje. Blokada jest zmienną skojarzoną z każdą daną w bazie danych, określającą dostępność danej ze względu na możliwość wykonania na niej określonych operacji. Ogólnie, z każdą daną mamy skojarzoną jedną blokadę.

Ze względu na proces blokowania, dane w bazie danych mogą występować w jednym z trzech stanów:

- dana nie zablokowana (***0)***,

- dana zablokowana dla odczytu ***R*** (współdzielona ***S***),

- dana zablokowana dla zapisu ***W*** (wyłączna ***X***).

Algorytmy blokowania (2)

- System zarządzania bazą danych musi realizować trzy dodatkowe operacje na bazie danych:
  - Blokowanie danej x do odczytu (LR(x))
  - Blokowanie danej x do zapisu (LW(x))
  - Odblokowanie danej x (UNL(x))
- Operacje blokowania muszą poprzedzać wykonanie operacji odczytu oraz zapisu danej

Wprowadzenie blokad pociąga za sobą konieczność modyfikacji zbioru operacji wykonywanych na bazie danych. Podstawowy zbiór operacji transakcji (odczyt, zapis, zatwierdzenie, wycofanie) rozszerzamy o 3 dodatkowe operacje:

- Blokowanie danej x do odczytu (LR(x)),
- Blokowanie danej x do zapisu (LW(x)),
- Odblokowanie danej x (UNL(x)).

Operacje blokowania muszą poprzedzać wykonanie operacji odczytu oraz zapisu danej. Z każdą operacją blokowania danej X skojarzona jest operacja odblokowania danej X.

Kompatybilność blokad

- Dwie blokady są kompatybilne jeżeli mogą być założone na tej samej danej przez dwie różne transakcje

Blokada żądana \ Blokada uzyskana	R	W
R	✓	-
W	-	-

Wprowadzimy obecnie pojęcie kompatybilności blokad. Mówimy, że dwie blokady są kompatybilne jeżeli mogą być założone na tej samej danej przez dwie różne transakcje. Macierz kompatybilności blokad przedstawiono na slajdzie. Jak łatwo zauważyć, kompatybilne są blokady do odczytu, natomiast pozostałe kombinacje blokad (RW, WR, WW) są niekompatybilne.

Systemy baz danych

Konwersja blokad

- Transakcja posiadająca blokadę określonego typu na danej może dokonać jej konwersji w blokadę innego typu

Blokada żądana \ Blokada uzyskana	R	W
R	✓	-
W	✓	✓

Wprowadzimy obecnie pojęcie konwersji blokad. Transakcja posiadająca blokadę określonego typu na danej może dokonać jej konwersji w blokadę innego typu zgodnie z przyjętą macierzą konwersji blokad. Macierz konwersji blokad przedstawiono na slajdzie. Jak łatwo zauważyć, dopuszczalna jest konwersja blokady typu R na blokadę typu W (nazywamy to operacją eskalacji blokad). Natomiast niedopuszczalna jest konwersja z blokady do zapisu (W) na blokadę do odczytu (R).

Obecnie przedstawimy implementację algorytmów blokowania. Implementacja ta obejmuje struktury danych wspierające blokowanie oraz algorytmy zakładania i zdejmowania blokad.

Z każdą daną jest związana blokada tej danej oraz dwie kolejki transakcji: kolejka transakcji, które uzyskały dostęp do danej oraz kolejka transakcji oczekujących na dostęp do danej. Dla ilustracji rozważmy przykład przedstawiony na slajdzie.

Dana x1 jest zablokowana przez transakcję T1 do zapisu (blokada W). W kolejce transakcji oczekujących na dostęp do x1 znajdują się dwie transakcje, tj. T2 i T3, które żądają odpowiednio blokady do odczytu (R) i zapisu (W). Dana x2 jest zablokowana do odczytu przez transakcje T1 i T2 (blokady do odczytu są kompatybilne dla obu tych transakcji). W kolejce transakcji oczekujących na dostęp do x2 znajduje się transakcja T4, która żąda blokady do zapisu (W). Dana x3 jest zablokowana przez transakcję T2 do zapisu, natomiast kolejka transakcji oczekujących na dostęp do x3 jest pusta.

Systemy baz danych

Implementacja algorytmów blokowania (2)

