Doc. dr inż. Maria Chałon

Ochrona i bezpieczeństwo danych

Integralność baz danych

„ Integralność (ang.data integrity) to formalna poprawność bazy danych, jej fizycznej organizacji, zgodność ze schematem bazy danych i regułami dostępu..”

Integralność baz danych.

Odpowiednie mechanizmy **zabezpieczające przed** skutkami przypadkowych błędów logicznych, konfliktów we współbieżnym dostępie do danych oraz skutkami awarii oprogramowania i sprzętu komputerowego.

System integralny to taki, który **dostarcza nam** wiarygodne dane i jest zabezpieczony przed nieautoryzowaną modyfikacją informacji.

Systemy baz danych **powinny zapewniać** możliwość sprawdzania i ewentualnej korekty wprowadzanych danych oraz powinny zawierać odpowiednie mechanizmy zapewniające prawidłowe przetwarzanie danych.

Integralność to zapewnienie **kompletności, poprawności i wiarygodności** danych zgromadzonych w bazie.

Proces ochrony integralności obejmuje:

* kontrolę danych wejściowych oraz synchronizację dostępu do danych,
* poprawianie czyli korektę danych, cofanie i odtwarzanie stanu bazy,
* archiwizacje poprzez tworzenie kopii bazy oraz zapisów działania systemu,
* testowanie czyli sprawdzanie poprawności zawartości bazy.

Pojęcie integralności obejmuje **integralność statyczną i transakcyjną**.

Źródłem naruszenia **integralności statycznej** są błędy logiczne w danych oraz brak poprawnie skonstruowanego schematu bazy danych.

Zagrożeniem **integralności transakcyjnej** są awarie oprogramowania i sprzętu oraz współbieżny dostęp do danych.

Integralność statyczna

Integralność statyczna dotyczy poprawnie zaprojektowanego schematu bazy danych jak również spełnienia ograniczeń nałożonych na wartości atrybutów opisujących obiekty w bazie.

Wyróżniamy:

INTEGRALNOŚĆ SEMANTYCZNĄ

INTEGRALNOŚĆ ENCJI

INTEGRALNOŚĆ REFERENCZJNĄ

Integralność semantyczna.

Jeżeli wartości danych spełniają wcześniej zdefiniowane i nałożone ograniczenia wówczas mówimy, że zachowana jest integralność semantyczna.
Niżej podano opisany w języku SQL przykład zapewnienia integralności semantycznej w bazie relacyjnej.

Przykład 1.

```
CREATE TABLE Studenci
(NrIndeksu Number(5) NOT NULL UNIQUE;
NazwiskoStud Varchar (15);
NazwaWydziału Varchar (20);
Stypendium Decimal (7,2);
PRIMARY KEY (NrIndeksu));
CHECK (NrStud BETWEEN 10 AND 500);
```

Na przykład utrata integralności semantycznej może wynikać z:

niewłaściwej wartości pola danych (np. wiek 200 lat),

niezgodności między wartościami pól tej samej jednostki danych,

niespójności między polem jednostki, a polem jednostki połączonej,

obecności pustych pól, nieważnych wartości.

Zapewnienie integralności semantycznej ma na celu zabezpieczenie danych przed celową lub przypadkową błędną modyfikacją danych, a więc odrzucenie wszelkich akcji powodujących niespójność bazy lub uruchomienie akcji, które przywracają poprawność i spójność bazy.

Integralność można wymusić w sposób **deklaratywny** poprzez **więzy integralności** oraz w sposób **proceduralny** poprzez tzw. **wyzwalacze** (ang. triggers).

Więzy integralności są to pewne warunki, które muszą być spełnione poprzez określony podzbiór danych w bazie. Warunki te muszą pozostać prawdziwe przy każdej operacji modyfikacji w bazie danych. Każda operacja naruszająca te więzy musi zostać anulowana. Typowy przykład narzuconych ograniczeń: wartości atrybutów w danej tabeli muszą pochodzić z określonej dziedziny. Więzy integralności dzielimy na statyczne i dynamiczne. Więzy statyczne muszą być spełnione w bieżącym i następnym stanie bazy, więzy dynamiczne, temporalne określają poprawność danych w odniesieniu do historii stanów przechowywanych w bazie.

Wyzwalacze są to procedury uruchamiane automatycznie przez system przy zajściu jakiegoś zdarzenia dotyczącego danego obiektu.

Przykład

Ich zadaniem jest kontrola poprawności wprowadzanych lub przetwarzanych danych do postaci akceptowalnej przez użytkownika.

Wyzwalacze nie dopuszczają lub cofają zmiany, które naruszają zasady integralności. Utworzenie wyzwalacza polega na określeniu zdarzenia uruchamiającego wyzwalacz tabeli będącej jego celem i zaprojektowaniu akcji jaka ma ta procedura wykonać.

W języku SQL instrukcje INSERT, UPDATE, DELETE służą do uruchamiania procedury wyzwalacza.

IF UPDATE nazwa kolumn
BEGIN
wyrażenie SQL
END
lub
IF UPDATE nazwa kolumn AND UPDATE nazwa kolumn
BEGIN
wyrażenie SQL
END

Akcja może być uruchamiana przed (wyzwalacz typu **BEFORE**) lub po (wyzwalacz typu **AFTER**) określonym zdarzeniu.
Może też chronić przed zajściem zdarzenia lub może zmienić wynik zdarzenia.

Wyzwalacz może zadziałać raz dla operacji, która go wywołuje lub wielokrotnie dla każdego wiersza (z klauzulą **FOR EACH ROW**). Wykorzystanie tej klauzuli umożliwia dostęp zarówno do wartości sprzed zajścia zdarzenia (OLD. atrybut relacji) jak i do nowych wartości atrybutu (NEW. atrybut relacji) powstałych w wyniku wstawienia, modyfikacji czy skreślenia wiersza w trakcie zdarzenia.

```
ALTER TRRIGER DELETE Klient;
ON Klient FOR DELETE;
AS;
BEGIN;
DELETE FROM Polisa;
WHERE Polisa.ubezpieczony.nr = Klient.klient.nr;
END
```

Przykłady wyzwalaczy

```
ALTER TRRIGER policz przychody;
ON Ksiega przychodu FOR INSERT;
AS;
BEGIN;
UPDATE Ksiega przychodu SET;
Przychód=składka*Prowizja/100;
END
```

Wyzwalacze **mogą zagnieżdżać w sobie inne wyzwalacze**. Każdy wyzwalacz może uruchamiać inny.

Liczba poziomów zagnieżdżenia zależy od systemu.

Bardzo efektywnym mechanizmem zabezpieczającym integralność danych są procedury składowane. Poprzez te procedury realizowany jest dostęp do danych. Są one wywoływane bezpośrednio przez aplikacje klienta. Klient poprzez podanie nazwy i parametrów wywołuje procedurę, która sprawdza, czy wprowadzone zmiany nie naruszają integralności semantycznej.

Integralność encji i integralność referencyjna.

Integralność encji zapewnia się na etapie definiowania schematu bazy danych.

Przykład

CREATE TABLE Studenci
(NrIndeksu Number(5) **NOT NULL UNIQUE**;
NazwiskoStud Varchar (15);
NazwaWydziału Varchar (20);
Stypendium Decimal (7,2);
PRIMARY KEY (NrIndeksu));

W schemacie bazy danych tablice powiązane są między sobą kluczami

Powiązania te realizowane są przez **klucze obce (ang. FOREIGN KEY**).

Powiązania między kluczami encji pociągają za sobą konieczność określenia reguł postępowania w wypadku wykonywania operacji na tabelach nadrzędnych w stosunku do innych tabel.

To właśnie **integralność referencyjna** określa stany w jakich może znajdować się wartość klucza obcego w danej tabeli. Wartość klucza obcego w danej tabeli musi być albo równa wartości klucza głównego w tabeli z nią powiązanej, ewentualnie wartości NULL. Niżej podano przykłady zapewnienia integralności referencyjnej w języku SQL.

Przykład

CREATE TABLE Wydziały;
(NazwaWydziału Char (15);
RokStudiów Smallint;
KodKursu Char (3);
NrStud Number (5);
PRIMARY KEY (NazwaWydziału);
FOREIGN KEY(NrIndeksuIDENTIFIES Studenci);
ON DELETE SET NULL;
ON UPDATE CASCADE)

Przykład

CREATE TABLE Wydziały;
(NazwaWydziału Char (15);
RokStudiów Smallint;
KodKursu Char (3);
NrStud Number (5);
PRIMARY KEY (NazwaWydziału);
FOREIGN KEY(NrIndeksu IDENTIFIES Studenci);
DELETE RESTRICTED;
ON UPDATE CASCADE)

Definicja dwóch tabel z uwzględnieniem więzów integralności oraz określenie asercji czyli więzów niezależnych od tabeli i dziedziny podano w przykładzie .

