BAZY DANYCH
8

GRZEGORZ SURÓWKA

2018

PLAN WYKŁADU

1. WPROWADZENIE
2. DBMS, MODELE DANYCH, MODEL RELACYJNY
3. DIAGRAMY E/R
4. E/R → MODEL RELACYJNY
5. ALGEBRA RELACJI
6. ZŁĄCZENIA
7. NORMALIZACJA BAZY DANYCH
8. **TRANSAKCJE**
9. OPTYMALIZACJA ZAPYTAŃ
10. WYDAJNOŚĆ
11. PODZAPYTANIA SQL
12. GRUPOWANIE
13. WIDOKI, KURSORY, WYZWALACZE, SEKWENCJE
14. SQL

PLAN

- POTRZEBA TRANSAKCJI
- ACID
- WSPÓŁBIEŻNOŚĆ
- MODEL PRZETWARZANIA
- SQL

TRANSAKCJE

definicje

- zadanie to minimalna (niepodzielna) jednostka przetwarzania
- transkacja to grupa zadań

ogólne potrzeby wprowadzenia transakcji

- baza danych modeluje świat rzeczywisty - zmiany zachodzące w świecie rzeczywistym muszą być zakodowane, by transformować bazę danych z jednego stanu spójnego do innego stanu spójnego
- niebezpieczeństwa związane z transformowaniem bazy danych
  - ✓ awaryjność środowiska sprzętowo-programowego: niezakończone operacje, buforowanie danych
  - ✓ współbieżny dostęp do danych: może naruszać spójność bazy lub generować niepoprawne wyniki
  - ✓ rozproszenie baz danych

szczególne potrzeby wprowadzenia transakcji

- pojedynczy użytkownik
  - ✓ ochrona szczególnie wrażliwych fragmentów
  - ✓ transakcja wykonuje się albo w całości, albo wcale
  - ✓ jeżeli w trakcie wykonywania transakcji wystąpi jakiś błąd, całą sekwencję operacji można odwołać, przywracając bazę do stanu sprzed rozpoczęcia tej sekwencji
- system wielodostępny
  - ✓ obowiązuje wszystko to, co dla pojedynczego użytkownika
  - ✓ różne procesy klienckie odwołujące się do tych samych tabel nie mogą się ze sobą kłócić

przykład

KLIENT 1	KLIENT 2	Wolne miejsce
Czy są wolne miejsca?		1
	Czy są wolne miejsca?	1
Oferuje miejsce		1
	Oferuje miejsce	1
Pytanie o kartę kredytową lub konto		1
	Pytanie o kartę kredytową lub konto	1
Podaje numer karty	Podaje numer konta	1
Autoryzacja		1
	Autoryzacja	1
Przypisanie miejsca		1
	Przypisanie miejsca	1
Obciążenie karty		1
	Obciążenie karty	1
Zmniejszenie liczby wolnych miejsc		0
	Zmniejszenie liczby wolnych miejsc	-1

przykład

- bez synchronizacji "równoległych" procesów: X=7

CZAS	PROCES A	PROCES B	WARTOŚĆ X_A	WARTOŚĆ X_B
1	Czyta X		5	
2		Czyta X	5	5
3	X=X+1		6	5
4		X=X+1	6	6
5	Zapisuje X		6	6
6		Zapisuje X	6	6

uwagi praktyczne

- należy unikać transakcji wtedy, gdy wymagana jest interakcja z użytkownikiem - należy najpierw zebrać wszystkie dane, a dopiero potem rozpoczynać transakcję lub blokować tabele
- długo działające transakcje blokują dostęp innych użytkowników do danych, na których działa transakcja, dopóki nie zostanie ona zatwierdzona lub odwołana

ACID

własności bazy danych gwarantujące:
dokładność, zupełność i nienaruszalność danych

Atomowość (Atomicity)

- transakcja musi być traktowana jako zadanie atomowe tzn. niepodzielne
  (wykonuje się wszystkie jej operacje, albo żadne)
- nie może być stanu bazy, w którym transakcja jest wykonana częściowo
- legalne stany bazy: przed wykonaniem transakcji, po wykonaniu transakcji, po odstąpieniu od transakcji, po awarii transakcji

Spójność (Consistency)

- transakcja nie może naruszać integralności danych
  (więzów narzuconych na dane w tabelach)
- transakcja przeprowadza bazę danych z jednego stanu spójnego do innego stanu spójnego - w trakcie wykonywania transakcji baza danych może być przejściowo niespójna
- po transakcji baza musi pozostać spójna
- żadna transakcja nie może szkodliwie wpływać na dane w bazie

Separacja (Isolation)

- własność odseparowania stanów różnych transakcji wykonywanych współbieżnie
- transakcje są autonomiczne i nie wpływają na siebie
- transakcje oddziałują na siebie poprzez dane - jedna transakcja nie może widzieć wyników działania jakiejś innej, niezatwierdzonej transakcji
- transakcja musi odbywać się tak, jakby żadna inna transakcja nie miała miejsca w tym samym czasie
- mimo współbieżnego wykonywania, transakcje widzą stan bazy danych tak, jak gdyby były wykonywane w sposób sekwencyjny

Trwałość (Durability)

- wyniki zatwierdzonych transakcji nie mogą zostać utracone w wyniku wystąpienia awarii lub restartu systemu
- jeśli transakcja zmienia jakieś dane w bazie, po zatwierdzeniu (commit) dane te są trwale zmienione
- jeśli nastąpi awaria po zatwierdzeniu transakcji, a dane nie zostały zapisane na dysku, wtedy dane zostaną zapisane/zmienione po przywróceniu systemu

przykład z przelewem

- transakcja Atomowa: jeżeli pieniądze zostaną poprawnie przetransferowane z konta A do B
- transakcja Spójna: jeżeli kwota odjęta z konta A jest równa kwocie dodanej do konta B
- transakcja Izolowana: jeżeli inne transakcje wykonywane współbieżnie, czytające i modyfikujące konta A i B, nie mają wpływu na transakcję
- transakcja Trwała: jeżeli po zakończeniu transakcji, baza danych trwale odzwierciedla nowe stany kont A i B

WSPÓŁBIEŻNOŚĆ

problemy związane z wielodostępnością

- niespójność odczytów: jedna transakcja może odczytać dane zmieniane przez drugą transakcję, chociaż transakcja ta nie zatwierdziła jeszcze zmian
- niepowtarzalność odczytów: transakcja odczytuje dane, nieco później odczytuje je ponownie, a odczytane dane są inne, mimo iż transakcja odczytująca nie została jeszcze zatwierdzona
- odczyty fantomowe: jedna tabela dodaje wiersz, druga transakcja aktualizuje wiersze - nowy wiersz powinien być zaktualizowany, a nie jest

realizacja izolacji

- w celu zapewnienia izolacji w systemie wielodostępnym, transakcje blokują tabele (fragmenty tabel), które są im potrzebne
- najczęściej stosowane mechanizmy blokad: 2PL, OCC
- 2PL jest efektywnie szybszy niż OCC, jeśli konflikty są częste
- aby zminimalizować blokowanie (i opóźnienie innych transakcji), stosuje się mechanizm OCC

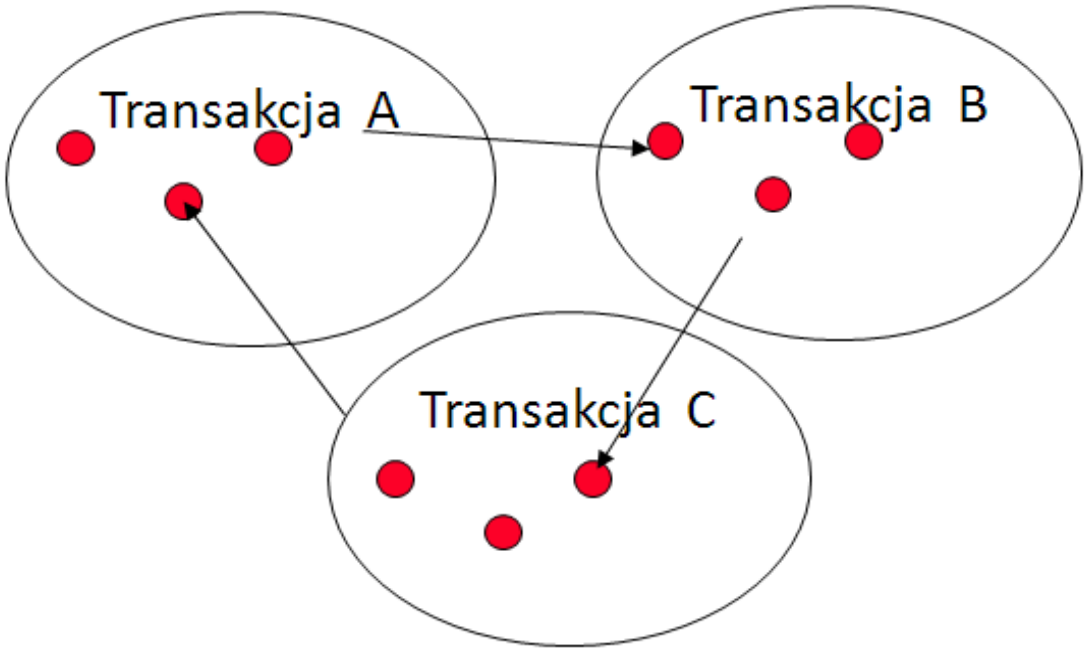
poziomy izolacji ANSI

poziom izolacji	niespójność odczytów	niepowtarzalność odczytów	odczyty fantomowe
read uncommitted	OK	OK	OK
read committed	NIE	OK	OK
repeatable read	NIE	NIE	OK
serializable	NIE	NIE	NIE

