# ACID

## Definition

**Transakcja** to ciąg poleceń odwołujących się do bazy danych (tj. `SELECT`, `UPDATE`, itd.) stanowiący podstawową, logiczną jednostkę pracy na bazie danych.

# ACID

## Definition

**Transakcja** to ciąg poleceń odwołujących się do bazy danych (tj. `SELECT`, `UPDATE`, itd.) stanowiący podstawową, logiczną jednostkę pracy na bazie danych.

A – atomowość każda transakcja musi być wykonana w całości albo wcale; zapewnienie atomowości do zadanie SZBD;

C – spójność (consistency) transakcja uruchomiona na spójnym stanie bazy danych przekształca go w także w stan spójny, czyli spełniający wszystkie warunki poprawności danych; SZBD ponosi tu odpowiedzialność tylko za warunki zdefiniowane po stronie bazy danych;

I – niezależność (isolation) każda transakcja powinna mieć wrażenie, że na czas wykonywania dostała bazę tylko dla siebie, jakby była jedyną aktywną transakcją w systemie; za niezależność odpowiada SZBD;

D – trwałość wyniki transakcji zakończonej z powodzeniem (wypełnionej, czyli *committed*) powinny być trwałe i nie powinny zostać utracone na skutek awarii systemu; za trwałość transakcji odpowiada SZBD;

# ACID

## Definition

**Transakcja** to ciąg poleceń odwołujących się do bazy danych (tj. `SELECT`, `UPDATE`, itd.) stanowiący podstawową, logiczną jednostkę pracy na bazie danych.

A – atomowość każda transakcja musi być wykonana w całości albo wcale; zapewnienie atomowości do zadanie SZBD;

C – spójność (consistency) transakcja uruchomiona na spójnym stanie bazy danych przekształca go w także w stan spójny, czyli spełniający wszystkie warunki poprawności danych; SZBD ponosi tu odpowiedzialność tylko za warunki zdefiniowane po stronie bazy danych;

I – niezależność (isolation) **każda transakcja powinna mieć wrażenie, że na czas wykonywania dostała bazę tylko dla siebie, jakby była jedyną aktywną transakcją w systemie; za niezależność odpowiada SZBD;**

D – trwałość wyniki transakcji zakończonej z powodzeniem (wypełnionej, czyli *committed*) powinny być trwałe i nie powinny zostać utracone na skutek awarii systemu; za trwałość transakcji odpowiada SZBD;

# Harmonogramy

**Transakcja** to ciąg operacji elementarnych odwołujących się do bazy danych (tj. odczyt i/lub zapis jednostki danych).

**Jednostka danych** to fragment bazy danych, do którego odwołuje się pojedyńcza operacja elementarna, tj. rekord bazy danych, pole rekordu lub tabela.

**Harmonogram transakcji** — mając do wykonania kilka transakcji dostajemy ciąg ich operacji elementarnych. W ciągu tym (harmonogramie) operacje każdej transakcji muszą występować w porządku, jak w transakcji, ale operacje z różnych transakcji mogą się dowolnie przeplatać. Na przyklad dla transakcji $T_1 = (p_1, p_2, p_3)$, $T_2 = (r_1, r_2)$ i $T_3 = (s_1, s_2, s_3, s_4)$ harmonogramem jest: $H = (p_1, s_1, s_2, p_2, r_1, r_2, s_3, s_4, p_3)$.

# Harmonogramy (cd.)

## Definition

**Harmonogram sekwencyjny** — harmonogram nazywamy sekwencyjnym, gdy operacje elementarne transakcji nie przeplatają się, choć transakcje jako całość występują w dowolnym porządku.
Na przykład $H_1 = (p_1, p_2, p_3, s_1, s_2, s_3, s_4, r_1, r_2)$ i $H_2 = (r_1, r_2, p_1, p_2, p_3, s_1, s_2, s_3, s_4)$ są dwoma harmonogramami sekwencyjnymi. Możemy je zapisać: $H_1 = (T_1, T_3, T_2)$ i $H_2 = (T_2, T_1, T_3)$.

Przypuśćmy, że mamy do wykonania transakcje $T_1, T_2, \ldots, T_n$. Dowolny *harmonogram sekwencyjny* tych transakcji powininien być uznany za dopuszczalne i poprawne z punktu widzenia niezależności wykonanie zadanych transakcji. Stąd za spełniający warunek niezależności uznamy każdy *harmonogram tych transakcji równoważny pewnemu harmonogramowi sekwencyjnemu*.

## Definition

**Harmonogram szeregowalny** — harmonogram transakcji $T_1, T_2, \ldots, T_n$ nazwiemy szeregowalnym, jeśli efekt jego wykonania jest równoważny pewnemu harmonogramowi sekwencyjnemu transakcji $T_1, T_2, \ldots, T_n$, czyli harmonogramowi $H_\pi = (T_{\pi(1)}, T_{\pi(2)}, \ldots, T_{\pi(n)})$ dla pewnej permutacji $\pi$.

# Równoważność harmonogramów

Zadaniem SZBD jest zaakceptowanie (jak największej liczby) harmonogramów, które nawet, jeśli nie są sekwencyjne, to są równoważne pewnemu harmonogramowi sekwencyjnemu. Aby zadecydować, czy dwa harmonogramy są równoważne, trzeba umieć sklasyfikować operacje ze względu na ich działanie na danych i określić wzajemny wpływ tych operacji. W typowej klasyfikacji wyróżniamy operacje:

odczyt (read) — operacja nie zmienia wartości jednostki; operacje odczytu mogą być wykonywane w dowolnym porządku i nie zmienia to ich efektów;

zapis (write = full access) — zapisuje nową wartość jednostki utworzoną (być może) w oparciu o poprzednią wartość; operacje read i write dotyczące tej samej jednostki są od siebie zależne i w równoważnych harmonogramach musza występować w takim samym porządku; podobnie dwie operacje zapisu tej samej jednostki — ich wyniki wpływaja na siebie, więc ich porządku nie mozna zmieniać;

ślepy zapis (blind write) — nadpisuje nową wartość jednostki w miejsce poprzedniej; ślepy zapis i operacje odczytu i zapisu są od siebie zależne; ślepe zapisy, których wyników nie odczytuje żadna transakcja są niezależne od siebie;

inkrementacja — zmiana wartości jednostki o ustaloną stałą (np. 1, -1); operacje inkrementacji są wzajemnie niezależne (mogą być wykonane w dowolnym porządku); operacje inkrementacji są jednak zależne od operacji odczytu i zapisu.

# Modele danych

By przeanalizować problem niezależności, musimy ustalić model danych wystarczająco ogólny, bo opisywał rozważany przypadek, i wystarczająco szczegółowy, by dało się wykorzystać specyficzne własności danych i operacji do sprawnego zarządzania wielodostępem. Kilka typowych modeli danych rozważanych z punktu widzenia transakcji, to:

# Modele danych

By przeanalizować problem niezależności, musimy ustalić model danych wystarczająco ogólny, bo opisywał rozważany przypadek, i wystarczająco szczegółowy, by dało się wykorzystać specyficzne własności danych i operacji do sprawnego zarządzania wielodostępem. Kilka typowych modeli danych rozważanych z punktu widzenia transakcji, to:

Model 1 (zbiór rekordów) — baza danych jest uważana za zbiór niezależnych jednostek, np. rekordów. Każda operacja elementarna oznacza odczyt (read) lub dostęp w całości (read+write) do jednej jednostki.