Przykład 5

CREATE TABLE **Grupy**;

(NrGrupy Integer (2), **NOT NULL UNIQUE**,

NazwaGrupy Varchar (30),

NrStarGrupy Integer (2),

PRIMARY KEY (NrGrupy));

CREATE TABLE **Studenci**;

(NrIndeksu Integer(2), **NOT NULL,**

Nazwisko Varchar (20),

Adres Varchar (50),

DataRozp.Stud DATE DEFAULT Sysdate,

Stypendium Decimal (8,2),

NrGrupy Integer (2);

PRIMARY KEY (NrIndeksu);

FOREIGN KEY (NrGrupy);

REFERENCES Grupy;

ON UPDATE CASCADE;

ON DELETE SET NULL;

CHECK (NrStud BETWEEN 10 AND 500);

```
CREATE ASSERTION MinStypendium;
AFTER
INSERT ON Studenci;
UPDATE OF Stypendium ON Studenci;
CHECK
( NOT EXIST
(SELECT *
FROM Studenci
WHERE Stypendium > 700));
```

Dzięki więzom asercji uniemożliwia się przyznanie studentom zbyt niskich stypendiów.

Innym sposobem zachowania integralności referencyjnej są wspomniane wyżej procedury zwane wyzwalaczami.

Integralność w różnych modelach baz danych.

Jeżeli mamy do czynienia z **obiektową bazą danych to problem integralności** jest bardziej skomplikowany niż miało to miejsce w bazach relacyjnych.

W bazie obiektowej każdy obiekt musi mieć unikatowy identyfikator w bazie. Ponieważ dostęp do obiektów odbywa się za pomocą metod tych obiektów ograniczenia integralności muszą być definiowane poprzez pewne warunki umieszczone w treściach metod.

Integralność danych wynika ze związków między obiektami i klasami.

Model obiektowy obejmuje cztery rodzaje integralności:

* integralność klasa-klasa- nadklasa nie może być usunięta dopóki nie zostaną usunięte powiązane z nią podklasy,

* integralność klasa-obiekt -klasa nie może być usunięta dopóki nie zostaną usunięte powiązane z nią obiekty,

* integralność dziedziny- atrybuty są definiowane na wcześniej utworzonych klasach lub na zbiorach identyfikatorów obiektów,

* integralność referencyjna- ponieważ klasy mogą być powiązane z innymi klasami poprzez związki, to w modelu obiektowym mamy do czynienia z integralnością referencyjną podobną do integralności referencyjnej w modelu relacyjnym.

Ciekawostkę stanowią **bazy temporalne,** czyli modelujące historię zmian rzeczywistości.
Model temporalny to model uwzględniający czas rejestracji danych oraz rzeczywisty czas zajścia zdarzeń.
Temporalne więzy integralności określają poprawność danych zarówno dla bieżącego stanu bazy danych jak i stanów poprzednich i przyszłych.

Ogólnie są one przydatne wszędzie tam, gdzie są istotne zależności czasowe między danymi. Np. można zadać taki warunek: pensja pracownika nie może wzrosnąć o więcej niż 12% w ciągu 4 kolejnych posiedzeń zarządu. W bazach statycznych możliwe jest tylko sprawdzenie warunku czy np. pensja pracownika nie może wzrosnąć jednorazowo o więcej niż 12%.

Do specyfikacji temporalnych więzów integralności stosowana jest logika temporalna. Wykorzystuje ona operatory odnoszące się do przeszłości. Jeżeli chce się dołączyć nowy stan należy zweryfikować poprzednie, jeżeli są spełnione to nowa historia stanów jest zapamiętana.

Integralność transakcyjna

Transakcja jest to ciąg operacji wykonywanych na danych w bazie, inaczej mówiąc ciąg instrukcji języka SQL tworzących pewną całość.

Transakcja od momentu jej rozpoczęcia, aż do chwili jej zakończenia może znajdować się w jednym z pieciu stanów: **aktywna, częściowo zatwierdzona, zatwierdzona, nieudana, odrzucona.**

Transakcja może zakończyć się:
-**powodzeniem**, wówczas jest zatwierdzana (ang. **Commit**),
-**niepowodzeniem**, wówczas jest odrzucana (ang. **Abort**), lub wycofywana (ang. **Rollback**)

Transakcja powinna posiadać następujące cechy:

* niepodzielność (ang. Atomicity) – gwarantuje, że transakcja musi być wykonana w całości lub całkowicie anulowana,

* spójność (ang. Consistency)- zapewnia nienaruszalność zasad integralności, czyli baza po operacji musi być spójna,

* izolacja (ang. Isolation)-mówi nam o tym, że dane, które są modyfikowane przez jedną transakcję przed jej zakończeniem muszą być niedostępne dla innych transakcji,

* trwałość (ang. Durability)- gwarantuje, że po pomyślnym zakończeniu transakcji zmiany w bazie będą na stałe zapisane i nie zostaną utracone w wyniku jakiejkolwiek późniejszej awarii.

Źródłem zagrożeń dla integralności transakcyjnej są:

* awarie programowe i sprzętowe zaistniałe w trakcie wykonywania transakcji. Na ogół wynikają one z utraty zawartości pamięci operacyjnej komputera,

* błędy transakcji spowodowane wykonywaniem zabronionych operacji (dzielenie przez zero), czy niemożnością wykonania danej operacji (niewystarczający stan konta),

* nieprawidłowa realizacja równoczesnego dostępu do tych samych danych przez różnych użytkowników,

* błędy zapisu lub odczytu danych z dysku podczas wykonywania transakcji,

* zanik napięcia, zamazanie danych przez operatora, pożar, kradzież, sabotaż.

Utrzymanie integralności wymaga:
* właściwego zarządzania transakcjami, aby po awarii przywrócić spójny stan bazy danych,
* odpowiednich mechanizmów sterowania współbieżnym dostępem do danych.

Można wyróżnić dwie strategie działania transakcji:

-transakcje działają na własnych kopiach, które są zamieniane z oryginalnymi obiektami w fazie zatwierdzenia. Rozwiązanie to, wymagające większego obszaru pamięci, jest rzadko stosowane ze względu na wiekszą możliwość awarii.
-transakcje działają bezpośrednio na bazie danych w specjalnych plikach zwanych dziennikami, potocznie nazywanych logami (ang.log). W logach zapisują wszelkie operacje aktualizacyjne, które wykonały, wraz z obiektami przed i ewentualnie po aktualizacji. W razie awarii baza jest odtwarzana poprzez analizę informacji zapisanej w logu.

Dzienniki transakcji.

Jeżeli w wyniku awarii systemy ciąg operacji tworzących transakcje zostanie przerwany wówczas stosuje się zasadę wszystko albo nic to znaczy należy wycofać z bazy efekty częściowego wykonania transakcji.

Jest to możliwe dzięki prowadzeniu **dziennika transakcji zwanego w skrócie logiem.**

Log jest to plik, w którym rejestruje się przebieg wykonywania operacji.

W logu rejestrowany jest:

* identyfikator transakcji,
* adresy wszystkich obiektów aktualizowanych przez daną transakcję,
* wartości obiektów przed i po modyfikacji
* informacje dotyczące przebiegu transakcji.

Dopiero po zapisaniu **polecenia zatwierdzającego transakcje (ang.COMMIT)**, a dokładnie punktu **zatwierdzenia transakcji (ang. COMMIT POINT)**, wszelkie modyfikacje wykonywane przez transakcje są przepisywane do fizycznej bazy danych. Przed osiągnięciem punktu zatwierdzenia wszystkie takie aktualizacje trzeba uważać za **tymczasowe, ponieważ mogą być odwołane.**

Technika ta zwana **wyprzedzającym zapisem do logu** pozwala łatwo wycofać transakcje, która nie może zostać dokończona. Czyli zostawia się stan bazy bez zmiany, pozostaje tylko poinformować użytkownika, że transakcja nie doszła do skutku.

Jeśli transakcja **została zatwierdzona**, to zmiany przez nią wprowadzone muszą być na trwałe zapamiętane w bazie, **nawet jeśli wystąpiła awaria i nie zostały przepisane do fizycznej bazy.**

Należy wówczas ustalić, które transakcje zostały zatwierdzone, a nie przepisane do fizycznej bazy, **a które nie były zatwierdzone i powinny zostać anulowane.** W tym celu jest określany tzw. **punkt kontrolny logu (CP)** czyli punkt zapisu do logu wszystkich transakcji i uaktualnianie ich.

Tworzenie punktu kontrolnego polega na:

* wstrzymaniu uruchomień nowych transakcji,

* zaczekaniu aż wszystkie rozpoczęte transakcje wprowadzą do logu zapisy potwierdzenia<COMMIT> lub odrzucenia <ABORT>,

* przesłaniu logu na dysk,

* wprowadzenie do logu zapisu <CP> i ponownym przesłaniu logu na dysk,

* wznowieniu wstrzymanych transakcji.

Jeśli wystąpi awaria to bada się przebieg transakcji od ostatniego punktu kontrolnego. Wszystkie transakcje, które po punkcie kontrolnym są zatwierdzone instrukcją COMMIT uaktualnia się i zapisuje się na dysku. Transakcje te mogły nie być fizycznie przepisane.

Inne transakcje anuluje się stosując instrukcje ROLLBACK.

Można wyróżnić kilka rodzajów logów. Na szczególną uwagę zasługują

* logi z unieważnieniem,
* logi z powtarzaniem,
* logi hybrydowe powstałe z połączenia tych dwóch typów.