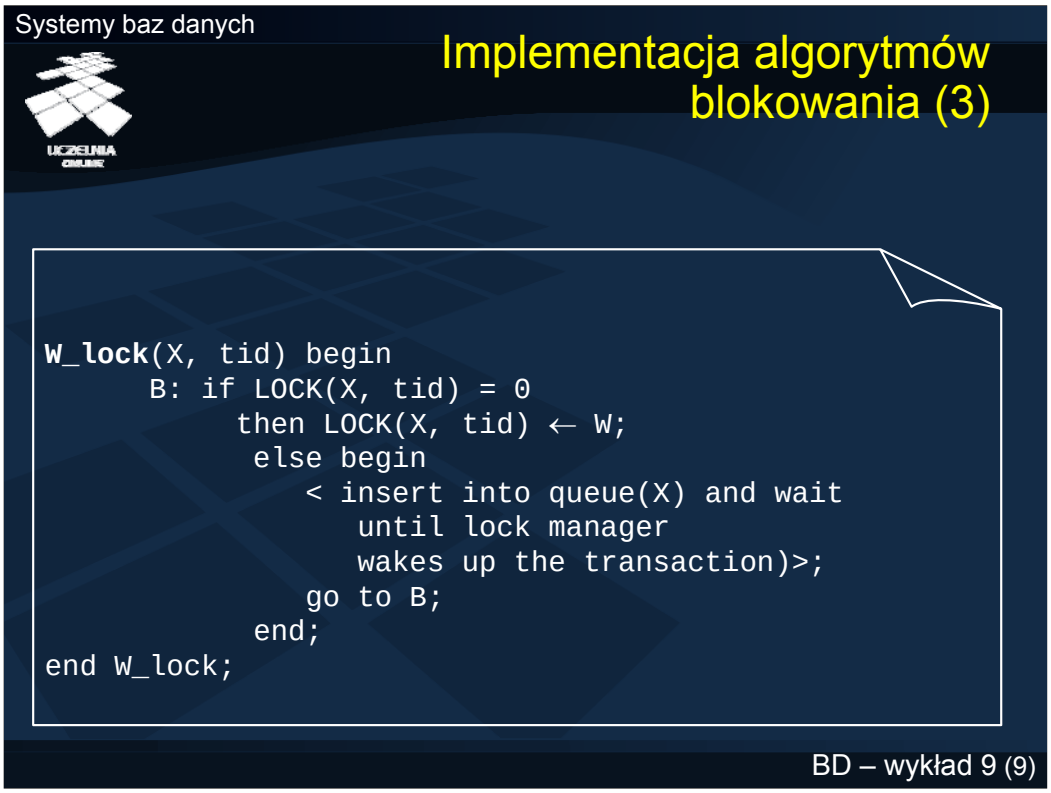
Operacje: ***LOCK, R_lock, W_lock, Unlock***

```
LOCK(X, tid) {0, R, W}
R_lock(X, tid) begin
      B: if (LOCK(X, tid)=0 or LOCK(X, tid)=R)
       then LOCK(X, tid) ← R;
       else begin
          < insert into queue(X) and wait
            until lock
            manager wakes up the transaction)>;
          go to B;
      end;
end R_lock;
```

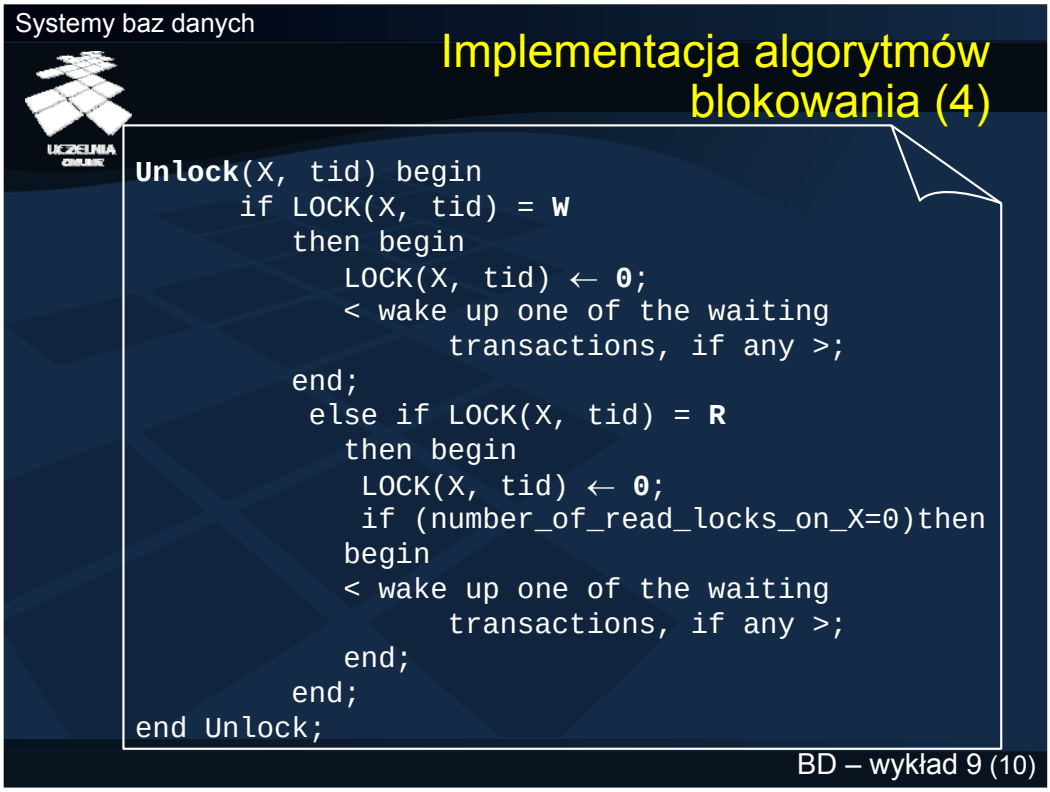
BD – wykład 9 (8)

Przedstawimy obecnie algorytmy zakładania i zdejmowania blokad. Rozpoczniemy od algorytmu zakładania blokady do odczytu, przedstawionego na slajdzie.

Algorytm przedstawia założenie blokady do odczytu przez transakcję tid na danej X. Jeżeli dana X jest niezablokowana lub dana X jest zablokowana do odczytu to transakcja tid zakłada blokadę do odczytu (R) danej X. W przeciwnym razie, transakcja tid jest umieszczana w kolejce transakcji oczekujących na dostęp do danej X i oczekuje tam do momentu zwolnienia blokady na danej X.



Algorytm zakładania blokady do zapisu przedstawiono na slajdzie. Algorytm przedstawia założenie blokady do zapisu przez transakcję tid na danej X. Jeżeli dana X jest niezablokowana to transakcja tid zakłada blokadę do zapisu (W) danej X. W przeciwnym razie, transakcja tid jest umieszczana w kolejce transakcji oczekujących na dostęp do danej X i oczekuje tam do momentu zwolnienia blokady na danej X.