# Modele danych

By przeanalizować problem niezależności, musimy ustalić model danych wystarczająco ogólny, bo opisywał rozważany przypadek, i wystarczająco szczegółowy, by dało się wykorzystać specyficzne własności danych i operacji do sprawnego zarządzania wielodostępem. Kilka typowych modeli danych rozważanych z punktu widzenia transakcji, to:

Model 1 (zbiór rekordów) — baza danych jest uważana za zbiór niezależnych jednostek, np. rekordów. Każda operacja elementarna oznacza odczyt (read) lub dostęp w całości (read+write) do jednej jednostki.

Model 2 (hierarchia) — w bazie wyróżniamy małe jednostki elementarne i większe jednostki, złożone z jednostek mniejszych (np. rekordy, tabele złożone z rekordów i bazę danych złożoną z tabel). Transakcja, która ma dostęp do elementu złożonego z mniejszych jenostek, może operować także na tych jednostkach. W tym modelu możemy rozważać tylko jeden rodzaj operacji — dostęp całkowity.

# Modele danych

By przeanalizować problem niezależności, musimy ustalić model danych wystarczająco ogólny, bo opisywał rozważany przypadek, i wystarczająco szczegółowy, by dało się wykorzystać specyficzne własności danych i operacji do sprawnego zarządzania wielodostępem. Kilka typowych modeli danych rozważanych z punktu widzenia transakcji, to:

Model 1 (zbiór rekordów) — baza danych jest uważana za zbiór niezależnych jednostek, np. rekordów. Każda operacja elementarna oznacza odczyt (read) lub dostęp w całości (read+write) do jednej jednostki.

Model 2 (hierarchia) — w bazie wyróżniamy małe jednostki elementarne i większe jednostki, złożone z jednostek mniejszych (np. rekordy, tabele złożone z rekordów i bazę danych złożoną z tabel). Transakcja, która ma dostęp do elementu złożonego z mniejszych jenostek, może operować także na tych jednostkach. W tym modelu możemy rozważać tylko jeden rodzaj operacji — dostęp całkowity.

Model 3 (drzewo) — dane to rekordy powiązane w strukturę drzewa. Dostęp do jednostek odbywa się za pomocą powiązań w drzewie, tzn. od rodzica możemy przejść do dzieci i odwrotnie. Model ten dobrze oddaje strukturę indeksów w postaci B-drzewa, R-drzewa itp.

# Graf pierwszeństwa

Rozważmy model 1 (zbiór niezależnych rekordów z operacjami `read` oraz `write` — pełny dostęp). Niech $H$ będzie harmonogramem transakcji $T_1, \ldots, T_n$. Zależności pomiędzy transakcjami wynikające z $H$ możemy opisać w formie **grafu pierwszeństwa** $G = (V, E)$, gdzie $V = \{T_1, T_2, \ldots, T_n\}$ i $E = \{(T_i, T_j) \mid T_i \text{ poprzedza } T_j\}$, gdzie $T_i$ poprzedza $T_j$ wtedy i tylko wtedy, gdy w harmonogramie $H$ zachodzi jeden z przypadków:

- transakcja $T_j$ czyta wartość jednostki $A$ zapisaną przez transakcję $T_i$,
- transakcja $T_j$ zmienia wartość jednostki $A$ zapisanej przez $T_i$,
- transakcja $T_i$ czyta wartość jednostki $A$ przed zapisaniem jej przez transakcję $T_j$.

# Graf pierwszeństwa

Rozważmy model 1 (zbiór niezależnych rekordów z operacjami `read` oraz `write` — pełny dostęp). Niech $H$ będzie harmonogramem transakcji $T_1, \ldots, T_n$. Zależności pomiędzy transakcjami wynikające z $H$ możemy opisać w formie **grafu pierwszeństwa** $G = (V, E)$, gdzie $V = \{T_1, T_2, \ldots, T_n\}$ i $E = \{(T_i, T_j) \mid T_i \text{ poprzedza } T_j\}$, gdzie $T_i$ poprzedza $T_j$ wtedy i tylko wtedy, gdy w harmonogramie $H$ zachodzi jeden z przypadków:

- transakcja $T_j$ czyta wartość jednostki $A$ zapisaną przez transakcję $T_i$,
- transakcja $T_j$ zmienia wartość jednostki $A$ zapisanej przez $T_i$,
- transakcja $T_i$ czyta wartość jednostki $A$ przed zapisaniem jej przez transakcję $T_j$.

**Theorem**

*Harmonogram jest szeregowalny wtedy i tylko wtedy, gdy jego graf pierwszeństwa jest acykliczny. Harmonogram jest równoważny harmonogramowi sekwencyjnemu będącemu liniowym rozszerzeniem porządku częściowego zadanego przez graf.*

# Graf pierwszeństwa

Rozważmy model 1 (zbiór niezależnych rekordów z operacjami `read` oraz `write` — pełny dostęp). Niech $H$ będzie harmonogramem transakcji $T_1, \ldots, T_n$. Zależności pomiędzy transakcjami wynikające z $H$ możemy opisać w formie **grafu pierwszeństwa** $G = (V, E)$, gdzie $V = \{T_1, T_2, \ldots, T_n\}$ i $E = \{(T_i, T_j) \mid T_i \text{ poprzedza } T_j\}$, gdzie $T_i$ poprzedza $T_j$ wtedy i tylko wtedy, gdy w harmonogramie $H$ zachodzi jeden z przypadków:

- transakcja $T_j$ czyta wartość jednostki $A$ zapisaną przez transakcję $T_i$,
- transakcja $T_j$ zmienia wartość jednostki $A$ zapisanej przez $T_i$,
- transakcja $T_i$ czyta wartość jednostki $A$ przed zapisaniem jej przez transakcję $T_j$.

**Theorem**

*Harmonogram jest szeregowalny wtedy i tylko wtedy, gdy jego graf pierwszeństwa jest acykliczny. Harmonogram jest równoważny harmonogramowi sekwencyjnemu będącemu liniowym rozszerzeniem porządku częściowego zadanego przez graf.*

**Uwagi:**

# Graf pierwszeństwa

Rozważmy model 1 (zbiór niezależnych rekordów z operacjami `read` oraz `write` — pełny dostęp). Niech $H$ będzie harmonogramem transakcji $T_1, \ldots, T_n$. Zależności pomiędzy transakcjami wynikające z $H$ możemy opisać w formie **grafu pierwszeństwa** $G = (V, E)$, gdzie $V = \{T_1, T_2, \ldots, T_n\}$ i $E = \{(T_i, T_j) \mid T_i \text{ poprzedza } T_j\}$, gdzie $T_i$ poprzedza $T_j$ wtedy i tylko wtedy, gdy w harmonogramie $H$ zachodzi jeden z przypadków:

- transakcja $T_j$ czyta wartość jednostki $A$ zapisaną przez transakcję $T_i$,
- transakcja $T_j$ zmienia wartość jednostki $A$ zapisanej przez $T_i$,
- transakcja $T_i$ czyta wartość jednostki $A$ przed zapisaniem jej przez transakcję $T_j$.

