Plan

- Transakcje, odtwarzanie oraz kontrola współbieżności
- Transakcje i zarządzanie transakcjami
- Odtwarzanie
- Kontrola współbieżności
- Własności ACID transakcji

Transakcje, odtwarzanie i kontrola współbieżności

- Większość baz danych dla wielu użytkowników (multi user databases)
- Równoczesny dostęp do tych samych danych może powodować różnego rodzaju anomalie
- Błędy mogą wystąpić w DBMS lub jego środowisku
- DBMS musi wspierać własności ACID (Atomicity, Consistency, Isolation, Durability)

Transakcje, odtwarzanie i kontrola współbieżności

- Transakcja: zbiór operacji bazy danych wywoływana przez pojedynczego użytkownika lub aplikację, którą można traktować jako jedną niepodzielną jednostkę pracy
  - np. transfer między dwoma rachunkami bankowymi
- Transakcja powinna zawsze „zakończyć się sukcesem” lub „niepowodzeniem” w całości
- Transakcja przenosi bazę danych z jednego stanu spójnego w inny stan spójny

Transakcje, odtwarzanie i kontrola współbieżności

- Rodzaje błędów: awaria dysku twardego, awaria aplikacji/DBMS, dzielenie przez 0, …
- **Odtwarzanie**: działanie polegające na zapewnieniu, że niezależnie od tego, który z problemów wystąpił, baza danych zostanie przywrócona do spójnego stanu bez późniejszej utraty danych
- **Kontrola współbieżności:** koordynacja transakcji, które są realizowane jednocześnie na tych samych danych, tak aby nie powodowały niespójności w danych z powodu wzajemnej ingerencji

Transakcje i zarządzanie transakcjami

- Określanie transakcji i cyklu życia transakcji
- Komponenty DBMS zaangażowane w zarządzanie transakcjami
- Plik dziennika

Określanie transakcji i cyklu życia transakcji

- Granice transakcji można określić niejawnie lub jawnie
  - jawnie: `begin_transaction` and `end_transaction`
  - niejawnie: pierwsza wykonywalna instrukcja SQL
- Po wykonaniu pierwszej operacji transakcja jest aktywna
- Jeśli transakcja zakończyła się pomyślnie, można ją **zatwierdzić** (**committed**)**.** Jeżeli nie – należy ją **wycofać** (**rolled back**).

Określanie transakcji i cyklu życia transakcji

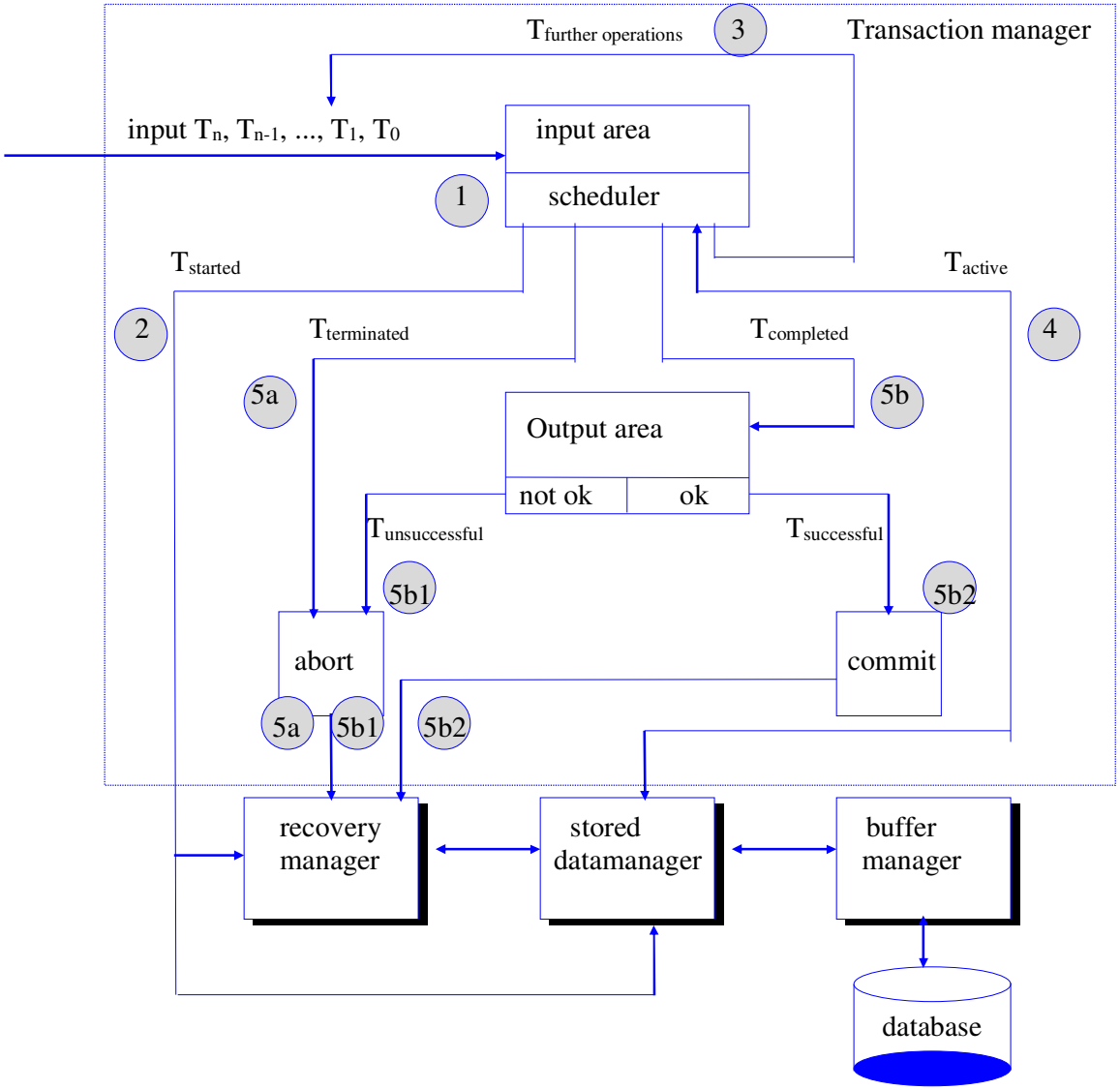
```
<begin_transaction>

UPDATE account
SET balance = balance - :amount
WHERE accountnumber = :account_to_debit

UPDATE account
SET balance = balance + :amount
WHERE accountnumber = :account_to_credit

<end_transaction>
```