W logach z **unieważnieniem** przy odtwarzaniu likwiduje się zmiany wprowadzone przez nie zatwierdzone transakcje. Odtwarzanie odbywa się przez przywrócenie uprzednich wartości wszystkich nie zatwierdzonych transakcji.

Zmiany w bazie danych muszą być zapamiętane na dysku zanim w logu na dysku zostaje zapamiętany zapis COMMIT.

Dane należy zapisywać na dysk natychmiast po zakończeniu transakcji, co wielokrotnie może prowadzić do zwiększenia liczby potrzebnych dostępów do dysku.

W logach z trybem powtarzania ignoruje się transakcje niezatwierdzone.

Zmiany wykonywane przez transakcje zatwierdzone powtarza się.

Odtwarzanie obejmuje **przypisanie nowych wartości** ze wszystkich transakcji zatwierdzonych.

Zapis COMMIT zostaje zapamiętany na dysku, **zanim wartości zmienione** zapisze się na dysku. Prowadzi to do konieczności przechowywania zmodyfikowanych bloków w buforach. Bloki te trzyma się aż do zatwierdzenia transakcji i zakończenia tworzenia zapisów w logu. Może to zwiększyć liczbę buforów potrzebnych przy wykonywaniu transakcji.

Oba typy logów mogą spowodować powstanie sprzecznych wymagań.

Dotyczy to obsługi buforów w trakcie wstawiania punktów kontrolnych, o ile elementy danych nie są całymi blokami lub zbiorami bloków.

Dlatego też stosuje się logi stanowiące **połączenie dwóch technik:** odtwarzania i powtarzania. Logi te są bardziej elastyczne w zakresie kolejności wykonywania czynności, ale z kolei wymagają zapisywania większej liczby danych do logu.

Odtwarzanie polega na powtórzeniu transakcji zatwierdzonych oraz unieważnieniu transakcji nie zatwierdzonych.

Aby w sposób jasny i klarowny opisać efekty wykonania transakcji w warunkach awaryjnych wprowadzono pseudojęzyk oparty na komendach SQL.

INPUT < X > pobieranie do bufora pamięci elementów bazy danych X z **dysku**

OUTPUT < X > kopiowanie bufora pamięci na **dysk**

READ < X, y > kopiowanie elementu bazy danych X do lokalnej zmiennej transakcji y

WRITE < X, y > kopiowanie wartości zmiennej y do elementu bazy danych X w buforze pamięci

SET < X > tworzenie przez transakcje nowej wartości w swojej przestrzeni adresowej

<SEND LOG> przesyłanie logu, czyli skopiowanie pliku na dysk

< BEGIN T > rozpoczęcie transakcji

<COMMIT T > zatwierdzenie transakcji

< ABORT T > unieważnienie transakcji

< T, y, old w > zapis aktualizujący w logu z unieważnieniem

< T, y, new w > zapis aktualizujący w logu z powtórzeniem

< T, y, old w, new w > zapis aktualizujący w logu hybrydowym (z unieważnieniem/powtórzeniem)

< CP> punkt kontrolny

< BEGIN CP (T1, T2,.....TK) > początek bezkolizyjnego punktu kontrolnego (transakcje aktywne)

< END CP> koniec punktu kontrolnego

Ri (X) transakcja Ti czyta element z bazy danych

Wi (X) transakcja Ti zapisuje element do bazy danych

P(Ti,Tj) plan sekwencyjny gdy i-ta transakcja poprzedza transakcję j-tą

P(Tj,Ti) plan sekwencyjny gdy j-ta transakcja poprzedza transakcję i-tą

Ok(Ti) k-ta operacja i-tej transakcji,

STOPi (X) – blokada do odczytu zakładana przez transakcję Ti

GOi (X)- zwolnienie blokady przez transakcję Ti

Zazwyczaj przebieg transakcji odbywa się w dwóch krokach:

Krok 1. Operacja odczytania elementu bazy danych:

INPUT < X > pobieranie do bufora pamięci elementów bazy danych X z dysku

READ < X, y > wczytanie zawartości buforów do przestrzeni adresowej transakcji

Krok 2. Operacja zapisania nowej wartości elementu do bazy danych:

SET < X > tworzenie przez transakcję nowej wartości w swojej przestrzeni adresowej

WRITE < X, y > kopiowanie wartości zmiennej y do elementu bazy danych X w buforze pamięci

OUTPUT < X > zapisanie z bufora pamięci na dysk

Wszystkie trzy typy logów mają podobna postać, różnią się tylko zapisem aktualizacyjnym. Niżej przedstawiono zapis w logach dla uproszczenia przyjmując, że mamy do czynienia z jedną transakcją składającą się z dwóch elementów A i B.

Postać logu typu unieważnienie:

< BEGIN T >

< T, A, old w >

< T, B, old w >

<COMMIT T >

Postać logu typu powtarzanie:

< BEGIN T >

< T, A, new w >

< T, B, new w >

<COMMIT T >

Postać logu typu unieważnienie/powtarzanie:

< BEGIN T >

< T, A, old w, new w >

< T, B, old w, new w >

<COMMIT T >

Cały problem polega na tym w jaki sposób i kiedy zapisuje się powstałe w wyniku transakcji zmiany na dysku.

W logu z **unieważnieniem** nie można skopiować A i B na dysk, zanim nie znajdzie się tam log z zapisem zmian. A więc najpierw wykonujemy operacje SEND LOG. Potem dopiero następuje zapisanie wartości A i B na dysk, zatwierdzenie transakcji T, zapisanie <COMMIT T > w logu. Kolejny krok to ponowne umieszczenie logu na dysku, aby zapewnić, że zapis <COMMIT T > jest trwały.

W przypadku **logu z powtarzaniem** w zapisach logu, które opisują zmiany są nowe wartości (new w) A i B. Po zakończeniu transakcji WRITE < X, y > następuje jej zatwierdzenia <COMMIT T >. Potem log jest przesyłany na dysk < SEND LOG>, czyli wszystkie zapisy dotyczące zmian wykonywanych przez transakcję T zostają zapamiętane na dysku. Dopiero wtedy można na dysku zapamiętać nowe wartości A i B.

W przypadku logu typu **unieważnienie/powtarzanie** istotne jest, aby zanim na dysku zapisze się zmianę wartości elementu X spowodowaną działaniem transakcji T, należy najpierw na dysk wprowadzić zapis
< T, X, old w, new w >.

Logi z unieważnieniem.

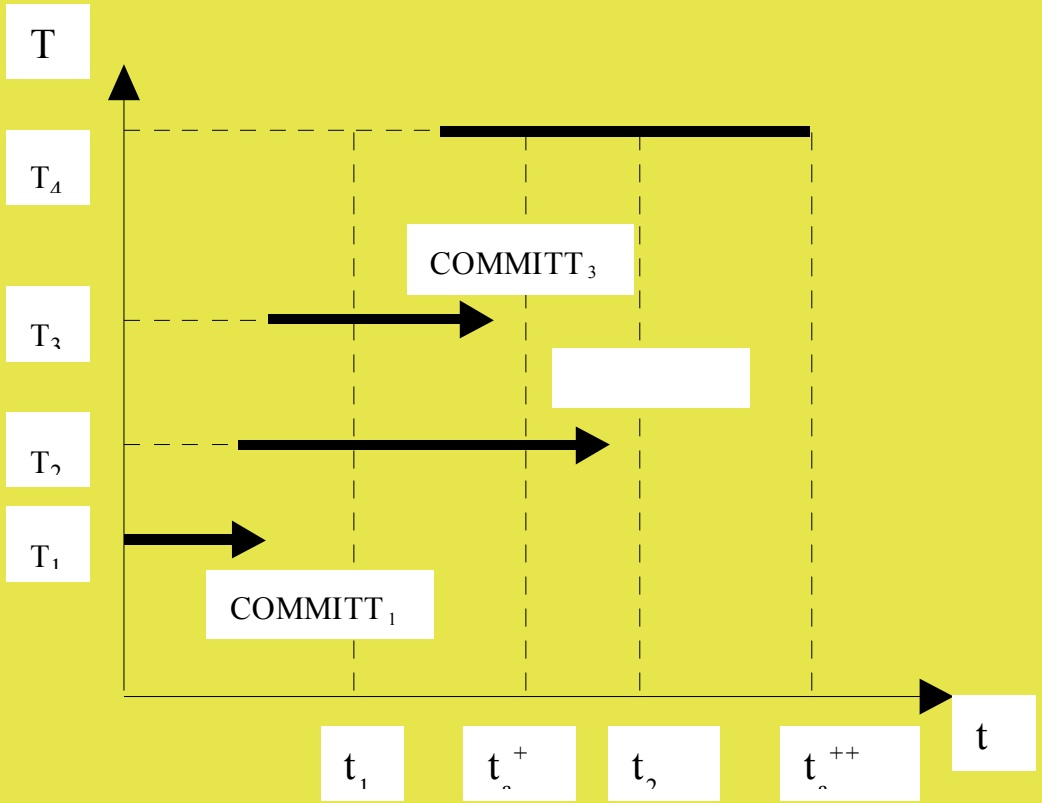
Jeżeli w wyniku awarii transakcja nie została wykonana w całości należy wówczas odtworzyć spójny stan bazy danych przy pomocy logu.
Algorytm.