Niniejszy slajd przedstawia algorytm odblokowania danej X przez transakcję tid. Jeżeli dana X była blokowana w trybie do zapisu (W) przez transakcję tid, wówczas, następuje zdjęcie blokady do zapisu (dana X jest niezablokowana) i dostęp do danej X uzyskuje pierwsza transakcja z kolejki transakcji oczekujących na dostęp do danej X. Jeżeli dana X była blokowana w trybie do odczytu (R) przez transakcję tid, wówczas następuje zdjęcie blokady do odczytu dla tej transakcji. Jeżeli transakcja tid była jedyną transakcją blokującą daną X do odczytu, wówczas, po zdjęciu blokady przez transakcję tid, dana X znajdzie się w stanie – nie zablokowana. Jeżeli natomiast, równolegle z transakcją tid dana X była blokowana do odczytu przez inne transakcje, wówczas, po zdjęciu blokady przez transakcję tid, dana X znajdzie się w stanie - zablokowana do odczytu.

Algorytm blokowania (1)

T_1	T_2
R_lock(T1, Y)	R_lock(T2, X)
read(Y)	read(X)
unlock(Y)	unlock(X)
W_lock(T1, X)	W_lock(T2, Y)
read(X)	read(Y)
X := X + Y;	Y := Y + X;
write(X);	write(Y);
unlock(X)	unlock(Y)

Wartości początkowe:
X = 20, Y = 30

Wyniki sekwencyjnych realizacji transakcji T_1 i T_2 :
($T_1 \rightarrow T_2$) : X = 50, Y = 80; ($T_2 \rightarrow T_1$) : X = 70, Y = 50;

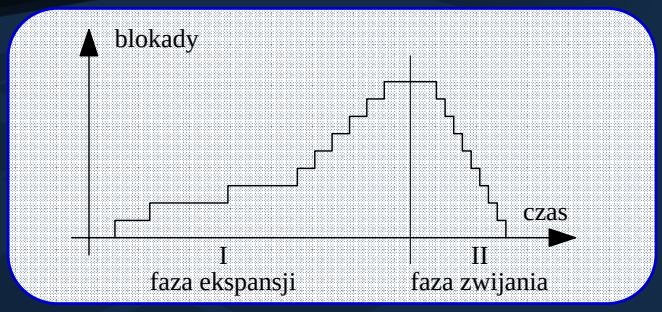
Rozważmy następujący przykład przedstawiony na slajdzie ilustrujący działanie mechanizmu zakładania i zdejmowania blokad. Dane są transakcje T1 i T2. Transakcja T1 odczytuje wartości danych X i Y. Następnie sumuje te wartości i zapisuje do danej X. Transakcja 2 odczytuje wartości danych X i Y. Następnie sumuje te wartości i zapisuje do danej Y.

Załóżmy, że wartości początkowe danych X i Y wynoszą, odpowiednio, 20 i 30. Współbieżne wykonanie transakcji T1 i T2 przebiega następująco. Transakcja T1 przed odczytem Y zakłada blokadę do odczytu tej danej. Po wykonaniu operacji odczytu, zdejmuje blokadę danej Y. Współbieżnie, transakcja T2 przed odczytem X zakłada blokadę do odczytu tej danej. Po wykonaniu operacji odczytu, zdejmuje blokadę danej X. Następnie, transakcja T1 zakłada blokadę do zapisu danej X, wykonuje odczyt danej X, aktualizuje wartość danej X, zapisuje zmodyfikowaną wartość danej X i zdejmuje blokadę do zapisu danej X. Następnie, transakcja T2 zakłada blokadę do zapisu danej Y, wykonuje odczyt danej Y, aktualizuje wartość danej Y zapisuje zmodyfikowaną wartość danej Y i zdejmuje blokadę do zapisu danej Y.

Końcowy stan bazy danych uzyskany w wyniku przedstawionego współbieżnego wykonania transakcji T1 i T2 wynosi X=50 i Y=50. Zauważmy, że stan bazy danych uzyskany w wyniku sekwencyjnej realizacji transakcji T1 i T2 różni się od stanu bazy danych uzyskanego w wyniku przedstawionego współbieżnego wykonania transakcji T1 i T2. Dla realizacji sekwencyjnej, w której transakcja T1 poprzedza transakcję T2 końcowy stan bazy danych wynosi X=50 i Y=80, natomiast dla realizacji sekwencyjnej, w której transakcja T2 poprzedza transakcję T1 końcowy stan bazy danych wynosi X=70 i Y=50. Oznacza to, że przedstawiona realizacja współbieżna transakcji jest nieuszeregowalna.

Stosowanie blokad na danych nie gwarantuje automatycznie uszeregowalności realizacji zbioru transakcji.

Algorytm blokowania dwufazowego (1)



Algorytm podstawowy:

1. Każda operacja *read(X)* danej transakcji *T* musi być poprzedzona operacją *R_lock(X, T)* lub *W_lock(X, T)*
2. Każda operacja *write(X)* danej transakcji *T* musi być poprzedzona operacją *W_lock(X, T)*
3. Operacje *unlock(x,T)* dla danej transakcji *T* są wykonywane po zakończeniu wszystkich operacji *read* i *write*

Jak widać z poprzedniego przykładu stosowanie blokad na danych nie gwarantuje automatycznie uszeregowalności realizacji zbioru transakcji. Okazuje się, że istotna jest kolejność zakładania i zdejmowania blokad. Ilustruje to podstawowy algorytm blokowania, nazywany algorytmem blokowania dwu-fazowego (2PL). Podstawowa wersja algorytmu 2PL ma następującą postać:

1. Każda operacja odczytu danej X przez transakcję T (*read(X)*) musi być poprzedzona operacją zablokowania danej X w trybie do odczytu (*R_lock(X, T)*) lub w trybie do zapisu (*W_lock(X, T*)).
2. Każda operacja zapisu danej X przez transakcję T (*write(X)*) musi być poprzedzona operacją *W_lock(X, T)*.
3. Operacje odblokowania danej X (*unlock(x,T)*) dla danej transakcji *T* są wykonywane po zakończeniu wszystkich operacji *read* i *write*.