Komponenty DBMS zaangażowane w zarządzanie transakcjami



Plik dziennika

- Plik dziennika rejestruje
  - unikalny numer sekwencji dziennika
  - unikalny identyfikator transakcji
  - oznaczenie początku transakcji, wraz z czasem rozpoczęcia transakcji i wskazaniem, czy transakcja jest tylko do odczytu, czy do odczytu/ zapisu
  - identyfikatory rekordów bazy danych zaangażowanych w transakcję, a także operacje, którym zostały poddane
  - **obrazy przed** (**before images**) wszystkich rekordów biorących udział w transakcji
  - **obrazy po** (**after images**) wszystkich rekordów, które zostały zmienione przez transakcję
  - aktualny stan transakcji (*active*, *committed* or *aborted*)

Plik dziennika

- Plik dziennika może również zawierać punkty kontrolne (checkpoints)
  - momenty, w których aktualizacje buforowane przez aktywne transakcje, obecne w buforze bazy danych, są natychmiast zapisywane na dysk
- Strategia zapisu dziennika do przodu (write ahead log)
  - wszystkie aktualizacje są rejestrowane w pliku dziennika przed zapisaniem na dysku
  - before images są zawsze rejestrowane w pliku dziennika przed nadpisaniem rzeczywistych wartości w fizycznych plikach bazy danych

Odtwarzanie

- Rodzaje awarii
- Odtwarzanie systemu
- Odtwarzanie nośników

Rodzaje awarii

- **Awaria transakcji** wynika z błędu w logice, która kieruje operacjami transakcji i/lub w logice aplikacji
- **Awaria systemu** występuje w przypadku awarii systemu operacyjnego lub bazy danych
- **Awaria nośnika** występuje, gdy dodatkowa pamięć masowa jest uszkodzona lub niedostępna

Odtwarzanie systemu

- W przypadku awarii systemu 2 rodzaje transakcji
  - osiągnęły już stan zatwierdzenia przed awarią
  - nadal w stanie aktywnym
- Plik dziennika jest niezbędny, aby uwzględnić, które aktualizacje zostały wykonane przez które transakcje (i kiedy) oraz aby śledzić obrazy przed i po, potrzebne do UNDO i REDO
- Strategia opróżniania bufora bazy danych ma wpływ na UNDO i REDO