Krok1: Badamy, czy dla danej transakcji T wykonała się operacja SEND LOG.
Krok2: Jeśli TAK to przechodzimy do Kroku 8, jeśli nie to
Krok3: Badamy, czy dla danej transakcji T <COMMIT T > jest zapisana na dysku.
Krok4: Jeśli TAK to przechodzimy do Kroku 8, jeśli nie to
Krok5: Kolejno od ostatniej komendy sprawdzamy wszystkie operacje i przypisujemy elementom stare wartości < T, y, old w > .
Krok6: Do logu wprowadzamy zapis < ABORT T >
Krok 7: Wykonujemy SEND LOG
Krok8: STOP

Jeżeli w trakcie odtwarzania wystąpiła awaria to powtarzamy algorytm.

Bardzo często wiele transakcji wykonuje się równocześnie. Aby określić jakie zmiany nastąpiły i w jaki sposób przywrócić spójny stan bazy danych wprowadza się punkty kontrolne.

Na rysunku przedstawiono przykład realizacji w pewnym przedziale czasowym czterech transakcji T1, T2, T3, T4.



t

Rys.1 Przykład transakcji zapisywanych w logach z unieważnieniem.

Bezkolizyjne punkty kontrolne oznaczono t1 i t2, moment wystąpienia awarii ta gdzie:

t1 : < BEGIN CP (T2, T3) >

t2 : < END CP >

Jeśli t1< ta< t2 czyli ta =ta+ to awaria nastąpiła między punktami kontrolnymi,

jeśli ta> t2 czyli ta =ta++ to awaria nastąpiła po zakończeniu punktu kontrolnego.

Ograniczymy się tylko dla transakcji aktywnych w trakcie wystawiania punktu kontrolnego.
A więc transakcja T1 nie jest brana pod uwagę.

Rozważymy 2 przypadki wystąpienia awarii, które w pełni oddają skutki awarii:

1.ta =ta+

W chwili ta+ transakcje T2, T4 są aktywne, nie zostały zakończone dlatego należy unieważnić zmiany wprowadzone przez te transakcje. Czyli wprowadzamy zapisy aktualizacyjne < T2, X, old w >, < T4, X, old w > i komendy< ABORT T2 >,< ABORT T4 >.

Badając operacje od punktu ta+ wstecz (w logu z unieważnieniem zaczynamy sprawdzanie wszystkich transakcji od końca logu) natrafiamy na punkt t1 : < BEGIN CP (T2, T3) >. Oczywiste jest, że awaria nastąpiła w tym przedziale kontrolnym. Oprócz transakcji T2, T4 tylko transakcja T3 może być nie zatwierdzona. Napotykamy jednak na zapis <COMMIT T3 >, a więc transakcja została zatwierdzona i nie bierze się jej pod uwagę. Dlatego nie ma sensu sprawdzania logu wstecz dalej niż do początku najwcześniejszej z tych transakcji w tym wypadku T2.

2.ta =ta++

Awaria nastąpiła w chwili ta++. Badając operacje od punktu ta++ wstecz napotkamy na koniec punktu kontrolnego w chwili t2 : < END CP >. Wiadomo, że wszystkie nie zakończone transakcje zaczęły się po poprzednim zapisie: < BEGIN CP (T2,T3) >. Sprawdzając wstecz należy uaktualnić niezatwierdzoną transakcję T4 wprowadzając <T4, X, old w>, następnie usuwamy transakcje T4 poprzez komendę <ABORT T4>. Wszystkie inne transakcje (T2,T3) są zatwierdzone przez <COMMIT T2 > i <COMMIT T3 > czyli nic się nie zmienia.

Logi z powtarzaniem.

Jeżeli w wyniku awarii transakcja nie została wykonana w całości należy wówczas odtworzyć spójny stan bazy danych przy pomocy logu.

Dla logów z powtarzaniem skupiamy się na transakcjach zatwierdzonych komendą COMMIT. Transakcje te należy powtarzyć. Jeśli transakcja T nie jest zatwierdzona, to znaczy, że zmiany nie zostały zapisane na dysku i traktujemy T jakby jej w ogóle nie było.

Dwie różne transakcje zatwierdzone mogą zmieniać w różnym czasie wartość tego samego elementu bazy danych. Zapisy w logach sprawdzamy od najwcześniejszego do najpóźniejszego. W wypadku transakcji zatwierdzonej pojawia się problem, które zmiany zostały na dysku zapamiętane.

Aby odzyskać dane po awarii wykonujemy następujące kroki:

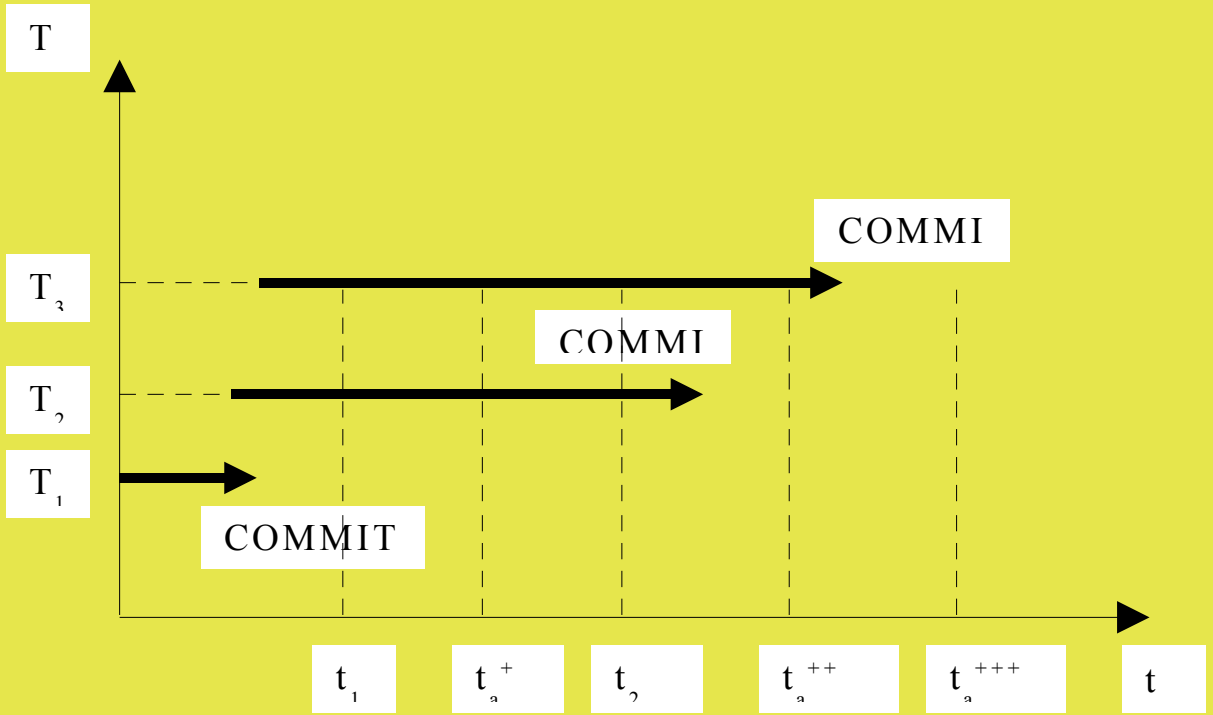
Krok1: Badamy, czy dla danej transakcji Ti wykonała się operacja <COMMIT Ti >,

Krok2: Jeśli NIE to przechodzimy do Krok6,

Krok3: Jeśli TAK badamy, czy dla danej transakcji Ti wykonano SEND LOG.

Krok4: Jeśli TAK to transakcja Ti jest zakończona, sprawdzamy log, aktualizujemy zapisy
< Ti, X, new w >
*Krok5:*Jeśli NIE (to mimo to, że zapis mógł znaleźć się w logu , ale może nie być na dysku) Ti traktujemy jako nie zakończoną i wprowadzamy zapis < ABORT Ti >
Krok6: STOP

Na rysunku przedstawiono przykład realizacji w pewnym przedziale czasowym trzech transakcji T1, T2, T3,



Rys.2 Przykład transakcji zapisywanych w logach z powtarzaniem.