**Theorem**

*Harmonogram jest szeregowalny wtedy i tylko wtedy, gdy jego graf pierwszeństwa jest acykliczny. Harmonogram jest równoważny harmonogramowi sekwencyjnemu będącemu liniowym rozszerzeniem porządku częściowego zadanego przez graf.*

**Uwagi:**

1. Skonstruowanie grafu to proste zadanie. Sprawdzenie, czy graf jest acykliczny, także jest łatwe (wystarczy sortowanie topologiczne czy DFS). Czy to oznacza, że mamy prostą metodę weryfikacji szeregowalności harmonogramów?

# Graf pierwszeństwa

Rozważmy model 1 (zbiór niezależnych rekordów z operacjami `read` oraz `write` — pełny dostęp). Niech $H$ będzie harmonogramem transakcji $T_1, \ldots, T_n$. Zależności pomiędzy transakcjami wynikające z $H$ możemy opisać w formie **grafu pierwszeństwa** $G = (V, E)$, gdzie $V = \{T_1, T_2, \ldots, T_n\}$ i $E = \{(T_i, T_j) \mid T_i \text{ poprzedza } T_j\}$, gdzie $T_i$ poprzedza $T_j$ wtedy i tylko wtedy, gdy w harmonogramie $H$ zachodzi jeden z przypadków:

- transakcja $T_j$ czyta wartość jednostki $A$ zapisaną przez transakcję $T_i$,
- transakcja $T_j$ zmienia wartość jednostki $A$ zapisanej przez $T_i$,
- transakcja $T_i$ czyta wartość jednostki $A$ przed zapisaniem jej przez transakcję $T_j$.

**Theorem**

*Harmonogram jest szeregowalny wtedy i tylko wtedy, gdy jego graf pierwszeństwa jest acykliczny. Harmonogram jest równoważny harmonogramowi sekwencyjnemu będącemu liniowym rozszerzeniem porządku częściowego zadanego przez graf.*

**Uwagi:**

1. Skonstruowanie grafu to proste zadanie. Sprawdzenie, czy graf jest acykliczny, także jest łatwe (wystarczy sortowanie topologiczne czy DFS). Czy to oznacza, że mamy prostą metodę weryfikacji szeregowalności harmonogramów?
2. Niestety, sposób ten nie jest przydatny w praktyce — wymaga znajomości kompletnego harmonogramu złożonego z zakończonych transakcji, czego zazwyczaj nie mamy.

# Blokady

Możemy oprzeć organizację wielodostępu do bazy danych na blokowaniu jednostek dla określonych transakcji. Blokada musi być przyznana transakcji, jeśli chce ona wykonać operację na jednostce, ale może też być utrzymywana dłużej, by zapobiec dostępowi innych transakcji do jednostki. Transakcja, która nie ma blokady jednostki, na której chce działać, musi czekać, aż system jej tę blokadę przyzna (zapewne po zwolnieniu jednostki przez inną transakcję). Zazwyczaj rozważamy dwa rodzaje blokad:

`RLOCK` lub `SHARED LOCK` potrzebna, gdy transalcja chce czytać jednostkę;

`LOCK` lub `EXCLUSIVE LOCK` potrzebna, gdy transalcja chce zmieniać jednostkę.

Po wykonaniu operacji transakcja może zwolnić blokadę jednostki (komenda `UNLOCK`). Blokady wyłączne (`EXCLUSIVE LOCK`) wykluczają się. Możliwe jest natomiast blokowanie jednostki przez kilka transakcji blokadą dzieloną (`SHARED LOCK`). Blokada dzielona i wyłączna już się jednak wykluczają.
Harmonogram z blokadamia jest **legalny**, jeśli:

- zakładane w nim blokady nie kolidują ze sobą (zgodnie z podanymi wyżej zasadami wzajemnego wykluczania się);
- każda transakcja wykonuje operację tylko wtedy, gdy ma odpowiednią blokadę.

# Blokady

Możemy oprzeć organizację wielodostępu do bazy danych na blokowaniu jednostek dla określonych transakcji. Blokada musi być przyznana transakcji, jeśli chce ona wykonać operację na jednostce, ale może też być utrzymywana dłużej, by zapobiec dostępowi innych transakcji do jednostki. Transakcja, która nie ma blokady jednostki, na której chce działać, musi czekać, aż system jej tę blokadę przyzna (zapewne po zwolnieniu jednostki przez inną transakcję). Zazwyczaj rozważamy dwa rodzaje blokad:

`RLOCK` lub `SHARED LOCK` potrzebna, gdy transalcja chce czytać jednostkę;

`LOCK` lub `EXCLUSIVE LOCK` potrzebna, gdy transalcja chce zmieniać jednostkę.

Po wykonaniu operacji transakcja może zwolnić blokadę jednostki (komenda `UNLOCK`). Blokady wyłączne (`EXCLUSIVE LOCK`) wykluczają się. Możliwe jest natomiast blokowanie jednostki przez kilka transakcji blokadą dzieloną (`SHARED LOCK`). Blokada dzielona i wyłączna już się jednak wykluczają.
Harmonogram z blokadamia jest **legalny**, jeśli:

- zakładane w nim blokady nie kolidują ze sobą (zgodnie z podanymi wyżej zasadami wzajemnego wykluczania się);
- każda transakcja wykonuje operację tylko wtedy, gdy ma odpowiednią blokadę.

# Protokół dwufazowy

**Protokół** to zbiór reguł, których powinna przestrzegać każda transakcja niezależnie od pozostałych. Np. poniższy protokół może zostać zastosowany do kontroli szeregowalności harmonogramu:

# Protokół dwufazowy

**Protokół** to zbiór reguł, których powinna przestrzegać każda transakcja niezależnie od pozostałych. Np. poniższy protokół może zostać zastosowany do kontroli szeregowalności harmonogramu:

**Definition (Blokowanie dwufazowe (*Two Phase Locking*, 2PL))**

- każda transakcja może zablokować dowolną dostępną jednostkę blokadą wyłączną (do zapisu) lub dzieloną (do odczytu);
- po zwolnieniu dowolnej blokady transakcji nie wolno blokować jednostek.

# Protokół dwufazowy

**Protokół** to zbiór reguł, których powinna przestrzegać każda transakcja niezależnie od pozostałych. Np. poniższy protokół może zostać zastosowany do kontroli szeregowalności harmonogramu:

**Definition (Blokowanie dwufazowe (*Two Phase Locking*, 2PL))**

- każda transakcja może zablokować dowolną dostępną jednostkę blokadą wyłączną (do zapisu) lub dzieloną (do odczytu);
- po zwolnieniu dowolnej blokady transakcji nie wolno blokować jednostek.