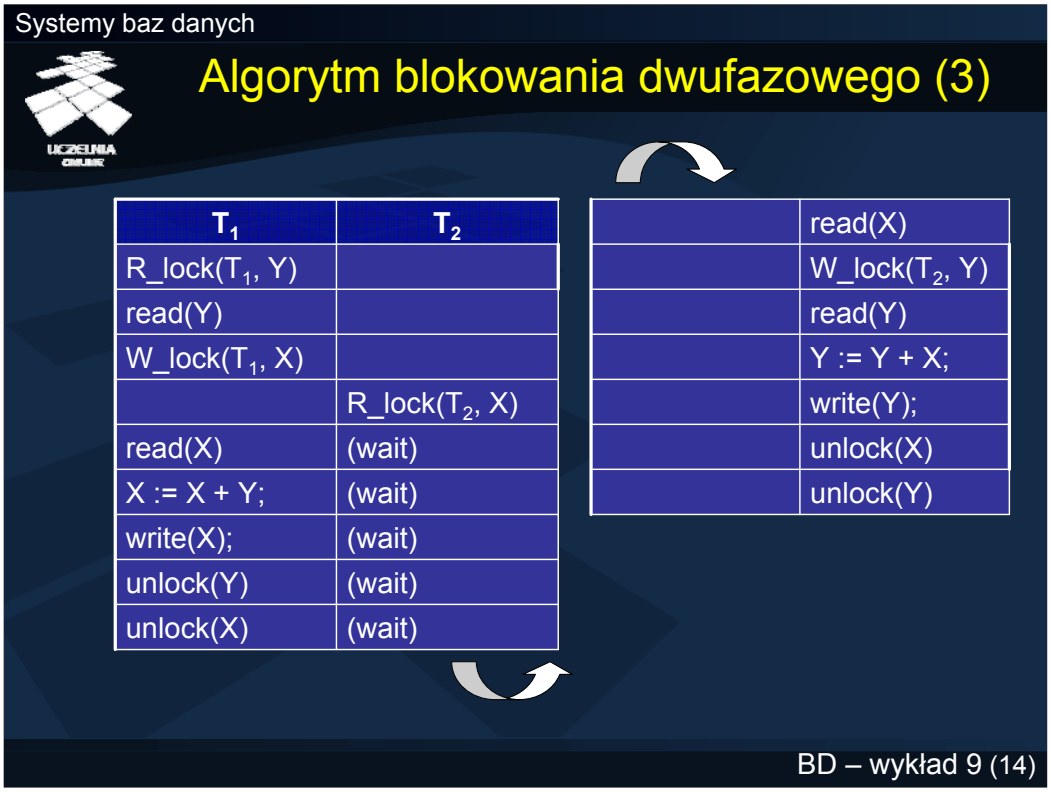
Jak widać z przedstawionego schematu, realizacja transakcji, zgodnie z algorytmem 2PL, przebiega w dwóch fazach (stąd nazwa algorytmu): w fazie ekspansji oraz w fazie zwijania. W fazie ekspansji transakcja zakłada blokady kolejnych danych, nie zwalniając żadnej z uzyskanych blokad, aż do osiągnięcia punktu akceptacji. W fazie zwijania transakcja zwalnia blokady wszystkich danych, nie może żądać natomiast założenia żadnej blokady.

Algorytm blokowania dwufazowego (2)

- **Algorytm statyczny:** (1., 2., 3.)
  Wszystkie blokady muszą być uzyskane przed rozpoczęciem transakcji (przez predeklarowanie zbioru odczytywanych i modyfikowanych danych)
- **Algorytm restryktywny:** (1., 2.)
  Operacje *unlock(x,T)* dla danej transakcji *T* są wykonywane po operacji *commit* lub *rollback*

Istnieje wiele wariantów podstawowego algorytmu blokowania dwu-fazowego. W algorytmie statycznym 2PL, składającym się z podstawowych kroków algorytmu 2PL z poprzedniego slajdu, wszystkie blokady muszą być uzyskane przed rozpoczęciem transakcji (przez predeklarowanie zbioru odczytywanych i modyfikowanych danych).

W algorytmie restryktywnym 2PL, kroki 1 i 2 są identyczne z algorytmem podstawowym 2PL, natomiast w kroku trzecim operacje *unlock(x,T)* dla danej transakcji *T* są wykonywane po operacji *commit* lub *rollback*.



Dla ilustracji działania algorytmu blokowania dwu-fazowego rozważmy ponownie przykładową realizację transakcji T1 i T2 ze slajdu nr 11. Transakcja T1 po uzyskaniu blokady do odczytu danej Y i uzyskaniu blokady do zapisu danej X wykonuje odczyt obu danych, aktualizuje i zapisuje daną X. Następnie, zdejmuje blokady danych X i Y. Współbieżnie, transakcja T2 żąda założenia blokady do odczytu danej X. Ponieważ dana X jest zablokowana przez transakcję T1 do zapisu, żądanie transakcji T2 nie może być zrealizowane i transakcja T2 zostaje umieszczona w kolejce transakcji oczekujących na uwolnienie danej X. Po odblokowaniu danej X przez transakcję T1, realizacja transakcji T2 jest wznowiona. T2 realizuje odczyt danych X i Y, aktualizuje i zapisuje nową wartość danej Y.