Punkty kontrolne oznaczono t1 i t2, moment wystąpienia awarii ta gdzie:

t1 : < BEGIN CP (T2, T3) >
t2 : < END CP >

jeśli t1< ta< t2 czyli ta =ta+ to awaria nastąpiła między punktami kontrolnymi oraz przed zatwierdzeniem transakcji T2, T3.
jeśli ta> t2 są 2 przypadki

a)ta =ta++ to awaria nastąpiła po zatwierdzeniu transakcji T2, i przed zatwierdzeniem transakcji T3 i po komendzie < END CP >

b)ta =ta+++ to awaria nastąpiła po zatwierdzeniu transakcji T2, T3 i po komendzie < END CP >

Rozważymy trzy przypadki wystąpienia awarii:

1.ta =ta+

Awaria nastąpiła miedzy punktami kontrolnymi przed <END CP>. Wówczas wyszukujemy wstecz najbliższy zapis początku bezkonfliktowego punktu kontrolnego < BEGIN CP (T2, T3) >. Zbiór wszystkich transakcji to{ T2, T3 }. Ponieważ < BEGIN CP (T2, T3) > jest to jedyny punkt kontrolny to sprawdzamy cały log. Jedyną zatwierdzoną transakcją jest T1. Powtarzamy jej działanie < T1, X, new w > i po odtworzeniu do logu wprowadza się < ABORT T2 > i < ABORT T3 >

2.ta =ta++

Awaria wystąpiła pomiędzy komendami <COMMIT T2 > i <COMMIT T3 >. Szukamy wstecz pierwszego wystąpienia komendy <END CP>. Jest to punkt t2. Wiadomo więc, że wystarczy powtórzenie tylko tych transakcji, które zaczęły się po zapisie < BEGIN CP >, albo są ze zbioru {T2, T3}. Znajdujemy zapis <COMMIT T2 > czyli powtarzamy T2. Aktualizujemy zapisy tylko dla transakcji T2 gdzie < T2, X, new w > nic nie zmieniając dla T3. Po odtworzeniu do logu wprowadza się zapis usunięcia transakcji T3 < ABORT T3 > .

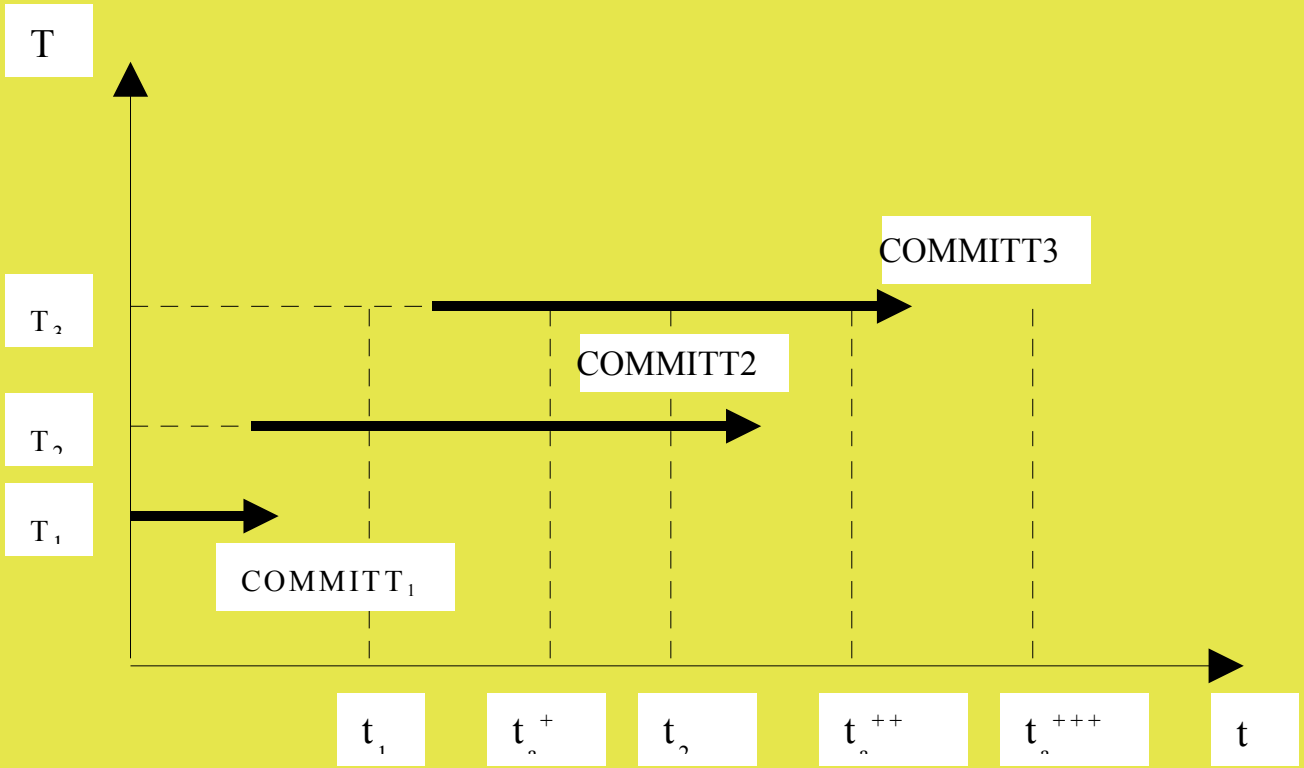
3.ta =ta+++

Awaria wystąpiła poza przedziałem kontrolnym. Szukamy wstecz pierwszego wystąpienia <END CP> . Jest to punkt t2. Wiadomo więc, że wystarczy powtórzenie tylko tych transakcji, które zaczęły się po zapisie < BEGIN CP (T2, T3) >, albo są ze zbioru {T2, T3}. Znajdujemy zapis <COMMIT T2 > i <COMMIT T3 >. Wiadomo, że trzeba je powtórzyć. Przeszukujemy log wstecz do napotkania rekordów: < BEGIN T2> , < BEGIN T3>, aktualizujemy wszystkie zapisy < T2, X, new w > i < T3, X, new w > dla transakcji T2 i T3.

Aby uniknąć przechowywania zbyt dużej liczby informacji można w zapisie: < BEGIN CP (T1 T2……. Tm) oprócz nazwy transakcji dodać wskaźnik do tych adresów w logu w których te transakcje się zaczynają.

Logi hybrydowe (unieważnienie/powtarzanie).

Algorytm odtwarzania po awarii za pomocą logów hybrydowych polega na powtórzeniu wszystkich transakcji zatwierdzonych poleceniem <COMMIT Ti > zaczynając od najwcześniejszych. Następnie unieważniamy wszystkie nie zakończone transakcje zaczynając od najpóźniejszych.



Rys.3 Przykład transakcji zapisywanych w logach hybrydowych.

Rozważymy trzy przypadki wystąpienia awarii:

1.ta =ta+

Awaria nastąpiła gdy obydwie transakcje T2 i T3 nie są zatwierdzone. Unieważniamy transakcje T2 i T3, aktualizujemy wszystkie zapisy <T2, X, old w > i <T3, X, old w > dla transakcji T2 i T3, cofamy transakcje T2 i T3 poprzez < ABORT T2 > i < ABORT T3 >

2.ta =ta++

Awaria nastąpiła gdy transakcja T2 jest zatwierdzona, a transakcja T3 nie jest kompletna. Powtarzamy transakcje T2, aktualizujemy wszystkie zapisy <T2, X, new w > dla transakcji T2 zapamiętując na dysku <new w>. Aktualizujemy wszystkie zapisy <T3, X, old w > dla transakcji T3 i cofamy transakcje T3 < ABORT T3 > .

3.ta =ta+++

Awaria nastąpiła gdy obydwie transakcje T2 i T3 są zatwierdzone. Ponieważ transakcja T1 była zatwierdzona przed < BEGIN CP >, to zakładamy, że jest zapisana na dysk. Powtarzamy transakcje T2 i T3, aktualizujemy wszystkie zapisy < T2, X, new w > i < T3,X, new w > dla transakcji T2 i T3. zapamiętując na dysku <new w>.

Współbieżność.

W przypadku sekwencyjnego wykonywania poszczególnych transakcji baza danych zawsze będzie w stanie spójnym, jeśli oczywiście będzie zachowana semantyczna integralność. W systemach wielodostępnych, a szczególnie w rozproszonych bazach danych, zachowanie spójności przy współbieżnym wykonywaniu transakcji jest dużym problemem. Nie mniej jednak ze względu na efektywność systemu, ważne jest, aby wiele transakcji było wykonywane jednocześnie.

Celem **sterowania współbieżnością** jest ochrona spójności bazy danych w sytuacji równoczesnego dostępu do tych samych danych przez wielu użytkowników. Wiąże się to z koniecznością zapewnienia **bezkolizyjności przebiegu wielu transakcji. Jedną z metod sterowania współbieżnością jest planowanie.**

Plany

Każda transakcja Ti składa się z ciagu operacji o1 (Ti),o2 (Ti)….on (Ti).

Plan P={ T1, T2,… Ti} jest to zbiór transakcji lub zbiór operacji uporządkowanych w czasie.

Zakładamy, ze kilka transakcji może mieć dostęp do tego samego elementu bazy danych. Mówimy wtedy o współbieżności dostępu do danych.

Przeanalizujemy następujące przypadki:

* Każda transakcja Ti wykonywana jest w całości.
Zachowana zostaje zasada izolacji.

Po wykonaniu transakcji Ti baza przechodzi z jednego stanu spójnego w drugi stan spójny.

Wprowadzimy pojęcie **planu sekwencyjnego**.

Plan sekwencyjny jest to taki plan, w którym wszystkie operacje jednej transakcji poprzedzają wszystkie operacje innej transakcji. Jeśli jakaś operacja on (Ti) poprzedza operacje ok (Tj) to wszystkie operacje transakcji Ti muszą poprzedzać wszystkie operacje transakcji Tj.

Przykład 6

Mamy dane dwie transakcje Ti i Tj.