System oparty na 2PL jest bardzo prosty w implementacji. Zachodzi także następujące twierdzenie:

# Protokół dwufazowy

**Protokół** to zbiór reguł, których powinna przestrzegać każda transakcja niezależnie od pozostałych. Np. poniższy protokół może zostać zastosowany do kontroli szeregowalności harmonogramu:

**Definition (Blokowanie dwufazowe (*Two Phase Locking*, 2PL))**

- każda transakcja może zablokować dowolną dostępną jednostkę blokadą wyłączną (do zapisu) lub dzieloną (do odczytu);
- po zwolnieniu dowolnej blokady transakcji nie wolno blokować jednostek.

System oparty na 2PL jest bardzo prosty w implementacji. Zachodzi także następujące twierdzenie:

**Theorem**

*Rozważmy harmonogram transakcji. Jeśli każda transakcja w harmonogramie przestrzega protokołu blokowania dwufazowego, to cały harmonogram jest szeregowalny.*

# Brudne dane

Każda transakcja w trakcie działania jest narażona błędy, awarie i inne zdarzenia, które mogą uniemożliwić jej zakończenie. Transakcja przerwana musi zostać wycofana, bo przeciwnie naruszylibyśmy zasadę atomowości. **Brudne dane (*dirty data*)**, inaczej — niepewne dane, to dane utworzone przez transakcję, która jeszcze nie zakończyła działania. Dane te przestają być brudne, gdy transakcja wykona operację `COMMIT`. Brudne dane są niepewne — być może trzeba będzie je wycofać i wrócić do poprzednich wartości jednostek, gdy nie uda się zakończyć z powodzeniem transakcji, która je utworzyła (czyli transakcja zakończy się operacją `ROLLBACK`).

# Brudne dane

Każda transakcja w trakcie działania jest narażona błędy, awarie i inne zdarzenia, które mogą uniemożliwić jej zakończenie. Transakcja przerwana musi zostać wycofana, bo przeciwnie naruszylibyśmy zasadę atomowości. **Brudne dane (*dirty data*)**, inaczej — niepewne dane, to dane utworzone przez transakcję, która jeszcze nie zakończyła działania. Dane te przestają być brudne, gdy transakcja wykona operację `COMMIT`. Brudne dane są niepewne — być może trzeba będzie je wycofać i wrócić do poprzednich wartości jednostek, gdy nie uda się zakończyć z powodzeniem transakcji, która je utworzyła (czyli transakcja zakończy się operacją `ROLLBACK`).

- Zezwolenie transakcjom na czytanie brudnych danych może spowodować *anulowanie kaskadowe*.
- Zakaz dostępu do brudnych danych może znacznie spowolnić działanie bazy danych.

# Brudne dane

Każda transakcja w trakcie działania jest narażona błędy, awarie i inne zdarzenia, które mogą uniemożliwić jej zakończenie. Transakcja przerwana musi zostać wycofana, bo przeciwnie naruszylibyśmy zasadę atomowości. **Brudne dane (*dirty data*)**, inaczej — niepewne dane, to dane utworzone przez transakcję, która jeszcze nie zakończyła działania. Dane te przestają być brudne, gdy transakcja wykona operację `COMMIT`. Brudne dane są niepewne — być może trzeba będzie je wycofać i wrócić do poprzednich wartości jednostek, gdy nie uda się zakończyć z powodzeniem transakcji, która je utworzyła (czyli transakcja zakończy się operacją `ROLLBACK`).

- Zezwolenie transakcjom na czytanie brudnych danych może spowodować *anulowanie kaskadowe*.
- Zakaz dostępu do brudnych danych może znacznie spowolnić działanie bazy danych.

**Definition (Blokowanie ściśle dwufazowe (Strict 2PL))**

Każda transakcja przestrzegająca protokołu ściśle dwufazowego jest dwufazowa i trzyma każdą blokadę do momentu wypełnienia (`COMMIT`).

# Brudne dane

Każda transakcja w trakcie działania jest narażona błędy, awarie i inne zdarzenia, które mogą uniemożliwić jej zakończenie. Transakcja przerwana musi zostać wycofana, bo przeciwnie naruszylibyśmy zasadę atomowości. **Brudne dane (*dirty data*)**, inaczej — niepewne dane, to dane utworzone przez transakcję, która jeszcze nie zakończyła działania. Dane te przestają być brudne, gdy transakcja wykona operację `COMMIT`. Brudne dane są niepewne — być może trzeba będzie je wycofać i wrócić do poprzednich wartości jednostek, gdy nie uda się zakończyć z powodzeniem transakcji, która je utworzyła (czyli transakcja zakończy się operacją `ROLLBACK`).

- Zezwolenie transakcjom na czytanie brudnych danych może spowodować *anulowanie kaskadowe*.
- Zakaz dostępu do brudnych danych może znacznie spowolnić działanie bazy danych.

**Definition (Blokowanie ściśle dwufazowe (Strict 2PL))**

Każda transakcja przestrzegająca protokołu ściśle dwufazowego jest dwufazowa i trzyma każdą blokadę do momentu wypełnienia (`COMMIT`).

W systemie, gdzie każda transakcja przestrzega protokołu ściśle dwufazowego, każdy harmonogram jest szeregowalny. Nie występują także brudne dane, stąd nie ma niebezpieczeństwa anulowania kaskadowego.

# Blokowanie w modelu hierarchicznym

## Model 2 (hierarchiczny)

Jednostki bazy danych mają strukturę hierarchii — większa jednostka składa się z mniejszych, np. baza danych składa się z tabel, tabele składają się z rekordów, a rekordy — z pól. Transakcje mogą blokować całe tabele lub pojedyncze rekordy, lub tylko pola rekordów.

- Przyjmujemy, że transakcja, która zablokowała jednostkę, musi mieć zagwarantowany dostęp do niej i wszystkich jej jednostek podrzędnych.
- Istotny zaczyna więc być problem efektywności i legalności blokowania.
- Procedura blokowania jednostki poprzez zablokowanie wszystkich jednostek najniższego rzędu składających się na daną jednostkę jest nieefektywna (konieczność założenie wielu blokad) i może skutkować zakleszczeniem transakcji (deadlock).
- Trzeba znaleźć rozwiązanie lepiej dostosowane do tego przypadku: gwarantujące **legalność** blokowania, **szeregowalność** i **redukujące możliwość zakleszczenia** transakcji.

# Protokół z ostrzeżeniami

**Model:** operacja **modify** i **hierarchiczna struktura** jednostek bazy danych (baza danych, tabele, rekordy, pola).

# Protokół z ostrzeżeniami

**Model:** operacja **modify** i **hierarchiczna struktura** jednostek bazy danych (baza danych, tabele, rekordy, pola).

## Protokół z ostrzeżeniami

# Protokół z ostrzeżeniami

**Model:** operacja **modify** i **hierarchiczna struktura** jednostek bazy danych (baza danych, tabele, rekordy, pola).

## Protokół z ostrzeżeniami

- Aby uzyskać dostęp do jednostki *A*, transakcja ustawia flagę `WARN` w korzeniu drzewa i na kolejnych wierzchołkach ścieżki prowadzącej do *A*.