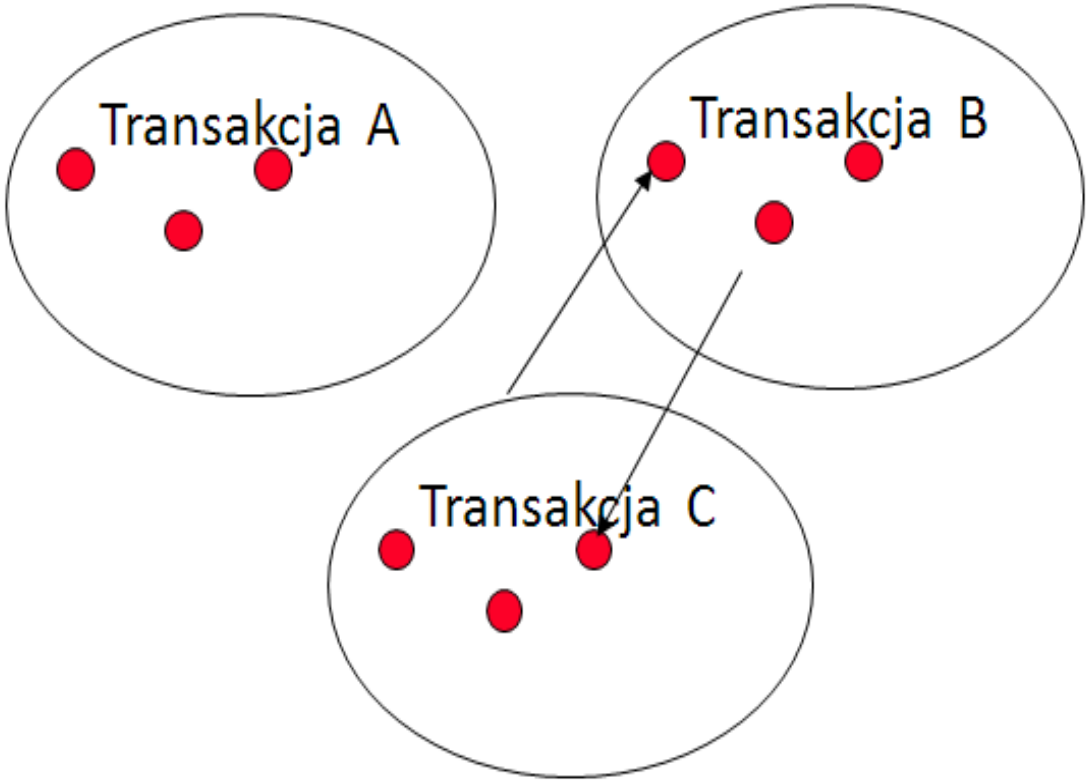
zakleszczenie

- jest efektem wprowadzenia zamków
- przykład dwóch transakcji
  Transakcja A zablokowała zasób X
  i żąda dostępu do zasobu Y.
  Transakcja B zablokowała zasób Y
  i żąda dostępu do zasobu X.
  Ani A, ani B nie mogą dalej kontynuować
  jakiejkolwiek akcji. (System "zawiesił się")

- przykład trzech transakcji



- zakleszczyły się dwie transakcje, pozostałe działają normalnie



walka z zakleszczeniem

- wykrywanie zakleszczeń i rozrywanie pętli zakleszczenia
  - ✓ detekcja zakleszczenia wymaga skonstruowania grafu czekania (Wait-For-Graph) i tranzytywnego domknięcia tego grafu (złożoność gorsza niż n*n)
  - ✓ rozrywanie pętli zakleszczenia polega na wybraniu transakcji "ofiary" uczestniczącej w zakleszczeniu i następnie, jej zerwaniu i uruchomieniu od nowa
  - ✓ wybór "ofiary" podlega różnym kryteriom, np. najmłodsza, najmniej pracochłonna, o niskim priorytecie, itd.

- niedopuszczanie do zakleszczeń
  - ✓ wstępne żądanie zasobów (Preclaiming)
  - ✓ czekaj-umieraj (WD, Wait-Die)

metody bez zakleszczeń oparte na stemplach czasowych

- każda transakcja w momencie startu dostaje unikalny stempel czasowy (po stemplach można rozróżniać transakcje)
- żadna zmiana nie jest nanoszona do bazy danych - transakcja działa na swoich własnych kopiach, aż do potwierdzenia
- każdy obiekt bazy danych przechowuje 2 stemple czasowe: transakcji, która ostatnio brała obiekt do czytania i transakcji, która ostatnio brała obiekt do modyfikacji
- w momencie potwierdzenia sprawdza się stemple transakcji oraz wszystkich pobranych przez nią obiektów

- można dość łatwo wyprowadzić reguły zgodności, np.
  - ✓ jeżeli obiekt był aktualizowany i stempel na obiekcie do aktualizacji jest taki sam jak stempel transakcji, to OK; inaczej zerwij i uruchom od nowa
  - ✓ jeżeli obiekt był tylko czytany i stempel na obiekcie do aktualizacji jest starszy niż stempel transakcji, to OK; inaczej zerwij i uruchom od nowa
- metoda jest o tyle przyjemna, że zerwanie transakcji oznacza zlikwidowanie tylko lokalnych kopii obiektów - nie potrzeba nic robić w bazie danych

metody optymistyczne

- zasada podobna jak w przypadku stempli czasowych, ale transakcje nie działają na swoich własnych kopiach, ale bezpośrednio na bazie danych
- "optymizm" polega na założeniu, że żadnych konfliktów nie będzie
- konflikty są jednak wykrywane i w przypadku ich wykrycia, transakcje są zrywane, a baza danych jest cofana do poprzedniego stanu
- metody optymistyczne są bardzo nieskuteczne, jeżeli konfliktów jest dużo: praktycznie system staje
- rzadko stosowane
- wymagają one również pewnej formy zamków i mogą prowadzić do zakleszczeń

OCC (Optimistic Concurrency Control)

- wiele transakcji może odczytywać i modyfikować fragment danych bez zakładania blokad
- transakcje zapamiętują historię dokonywanych odczytów i zapisów
- przed zatwierdzeniem transakcja sprawdza historię w celu wykrycia ewentualnych konfliktów z innymi transakcjami
- jeśli jakieś konflikty zostaną wykryte, jedna z transakcji wywołujących konflikt zostaje odwołana
- działa dobrze, gdy konflikty są rzadkie,
  ale może jednak wygenerować duży koszt,
  jeżeli konflikty nie są rzadkie

ziarnistość mechanizmu transakcji (granularity)

- grube ziarna (baza danych, relacja)
  → mały stopień współbieżności
- małe ziarna (np. element krotki) → pracochłonność zakładania zamków, duże zapotrzebowanie na dodatkową pamięć na zamki, przeciążenie sieci
- ziarnistość trzeba zgrać z innymi mechanizmami, np. buforowaniem w RAM
- testy dla systemów relacyjnych pokazują, że optymalną ziarnistość zapewnia strona fizyczna (to uniemożliwia bardziej "inteligentne" mechanizmy przetwarzania transakcji, które "rozumieją" semantykę tego, co blokują (np. predicate locking))
- zmienna ziarnistość w praktyce nie jest implement.

typy awarii

- zerwanie transakcji: z różnych powodów, wewnętrznych (np. czekaj-umieraj) lub zewnętrznych
  - ✓ transakcja może być powtarzana automatycznie lub (jeżeli to niemożliwe) stracona
  - ✓ program wywołujący transakcje powinien przewidywać sytuację, że transakcja może być zerwana, wykryć to i ewentualnie powtórzyć
- awaria systemu, ze stratą zawartości pamięci operacyjnej
- awaria nośników pamięci, ze stratą zawartości dysku