Odtwarzanie systemu

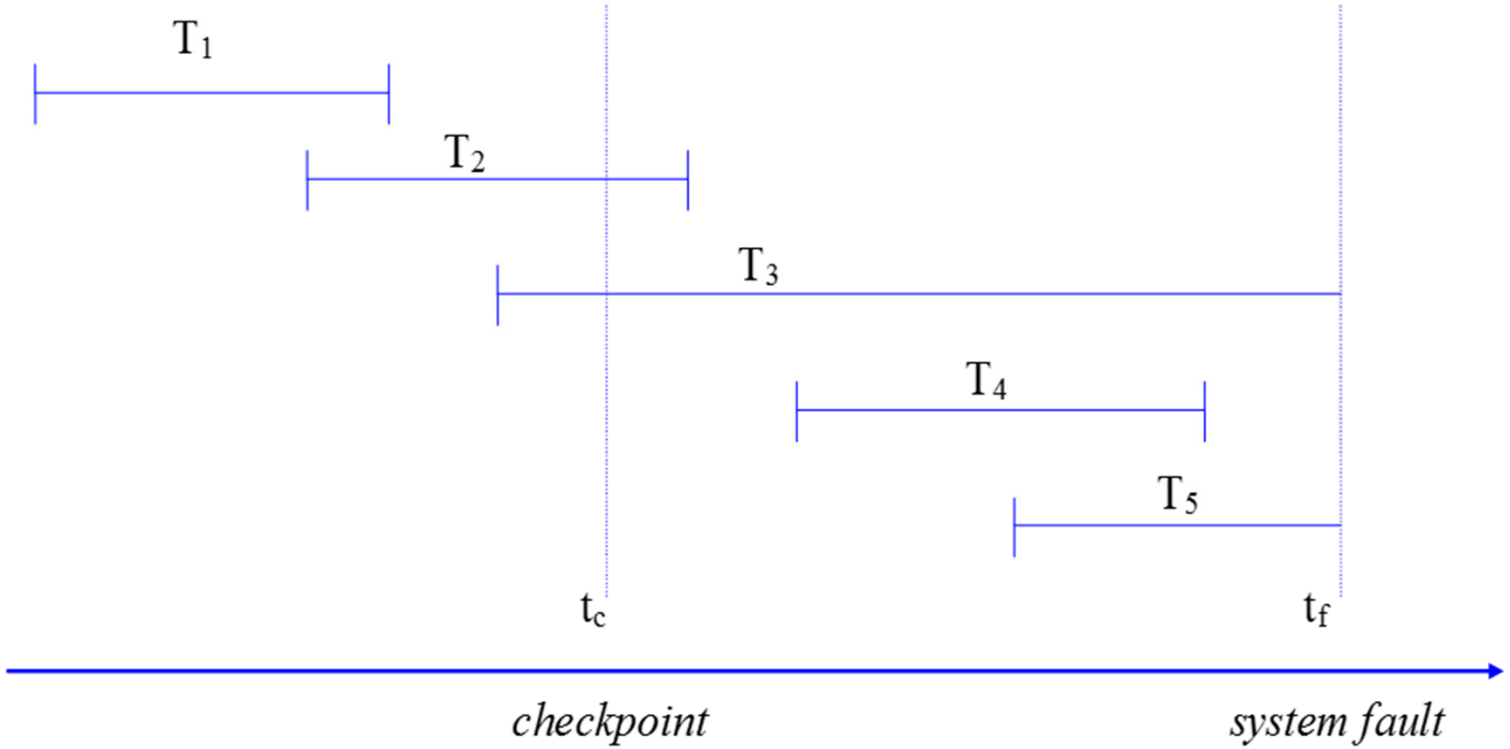
T_1: nothing

T_2: REDO

T_3: UNDO

T_4: REDO

T_5: nothing



Uwaga 1: punkt kontrolny oznacza moment, w którym menedżer bufora ostatnio „opróżnił” bufor bazy danych na dysk!

Uwaga 2: podobne rozumowanie można zastosować w przypadku niepowodzenia transakcji (np. T_3, T_5)

Odtwarzanie nośników

- Odzyskiwanie nośników opiera się na pewnym typie nadmiarowości danych
  - Przechowywanych na offline (np. taśmy) lub online mediach (np. zapasowy dysk twardy online)
- Kompromis między kosztami utrzymania nadmiarowych danych a czasem potrzebnym do przywrócenia systemu
- Dwa rodzaje: mirroring dysku i archiwizacja

Odtwarzanie nośników

- Mirroring dysku
  - podejście w czasie (prawie) rzeczywistym, w którym te same dane są zapisywane jednocześnie na 2 lub więcej dyskach fizycznych
  - ograniczony czas przełączania awaryjnego, ale często droższe niż archiwizacja
  - (ograniczony) negatywny wpływ na wydajność zapisu, ale możliwości równoległego dostępu do odczytu
- Archiwizacja
  - pliki baz danych są okresowo kopiowane na inne nośniki danych (np. taśma, dysk twardy)
  - kompromis między kosztem częstszych kopii zapasowych a kosztem utraty danych
  - pełna vs przyrostowa kopia zapasowa

Kontrola współbieżności

- Typowe problemy ze współbieżnością
- Harmonogramy i harmonogramy seryjne
- Serializowalne harmonogramy
- Optymistyczne i pesymistyczne harmonogramy
- Blokowania i protokoły blokowania

Typowe problemy ze współbieżnością

- Scheduler odpowiada za planowanie wykonywania transakcji i ich operacji
- Proste wykonanie seryjne byłoby bardzo nieefektywne
- Scheduler zapewni, że operacje transakcji mogą być wykonywane z przeplotem
- Mogą wystąpić problemy z interferencją
  - problem utraconej aktualizacji (utraty zmian)
  - problem niezatwierdzonych zależności
  - problem analizy niespójności

Typowe problemy ze współbieżnością

- Problem **utraconej aktualizacji** (lost update problem) występuje, jeśli udana aktualizacja danych przez transakcję jest nadpisywana przez inną transakcję, która nie była „świadoma” pierwszej aktualizacji

time	T_1	T_2	$amount_x$
t_1		begin transaction	100
t_2	begin transaction	read($amount_x$)	100
t_3	read($amount_x$)	$amount_x = amount_x + 120$	100
t_4	$amount_x = amount_x - 50$	write($amount_x$)	220
t_5	write($amount_x$)	commit	50
t_6	commit		50

Typowe problemy ze współbieżnością

- Jeśli transakcja odczytuje jedną lub więcej danych, które są aktualizowane przez inną, jeszcze niezatwierdzoną transakcję, możemy napotkać problem **niezatwierdzonych zależności** (**odczyt niepewnych/brudnych danych**) (uncommited dependency problem/dirty read problem)

time	T_1	T_2	$amount_x$
t_1		begin transaction	100
t_2		read($amount_x$)	100
t_3		$amount_x = amount_x + 120$	100
t_4	begin transaction	write($amount_x$)	220
t_5	read($amount_x$)		220
t_6	$amount_x = amount_x - 50$	rollback	100
t_7	write($amount_x$)		170
t_8	commit		170