# Protokół z ostrzeżeniami

**Model:** operacja **modify** i **hierarchiczna struktura** jednostek bazy danych (baza danych, tabele, rekordy, pola).

## Protokół z ostrzeżeniami

- Aby uzyskać dostęp do jednostki *A*, transakcja ustawia flagę `WARN` w korzeniu drzewa i na kolejnych wierzchołkach ścieżki prowadzącej do *A*.
- Transakcja nie może ustawić swojej flagi `WARN` na wierzchołku, na którym jest blokada `LOCK` założona przez inną transakcję.

# Protokół z ostrzeżeniami

**Model:** operacja **modify** i **hierarchiczna struktura** jednostek bazy danych (baza danych, tabele, rekordy, pola).

## Protokół z ostrzeżeniami

- Aby uzyskać dostęp do jednostki *A*, transakcja ustawia flagę `WARN` w korzeniu drzewa i na kolejnych wierzchołkach ścieżki prowadzącej do *A*.
- Transakcja nie może ustawić swojej flagi `WARN` na wierzchołku, na którym jest blokada `LOCK` założona przez inną transakcję.
- Transakcja nakłada blokadę `LOCK` na jednostkę *A*, gdy ma flagi `WARN` na całej ścieżce od korzenia do *A*.

# Protokół z ostrzeżeniami

**Model:** operacja **modify** i **hierarchiczna struktura** jednostek bazy danych (baza danych, tabele, rekordy, pola).

## Protokół z ostrzeżeniami

- Aby uzyskać dostęp do jednostki *A*, transakcja ustawia flagę `WARN` w korzeniu drzewa i na kolejnych wierzchołkach ścieżki prowadzącej do *A*.
- Transakcja nie może ustawić swojej flagi `WARN` na wierzchołku, na którym jest blokada `LOCK` założona przez inną transakcję.
- Transakcja nakłada blokadę `LOCK` na jednostkę *A*, gdy ma flagi `WARN` na całej ścieżce od korzenia do *A*.
- Transakcja może założyć blokadę `LOCK` na wierzchołku *A* tylko wtedy, gdy nie ma na nim ani `LOCK` ani `WARN` założonych przez inną transakcję.

# Protokół z ostrzeżeniami

**Model:** operacja **modify** i **hierarchiczna struktura** jednostek bazy danych (baza danych, tabele, rekordy, pola).

## Protokół z ostrzeżeniami

- Aby uzyskać dostęp do jednostki *A*, transakcja ustawia flagę `WARN` w korzeniu drzewa i na kolejnych wierzchołkach ścieżki prowadzącej do *A*.
- Transakcja nie może ustawić swojej flagi `WARN` na wierzchołku, na którym jest blokada `LOCK` założona przez inną transakcję.
- Transakcja nakłada blokadę `LOCK` na jednostkę *A*, gdy ma flagi `WARN` na całej ścieżce od korzenia do *A*.
- Transakcja może założyć blokadę `LOCK` na wierzchołku *A* tylko wtedy, gdy nie ma na nim ani `LOCK` ani `WARN` założonych przez inną transakcję.
- Po wykorzystaniu jednostki transakcja zwalania blokady `LOCK` i flagi `WARN` w kolejności od jednostki w kierunku korzenia hierarchii.

# Protokół z ostrzeżeniami

**Model:** operacja **modify** i **hierarchiczna struktura** jednostek bazy danych (baza danych, tabele, rekordy, pola).

## Protokół z ostrzeżeniami

- Aby uzyskać dostęp do jednostki *A*, transakcja ustawia flagę `WARN` w korzeniu drzewa i na kolejnych wierzchołkach ścieżki prowadzącej do *A*.
- Transakcja nie może ustawić swojej flagi `WARN` na wierzchołku, na którym jest blokada `LOCK` założona przez inną transakcję.
- Transakcja nakłada blokadę `LOCK` na jednostkę *A*, gdy ma flagi `WARN` na całej ścieżce od korzenia do *A*.
- Transakcja może założyć blokadę `LOCK` na wierzchołku *A* tylko wtedy, gdy nie ma na nim ani `LOCK` ani `WARN` założonych przez inną transakcję.
- Po wykorzystaniu jednostki transakcja zwalania blokady `LOCK` i flagi `WARN` w kolejności od jednostki w kierunku korzenia hierarchii.
- Transakcja przestrzega protokołu dwufazowego, w którym operacje `LOCK` i `WARN` są wykonywane w fazie wzrostu, a operacje zdejmowania blokad i flag — w fazie zwalniania zasobów.

# Protokół z ostrzeżeniami

**Model:** operacja **modify** i **hierarchiczna struktura** jednostek bazy danych (baza danych, tabele, rekordy, pola).

## Protokół z ostrzeżeniami

- Aby uzyskać dostęp do jednostki *A*, transakcja ustawia flagę `WARN` w korzeniu drzewa i na kolejnych wierzchołkach ścieżki prowadzącej do *A*.
- Transakcja nie może ustawić swojej flagi `WARN` na wierzchołku, na którym jest blokada `LOCK` założona przez inną transakcję.
- Transakcja nakłada blokadę `LOCK` na jednostkę *A*, gdy ma flagi `WARN` na całej ścieżce od korzenia do *A*.
- Transakcja może założyć blokadę `LOCK` na wierzchołku *A* tylko wtedy, gdy nie ma na nim ani `LOCK` ani `WARN` założonych przez inną transakcję.
- Po wykorzystaniu jednostki transakcja zwalania blokady `LOCK` i flagi `WARN` w kolejności od jednostki w kierunku korzenia hierarchii.
- Transakcja przestrzega protokołu dwufazowego, w którym operacje `LOCK` i `WARN` są wykonywane w fazie wzrostu, a operacje zdejmowania blokad i flag — w fazie zwalniania zasobów.

*W harmonogramie transakcji przestrzegających protokołu z ostrzeżeniami blokady są **legalne** a harmonogram jest **szeregowalny**.*

# Blokowanie jednostek powiązanych w strukturę drzewiastą

## Model 3 (drzewo)

Baza składa się rekordów powiązanych w drzewo. Jednostki (rekordy) można blokować niezależnie, ale typowy dostęp do jednostki jest poprzez nawigację w drzewie: transakcje poruszają się w kierunku od ojca do syna i od syna do ojca.

- Model dobrze opisuje strukturę B-drzewa, podstawowej struktury indeksowej w bazach danych.
- W B-drzewach możemy dodatkowo przyjąć, że każda transakcja rozpoczyna dostęp (blokowanie) od korzenia drzewa.
- Zastosowanie protokołu dwufazowego powoduje, że transakcja, która zablokowała korzeń drzewa nie może go odblokować, zanim nie rozpocznie fazy zwalniania zasobów. To może spowodować długą kolejkę transakcji oczekujących na dostęp do indeksu.
- Poszukiwania dobrej metody blokowania dla B-drzewa powinny dążyć do znalezienia protokołu, który gwarantuje szeregowalność, ale pozwala transakcjom zwalaniać jednostki szybciej niż w 2PL.

# Protokół drzewa

**Model:** operacja **modify**; jednostki powiązane w **drzewo**; jednostki można blokować niezależnie od pozostałych, ale poruszamy się zgodnie z relacją ojciec-syn.