- absolutna niezawodność → absolutna złożoność, nieskończony koszt (nawet z transakcjami absolutna niezawodność jest niemożliwa)
  rozsądna niezawodność → mały wpływ na wydajność
- transakcje rozsądnie podwyższają niezawodność, nie prowadząc do nadmiernych obciążeń czasu wykonania ani dodatkowego zapotrzebowania na pamięć

odtwarzanie po awarii

- zrzut (back-up): cykliczne przegrywanie zawartości bazy danych lub jej najważniejszych fragmentów na inny nośnik, cykl: co 30 min (banki), na koniec dnia (instytucje), co tydzień (uniwersytety), itp.
- odtwarzanie (recovery): pół-automatyczne lub automatyczne stanu bazy danych po awarii na podstawie ostatniego backup'u i ew. logu
- dziennik (log): rejestracja wszystkich operacji zachodzących na bazie danych wraz ze zmienianymi obiektami, z dokładnością do transakcji, które to robią

- cofanie (rollback): cofanie operacji w bazie danych na podstawie logu, np. po zerwaniu transakcji
- punkty kontrolne (checkpoint): migawkowe zdjęcie stanu przetwarzania (trwałych i nietrwałych obiektów, stanu sterowania, itd.) stosowane w niektórych systemach celem powrotu do poprzedniego stanu, o ile programista/użytkownik tego sobie zażyczył
- replikacja: zdublowanie informacji na fizycznie i geograficznie oddzielonych nośnikach, celem zwiększenia niezawodności i zmniejszenia kosztów transmisji

odtwarzanie po zerwaniu

- warunek atomowości transakcji wymaga, aby w przypadku zerwania wszelkie wykonane przez nią czynności zostały odwrócone: baza danych ma wrócić do stanu sprzed transakcji
- strategia 1
  - ✓ transakcje działają na własnych kopiach, które podmieniają z oryginalnymi obiektami w fazie commit
  - ✓ wady: zwiększone zapotrzebowanie na pamięć, większa możliwość awarii w fazie commit (bardzo niebezpieczne)

- strategia 2
  - ✓ transakcje działają bezpośrednio na bazie danych, ale w specjalnym pliku zwanym dziennikiem (log) zapisują wszelkie operacje aktualizacyjne których dokonały, wraz z obiektami przed i ew. po aktualizacji
  - ✓ w razie zerwania - baza danych jest odtwarzana do poprzedniego stanu poprzez czytanie logu "od tyłu"

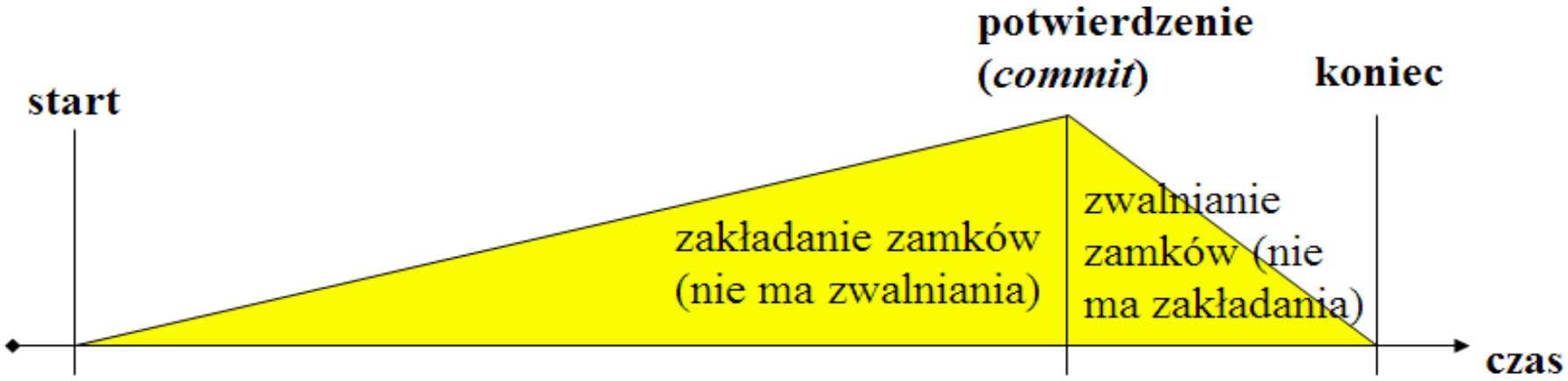
- strategia 3
  - ✓ write ahead log: w dzienniku zapisywane są zmiany przed ich dokonaniem
  - ✓ w razie awarii wiadomo, co było zrobione, czyli możliwe jest poprawne cofnięcie
- czasami stosuje się tryb oszczędny (SQL) , w którym zmiany są zaznaczane w dzienniku, ale nie są naniesione w bazie danych - prowadzi to do tego, że transakcja "nie widzi" własnych zmian, co jest sporą uciążliwością przy programowaniu i prowadzi do błędów

podstawowe algorytmy z zamkami

Dwu-Fazowe Blokowanie (2PL, [Strict] Two-Phase Locking)

- każda transakcja zakłada blokadę na każdy rekord, który chce odczytać, przed dokonaniem odczytu
- blokady do odczytu mogą być współdzielone z innymi transakcjami
- każda transakcja zakłada też wyłączną blokadę na każdy fragment danych, który chce zapisać
- wszystkie blokady są utrzymywane aż do zakończenia transakcji
- system prowadzi dziennik i graf czekania, umożliwiając odtworzenie po zerwaniu i przeciwdziałając zakleszczeniu

- implementacja 2PL
  - ✓ zamek wyłączny X (Exclusive lock): transakcja zablokowuje jakikolwiek dostęp do obiektu dla innych transakcji
  - ✓ zamek dzielony S (Shared lock): inne transakcje mogą czytać, ale nie mogą modyfikować
  - ✓ szeregowalność jest zachowana, jeżeli cała transakcja ma dwie fazy: rosnącą i skracającą (awarie są możliwe w obydwu fazach, ale faza commit jest szybka (milisekundy))



- założenia
  - ✓ blokadę S (czytanie) dla tej samej danej może uzyskać dowolna liczba transakcji
  - ✓ blokadę X (zapis, usunięcie) dla konkretnej danej może uzyskać tylko jedna transakcja
  - ✓ S i X nie mogą być jednocześnie założone na tę samą daną dla dwóch różnych transakcji
  - ✓ zdejmowanie blokad następuje natychmiast po zakończeniu operacji
  - ✓ dana zablokowana przez S jest po odczycie natychmiast odblokowywana
  - ✓ dana zablokowana przez X jest po modyfikacji natychmiast zamieniana na blokadę S
  - ✓ blokada dla operacji modyfikacji jest całkowicie usuwana po zatwierdzeniu transakcji

Czekasz-umieraj (WD, Wait-Die)

- jak w 2PL, ale jeżeli transakcja próbuje dostać się do zasobu, który jest zablokowany, to natychmiast jest zrywana i powtarzana od nowa
- metoda może być nieskuteczna dla systemów interakcyjnych (użytkownik może się denerwować koniecznością dwukrotnego wprowadzania tych samych danych) oraz prowadzi do spadku efektywności (sporo pracy idzie na marne)
- odmiany algorytmu: z dwóch transakcji w konflikcie ginie młodsza / starsza / ta, która się mniej napracowała / ta, która jest mniej ważna, itd.