Typowe problemy ze współbieżnością

- Problem **analizy niespójności** (incosistent analysis problem) oznacza sytuację gdy transakacja odczytuje częściowe wyniki innej transakcji, która równolegle oddziałuje na te same dane (i aktualizuje je).

time	T_1	T_2	$amount_x$ $_y$ $_z$	*sum*
t_1		begin transaction	100 75 60	
t_2	begin transaction	sum = 0	100 75 60	0
t_3	read($amount_x$)	read($amount_x$)	100 75 60	0
t_4	$amount_x$ = $amount_x$ – 50	sum = sum + $amount_x$	100 75 60	100
t_5	write($amount_x$)	read($amount_y$)	50 75 60	100
t_6	read($amount_z$)	sum = sum + $amount_y$	50 75 60	175
t_7	$amount_z$ = $amount_z$ + 50		50 75 60	175
t_8	write($amount_z$)		50 75 110	175
t_9	commit	read($amount_z$)	50 75 110	175
t_{10}		sum = sum + $amount_z$	50 75 110	285
t_{11}		commit	50 75 110	285

Typowe problemy ze współbieżnością

- Inne problemy związane ze współbieżnością
  - **niepowtarzalny odczyt** (nonrepeatable read) występuje gdy transakcja T_1 odczytuje ten sam wiersz wiele razy, ale uzyskuje różne kolejne wartości, ponieważ w międzyczasie inna transakcja T_2 zaktualizowała ten wiersz
  - **odczyt fantomów** (Phantom reads) może wystąpić, gdy transakcja T_2 wykonuje operacje wstawiania lub usuwania na zestawie wierszy odczytywanych przez transakcję T_1

Harmonogramy i harmonogramy szeregowe

- Definiujemy *harmonogram* S jako zbiór *n* transakcji I sekwencyjne uporządkowanie instrukcji tych transakcji, dla których zachodzi następująca własność:
  "Dla każdej transakcji T która uczestniczy w harmonogramie S i dla wszystkich instrukcji s_i i s_j które należą do tej samej transakcji T: jeżeli instrukcja s_i poprzedza instrukcję s_j w T, to s_i ma zostać wykonana przed s_j w S."
- Harmonogram zachowuje kolejność poszczególnych instrukcji *wewnątrz* każdej transakcji, ale pozwala na dowolną kolejność instrukcji *między* transakcjami

Harmonogramy i szeregowe harmonogramy

- Harmonogram S jest *szeregowy* jeżeli wszystkie instrukcje s_i tej samej transakcji T są zaplanowane kolejno, bez przeplotu z instrukcjami z innej transakcji
- Szeregowe harmonogramy uniemożliwiają równoległe wykonywanie transakcji
- Potrzeba nieszeregowego poprawnego harmonogramu

Harmonogramy szeregowalny

- Harmonogram **szeregowalny** to hamonogram niesekwencyjny, który jest równoważny harmonogramowi sekwencyjnemu
- 2 harmonogramy S_1 i S_2 (z tymi samymi transakcjami T_1, T_2, ..., T_n) są równoważne jeżeli
  - *Dla każdej operacji $read_x$ transakcji T_i w S_1 zachodzi: jeżeli wartość x która jest odczytana przez tę operację, została ostatnio zapisana przez operację $write_x$ transakcji T_j w S_1, to ta sama operacja $read_x$ transakcji T_i w S_2 powinna odczytać wartość x, zapisaną przez tę samą operację $write_x$ T_j w S_2*
  - *Dla każdej wartości x na którą ma wpływ operacja zapisu w tych harmonogramach, ostatnia operacja zapisu $write_x$ w harmonogramie S_1, wykonana jako część transakcji T_i, powinna również być ostatnią operacją zapisu x w harmonogramie S_2, ponownie jako część transakcji T_i.*

Harmonogramy szeregowalne

	schedule S_1 serial schedule		*schedule S_2 non serial schedule*	
time	T_1	T_2	T_1	T_2
t_1	begin transaction		begin transaction	
t_2	read($amount_x$)		read($amount_x$)	
t_3	$amount_x$ = $amount_x$ + 50		$amount_x$ = $amount_x$ + 50	
t_4	write($amount_x$)		write($amount_x$)	begin transaction
t_5	read($amount_y$)			read($amount_x$)
t_6	$amount_y$ = $amount_y$ - 50			$amount_x$ = $amount_x$ x 2
t_7	write($amount_y$)			write($amount_x$)
t_8	end transaction			read($amount_y$)
t_9				$amount_y$ = $amount_y$ x 2
t_{10}		begin transaction		write($amount_y$)
t_{11}		read($amount_x$)	read($amount_y$)	end transaction
t_{12}		$amount_x$ = $amount_x$ x 2	$amount_y$ = $amount_y$ - 50	
t_{13}		write($amount_x$)	write($amount_y$)	
t_{14}		read($amount_y$)	end transaction	
t_{15}		$amount_y$ = $amount_y$ x 2		
t_{16}		write($amount_y$)		
t_{17}		end transaction		