# Protokół drzewa

**Model:** operacja **modify**; jednostki powiązane w **drzewo**; jednostki można blokować niezależnie od pozostałych, ale poruszamy się zgodnie z relacją ojciec-syn.

## Protokół drzewa

- Transakcja zakłada pierwszą blokadę `LOCK` na wybraną jednostkę (nie musi to być korzeń, choć zazwyczaj jest to korzeń).
- Blokadę `LOCK` transakcja może założyć tylko na jednostkę, na której nie ma innej blokady.
- Transakcja może zablokować dowolną kolejną jednostkę tylko wówczas, jeżeli ma zablokowanego jej rodzica.
- Po odblokowaniu dowolnej jednostki transakcja nie może zablokować jej ponownie.

# Protokół drzewa

**Model:** operacja **modify**; jednostki powiązane w **drzewo**; jednostki można blokować niezależnie od pozostałych, ale poruszamy się zgodnie z relacją ojciec-syn.

## Protokół drzewa

- Transakcja zakłada pierwszą blokadę `LOCK` na wybraną jednostkę (nie musi to być korzeń, choć zazwyczaj jest to korzeń).
- Blokadę `LOCK` transakcja może założyć tylko na jednostkę, na której nie ma innej blokady.
- Transakcja może zablokować dowolną kolejną jednostkę tylko wówczas, jeżeli ma zablokowanego jej rodzica.
- Po odblokowaniu dowolnej jednostki transakcja nie może zablokować jej ponownie.

## Theorem

*Harmonogram transakcji przestrzegających protokołu drzewa jest szeregowalny. Dowód: nie jest trywialny. Polega na prześledzeniu działania transakcji i uzasadnieniu, że dwie transakcje do wszystkich jednostek, do których odwołują się wspólnie, odwołują się w tym samym porządku. Pozwala to skonstruować graf pierwszeństwa i przeanalizować możliwość powstawania w nim cykli.*

# Metody optymistyczne *versus* metody zachowawcze

**Metody zachowawcze:** do metod zachowawczych zaliczamy metody oparte na blokowaniu jednostek, tj. 2PL, protokół drzewa i protokół z ostrzeżeniami:

- transakcja gwarantuje sobie dostęp do wszystkich potrzebnych jednostek;
- oczekiwanie na uzyskanie blokady może trwać, może też dojść do:
  - **zakleszczenia**, gdy transakcje nazwajem blokują potrzebne sobie zasoby,
  - **zagłodzenia**, gdy transakcja cały czas przegrywa z innymi przy próbe dostępu do obleganego zasobu;
- transakcja, która zablokowała wszystkie potrzebne jednostk,i ma duże szanse na ukończenie z powodzeniem;
- wycofanie transakcji może nastąpić z powodu czytania brudnych danych, błędów wewnętrznych i/lub awarii systemu, ale nie z powodu nieszeregowalności.

# Metody optymistyczne *versus* metody zachowawcze

**Metody zachowawcze:** do metod zachowawczych zaliczamy metody oparte na blokowaniu jednostek, tj. 2PL, protokół drzewa i protokół z ostrzeżeniami:

- transakcja gwarantuje sobie dostęp do wszystkich potrzebnych jednostek;
- oczekiwanie na uzyskanie blokady może trwać, może też dojść do:
  - **zakleszczenia**, gdy transakcje nazwajem blokują potrzebne sobie zasoby,
  - **zagłodzenia**, gdy transakcja cały czas przegrywa z innymi przy próbe dostępu do obleganego zasobu;
- transakcja, która zablokowała wszystkie potrzebne jednostk,i ma duże szanse na ukończenie z powodzeniem;
- wycofanie transakcji może nastąpić z powodu czytania brudnych danych, błędów wewnętrznych i/lub awarii systemu, ale nie z powodu nieszeregowalności.

**Metody optymistyczne:** alternatywnym podejściem jest zezwolenie transakcjom na działanie i dopiero późniejsze sprawdzanie, czy efekty ich działania są szeregowalne. Może to powodować, że będąca w toku transakcja nie spełni warunków szeregowalności i zostanie wycofana. Podstawowe metody, to:

metoda znaczników czasowych: sprawdzamy szeregowalność za każdym razem, gdy transakcja odwołuje się do jednostki

metoda walidacji: sprawdzamy szeregowalność w momencie zakończenia transakcji.

# Metody optymistyczne *versus* metody zachowawcze

**Metody zachowawcze:** do metod zachowawczych zaliczamy metody oparte na blokowaniu jednostek, tj. 2PL, protokół drzewa i protokół z ostrzeżeniami:

- transakcja gwarantuje sobie dostęp do wszystkich potrzebnych jednostek;
- oczekiwanie na uzyskanie blokady może trwać, może też dojść do:
  - **zakleszczenia**, gdy transakcje nazwajem blokują potrzebne sobie zasoby,
  - **zagłodzenia**, gdy transakcja cały czas przegrywa z innymi przy próbe dostępu do obleganego zasobu;
- transakcja, która zablokowała wszystkie potrzebne jednostk,i ma duże szanse na ukończenie z powodzeniem;
- wycofanie transakcji może nastąpić z powodu czytania brudnych danych, błędów wewnętrznych i/lub awarii systemu, ale nie z powodu nieszeregowalności.

**Metody optymistyczne:** alternatywnym podejściem jest zezwolenie transakcjom na działanie i dopiero późniejsze sprawdzanie, czy efekty ich działania są szeregowalne. Może to powodować, że będąca w toku transakcja nie spełni warunków szeregowalności i zostanie wycofana. Podstawowe metody, to:

metoda znaczników czasowych: sprawdzamy szeregowalność za każdym razem, gdy transakcja odwołuje się do jednostki

metoda walidacji: sprawdzamy szeregowalność w momencie zakończenia transakcji.

*Rozważymy obie powyższe metody w modelu z niezależnymi jednostkami i operacjami* `read` *i* `write` *(ślepy zapis!).*

# Znaczniki czasowe

**Znacznik czasowy** to unikalna wartość, którą można uważać za moment uruchomienia transakcji w systemie. Przypisujemy:

time(T) — znacznik czasowy transakcji $T$ różny od znaczników czasowych wszystkich pozostałych transakcji; zazwyczaj jest to moment uruchomienia $T$ w systemie.

rtime(A) — maksymalny znacznik czasowy transakcji, która odczytała $A$, możemy go nazywać czasem ostatniego odczytu $A$;

wtime(A) — znacznik czasowy transakcji, która utworzyła aktualną wartość $A$, możemy go nazywać czasem ostatniego zapisu $A$.

# Znaczniki czasowe

**Znacznik czasowy** to unikalna wartość, którą można uważać za moment uruchomienia transakcji w systemie. Przypisujemy:

time(T) — znacznik czasowy transakcji $T$ różny od znaczników czasowych wszystkich pozostałych transakcji; zazwyczaj jest to moment uruchomienia $T$ w systemie.

rtime(A) — maksymalny znacznik czasowy transakcji, która odczytała $A$, możemy go nazywać czasem ostatniego odczytu $A$;

wtime(A) — znacznik czasowy transakcji, która utworzyła aktualną wartość $A$, możemy go nazywać czasem ostatniego zapisu $A$.