Ograniczamy się tylko do operacji na buforach READ < X, y > i
WRITE < X, y > bez przepisywania informacji na dysk.

Transakcja Ti składa się z następujących operacji:

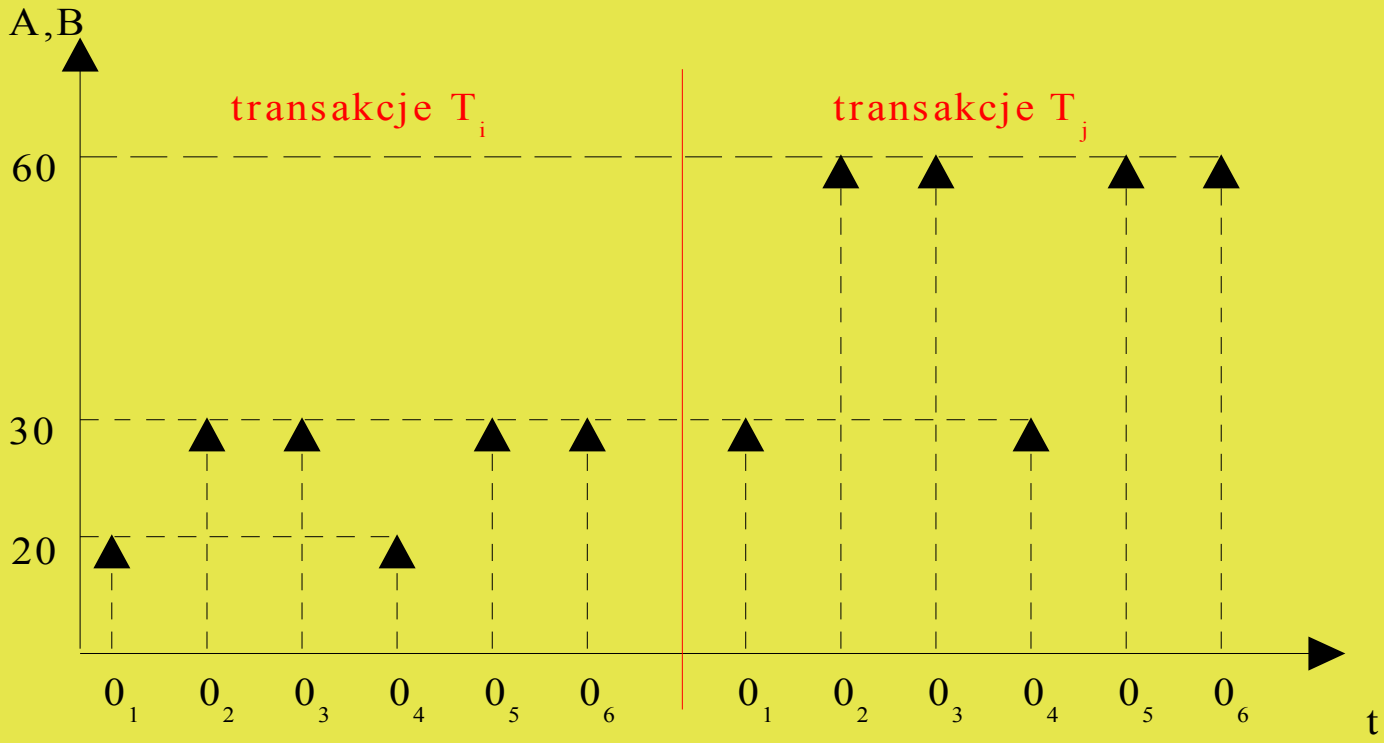
READ < A, x >	operacja o1 (Ti).
x:=x+10	operacja o2 (Ti).
WRITE < A, x >	operacja o3 (Ti).
READ < B, x >	operacja o4 (Ti).
x:=x+10	operacja o5 (Ti).
WRITE < B, x >	operacja o6 (Ti).

Transakcja Tj składa się z następujących operacji:

READ < A, y >	operacja o1 (Tj).
y:=y*2	operacja o2 (Tj).
WRITE < A, y >	operacja o3 (Tj).
READ < B, y >	operacja o4 (Tj).
y:=y*2	operacja o5 (Tj).
WRITE < B, y >	operacja o6 (Tj).

gdzie A=B=20

Na rysunku przedstawiono plan sekwencyjny P={Ti,Tj). Najpierw są wykonywane wszystkie operacje transakcji Ti, a potem wszystkie operacje transakcji Tj.



Rys. Plan sekwencyjny P={Ti,Tj).

Wartości początkowe A=B=20

Po wykonaniu transakcji Ti wartość A=30, wartość B=30

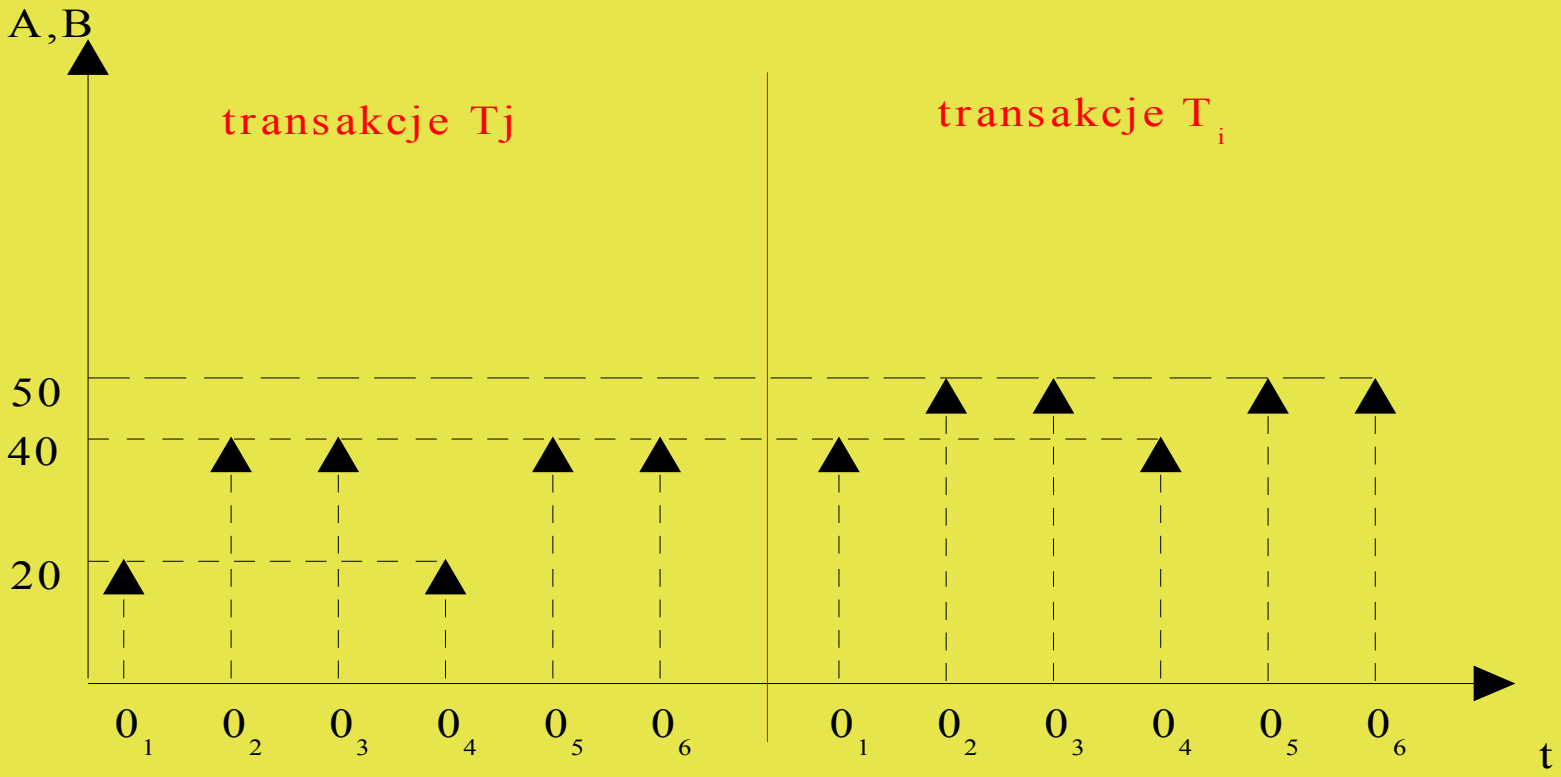
Po wykonaniu transakcji Tj wartość A=60, wartość B=60

Stan bazy po wykonaniu operacji jest spójny.

Ciąg operacji planu jest następujący:

P{Ti,Tj)=Ri(A);Wi(A);Ri(B);Wi(B);Rj(A);Wj(A); Rj(B);Wj(B);

Na rysunku 5 przedstawiono plan sekwencyjny P={Tj,Ti). Najpierw są wykonywane wszystkie operacje transakcji Tj, a potem wszystkie operacje transakcji Ti.



Rys. 5 Plan sekwencyjny P={Tj,Ti).