Harmonogramy szeregowalne

- **Graf pierwszeństwa** do testowania harmonogramu pod kątem szeregowalności
  - utwórz węzeł dla każdej transakcji T_i
  - utwórz skierowaną krawędź $T_i \rightarrow T_j$ jeżeli T_j czyta wartość po zapisaniu jej przez T_i
  - utwórz skierowaną krawędź $T_i \rightarrow T_j$ jeżeli T_j zapisuje wartość po jej odczytaniu przez T_i
  - utwórz skierowaną krawędź $T_i \rightarrow T_j$ jeżeli T_j zapisuje wartość po jej zapisaniu przez T_i
- Jeżeli graf pierwszeństwa zawiera cykl – harmonogram nie jest szeregowalny

Harmonogramy optymistyczne i pesymistyczne

- Scheduler wykorzystuje protokół harmonogramowania
- Protokół optymistyczny
  - konflikty między równoległymi transakcjami są rzadkie
  - operacje transakcji są planowane bez opóźnień
  - gdy transakcja jest gotowa do zatwierdzenia, jest weryfikowana pod kątem konfliktów
  - jeśli nie ma konfliktów, transakcja jest zatwierdzana; w przeciwnym razie - wycofywana
- Protokół pesymistyczny
  - jest prawdopodobne, że transakcje będą kolidować i powodować konflikty
  - realizacja operacji transakcyjnych opóźniona do czasu, gdy scheduler będzie mógł je zaplanować w taki sposób, aby zminimalizować ryzyko konfliktów
  - do pewnego stopnia zmniejszenie przepustowości
  - np. harmonogram sekwencyjny

Harmonogramy optymistyczne i pesymistyczne

- Blokowanie może być używane do harmonogramowania optymistycznego i pesymistycznego
  - Pesymistyczne harmonogramowanie: blokowanie służy do ograniczenia jednoczesnego wykonania transakcji
  - Optimistyczne harmonogramowanie: blokowanie służy do wykrywania konfliktów podczas realizacji transakcji
- Znacznik czasowy (timestamping)
  - Znaczniki czasowe odczytu i zapisu to atrybuty powiązane z obiektem bazy danych
  - Znaczniki czasowe służą do wymuszenia wykonania zbioru operacji transakcji w odpowiedniej kolejności

Blokowanie i protokoły blokowania

- Cele blokowania
- Protokół dwufazowego blokowania Two-Phase Locking Protocol (2PL)
- Kaskadowe wycofywania (Cascading Rollbacks)
- Radzenie sobie z zakleszczeniaim (Dealing with Deadlocks)
- Poziomy izolacji
- Ziarnistość blokad

Cele blokowania

- Zapewnienie, że w sytuacjach, w których różne współbieżne transakcje próbują uzyskać dostęp do tego samego obiektu bazy danych, dostęp jest udzielany tylko w taki sposób, aby nie mogły wystąpić konflikty
- Blokada to zmienna powiązana z obiektem bazy danych, w której wartość zmiennej ogranicza typy operacji, które mogą być wykonywane na obiekcie w danym czasie
- Menedżer blokad odpowiada za przyznawanie blokad i zwalnianie blokad poprzez zastosowanie protokołu blokowania

Cele blokowania

- **Blokada na wyłączność** (**exclusive lock**) (x-lock lub blokada zapisu) oznacza, że pojedyncza transakcja uzyskuje wyłączne uprawnienie do interakcji z tym konkretnym obiektem bazy danych w danym czasie
  - żadne inne transakcje nie mogą go czytać ani zapisywać
- **Blokada współdzielona** (**shared lock**) (blokada s-lock lub blokada odczytu) gwarantuje, że żadne inne transakcje nie zaktualizują tego samego obiektu tak długo, jak utrzymywana jest blokada
  - inne transakcje mogą również posiadać współdzieloną blokadę na tym samym obiekcie, jednak mogą tylko go czytać

Cele blokowania

- Jeśli transakcja chce zaktualizować obiekt, wymagana jest blokada na wyłączność
  - możliwa do uzyskania tylko wtedy, gdy żadna inna transakcja nie blokuje obiektu
- Macierz kompatybilności (compatibility matrix)

Type of lock requested	Type of lock(s) currently held on object: unlocked	shared	exclusive
unlock	-	yes	yes
shared	yes	yes	no
exclusive	yes	no	no

Cele blokowania

- Menadżer blokad implementuje protokół blokowania
  - zbiór reguł określających jakie blokady można przyznać w jakiej sytuacji (na podstawie np. macierzy kompatybilności)
- Menadżer blokad wykorzystuje również tabelę blokad
  - które blokady są aktualnie utrzymywane przez jaką transakcję, które transakcje czekają na zdobycie określonych blokad itp.
- Menadżer blokad musi zapewnić „uczciwość” planowania transakcji, aby np. uniknąć zagłodzenia