Dynamiczne Dwufazowe Blokowanie bez podwyższania zamków

- tak jak 2PL, ale nie ma podwyższania zamków z S do X

Wstępne Żądanie Zasobów (Preclaiming)

- przed uruchomieniem każda transakcja określa potrzebne jej zasoby, później nie może nic żądać
- wada: zgrubne oszacowanie żądanych zasobów prowadzi do zmniejszenia stopnia współbieżności

transakcje zagnieżdżone

- powody: ułatwienie programowania, dostarczenie nowego mechanizmu abstrakcji
- programista może kilka transakcji zamknąć w jedną większą bez zmiany tekstu programu
- programista może większą transakcję rozbić na mniejsze bez zmiany koncepcji programu
- filozofie
  - ✓ zagnieżdżona transakcja jest potwierdzana wyłącznie dla swojej macierzystej transakcji, przez co aktualizacje zrobione przez pod-transakcję stają się widoczne dla innych pod-transakcji
    - ❖ zamki pod-transakcji są dziedziczone przez jej matkę

- ostateczne potwierdzenie pod-transakcji i zwolnienie zamków następuje po potwierdzeniu transakcji stojącej najwyżej w hierarchii

✓ każda pod-transakcja po potwierdzeniu bezpośrednio wprowadza zmiany do bazy danych

- zamki nie są dziedziczone (są zwalniane)
- każda pod-transakcja posiada bliźniaczą pod-transakcję kompensującą, która określa co robić, jeżeli transacja-matka nie zostanie potwierdzona
- ta filozofia jest także określana jako saga

długie transakcje (transakcje projektowe)

- jeżeli transakcje są bardzo długie wówczas klasyczne własności ACID są niewygodne
- przykład
  Załóżmy, że kilku autorów opracowuje ten sam projekt. Czy dopuszczalna jest sytuacja, że koledzy nie mają możliwości obejrzenia jakiegoś fragmentu projektu, ponieważ ostatni edytor, nie dokończył sprawy i poszedł na dłuższy urlop?
- długie transakcje wymagają osłabienia warunku izolacji lub jakiegoś innego pojęcia szeregowalności - pewnym rozwiązaniem są poziomy izolacji w SQL
- rozwiązanie trywialne: specjalny tryb czytania obiektu, który omija nałożone na niego zamki (tzw. "kuchenne drzwi", backdoor)

zabezpieczenie transakcji

szeregowalność (serialization)

- oznacza, że wynik sekwencji przeplatających się działań, wykonywanych przez zatwierdzane transakcje, musi ściśle odpowiadać sytuacji, w której wszystkie transakcje wykonywane są kolejno, jedna po zakończeniu drugiej

plan (schedule)

- chronologiczna sekwencja transakcji, każda transakcja składa się z pewnej liczby zadań

plan szeregowy (serial schedule)

- transakcje poukładane są wg. kolejności
- kiedy jedna transakcja się skończy, zostaje wykonana następna
- w środowiskach wielo-transakcyjnych ten sposób jest uważany za odnośnik
- jeśli transakcje są niezależne i operują na różnych danych, to ich porządek nie ma znaczenia, ale jeśli dwie transakcje operują na tych samych danych, to plan szeregowy może doprowadzić do straty zgodności bazy → dlatego dopuszcza się przetwarzanie równoległe transakcji (jeśli transakcje są szeregowalne lub istnieje relacja równoważności pomiędzy nimi)

typy planów równoważności

równoważność wyniku

- definicja: jeśli dwa plany produkują ten sam wynik
- równoważność ta nie jest istotna, ponieważ identyczny wynik dla pewnych danych nie oznacza identyczności wyników dla innych

równoważność widoku

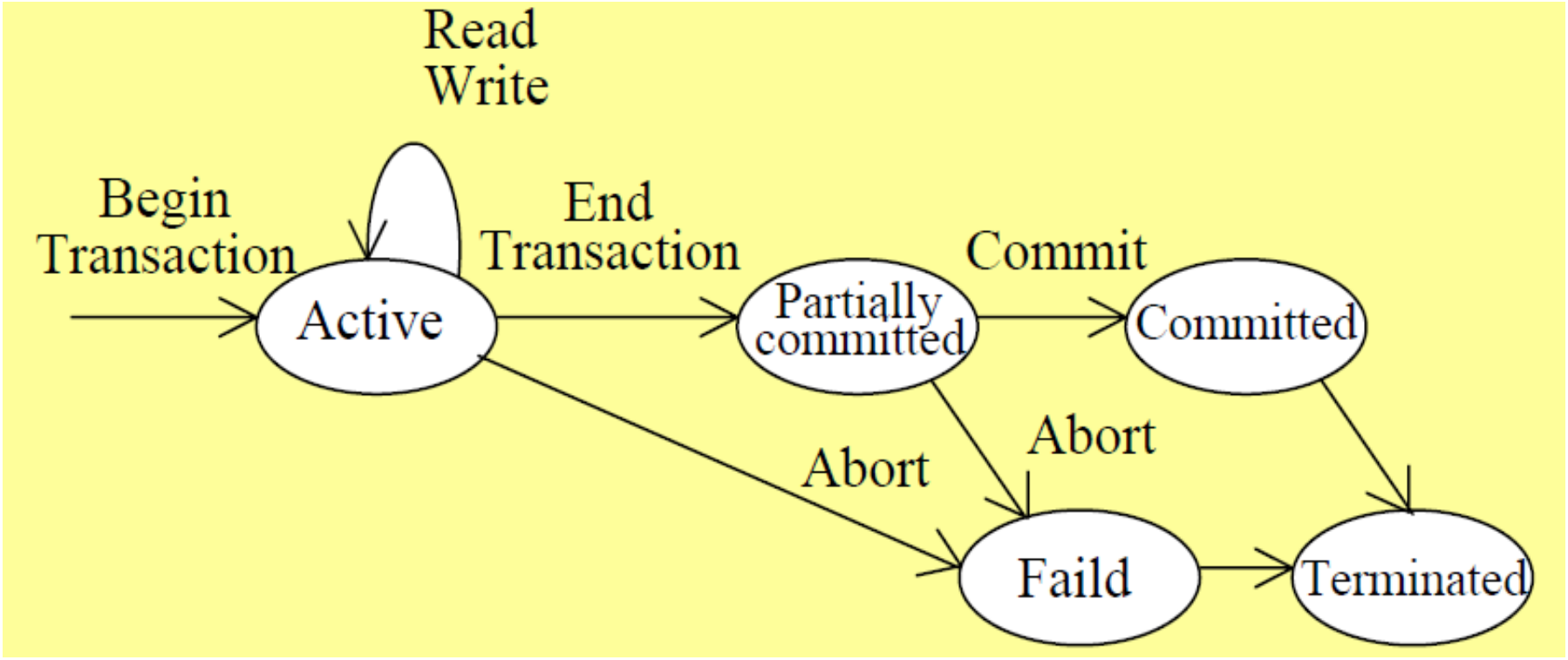
- definicja: jeśli transakcje w obu planach wykonują podobne działania w podobny sposób
- przykład
  jeśli T czyta dane początkowe w S1,
  to czyta także dane początkowe w S2
  jeśli T czyta wartość wpisaną przez J do S1,
  to czyta także wartość wpisaną przez J do S2
  jeśli T wykonuje ostateczny wpis na danych w S1,
  wykonuje też ostateczny wpis na danych w S2

równoważność kolizyjna

- dwa plany będą kolizyjne jeśli
  - ✓ należą do oddzielnych transakcji
  - ✓ mają dostęp do tych samych danych
  - ✓ przynajmniej jeden z nich wykonuje operację zapisu
- definicja: dwa plany mające wiele transakcji o operacjach kolizyjnych są w konflikcie wtedy i tylko wtedy, gdy oba plany zawierają ten sam zbiór transakcji
- porządek kolizyjnych par operacji jest zachowany w obu planach

wszystkie plany o równoważności kolizyjnej są również planami o równoważności widoku

STANY TRANSAKCJI



- aktywna (active): wykonywanie transakcji, stan początkowy każdej transakcji
- częściowo dokonana (partially committed): transakcja wykonuje ostatnią operację
- nieudana (failed): jeśli jakikolwiek test przeprowadzony przez DBMS wyjdzie negatywnie; taka transakcja nie może być kontynuowana
- zatrzymana (aborted): jeśli transakcja jest w stanie "nieudana", to DBMS cofa wszystkie operacje zapisu i przywraca bazę do stanu sprzed transakcji, po tym DBMS zazwyczaj wykonuje jedną z dwóch operacji: restart transakcji lub usunięcie transakcji
- dokonana (committed): transakcja wykonuje wszystkie operacje, efekty transakcji są zapisane na stałe w bazie

dwufazowe zatwierdzanie (2-Phase Commit)