Końcowy stan bazy danych uzyskany w wyniku przedstawionego współbieżnego wykonania transakcji T1 i T2 wynosi tym razem X=50 i Y=80. Zauważmy, że tym razem stan końcowy bazy danych jest równoważny stanowi bazy danych uzyskanemu w wyniku sekwencyjnej realizacji transakcji T1 i T2. Oznacza to, że przedstawiona realizacja współbieżna transakcji jest uszeregowalna.

Zakleszczenie transakcji (1)

T_1	T_2
R_lock(T_1, Y)	
read(Y)	
	R_lock(T_2, X)
	read(X)
	W_lock(T_2, Y)
W_lock(T_1, X)	(wait)
(wait)	(wait)
(wait)	(wait)
dead	lock

Dwa podejścia do problemu zakleszczenia transakcji:
- **Wykrywanie i rozwiązywanie** zakleszczenia
- **Zapobieganie wystąpieniu** zakleszczenia

Rozważmy przykład współbieżnej realizacji transakcji T1 i T2, zgodnie z algorytmem 2PL, przedstawiony na slajdzie. Transakcja T1 zakłada blokadę do odczytu danej Y, a następnie żąda założenia blokady do zapisu danej X. Współbieżnie transakcja T2 zakłada blokadę do odczytu danej X, a następnie żąda założenia blokady do zapisu danej Y. Zauważmy, że realizacja obu transakcji ulega zawieszeniu. Transakcja T1 oczekuje na uwolnienie danej X przez transakcję T2, natomiast transakcja T2 oczekuje na uwolnienie danej Y blokowanej przez transakcję T1. Stan wzajemnego oczekiwania transakcji na uwolnienie zasobów nazywamy zakleszczeniem transakcji.

W systemach baz danych stosowane są dwa podejścia do problemu zakleszczenia transakcji:
- wykrywanie i rozwiązywanie zakleszczenia,
- zapobieganie wystąpieniu zakleszczenia.

Algorytmy zapobiegania zakleszczeniom (1)

- Algorytmy wykorzystujące ***znaczniki czasowe*** **transakcji** - *TS(T)*, nadawane w momencie inicjacji transakcji:
- **wait-die**: Transakcja T_i próbuje uzyskać blokadę na danej *X*, tymczasem dana ta jest już zablokowana przez transakcję T_j. Jeżeli $TS(T_i) < TS(T_j)$ (T_i jest starsza T_j) wtedy transakcja T_i będzie czekać na zwolnienie blokady. W przeciwnym wypadku T_i będzie wycofana i restartowana z tym samym znacznikiem czasowym
- **wound-wait**: Transakcja T_i próbuje uzyskać blokadę na danej *X*, tymczasem dana ta jest już zablokowana przez transakcję T_j. Jeżeli $TS(T_i) < TS(T_j)$ (T_i jest starsza T_j) wtedy transakcja T_j będzie wycofana i restartowana z tym samym znacznikiem czasowym. W przeciwnym wypadku T_i będzie czekać na zwolnienie blokady

Przedstawimy obecnie krótko dwa podstawowe algorytmy zapobiegania zakleszczeniom. Oba algorytmy do rozwiązywania problemu zakleszczenia wykorzystują tzw. znaczniki czasowe transakcji. Znacznik czasowy transakcji T, oznaczany jako TS(T), jest nadawany transakcji w momencie jej inicjacji i stanowi konkatenację stanu zegara fizycznego (lub logicznego) oraz identyfikatora stanowiska. Cechą charakterystyczną znacznika czasowego transakcji jest jego unikalność.

Pierwszy z wymienionych algorytmów zapobiegania zakleszczeniom, algorytm wait-die, ma następującą postać:

Transakcja T_i próbuje uzyskać blokadę na danej *X*, tymczasem dana ta jest już zablokowana przez transakcję T_j. Jeżeli $TS(T_i) < TS(T_j)$ (T_i jest starsza T_j) wtedy transakcja T_i będzie czekać na zwolnienie blokady. W przeciwnym wypadku T_i będzie wycofana i restartowana z tym samym znacznikiem czasowym.

Drugi z wymienionych algorytmów zapobiegania zakleszczeniom, algorytm wound-wait, ma podobną filozofię działania ale nadaje inne priorytety działania transakcjom starszym i transakcjom młodszym. Transakcja T_i próbuje uzyskać blokadę na danej *X*, tymczasem dana ta jest już zablokowana przez transakcję T_j. Jeżeli $TS(T_i) < TS(T_j)$ (T_i jest starsza T_j) wtedy transakcja T_j będzie wycofana i restartowana z tym samym znacznikiem czasowym. W przeciwnym wypadku T_i będzie czekać na zwolnienie blokady.

Oba algorytmy zapobiegają wystąpieniu zakleszczenia. Ich wadą jest to, że czasami prowadzą do wycofania transakcji nawet jeżeli nie występuje niebezpieczeństwo zakleszczenia.