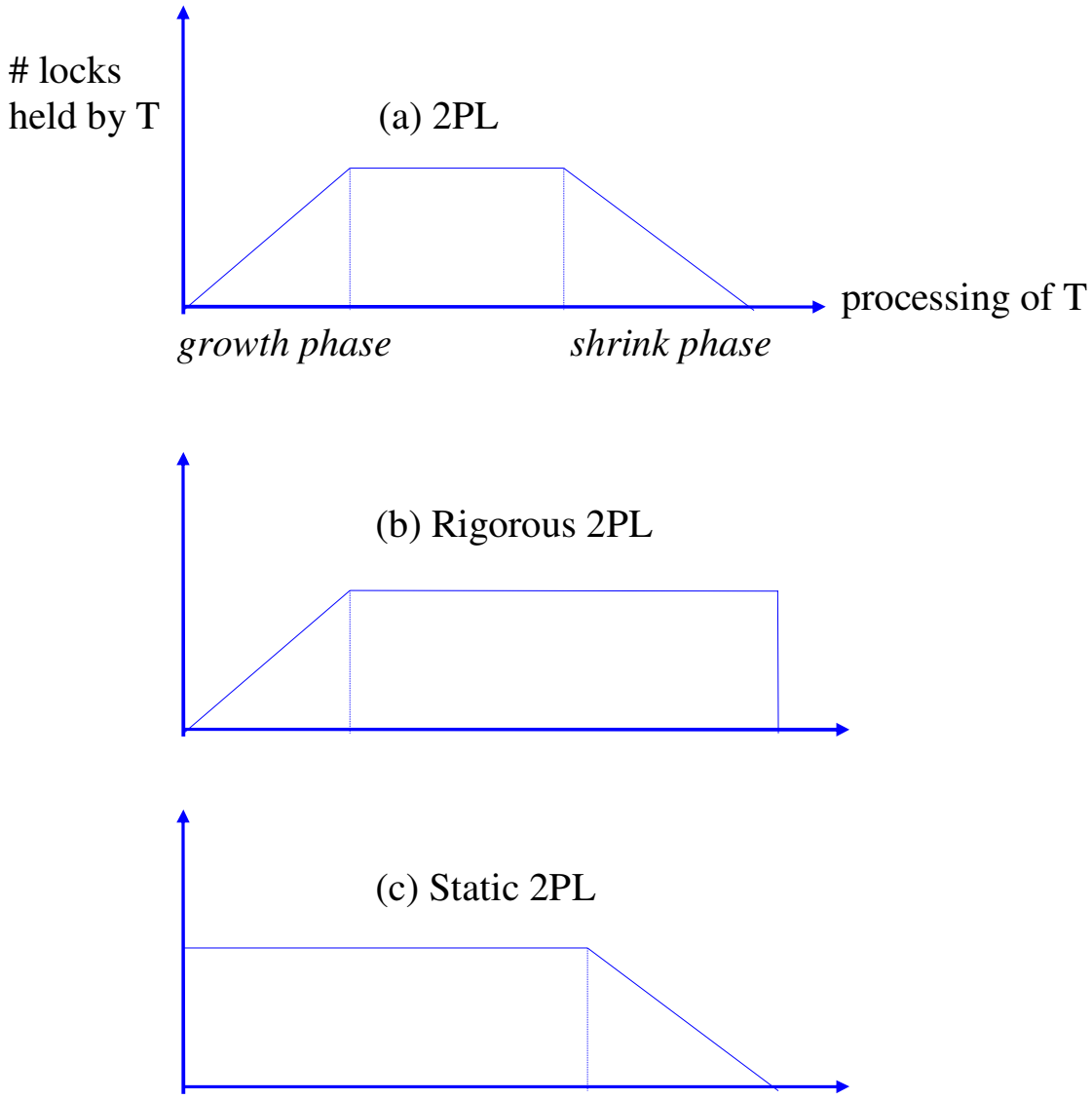
Protokół dwufazowego blokowania (2PL)

- Protokół 2PL działa w następujący sposób:
  1. Zanim transakcja będzie mogła odczytać (zaktualizować) obiekt bazy danych, powinna uzyskać współdzieloną (wyłączną) blokadę tego obiektu
  2. Menadżer blokad określa, czy można przyznać żądane blokady, na podstawie macierzy kompatybilności
  3. Uzyskiwanie i zwalnianie blokad odbywa się w 2 fazach
     - faza wzrostu: można uzyskiwać blokady, ale nie można ich zwalniać
     - faza zmniejszania: blokady są stopniowo zwalniane i nie można uzyskać dodatkowych blokad

Protokół dwufazowego blokowania (2PL)

- Warianty
  - Rygorystyczny 2PL: transakcja utrzymuje wszystkie swoje blokady, dopóki nie zostanie zatwierdzona
  - Statyczny 2PL (Zachowawczy 2PL): transakcja uzyskuje wszystkie swoje blokady od razu na początku transakcji

Protokół dwufazowego blokowania (2PL)



Protokół dwufazowego blokowania (2PL)

- Problem utraconej aktualizacji z blokowaniem

time	T_1	T_2	$amount_x$
t_1		begin transaction	100
t_2	begin transaction	x-lock($amount_x$)	100
t_3	x-lock($amount_x$)	read($amount_x$)	100
t_4	wait	$amount_x = amount_x + 120$	100
t_5	wait	write($amount_x$)	220
t_6	wait	commit	220
t_7	wait	unlock($amount_x$)	220
t_8	read($amount_x$)		220
t_9	$amount_x = amount_x - 50$		220
t_{10}	write($amount_x$)		170
t_{11}	commit		170
t_{12}	unlock($amount_x$)		170

Protokół dwufazowego blokowania (2PL)

- Problem niezatwierdzonych zależności z blokowaniem

time	T_1	T_2	$amount_x$
t_1		begin transaction	100
t_2		x-lock($amount_x$)	100
t_3		read($amount_x$)	100
t_4	begin transaction	$amount_x = amount_x + 120$	100
t_5	x-lock($amount_x$)	write($amount_x$)	220
t_6	wait	rollback	100
t_7	wait	unlock($amount_x$)	100
t_8	read($amount_x$)		100
t_9	$amount_x = amount_x - 50$		100
t_{10}	write($amount_x$)		50
t_{11}	commit		50
t_{12}	unlock($amount_x$)		50