- jest stosowane dla zapewnienia własności atomowości transakcji rozproszonych
- **Faza 1**: (Prepare) węzły (serwery) biorące udział w transakcji przygotowują się do jej zatwierdzenia (na żądanie koordynatora transakcji)
- **Faza 2**: (Commit) gdy wszystkie węzły są gotowe do zatwierdzenia transakcji następuje jej rzeczywiste zatwierdzenie

niejawne przetwarzanie transakcji

- automatycznie zatwierdzanie transakcji (COMMIT)
  - ✓ instrukcje DDL (Data Description Language), czyli instrukcje tworzące i usuwające bazy oraz tworzące, usuwające i modyfikujące tabele nie są "transakcyjne" - nie można ich wycofać
  - ✓ instrukcje DCL (Data Control Language)
  - ✓ wyjście bez jawnego COMMIT/ROLLBACK
- automatyczne unieważnienie transakcji (ROLLBACK)
  - ✓ nieprawidłowe zakończenie lub błąd systemowy

poziomy transakcji

- transakcja logiczna: zbiór poleceń języka SQL (select, insert, update, delete, commit, rollback)
- transakcja fizyczna: na poziomie systemu zarządzania bazą danych (DBMS) - składa się z operacji elementarnych (rozpoczęcia transakcji, zaalokowania zasobów, blokowania danych, manipulacji na danych, kończenia transakcji, zwalniania zasobów systemowych)

stan danych przed COMMIT/ROLLBACK

- można odtworzyć ostatni stan danych
- użytkownik bazy może przeglądnąć skutki operacji DML (Data Manipulation Language) za pomocą polecenia SELECT
- inni użytkownicy nie mogą przeglądnąć skutków operacji DML bieżącego użytkownika
- wiersze (krotki) dotknięte zmianą są zablokowane - inni użytkownicy nie mogą zmienić danych z zablokowanych krotek

stan danych po COMMIT

- zmiany w bazie stają się trwałe
- poprzedni stan bazy jest trwale stracony
- wszyscy użytkownicy mogą przeglądnąć zmiany
- blokady ze zmienionych krotek są zwolnione – krotki te stają się dostępne do zmiany przez innych użytkowników
- punkty kontrolne zostają usunięte

```
DELETE FROM employees WHERE employee_id = 99999;
INSERT INTO departments VALUES (290, 'Corporate Tax',
NULL, 1700);
COMMIT
```

stan danych po ROLLBACK

- odrzucenie wszystkich oczekujących zmian
- przywrócenie poprzedniego stanu danych
- zwolnienie blokad z krotek,
  które podlegały zmianom

DELETE FROM emploees
ROLLBACK

MODEL TRANSAKCJI

reprezentacja

- Transakcją T_i nazywamy uporządkowaną parę:
  (zbiór operacji na bazie danych, relacja częściowego porządku na zbiorze tych operacji)
  $\overline{T_j}=(T_i, <T_j)$
- zbiór operacji: $\overline{T_i}$ ={r, w, c, a}
  r=odczyt,
  w=zapis,
  c=zatwierdzenie transakcji,
  a=wycofanie transakcji

- notacja:

  $r_i(x)$ lub $r_i(x, \text{wartość})$
  odczyt danej x przez i-tą transakcję

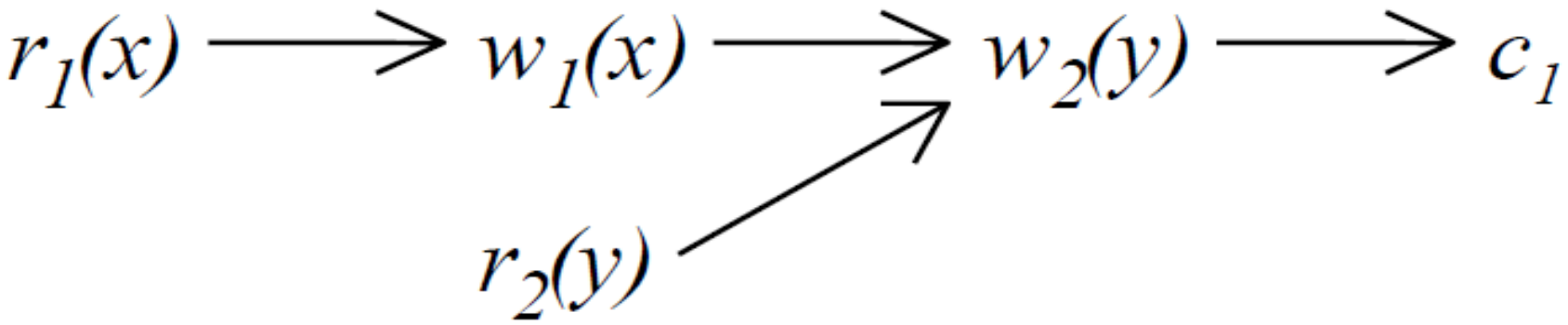
  $w_i(x)$ lub $w_i(x, \text{wartość})$
  zapis danej x przez i-tą transakcję

  c_i zatwierdzenie i-tej transakcji

  a_i wycofanie i-tej transakcji

  x: jednostka danych na różnym poziomie granulacji (dana elementarna, krotka, zbiór krotek wyznaczony przez warunek, tabela bazy danych)

- każda transakcja może być reprezentowana przez graf skierowany G=(V, A), gdzie:
  V: zbiór węzłów odpowiadających operacjom transakcji T_i
  A: zbiór krawędzi reprezentujących porządek na zbiorze operacji



klasyfikacja transakcji

- ze względu na porządek operacji: transakcja sekwencyjna, transakcja współbieżna
- ze względu na zależność operacji: transakcja zależna od danych, transakcja niezależna od danych
- ze względu na typy operacji: zapytania lub transakcja odczytu (read only), transakcja aktualizująca - transakcja (read/write)

realizacje transakcji

- w jednym systemie bazy danych działa równocześnie wiele transakcji - należy zapewnić, aby transakcje te były wykonane w takiej kolejności, która nie wprowadzi danych niepoprawnych
- definicja: Realizacja (Historia) transakcji to częściowo uporządkowana sekwencja operacji
- definicja:
  Realizacja S zbioru n transakcji T_1, T_2, ..., T_n to takie uporządkowanie operacji współbieżnie wykonywanych transakcji, w którym, dla każdej transakcji T_i w realizacji S, porządek wykonania operacji transakcji T_i jest taki sam jak częściowy porządek w zbiorze operacji transakcji $<T_i$

$S(\tau)=(\overline{T}_r(\tau), <r)$

zbiór operacji wszystkich transakcji

$\overline{T}_r(\tau)$ należących do zbioru τ

$<r$ relacja częściowego porządku na zbiorze $\overline{T}_r(\tau)$

dla dowolnej pary operacji o_i, o_j należących do zbioru operacji transakcji ze zbioru τ takich, że żądają one dostępu do tej samej danej i co najmniej jedna z nich jest operacją zapisu, zachodzi $o_{i<r}\ o_j$ lub $o_{j<r}\ o_i$

- definicja:

  Zaakceptowana projekcja to realizacja zawierająca tylko operacje zatwierdzonych transakcji

  (dalsze rozważania dotyczyć będą tyko realizacji spełniających powyższy warunek)

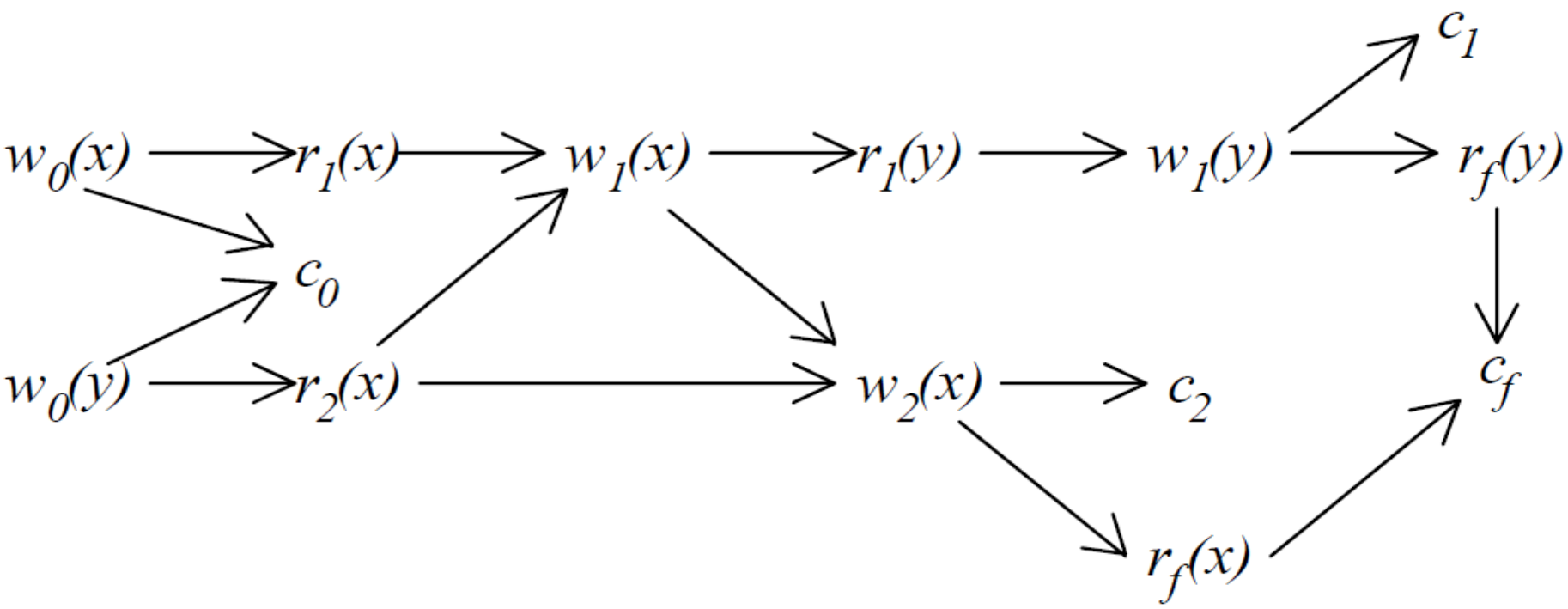
  przykład: r: $w_0(x), w_0(y), c_0, r_1(x), r_2(x), w_1(x), r_1(y), w_2(x), c_2, w_1(y), c_1, r_f(x), r_f(y), c_f$