Wartości początkowe A=B=20.
Po wykonaniu transakcji Tj wartość A=40, wartość B=40
Po wykonaniu transakcji Ti wartość A=50, wartość B=50
Stan bazy po wykonaniu operacji jest spójny A=B

Ciąg operacji planu jest następujący:

P{Tj,Ti)=Rj(A);Wj(A);Rj(B);Wj(B);Ri(A);Wi(A); Ri(B);Wi(B);

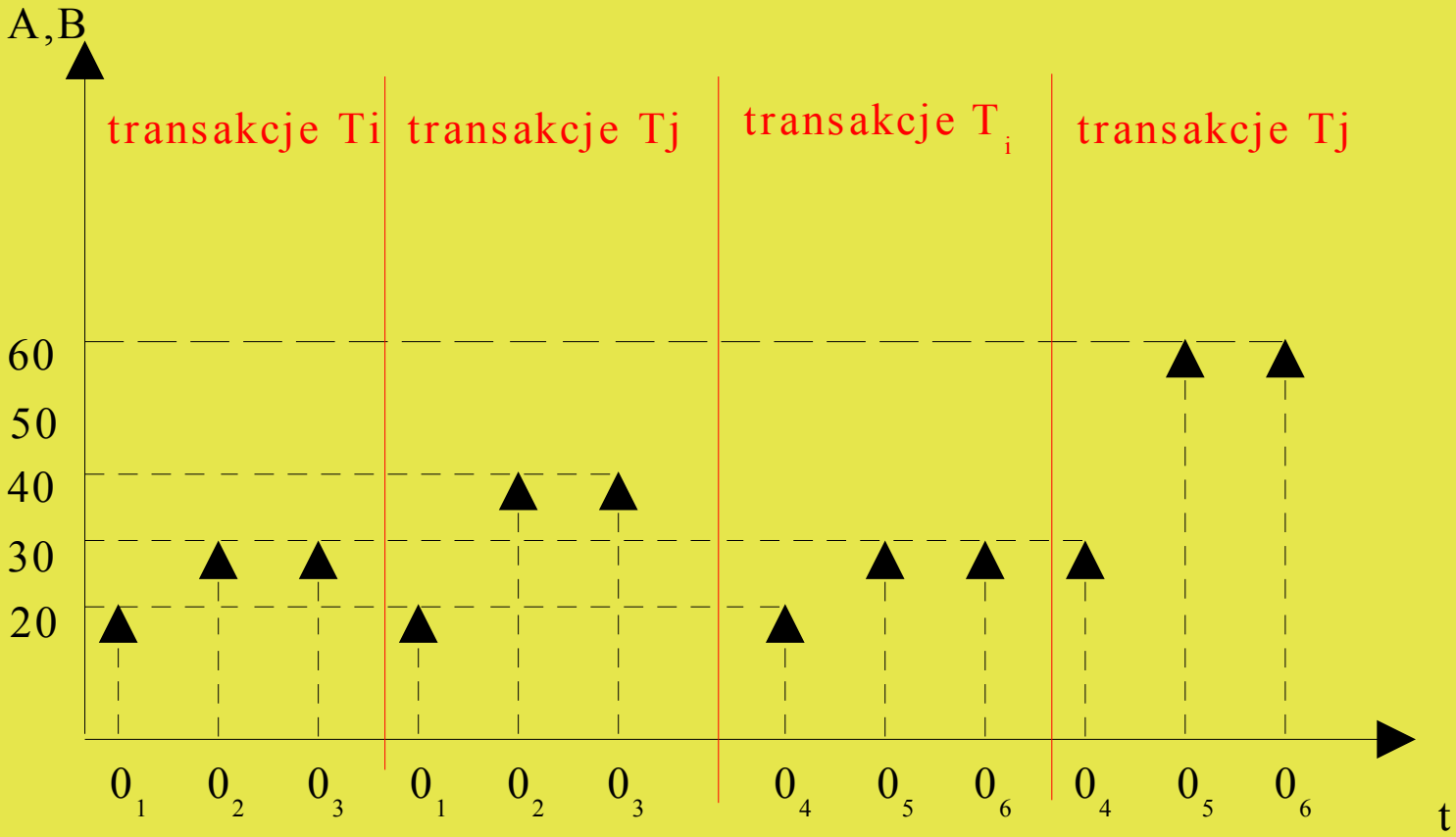
Końcowe wartości A i B w obu planach są różne. W wyniku wykonania planu P={Ti,Tj)najpierw wykonuje się transakcję Ti a A i B uzyskują wartość 60, natomiast gdy pierwsza jest transakcja Tj w planie P={Tj,Ti) to wartość A i B wynosi 50. Końcowa wartość nie jest istotna. Ważne jest, że przy planach sekwencyjnych kolejność wykonywania operacji zależy tylko od kolejności wykonywania całych transakcji, a więc spójność jest zachowana.

*** Transakcje nie są wykonywane w całości.**

Wprowadzimy pojęcie planu szeregowanego.

Plan szeregowany to taki plan, którego wpływ na bazy danych jest taki sam jak planu sekwencyjnego niezależnie od stanu początkowego bazy danych.

Korzystając z przykładu zawierającego transakcje Ti i Tj na rysunku przedstawiono plan szeregowany.


A,B
transakcje Ti
transakcje Tj
transakcje T i
transakcje Tj
60
50
40
30
20
0 1
0 2
0 3
0 1
0 2
0 3
0 4
0 5
0 6
0 4
0 5
0 6
t

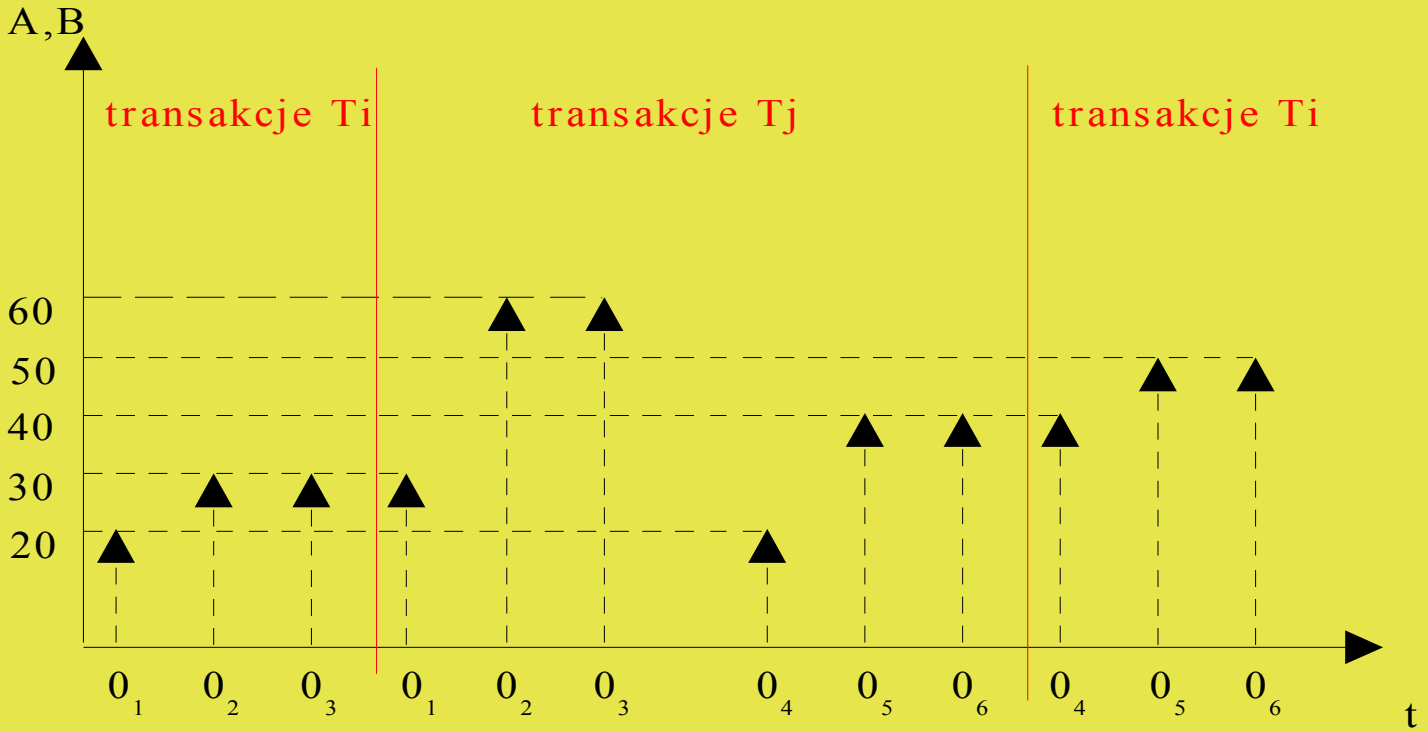
Psz={Ti,Tj).

Wynik przetwarzania tego planu jest taki sam jak planu P={Ti,Tj).

Ciąg operacji planu:

P{Ti,Tj)=Ri(A);Wi(A);Rj(A);Wj(A);Ri(B);Wi(B); Rj(B);Wj(B);

*** Transakcje nie są wykonywane w całości.**
Plan nie jest ani sekwencyjny, ani szeregowany.
Korzystając z przykładu zawierającego transakcje Ti i Tj na rysunku przedstawiono trzeci przypadek.