Kaskadowe wycofywanie

- Problem niezatwierdzonych zależności
  - problem zostanie rozwiązany, jeżeli T_2 utrzyma wszystkie swoje blokady aż nie zostanie wycofana
  - przy protokole 2PL blokady można zwolnić już przed zatwierdzeniem lub przerwaniem transakcji (faza zmniejszania)

time	T_1	T_2	$amount_x$
t_1		begin transaction	100
t_2		x-lock($amount_x$)	100
t_3		read($amount_x$)	100
t_4	begin transaction	$amount_x = amount_x + 120$	100
t_5	x-lock($amount_x$)	write($amount_x$)	220
t_6	wait	unlock($amount_x$)	220
t_7	read($amount_x$)		220
t_8	$amount_x = amount_x - 50$	rollback	220
t_9	write($amount_x$)		170
t_{10}	commit		170
t_{11}	unlock($amount_x$)		170
t_{12}			

Kaskadowe wycofywania

- Zanim transakcja T_1 będzie mogła zostać zatwierdzona, SZBD powinien upewnić się, że wszystkie transakcje, które dokonały zmian w danych, które zostały następnie odczytane przez T_1 są najpierw zatwierdzone
- Jeżeli transakcja T_2 zostanie wycofana, wszystkie niezatwierdzone transakcje T_u które mają odczytane wartości zapisane prze T_2 muszą zostać wycofane
- Wszystkie transakcje, które z kolei mają odczytane wartości zapisane przez T_u również muszą zostać wycofane, itd
- Kaskadowe wycofywanie zmian powinno być stosowane rekurencyjnie
  - może być czasochłonne
  - najlepszym sposobem na uniknięcie tego jest trzymanie blokad przez wszystkie transakcje, dopóki nie osiągną stanu „zatwierdzonego” (np. rygorystyczne 2PL)

Radzenie sobie z zakleszczeniami

- Zakleszczenie (deadlocks) występuje, gdy 2 lub więcej transakcji czeka na zwolnienie swoich blokad
- Np.

time	T_1	T_2
t_1	begin transaction	
t_2	x-lock($amount_x$)	begin transaction
t_3	read($amount_x$)	x-lock($amount_y$)
t_4	$amount_x$ = $amount_x$ - 50	read($amount_y$)
t_5	write($amount_x$)	$amount_y$ = $amount_y$ - 30
t_6	x-lock($amount_y$)	write($amount_y$)
t_7	wait	x-lock($amount_x$)
t_8	wait	wait

Radzenie sobie z zakleszczeniami

- Zapobieganie zakleszczeniu można osiągnąć dzięki statycznemu 2PL
  - transakcja musi uzyskać wszystkie swoje blokady na początku
- Wykrywanie i rozwiązywanie
  - **graf oczekiwań (wait for graph)** składa się z węzłów reprezentujących aktywne transakcje i skierowanych krawędzi $T_i \rightarrow T_j$ dla każdej transakcji T_i oczekującej na uzyskanie blokady aktualnie posiadanej prze transakcję T_j
  - zakleszczenie istnieje, jeżeli graf oczekiwań zawiera cykl
  - wybór ofiary

Poziomy izolacji

- Poziom izolacji transakcji oferowany przez 2PL może być zbyt rygorystyczny
- Ograniczona ilość inferencji może być akceptowalna dla lepszej przepustowości
- Blokada długoterminowa jest przyznawana i zwalniana zgodnie z protokołem i jest utrzymywana przez dłuższy czas, aż do zatwierdzenia transakcji
- Blokada krótkoterminowa jest utrzymywana tylko w przedziale czasu potrzebnym do zakończenia powiązanej operacji
  - użycie krótkoterminowych blokad narusza regułę 3 protokołu 2PL
  - może służyć do poprawy przepustowości

Poziomy izolacji

- **Niezatwierdzony odczyt (Read uncommitted) 0 -** to najniższy poziom izolacji. Blokady długoterminowe nie są brane pod uwagę; zakłada się, że konflikty współbieżności nie występują lub po prostu ich wpływ na transakcje z tym poziomem izolacji nie jest problematyczny. Ten poziom izolacji jest zwykle dozwolony tylko dla transakcji tylko do odczytu, które i tak nie wykonują aktualizacji .
- **Odczyt zatwierdzonych danych (Read committed) 1** - używa długoterminowych blokad zapisu, ale krótkoterminowych blokad odczytu. W ten sposób transakcja ma gwarancję, że nie odczyta żadnych danych, które są wciąż aktualizowane przez jeszcze niezatwierdzoną transakcję. To rozwiązuje utraconą aktualizację, a także problem niezatwierdzonych zależności. Jednak problem z analizą niespójności może nadal występować na tym poziomie izolacji, a także w przypadku odczytów niepowtarzalnych i odczytów fantomowych.