- dowolną realizację można przedstawić w postaci grafu skierowanego, nazywanego grafem realizacji,

$$GR(s(\tau))=(V, A)$$

węzły grafu odpowiadają operacjom ze zbioru $Tr(\tau)$, natomiast krawędzie grafu reprezentują relację częściowego porządku $< r$

- definicja:
  Realizacja jest sekwencyjna jeżeli, dla każdych dwóch transakcji, wszystkie operacje jednej z nich poprzedzają wszystkie operacje drugiej
- definicja:
  Realizacja jest współbieżna jeżeli, dla każdych dwóch transakcji, nie zachodzi powyższe



stan bazy danych

- reprezentuje zbiór wartości wszystkich danych w bazie

obraz bazy danych widziany przez transakcję T_i

- to zbiór wartości danych odczytywanych przez transakcję T_i

uszeregowalność realizacji

- Założenie 1:
  każda realizacja sekwencyjna jest poprawna
- Założenie 2:
  każda realizacja współbieżna równoważna dowolnej realizacji sekwencyjnej tego samego zbioru transakcji jest również poprawna

przykład

- Dane (początkowe wartości): a=50; b=50
  Transakcja T1: sumuje konta a i b
  Transakcja T2: przelewa 30 z konta a na konto b
- Dana realizacja postaci:

s: ...$r_2(a, 50)$ $w_2(a, 20)$ $r_1(a,20)$ $r_1(b, 50)$ $r_2(b,50)$ $w_2(b, 80)$ c_1 c_2

Czy dana realizacja jest poprawna?

Nie jest poprawna ponieważ obraz bazy danych widziany przez T1: a+b=70, zamiast 100

- Realizacje sekwencyjne transakcji T1 i T2:

s_1:.$r_1(a, 50)$ $r_1(b, 50)$ c_1 $r_2(a, 50)$ $w_2(a, 20)$ $r_2(b, 50)$ $w_2(b, 80)$ c_2..

Końcowy stan bazy danych: a= 20; b= 80

Obraz bazy danych widziany przez T2: a = 50; b = 50

Obraz bazy danych widziany przez T1: a = 50; b = 50

konflikt

- dwie operacje $o_i(x)$, $o_j(y)$ współbieżnej realizacji są konfliktowe, wtedy i tylko wtedy, gdy są spełnione następujące trzy warunki:
  1. $x = y$ Operacje na różnych danych nigdy nie są konfliktowe
  2. $i \neq j$ Operacje konfliktowe muszą należeć do różnych transakcji
  3. Jedna z dwóch operacji o_i lub o_j musi być operacją zapisu
- dwie transakcje T_i, T_j są konfliktowe, jeżeli zawierają wzajemnie konfliktowe operacje

- definicja: (relacja poprzedzania dla operacji)
  Operacja $o_i(x)$ poprzedza operację $o_j(y)$
  w realizacji $r(\tau)$, co zapisujemy jako $o_i(x)o_j(y)$,
  jeżeli operacje te są konfliktowe i $o_i(x) <_r o_j(y)$
- następujące pary operacji mogą znajdować się
  w konflikcie:
  $r_i(x)\ w_j(x)$
  $w_i(x)\ r_j(x)$
  $w_i(x)\ w_j(x)$

konfliktowa równoważność

- definicja: (relacja poprzedzania dla transakcji)
  Transakcja T_i poprzedza transakcję T_j w realizacji r(τ), co zapisujemy jako $T_i \rightarrow T_j$, jeżeli zawierają odpowiednio operacje $o_i(x)$ i $o_j(x)$, między którymi zachodzi związek poprzedzania
- definicja:
  Dwie realizacje r(τ)=(Tr(τ) , <r) i r'(τ)=(Tr(τ) , <r') są konfliktowo równoważne, jeżeli dla każdej pary operacji $o_i(x)$ i $o_j(y)$ w realizacji r(τ), takich, że $o_i(x) \rightarrow o_j(x)$, zachodzi również $o_i(x) \rightarrow o_j(y)$ w realizacji r'(τ)

kryterium konfliktowej uszeregowalności

- definicja:
  Realizacja r(τ) zbioru transakcji τ jest konfliktowo uszeregowalna wtedy i tylko wtedy, gdy jest ona konfliktowo równoważna dowolnej sekwencyjnej realizacji τ
- definicja:
  Grafem konfliktowej-uszeregowalności realizacji r(τ) nazywamy skierowany graf CSRG(r(τ))=(V, A) taki, w którym zbiór wierzchołków V odpowiada transakcjom ze zbioru, natomiast zbiór krawędzi $A = \{(T_i, T_j) : T_i \rightarrow T_j \}$

- tw: (o konfliktowej uszeregowalności)
  Realizacja r(τ) zbioru transakcji jest konfliktowo-uszeregowalna wtedy i tylko wtedy, gdy jej graf konfliktowej uszeregowalności CSRG(r(τ)) jest acykliczny, dowód:
  (wynika bezpośrednio z własności spójności transakcji)
  - ✓ każda transakcja transformuje bazę danych z jednego stanu spójnego do innego stanu spójnego
  - ✓ stąd wynika, że każda realizacja sekwencyjna zbioru transakcji zachowuje spójność bazy danych, gdyż jest ona sekwencją transformacji odwzorowujących bazę danych z jednego do innego stanu spójnego

- z definicji grafu konfliktowej uszeregowalności wynika, że graf ten, dla dowolnej realizacji sekwencyjnej, musi być acykliczny
- z definicji kryterium konfliktowej uszeregowalności wynika, że dowolna realizacja współbieżna jest poprawna, jeżeli jest ona równoważna dowolnej realizacji sekwencyjnej tego samego zbioru transakcji
- z definicji równoważności realizacji wynika, że graf konfliktowej uszeregowalności realizacji współbieżnej musi być również acykliczny, jeżeli realizacja ta jest konfliktowo-uszeregowalna

realizacje odtwarzalne

- własność uszeregowalności gwarantuje poprawność dowolnej realizacji transakcji, w szczególności, gwarantuje wolność od anomalii współbieżnego wykonywania transakcji jeżeli rozważamy wyłącznie realizacje będące zaakceptowanymi projekcjami, tj. realizacje zawierające tylko operacje zatwierdzonych transakcji
- w rzeczywistości, realizacje zawierają nie tylko operacje zatwierdzonych transakcji, ale transakcje są wycofywane przez użytkowników, podlegają awariom, są wycofywane przez system, np. na skutek wystąpienia zakleszczenia

- jeżeli rozważamy realizacje, które zawierają operacje wycofywanych transakcji, to własność uszeregowalności nie gwarantuje poprawności dowolnej realizacji transakcji, w szczególności, nie gwarantuje wolność od anomalii współbieżnego wykonywania transakcji
- przykład:
  $H = r_1[x]\ w_1[x]\ r_1[y]\ r_2[x]\ w_1[y]\ r_2[y]\ c_2\ r_1[z]\ w_1[z]$ <crash> c_1
  Historia H jest uszeregowalna, ale nie jest wolna od anomalii (brudny odczyt) - po restarcie systemu transakcja T2 nie zostanie poprawnie odtworzona gdyż powinna ona odczytać stan bazy danych sprzed wykonania transakcji T1

- definicja: (własności realizacji wykluczające anomalie będące wynikiem awarii systemu)
  Transakcja Ti czyta daną x z transakcji Tj w realizacji H jeżeli:
  $w_j[x] < r_i[x]$
  $a_j < r_i[x]$
  jeżeli istnieje operacja $w_k[x]$ taka, że $w_j[x] < w_k[x] < r_i[x]$, wtedy $a_k < r_i[x]$
- definicja:
  Transakcja T_i czyta z transakcji T_j w realizacji H, jeżeli T_i czyta jakąś daną z transakcji T_j w realizacji H
- definicja:
  Realizacja H jest odtwarzalna (recoverable) (RC) wówczas, jeżeli transakcja T_i czyta z transakcji T_j ($i \neq j$) w realizacji H i $c_i \in H$, to $c_j < c_i$