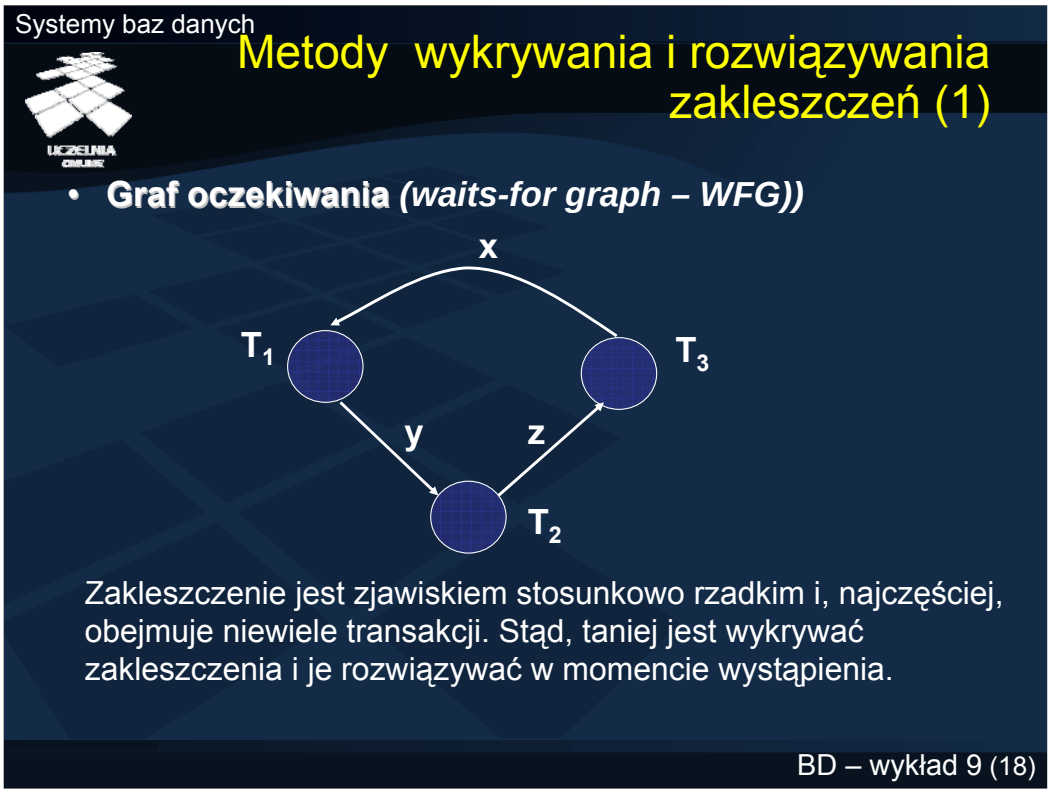
Algorytmy zapobiegania zakleszczeniom (2)

- Algorytmy nie korzystające ze **znaczników czasowych**.
- **no waiting**: Transakcja T_i próbuje uzyskać blokadę na danej X, tymczasem dana ta jest już zablokowana przez transakcję T_j. Transakcja T_i będzie wycofana i restartowana z pewnym opóźnieniem czasowym.
- **cautious waiting**: Transakcja T_i próbuje uzyskać blokadę na danej X, tymczasem dana ta jest już zablokowana przez transakcję T_j. Jeżeli transakcja T_j nie czeka na uzyskanie innej blokady, T_i będzie czekać na zwolnienie blokady przez T_j. W przeciwnym wypadku T_i będzie wycofana i restartowana

Drugą grupą algorytmów zapobiegania zakleszczeniom są algorytmy nie korzystające ze znaczników czasowych.

no waiting: Transakcja T_i próbuje uzyskać blokadę na danej X, tymczasem dana ta jest już zablokowana przez transakcję T_j. Transakcja T_i będzie wycofana i restartowana z pewnym opóźnieniem czasowym.

cautious waiting: Transakcja T_i próbuje uzyskać blokadę na danej X, tymczasem dana ta jest już zablokowana przez transakcję T_j. Jeżeli transakcja T_j nie czeka na uzyskanie innej blokady, T_i będzie czekać na zwolnienie blokady przez T_j. W przeciwnym wypadku T_i będzie wycofana i restartowana.

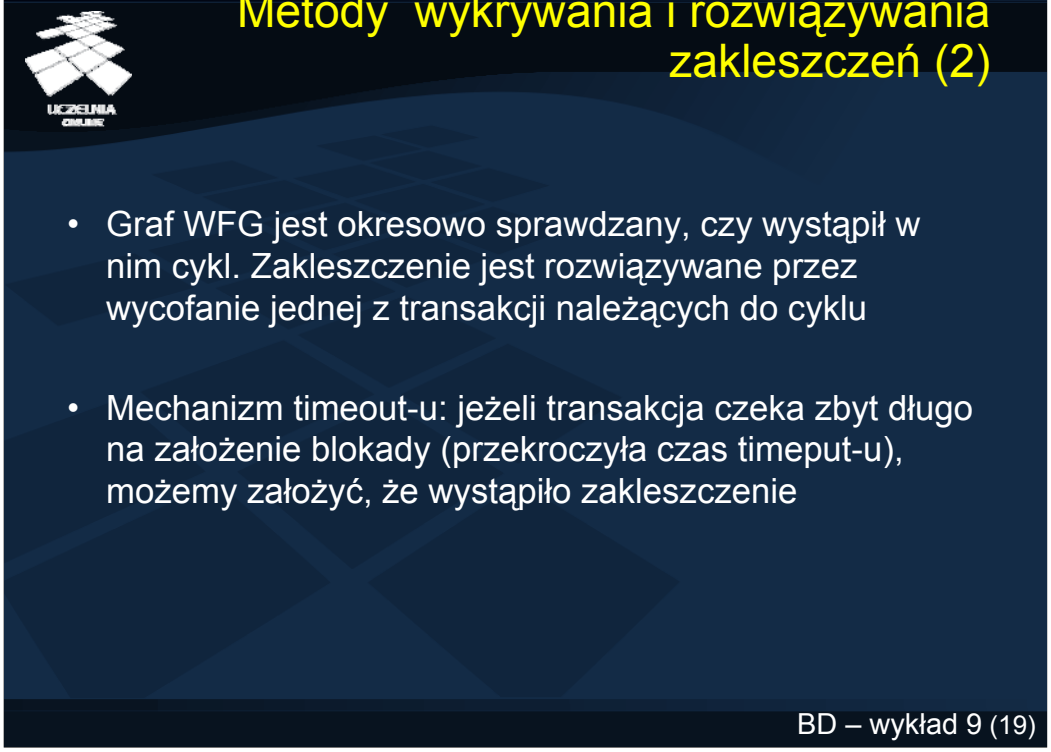


Drugą z popularnych metod rozwiązywania problemu zakleszczenia jest metoda wykrywania i rozwiązywania zakleszczeń. Idea tej metody polega na wykrywaniu zakleszczenia i następnie na jego rozwiązywaniu.