Poziomy izolacji

- **Powtarzalne odczyty (Repeatable read) 2 -** używa zarówno długoterminowych blokad odczytu, jak i blokad zapisu. W ten sposób transakcja może wielokrotnie odczytywać ten sam wiersz, bez ingerencji w operacje wstawiania, aktualizowania lub usuwania przez inne transakcje. Mimo to problem odczytów fantomowych pozostaje nierozwiązany przy tym poziomie izolacji .
- **Serializowalny (Serializable) 3 -** jest najsilniejszym poziomem izolacji i odpowiada mniej więcej implementacji 2PL. Teraz unika się również odczytów fantomowych. Należy zauważyć, że w praktyce definicja serializacji w kontekście poziomów izolacji sprowadza się jedynie do braku problemów ze współbieżnością, takich jak odczyty niepowtarzalne i odczyty fantomowe .

Poziomy izolacji

Poziom izolacji	Utracone aktualizacje	Niezatwierdzone zależności	Analiza niespójności	Niepowtarzalny odczyt	Odczyt fantomów
Read uncommited	Tak	Tak	Tak	Tak	Tak
Read commited	Nie	Nie	Tak	Tak	Tak
Repeatable read	Nie	Nie	Nie	Nie	Tak
Serializable	Nie	Nie	Nie	Nie	Nie

Ziarnistość blokad

- Obiekt bazy danych do blokowania może być krotką, kolumną, tabelą, przestrzenią tabel, blokiem dysku itp .
- Kompromis między kosztem blokowania, a przepustowością transakcji
- Wiele DBMS zapewnia opcję posiadania optymalnego poziomu granulacji określonego przez system bazy danych
- Protokół wielokrotnej ziarnistości (Multiple Granularity Locking - MGL) zapewnia, że odpowiednie transakcje, które uzyskały blokady na obiektach bazy danych, które są ze sobą powiązane hierarchicznie, nie mogą być ze sobą w konflikcie

Ziarnistość blokad

- Protokół MGL wprowadza dodatkowe blokady
  - intencyjna blokada do odczytu (is-lock): koliduje tylko z x-locks
  - intencyjna blokada do zapisu (ix-lock): konfliktuje za zarówno z x-locks i s-locks
  - blokada dzielona i intencja blokady wyłącznej (six-lock): jest w konflikcie ze wszystkimi innymi typami blokad, z wyjątkiem is-lock

Ziarnistość blokad

Type of lock requested	Type of lock(s) currently held on object: unlocked	is-lock	ix-lock	s-lock	six-lock	x-lock
unlocked	-	yes	yes	yes	yes	yes
is-lock	yes	yes	yes	yes	yes	no
ix-lock	yes	yes	yes	no	no	no
s-lock	yes	yes	no	yes	no	no
six-lock	yes	yes	no	no	no	no
x-lock	yes	no	no	no	no	no

Ziarnistość blokad

- Zanim blokada na obiekcie x będzie mogła zostać przyznana, blokada intencji jest umieszczana na wszystkich większych ziarnistościach obiektów obejmujących x
  - Np. jeśli transakcja żąda blokady s-lock (x-lock) na określonej krotce, is-lock (ix-lock) zostanie umieszczona w odpowiednim obszarze tabel, tabeli i bloku dysku

Ziarnistość blokad

- Według MGL, transakcja T_i może zablokować obiekt będący częścią struktury hierarchicznej, jeśli :
  1. przestrzegane są wszystkie kompatybilności w macierzy kompatybilności
  2. blokadę początkową należy umieścić w katalogu głównym hierarchii
  3. zanim T_i może uzyskać s-lock albo is-lock na obiekcie x, powinn uzyskać is-lock lub ix-lock na rodzicu x
  4. zanim T_i może uzyskać x-lock, six-lock lub ix-lock na obiekcie x, powinien uzyskać ix-lock lub six-lock na rodzicu x
  5. T_i może uzyskać dodatkowe blokady tylko wtedy, gdy nie zwolnił jeszcze żadnej blokady
  6. Zanim T_i będzie mógł zwolnić blokadę na x, powinny zostać zwolnione wszystkie blokady na wszystkich dzieciach x
- W protokole MGL wszystkie blokady są uzyskiwane top-down, a zwalniane bottom-up

Własności ACID transakcji

- ACID: Atomicity, Consistency, Isolation, Durability
- Atomowość (Atomicity) gwarantuje, że wiele operacji bazodanowych zmieniających stan bazy danych można traktować jako jedną niepodzielną jednostkę pracy
  - menedżer odtwarzania może w razie potrzeby wywołać wycofanie, za pomocą operacji UNDO
- Spójność (Consistency) odnosi się do faktu, że transakcja, jeśli jest wykonywana w izolacji, powoduje przekształcenie bazy danych z jednego spójnego stanu w inny spójny stan
  - odpowiedzialność dewelopera
  - również nadrzędna odpowiedzialność systemu zarządzania transakcjami DBMS

Własności ACID transakcji

- Izolacja (Isolation) oznacza, że w sytuacjach, gdy wiele transakcji jest wykonywanych jednocześnie, wynik powinien być taki sam, jak gdyby każda transakcja była wykonywana oddzielnie
  - odpowiedzialność za mechanizmy kontroli współbieżności DBMS, koordynowane przez scheduler
- Trwałość (Durability) odnosi się do faktu, że skutki zatwierdzonej transakcji powinny być zawsze na trwale zapisywane w bazie danych
  - Odpowiedzialność managera odzyskiwania (np. przez operacje REDO lub redundancję danych)