Gdy transakcja chce przeczytać lub zapisać jednostkę, porównuje swój znacznik czasowy ze znacznikami zapisu i/lub odczytu jednostki (właściwie robi to za nią SZBD). Operacja może zostać wykonana, jesli spełnione są warunki szeregowalności. W tej metodzie warunki te mają zapewnic, że działanie harmonogramu transakcji jest równoważne harmonogramowi sekwencyjnemu wyznaczonemu przez znaczniki czasowe transakcji.

# Znaczniki czasowe

**Znacznik czasowy** to unikalna wartość, którą można uważać za moment uruchomienia transakcji w systemie. Przypisujemy:

time(T) — znacznik czasowy transakcji $T$ różny od znaczników czasowych wszystkich pozostałych transakcji; zazwyczaj jest to moment uruchomienia $T$ w systemie.

rtime(A) — maksymalny znacznik czasowy transakcji, która odczytała $A$, możemy go nazywać czasem ostatniego odczytu $A$;

wtime(A) — znacznik czasowy transakcji, która utworzyła aktualną wartość $A$, możemy go nazywać czasem ostatniego zapisu $A$.

Gdy transakcja chce przeczytać lub zapisać jednostkę, porównuje swój znacznik czasowy ze znacznikami zapisu i/lub odczytu jednostki (właściwie robi to za nią SZBD). Operacja może zostać wykonana, jesli spełnione są warunki szeregowalności. W tej metodzie warunki te mają zapewnic, że działanie harmonogramu transakcji jest równoważne harmonogramowi sekwencyjnemu wyznaczonemu przez znaczniki czasowe transakcji.

Transakcja, która nie przejdzie testu szeregowalności, musi zostać wycofana. W szczególności muszą zostać wycofane wszystkie jej efekty działania. Transakcja ta jest uruchamiana ponownie z nowym (aktualnym) znacznikiem czasowym.

# Testy szeregowalności w metodzie znaczników czasowych

```
T: read(A)
   if time(T)<wtime(A) then rollback T;
   if rtime(A)<time(T) then rtime(A) := time(T);
   execute read A by T;

T: write(A)
   if time(T)<rtime(A) then rollback T;
   if time(T)<wtime(A) then {
      skip write A by T;
      continue T;
   }
   else {
     wtime(A) := time(T);
     execute write A by T;
   }
```

# Modyfikacje metody znaczników czasowych

Dodanie historii jednostki — możemy przechowywać w bazie wszystkie wartości, jakie miała jednostka; wtedy zawsze możemy odczytać wartość z przeszłości i nie musimy wycofywać transakcji, która spóźniła się z odczytem jednostki;

Bit wypełnienia — w metodzie znaczników czasowych transakcje nie są chronione przez dostępem do brudnych danych, co może spowodowć anulowanie kaskadowe nawet zakończonych transakcji. Aby temu zapobiec, możemy do każdej jednostki $A$ dodać bit wypełnienia $C(A)$, który (równy 1) oznacza, że wartość jednostki została utworzona przez transakcję zakończoną (nie są to więc brudne dane). Wartość $C(A)$ równa 0 oznacza, że transakcje chcące wykonać operację na jednostce muszą czekać na zmianę wartości bitu. Każda transakcja, która kończy działanie, ustawia bit wypełnienia na we wszystkich jednostkach, które modyfikowała. Także w przypadku wycofania transakcji, wartość jednostki jest odtwarzana i bit wypełnienia ustawiany na 1.

Blokady w metodzie znaczników czasowych — blokady służą tylko poprawnemu wykonaniu operacji, np. zapisu. Gdy dwie transakcje usiłują zapisać jednostkę, to operacje te są wykonywane w pewnym porządku i nie dochodzi do sytuacji, gdy operacje te mogą ze sobą interferować dając nieoczekiwany wynik lub gdy jedna transakcja zapisała wartość a inna swój znacznik czasowy.

# Metoda walidacji

W metodzie walidacji schemat działania transakcji jest następujący:

- każda transakcja w chwili uruchomienia otrzymuje swój prywatny obszar roboczy;
- transakcja w trakcie działania odczytuje potrzebne jej jednostki z bazy danych i kopiuje je do obszaru roboczego;
- wszystkie operacje zapisu i modyfikacji transakcja wykonuje tylko w swoim obszarze roboczym;
- gdy transakcja jest gotowa do zakończenia zgłasza się do systemu do "walidacji":
  - jeśli transakcja pozytywnie przejdzie proces walidacji, to wyniki z jej obszaru roboczego są zapisywane do bazy danych i transakcja zostaje wypełniona (zakończona pozytywnie);
  - jeśli transakcje zostaje w czasie walidacji odrzucona, to jest anulowana (jej obszar roboczy jest zwalniany) i uruchamiana ponowanie.

# Etapy transakcji

## ”Życie” transakcji:

# Etapy transakcji

## ”Życie” transakcji:

START(T) — transakcja jest uruchamiana, otrzymuje swój pierwszy znacznik odpowiadający momentowi uruchomienia w systemie. Otrzymuje także swój **prywatny obszar roboczy**.

# Etapy transakcji

## ”Życie” transakcji:

START(T) — transakcja jest uruchamiana, otrzymuje swój pierwszy znacznik odpowiadający momentowi uruchomienia w systemie. Otrzymuje także swój **prywatny obszar roboczy**.

działanie transakcji — transakcja czyta nowe jednostki z bazy danych, modyfikacje i ponowne odczyty wykonuje w swoim obszarze roboczym. Przez $RS(T)$ oznaczymy zbiór wszystkich jednostek odczytanych przez $T$ z bazy danych. Przez $WS(T)$ oznaczymy zbiór wszystkich jednostek zapisanych przez transakcję (na razie tylko w jej obszarze roboczym).

# Etapy transakcji

## ”Życie” transakcji:

START(T) — transakcja jest uruchamiana, otrzymuje swój pierwszy znacznik odpowiadający momentowi uruchomienia w systemie. Otrzymuje także swój **prywatny obszar roboczy**.

działanie transakcji — transakcja czyta nowe jednostki z bazy danych, modyfikacje i ponowne odczyty wykonuje w swoim obszarze roboczym. Przez $RS(T)$ oznaczymy zbiór wszystkich jednostek odczytanych przez $T$ z bazy danych. Przez $WS(T)$ oznaczymy zbiór wszystkich jednostek zapisanych przez transakcję (na razie tylko w jej obszarze roboczym).

VAL(T) — transakcja kończy obliczenia i zgłasza się do systemu. System sprawdza, czy transakcja działała na jednostkach we właściwej kolejności w stosunku do innych transakcji (szczegóły za chwilę).

- Jeśli test wypada negatywnie, to transakcja jest anulowana — kasowany jest jej obszar roboczy.
- Jeśli test wypada pozytywnie, to transakcja przechodzi do następnego etapu (patrz niżej).