Idea algorytmu wynika ze spostrzeżenia, że zakleszczenie jest zjawiskiem stosunkowo rzadkim i, najczęściej, obejmuje niewiele transakcji. Stąd, taniej jest wykrywać zakleszczenia i je rozwiązywać w momencie wystąpienia, niż zapobiegać wystąpieniu zakleszczeń. Do wykrywania zakleszczeń metody te wykorzystują graf oczekiwania transakcji (waits-for-graphs – WFG), który reprezentuje stan wzajemnego oczekiwania transakcji na uwolnienie zasobów. Węzłami grafu są transakcje, natomiast łuki reprezentują stan oczekiwania transakcji na uwolnienie zasobu.

Przykładowo, na grafie FWG przedstawionym na slajdzie transakcja T1 oczekuje na zwolnienie danej Y blokowanej przez transakcję T2. Transakcja T2 oczekuje na uwolnienie danej Z blokowanej przez transakcję T3, która, z kolei oczekuje na zwolnienie danej X blokowanej przez transakcję T1.

Cykl w grafie FWG oznacza wystąpienie zakleszczenia w systemie.



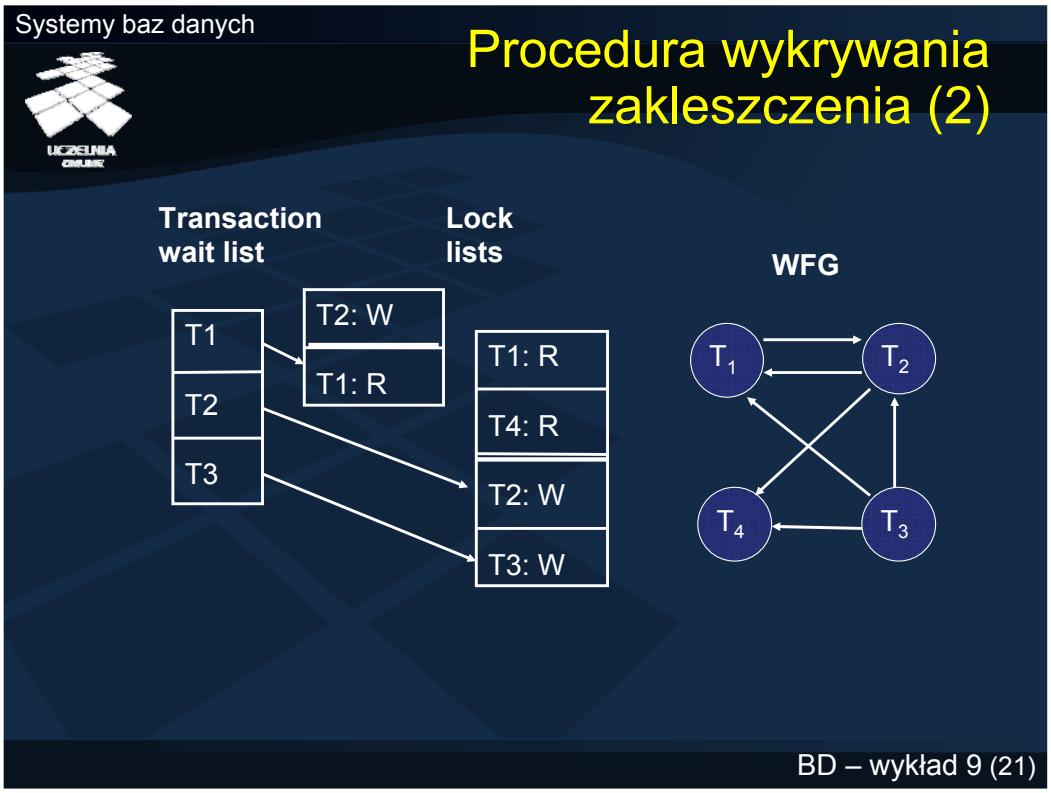
Algorytm wykrywania i rozwiązywania zakleszczeń konstruuje okresowo graf FWG i sprawdza, czy wystąpił w nim cykl. Zakleszczenie jest rozwiązywane przez wycofanie jednej z transakcji należących do cyklu. Graf FWG jest konstruowany przez specjalny proces systemowy w oparciu o mechanizm timeout-u. Jeżeli transakcja czeka zbyt długo na założenie blokady (przekroczyła limit czasu przydzielony jej przez system), możemy założyć, że wystąpiło zakleszczenie.