- definicja:
  Realizacja H unika kaskadowych wycofań (ACA, Avoids Cascading Aborts) wówczas, jeżeli transakcja T_i czyta z transakcji T_j ($i \neq j$), to $c_j < r_i[x]$
- definicja:
  Realizacja H jest ścisła (ST, strict) wówczas, jeżeli $w_j[x] < o_i[x]$ ($i \neq j$), zachodzi $a_j < o_i[x]$ lub $c_j < o_i[x]$, gdzie $o_i[x]$ jest jedną z operacji $r_i[x]$ lub $w_i[x]$
- przykład:
  $T_1 = w_1[x]\ w_1[y]\ w_1[z]\ c_1$
  $T_2 = r_2[u]\ w_2[x]\ r_2[y]\ w_2[y]\ c_2$
  $H_1 = w_1[x]\ w_1[y]\ r_2[u]\ w_2[x]\ r_2[y]\ w_2[y]\ c_2\ w_1[z]\ c_1$
  $H_2 = w_1[x]\ w_1[y]\ r_2[u]\ w_2[x]\ r_2[y]\ w_2[y]\ w_1[z]\ c_1\ c_2$
  $H_3 = w_1[x]\ w_1[y]\ r_2[u]\ w_2[x]\ w_1[z]\ c_1\ r_2[y]\ w_2[y]\ c_2$
  $H_4 = w_1[x]\ w_1[y]\ r_2[u]\ w_1[z]\ c_1\ w_2[x]\ r_2[y]\ w_2[y]\ c_2$

-Realizacja H_1 jest realizacją konfliktowo-uszeregowalną, ale nie jest realizacja odtwarzalną. Transakcja T_2 czyta daną y z transakcji T_1, ale $c_2<c_1$. Realizacja H_1 nie należy do klasy realizacji ACA jak również ST.

-Realizacja H_2 jest realizacją konfliktowo-uszeregowalną. Ponadto, jest również realizacją odtwarzalną. Transakcja T_2 czyta daną y z transakcji T_1, ale tym razem $c_1<c_2$. Realizacja H_2 nie należy do klasy realizacji ACA jak również ST.

-Realizacja H_3 jest realizacją konfliktowo-uszeregowalną i odtwarzalną. Ponadto, jest ona również realizacją unikającą kaskadowych wycofań (należy do klasy ACA). Transakcja T_2 czyta daną y z transakcji T_1 i spełniony jest warunek $c_1 < r_2[y]$. Realizacja H_3 nie jest natomiast realizacją ścisłą.

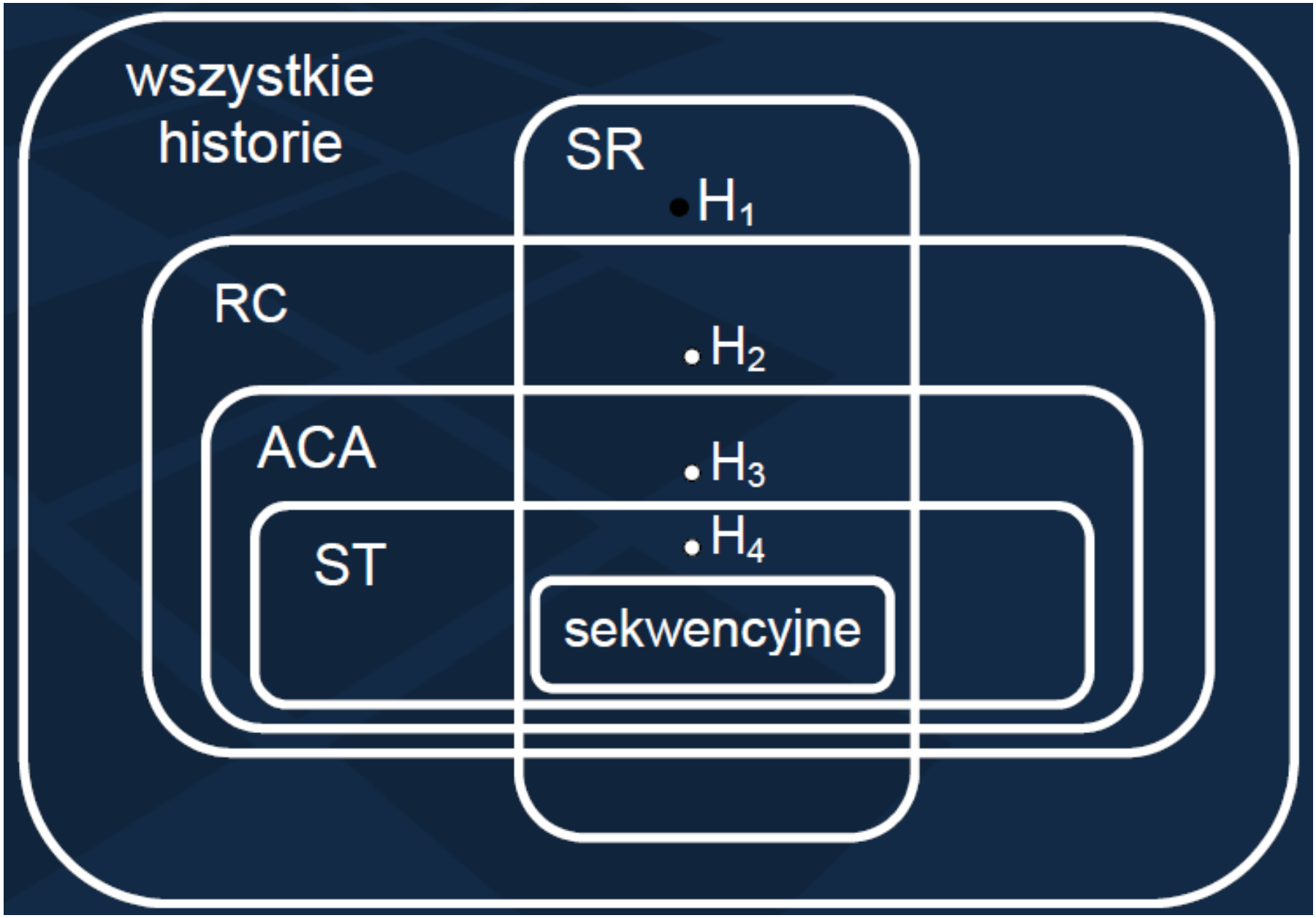
-Realizacja H_4 jest realizacją konfliktowo uszeregowalną, odtwarzalną unikającą kaskadowych wycofań oraz ścisłą.

zależności między zbiorami realizacji RC, ACA i ST

- tw: ST $\subset$ ACA $\subset$ RC $\subset$ wszystkie historie

izolacja

- definicja:
  Izolacja = uszeregowalność $\cap$ ST


wszystkie historie
SR
H_1
RC
H_2
ACA
H_3
H_4
ST
sekwencyjne

realizacje uszeregowalne

- definicja: (kryterium uszeregowalności)
  Realizacja r(T) zbioru transakcji T jest poprawna (uszeregowalna), jeżeli jest ona obrazowo i stanowo równoważna jakiejkolwiek sekwencyjnej realizacji zbioru transakcji T.
  Realizację r(T), spełniającą zdefiniowane powyżej kryterium, nazywamy realizacją uszeregowalną (należy do klasy SR).
- równoważność konfliktowa realizacji została zastąpiona w kryterium uszeregowalności równoważnością obrazową i stanową realizacji