# Etapy transakcji

## ”Życie” transakcji:

START(T) — transakcja jest uruchamiana, otrzymuje swój pierwszy znacznik odpowiadający momentowi uruchomienia w systemie. Otrzymuje także swój **prywatny obszar roboczy**.

działanie transakcji — transakcja czyta nowe jednostki z bazy danych, modyfikacje i ponowne odczyty wykonuje w swoim obszarze roboczym. Przez $RS(T)$ oznaczymy zbiór wszystkich jednostek odczytanych przez $T$ z bazy danych. Przez $WS(T)$ oznaczymy zbiór wszystkich jednostek zapisanych przez transakcję (na razie tylko w jej obszarze roboczym).

VAL(T) — transakcja kończy obliczenia i zgłasza się do systemu. System sprawdza, czy transakcja działała na jednostkach we właściwej kolejności w stosunku do innych transakcji (szczegóły za chwilę).

- Jeśli test wypada negatywnie, to transakcja jest anulowana — kasowany jest jej obszar roboczy.
- Jeśli test wypada pozytywnie, to transakcja przechodzi do następnego etapu (patrz niżej).

FIN(T) — transakcja, która przeszła walidację, zapisuje dane ze swojego obszaru roboczego do bazy danych; po zapisaniu otrzymuje ostatni znacznik oznaczający moment ukończenia: $FIN(T)$.

# Procedura walidacji

Dla każdej transakcji $T$ nazwijmy przedział czasowy $START(T)..VAL(T)$ **okresem czytania**, a przedział czasowy $VAL(T)..FIN(T)$ — **okresem zapisywania**.

# Procedura walidacji

Dla każdej transakcji $T$ nazwijmy przedział czasowy $START(T)..VAL(T)$ **okresem czytania**, a przedział czasowy $VAL(T)..FIN(T)$ — **okresem zapisywania**.

## Test szeregowalność $T$ z wcześniejszymi transakcjami:

Dla każdej transakcji wcześniejszej $S$, czyli takiej, dla której $VAL(S) < VAL(T)$:

# Procedura walidacji

Dla każdej transakcji $T$ nazwijmy przedział czasowy $START(T)..VAL(T)$ **okresem czytania**, a przedział czasowy $VAL(T)..FIN(T)$ — **okresem zapisywania**.

## Test szeregowalność $T$ z wcześniejszymi transakcjami:

Dla każdej transakcji wcześniejszej $S$, czyli takiej, dla której $VAL(S) < VAL(T)$:

- jeśli $WS(S) \cap RS(T) \neq \emptyset \ \wedge FIN(S) > START(T)$, to odrzuć $T$ (istnieje możliwość, że $T$ odczytała jednostkę $A \in WS(S) \cap RS(T)$ zanim $S$ zdążyła ja zapisać w bazie danych.

# Procedura walidacji

Dla każdej transakcji $T$ nazwijmy przedział czasowy $START(T)..VAL(T)$ **okresem czytania**, a przedział czasowy $VAL(T)..FIN(T)$ — **okresem zapisywania**.

## Test szeregowalność $T$ z wcześniejszymi transakcjami:

Dla każdej transakcji wcześniejszej $S$, czyli takiej, dla której $VAL(S) < VAL(T)$:

- jeśli $WS(S) \cap RS(T) \neq \emptyset \ \wedge FIN(S) > START(T)$, to odrzuć $T$ (istnieje możliwość, że $T$ odczytała jednostkę $A \in WS(S) \cap RS(T)$ zanim $S$ zdążyła ja zapisać w bazie danych.
- jeśli $WS(S) \cap WS(T) \neq \emptyset \ \wedge FIN(S) > VAL(T)$, to odrzuć $T$ (istnieje możliwość, że $T$ zapisze jednostkę $A \in WS(S) \cap WS(T)$ przed $S$.

# Procedura walidacji

Dla każdej transakcji $T$ nazwijmy przedział czasowy $START(T)..VAL(T)$ **okresem czytania**, a przedział czasowy $VAL(T)..FIN(T)$ — **okresem zapisywania**.

## Test szeregowalność $T$ z wcześniejszymi transakcjami:

Dla każdej transakcji wcześniejszej $S$, czyli takiej, dla której $VAL(S) < VAL(T)$:

- jeśli $WS(S) \cap RS(T) \neq \emptyset \ \wedge FIN(S) > START(T)$, to odrzuć $T$ (istnieje możliwość, że $T$ odczytała jednostkę $A \in WS(S) \cap RS(T)$ zanim $S$ zdążyła ja zapisać w bazie danych.
- jeśli $WS(S) \cap WS(T) \neq \emptyset \ \wedge FIN(S) > VAL(T)$, to odrzuć $T$ (istnieje możliwość, że $T$ zapisze jednostkę $A \in WS(S) \cap WS(T)$ przed $S$.

## Theorem

*Metoda walidacji gwarantuje szeregowalność harmonogramu transakcji w porządku zadanym wartościami VAL. Może powodować wycofywanie transakcji, ale nie występuje w niej anulowanie kaskadowe i brudne dane.*

# Poziomy niezależności SQL

## SET ISOLATION LEVEL

```
SET TRANSACTION ISOLATION LEVEL
{ READ ONLY | RED COMMITED | REPEATABLE READ | SERIALIZABLE };
```

| poziom niezależności | czas blokady rekordu odczytanego | czas blokady rekordu zapisanego | czas blokady odwiedzonej tabeli |
|---|---|---|---|
| `READ ONLY` | wcale | wcale | wcale |
| `READ COMITTED` | na czas operacji | do końca transakcji | wcale |
| `REPEATABLE READ` | do końca transakcji | do końca transakcji | wcale |
| `SERIALIZABLE` | do końca transakcji | do końca transakcji | do końca transakcji |

# Poziomy niezależności SQL

| poziom niezależności | czas blokady rekordu odczytanego | czas blokady rekordu zapisanego | czas blokady odwiedzonej tabeli |
|---|---|---|---|
| `READ ONLY` | wcale | wcale | wcale |
| `READ COMITTED` | na czas operacji | do końca transakcji | wcale |
| `REPEATABLE READ` | do końca transakcji | do końca transakcji | wcale |
| `SERIALIZABLE` | do końca transakcji | do końca transakcji | do końca transakcji |

- transakcja o poziomie `READ ONLY` może tylko czytać dane; nie zakłada przy tym blokady na dane i może czytać dane niepewne;
- transakcja o poziomie `READ COMITTED` nie uwalnia brudnych danych (blokuje zmienione dane do końca działania); ponieważ do odczytu potrzebuje blokady, więc nie czyta także brudnych danych;
- transakcja o poziomie `REPEATABLE READ` dzieki blokowaniu odczytanych jednostek ma gwarancje, że czytane nie zmienią się one w czasie jej działania;
- transakcja o poziomie `SERILIZABLE` może oprócz operacji na rekordach wykonywać bezpiecznie operacje na całych tabelach — ma gwarancje, że nie pojawią się w nich **fantomy** — jednostki wstawione przez inne transakcje po tym, jak dana transakcja przeszła przez dany obszar tabeli, ale jeszcze się nie zakończyła.