Procedura wykrywania zakleszczenia (1)

- Do budowy grafu WFG wykorzystujemy struktury opisujące blokady. Z każdą blokadą są związane dwie listy: lista transakcji, które uzyskały blokadę, oraz lista transakcji oczekujących na przydział blokady. Obie listy mają postać $((T_i, m_i),...)$, gdzie T_i oznacza transakcję, natomiast mi oznacza rodzaj blokady. Do grafu WFG dodajemy łuk $T_i \rightarrow T_j$, jeżeli zachodzi warunek:
  - transakcja T_j należy do listy transakcji, które uzyskały blokadę, natomiast T_i jest na liście transakcji oczekujących, lub
  - transakcja T_j jest przed transakcją T_i na liście transakcji oczekujących
  - blokady m_i i m_j są niekompatybilne

Do budowy grafu WFG wykorzystujemy struktury opisujące blokady. Z każdą blokadą są związane dwie listy: lista transakcji, które uzyskały blokadę, oraz lista transakcji oczekujących na przydział blokady. Obie listy mają postać $((T_i, m_i),...)$, gdzie T_i oznacza transakcję, natomiast mi oznacza rodzaj blokady. Do grafu WFG dodajemy łuk $T_i \rightarrow T_j$, jeżeli zachodzi warunek:

- transakcja T_j należy do listy transakcji, które uzyskały blokadę, natomiast T_i jest na liście transakcji oczekujących, lub
- transakcja T_j jest przed transakcją T_i na liście transakcji oczekujących,
- blokady m_i i m_j są niekompatybilne.



Dla ilustracji działania procedury wykrywania zakleszczenia rozważmy przykład przedstawiony na slajdzie. Mamy dwie dane oraz dwie kolejki transakcji, które uzyskały dostęp do tych danych. Pierwsza z danych jest blokowana przez transakcję T2 do zapisu (W), natomiast druga jest blokowana przez transakcje T1 i T4 do odczytu (R). Transakcja T1 oczekuje na transakcję T2 na uwolnienie pierwszego z zasobów, natomiast transakcje T2 i T3 oczekują na uwolnienie drugiego z zasobów.

Graf WFG składa się z 4 wierzchołków reprezentujących transakcje T1, T2, T3, T4. Łuk (T1,T2) reprezentuje oczekiwanie transakcji T1 na T2 w odniesieniu do pierwszego zasobu. Łuk (T2,T1) reprezentuje oczekiwanie transakcji T2 na T1 w odniesieniu do drugiego zasobu. Łuk (T2,T4) reprezentuje oczekiwanie transakcji T2 na T4 w odniesieniu do drugiego zasobu. Podobnie, łuki T3T4 i T3T1. Łuk T3T2 reprezentuje sytuację, gdy obie transakcje T3 i T2 znajdują się kolejce transakcji oczekujących na uwolnienie zasobu, ale ich blokady są niekompatybilne. Zauważmy, że w przykładowym grafie występuje cykl obejmujący wierzchołki T1 i T2.

Problem „duchów” (1)

Zakładaliśmy dotychczas, że baza danych jest zbiorem stałych i niezależnych obiektów. Rodzi to pewien problem

```
T1: select * from emp
where eyes=”blue” and hair=”red”;
T2: insert into emp
where eyes=”blue” and hair=”red”;
```

Zakładając, że blokowania jest realizowane na poziomie rekordów bazy danych, pojawia się niebezpieczeństwo nieuszeregowalności realizacji: Zauważmy, że transakcja T1 nie widzi rekordu wprowadzanego przez transakcję T2.
W odniesieniu do innej relacji, kolejność operacji transakcji T1 i T2 może być odwrotna

Przedstawimy obecnie problem duchów, który jest konsekwencją przyjęcia określonej jednostki blokowania. Zakładaliśmy dotychczas, że baza danych jest zbiorem stałych i niezależnych obiektów. Rodzi to pewien problem.

Rozważmy transakcje T1 i T2 przedstawione na slajdzie. Załóżmy, że transakcje T1 i T2 są wykonywane sekwencyjnie: najpierw T1, a później T2. Zakładając, że blokowanie jest realizowane na poziomie rekordów bazy danych, pojawia się niebezpieczeństwo nieuszeregowalności realizacji. Zauważmy, że transakcja T1 nie widzi rekordu wprowadzanego przez transakcję T2. Stąd, transakcja T1 nie mogła założyć blokady odczytu tego rekordu. W odniesieniu do innej relacji, kolejność operacji transakcji T1 i T2 może być odwrotna. W konsekwencji, współbieżna realizacja transakcji T1 i T2 będzie nieuszeregowalna. W grafie uszeregowalności transakcja T1 będzie poprzedzała transakcję T2 w odniesieniu do relacji emp przedstawionej na slajdzie, natomiast transakcja T2 będzie poprzedzała T1 w odniesieniu do drugiej relacji. Łatwo zauważyć, że w grafie uszeregowalności wystąpi cykl świadczący o nieuszeregowalności realizacji.

Problem „duchów” (2)

- W jaki sposób zapobiec, aby transakcja T2 nie wprowadzała nowych rekordów do relacji *emp* w trakcie realizacji transakcji T1?
- Łatwo zauważyć, że współbieżne wykonanie transakcji T1 i T2 może być nieuszeregowalne
- Takie nowowprowadzane lub usuwane rekordy nazywamy **duchami**
- Nie istnieje żaden mechanizm blokowania, realizowany na poziomie blokad rekordów, który gwarantowałby rozwiązanie problemu „duchów”
- Rozwiązanie problemu „duchów” wymaga wprowadzenia blokad hierarchicznych

Takie nowowprowadzane lub usuwane rekordy nazywamy „duchami”.

W jaki sposób zapobiec, aby transakcja T2 nie wprowadzała „duchów” do relacji *emp* w trakcie realizacji transakcji T1? Jak pokazaliśmy na poprzednim slajdzie pojawienie się rekordów duchów może prowadzić do nieuszeregowalności transakcji. Okazuje się, że nie istnieje żaden mechanizm blokowania, realizowany na poziomie blokad rekordów, który gwarantowałby rozwiązanie problemu „duchów”. Rozwiązanie problemu „duchów” wymaga wprowadzenia blokad hierarchicznych.