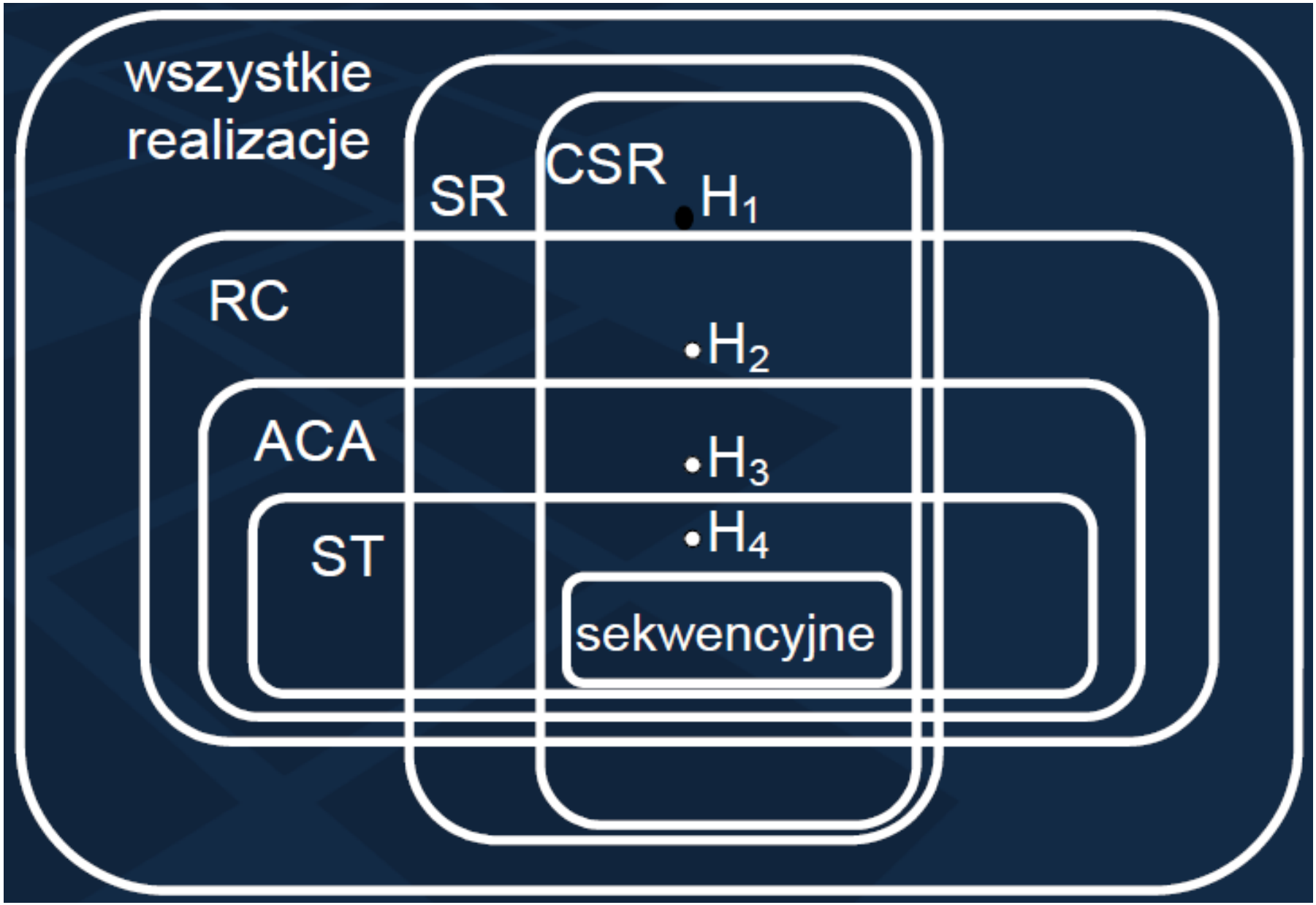
graf uszeregowalności

- definicja:
  Grafem uszeregowalności realizacji r(τ) nazywamy skierowany graf SG(r(τ)) = (V, A) taki, w którym zbiór wierzchołków V odpowiada transakcjom ze zbioru τ, natomiast zbiór krawędzi jest zdefiniowany następująco:
  Jeżeli istnieje dana x, i operacje T_i:r(x), T_j:w(x) ∈ Tr(τ), takie, że T_i:r(x) czyta wartość danej x zapisanej przez operację T_j:w(x), to:
  1. $(T_j, T_i) \in A$
  2. Jeżeli $T_j \neq T_0$, $T_i \neq T_f$ i istnieje operacja T_k:w(x) ∈ Tr(τ), $T_k \neq T_0$, to $(T_k, T_j) \in A$ lub $(T_i, T_k) \in A$
  3. Jeżeli $T_j \neq T_0$, to $(T_0, T_j) \in A$

4. Jeżeli $T_j=T_0$, $T_i \neq T_f$ i istnieje operacja
   $T_k: w(x) \in Tr(\tau)$, $T_k \neq T_0$, to $(T_i, T_k) \in A$;
5. Jeżeli $T_i=T_f$, i istnieje operacja
   $T_k: w(x) \in Tr(\tau)$, to $(T_k, T_j) \in A$

- dana realizacja r(τ) jest uszeregowalna $\Leftrightarrow$ można skonstruować dla niej acykliczny skierowany graf uszeregowlaności SG(r(τ))
- problem weryfikacji, czy dana realizacja jest uszeregowalna, jest problemem NP-zupełnym, trudność weryfikacji krytetium uszeregowalności wynika z konstrukcji tego grafu: z definicji grafu uszeregowalności wynika, że wynikowy graf jest poligrafem (patrz warunek 2); z teorii grafów wynika, że weryfikacja czy dany poligraf zawiera cykl jest problemem NP-zupełnym

- w praktyce, kryterium uszeregowalności (SR) zastąpiono łatwiejszym do weryfikacji kryterium konfliktowej uszeregowalności (CSR): problem weryfikacji, czy dana realizacja jest realizacją konfliktowo-uszeregowalną jest problemem obliczeniowo łatwym (problem należy do klasy problemów P)


wszystkie
realizacje
SR
CSR
H_1
RC
H_2
ACA
H_3
ST
H_4
sekwencyjne

SQL

START TRANSACTION;
zapytanie1;
zapytanie2;
. . . . . .
zapytanieN;
COMMIT;

START TRANSACTION;
zapytanie1;
zapytanie2;
. . . . . .
zapytanieN;
ROLLBACK;

COMMIT

- kończy transakcję
- zmiany zrobione w transakcji stają się trwałe
- zwalnia istniejące punkty kontrolne (savepoint - punkt/stan, do którego możliwy jest powrót bez oddziaływania na akcje wcześniejsze)

ROLLBACK (WORK) [(TO SAVEPOINT) <name>]

- odwrotność commit
- kończy transakcje i unieważnia wszystkie zmiany
- zwalnia wszystkie blokady nałożone na tabele podczas transakcji

korzyści z COMMIT i ROLLBACK

- zapewnienie zgodności danych
- przegląd zmian przed ich utrwaleniem
- grupowanie operacji połączonych logicznych

SAVEPOINT <name>

- oznacza i zachowuje dany punkt kontrolny w transakcji
- pozwala na unieważnienie części transakcji do tego punktu kontrolnego
- aktywny punkt kontrolny - punkt kontrolny utworzony od ostatniego polecenia COMMIT lub ROLLBACK

cofnięcie zmian do punktu kontrolnego

- utworzenie punktu kontrolnego: SAVEPOINT
- cofnięcie do punktu kontrolnego

```
UPDATE …
SAVEPOINT update_point
INSERT …
ROLLBACK TO update_point
```

metoda przechowywania tabel w MySQL

- ten sposób to inaczej silnik (engine)
- jest definiowany przy zakładaniu tabeli
- engine=MyISAM
  - ✓ domyślny (natywny) w MySQL
  - ✓ szybki dostęp do tabel
  - ✓ nie realizuje niektórych rzeczy zdefiniowanych w standardzie SQL, w szczególności nie obsługuje mechanizmu kluczy obcych ani transakcji
- engine=InnoDB
  - ✓ wolniejszy dostęp do tabel
  - ✓ pozwala na stosowanie kluczy obcych i transakcji

```
CREATE TABLE Pracownicy(...) ENGINE=InnoDB
```

blokowanie tabel

- tryb READ: chcę czytać tabelę i w tym czasie nie zezwalam innym na zapis
- tryb WRITE: chcę zmieniać zawartość tabeli i w tym czasie nie zezwalam innym ani na zapis, ani na odczyt
- tryb LOW PRIORITY WRITE: pozwala innym wątkom na założenie blokady READ - w tym czasie wątek, który chce nałożyć blokadę LOW PRIORITY WRITE, musi czekać, aż tamten wątek zwolni blokadę

LOCK TABLES nazwa_tabeli [READ | [LOW PRIORITY] WRITE];

zwalnianie blokady tabel

- zwalnia wszystkie zablokowane przez dany wątek tabele
- tabel nie należy blokować zbyt długo lub niepotrzebnie

UNLOCK TABLES;

poziomy izolacji (isolations levels)

- przykładowy problem:
  jak zrobić np. "raport dzienny" z operacji, jeśli system jest w ciągłym ruchu? - zawsze jakieś dane byłyby zablokowane …
- idea: dla pewnych sytuacji warunek pełnej izolacji okazuje się zbyt mocny – jego osłabienie jest czasami konieczne, ale powinno to być traktowane wyjątkowo, ze specjalnym statusem, ze specjalnymi środkami bezpieczeństwa
- rozwiązanie SQL jest krytykowane jako dające programiście zbyt wiele "brudnych" możliwości
- oznacza i zachowuje dany punkt kontrolny w transakcji