Rys. 7 Plan P={Ti,Tj).

Wartości początkowe A =B=20, po wykonaniu operacji A=60, B=50 czyli zaczynając od stanu spójnego przechodzimy w stan niespójny.

Jeśli jedna z operacji transakcji Ti działa jako pierwsza na A to również powinna działać jako pierwsza na B, Inaczej mamy do czynienia z pojawieniem się stanu niespójnego.

W naszym przypadku po wykonaniu obu transakcji A=2(A+10), a B=2B+10.

Ciąg operacji planu:

P=Ri(A);Wi(A);Rj(A);Wj(A);Rj(B);Wj(B); Ri(B);Wi(B);

Konflikty

Zasada poprawności bazy danych brzmi, **że transakcje wykonywane w izolacji przekształcają spójny stan bazy w inny spójny stan.**

Stanem spójnym nazywamy stan bazy danych, w którym są spełnione wszystkie deklarowane lub ustalone przez projektanta więzy.

Jeżeli nie zapewnimy izolacji transakcji mogą pojawić się następujące problemy:

* **brudny odczyt** ***(ang.dirty reads)*** występuje wtedy, gdy wiele transakcji próbuje uzyskać dostęp do tej samej tabeli w tym samym czasie. Może zdarzyć się, że jedna transakcja modyfikuje dane, inna czyta te dane przed ich potwierdzeniem, następnie pierwsza zostaje cofnięta i stan bazy powraca do stanu sprzed zmian. Druga transakcja może później próbować modyfikować tabelę w oparciu o dane uzyskane podczas odczytu, które jednak nie są już poprawne.

* **niepowtarzalny odczyt** ***(ang.nonrepeatable reads)*** występuje wtedy, gdy jedna transakcja czyta z tabeli, a potem inna transakcja modyfikuje dane w tabeli. Jeśli pierwsza transakcja próbuje potem jeszcze raz odczytać dane z tej tabeli, odczyt powoduje uzyskanie innych danych.

* **odczyt fantomów** ***(ang.phantom reads)*** występuje wtedy, gdy jedna transakcja odczytuje dane w oparciu o pewne warunki wyszukiwania, inna transakcja modyfikuje dane w tabeli, a następnie pierwsza transakcja odczytuje dane z tabeli ponownie, opierając się na tych samych warunkach wyszukiwania. W wyniku zmiany danych w tabeli, jako rezultat wyszukiwania otrzymuje zupełnie inne dane.

Aby zapobiec tego typu problemom, w standardzie SQL **wprowadzono cztery poziomy izolacji.**

Poziom **izolacji** to stopień w jakim dana transakcja może mieć dostęp do danych modyfikowanych przez inne transakcje przed ich zatwierdzeniem. Inaczej mówiąc jest to **wymuszanie szeregowalności transakcji.**

Wyróżniamy poziomy:

*** READ UNCOMMITED-** ***odczyt nie zatwierdzony***. Jest to najmniej restrykcyjny z poziomów izolacji. Dopuszcza czytanie przez transakcje danych jeszcze nie zatwierdzonych. Jest używany do transakcji zawierających ogólne informacje, takie jak dane statystyczne, które nie powinny być modyfikowane. Poziom ten dopuszcza: brudny i niepowtarzalny odczyt oraz odczyt fantomów.

*** READ COMMITED** ***odczyt zatwierdzony***. Poziom ten zabrania odczytu danych niezatwierdzonych, umożliwia jednak zapisywanie danych w transakcjach niezatwierdzonych. Dopuszcza występowanie niepowtarzalnych odczytów i odczytu fantomów, zabrania występowania brudnych odczytów.

*** REPEATABLE READ** ***odczyt powtarzalny.*** Poziom ten zabrania zapisywania w transakcjach niezatwierdzonych. Jeśli więc transakcja niezatwierdzona przeczytała daną, to dana ta może być tylko czytana przez inna transakcję. Jeśli transakcja niezatwierdzona zapisała jakąś daną, to nie można jej ani odczytać, ani zapisać dopóki ta transakcja nie zostanie zatwierdzona. Poziom ten dopuszcza występowanie odczytu fantomów, jednak blokuje występowania brudnych odczytów i niepowtarzalnych odczytów.

* **SERIALIZABLE** ***szeregowalność*** czyli jedyny bezpieczny poziom izolacji. Poziom ten nie dopuszcza występowania brudnych i niepowtarzalnych odczytów oraz odczytu fantomów.

Zapewnienie zachowania wysokiego poziomu poprawności danych przy jednoczesnym realizowaniu transakcji nazywamy szeregowalnością, a nawet szeregowalnością ze względu na konflikt.
Sytuacje konfliktowe mogą wystąpić wtedy, gdy dwie transakcje Ti i Tj gdzie $i \neq j$ są zainteresowane dostępem do tego samego obiektu lub kolejność występowania operacji w transakcji zostaje zmieniona.

Mamy dane dwie transakcje Ti i Tj gdzie i≠j.

Rozważymy następujące przypadki:

* Odczyt (Ri(X)) – Odczyt (Rj(Y)) dla X=Y i X≠ Y.

Para ta nie stanowi konfliktu ponieważ żadna z operacji odczytu Ri(X)) i (Rj(Y)) nie zmienia ani X ani Y. Każda z tych operacji może być wykonywana w dowolnej kolejności.

* Odczyt (Ri(X)) – Zapis (Wj(Y)) dla X≠ Y.

Para ta nie stanowi konfliktu ponieważ X nie jest zmieniane i transakcja Tj może zapisać wartość Y przed i po transakcji Ti .

* Zapis (Wi(X)) – Odczyt (Rj(Y)) dla X≠ Y.

Para ta nie stanowi konfliktu ponieważ Y nie jest zmieniane i transakcja Ti może zapisać wartość X przed i po transakcji Tj .

* Zapis (Wi(X)) – Zapis (Wj(Y)) dla X□Y. Para ta nie stanowi konfliktu

* Zapis (Wi(X)) – Zapis (Wj(Y)) dla X=Y.

Dwa zapisy tego samego elementu przez dwie różne transakcje Ti i Tj są konfliktem. Zmiana kolejności wykonywania operacji powoduje, że wartości wyliczane przez Ti i Tj mogą być różne.

* Odczyt (Ri(X)) – Zapis (Wi(Y))

Odczyt i zapis tej samej transakcji stanowi konflikt. Kolejność działań danej transakcji jest ustalona i nie może być zmieniona.

* Odczyt (Ri(X)) – Zapis (Wj(X)) i Zapis (Wi(X)) – Odczyt (Rj(X))

Odczyt i zapis oraz zapis i odczyt tego samego elementu bazy danych wykonywany przez różne transakcje stanowi konflikt. Zmiana kolejności operacji Ri(X) i Wj(X)) ma wpływ na wartość X.

Z przytoczonych przypadków wynika, że nie można zmieniać kolejności wykonywania operacji, jeśli dotyczą one tego samego elementu bazy danych lub co najmniej jedną czynnością jest operacja zapisu WRITE.

W przykładzie 7 pokazano przekształcenie planu szeregowanego na plan sekwencyjny z r bez konfliktów.

Przykład 7

Ri(A);Wi(A);Rj(A);Wj(A);Ri(B);Wi(B); Rj(B);Wj(B);	zapis-odczyt x≠y
Ri(A);Wi(A);Rj(A);Ri(B);Wj(A);Wi(B);Rj(B);Wj(B);	odczyt-odczyt
Ri(A);Wi(A); Ri(B); Rj(A);Wj(A);Wi(B);Rj(B);Wj(B);	zapis-zapis x≠y
Ri(A);Wi(A); Ri(B); Rj(A); Wi(B); Wj(A);Rj(B);Wj(B);	zapis-odczyt x≠y
Ri(A);Wi(A);Ri(B);Wi(B);Rj(A);Wj(A); Rj(B);Wj(B);	

Rozróżnia się trzy metody synchronizacji współbieżnego dostępu do danych:

blokady, optymistyczne sterowanie dostępem oraz znaczniki czasowe.

Blokady

Mechanizm blokad polega na tym, że każda transakcja przed uzyskaniem dostępu do danego obiektu bazy danych ***musi założyć odpowiednia blokadę na tym obiekcie.***

Zapewnia to użytkownikowi wyłączność dostępu do modyfikowanych przez siebie danych

Blokada uniemożliwia dostęp do obiektu innym transakcjom i jest usuwana w momencie zakończenia transakcji. ***Żadne dwie transakcje nie mogą zablokować tego samego obiektu, jeśli blokada nie została wcześniej zwolniona.***

Rozróżnia się: blokadę do odczytu STOPi(X) oraz blokadę do zapisu GOi (X).

Mechanizm blokad w pewnym stopniu wymusza szeregowalność transakcji.

Ogólnie mechanizm blokad jest bardzo złożony, gdyż blokady mogą być zakładane na pojedyncze krotki, wartości atrybutów, tabele i całe bazy danych.

Nazywa się to ziarnistością blokowania.

Wyróżnia się:

* **grube ziarna** to znaczy blokady zakłada się na całą bazę danych, poszczególne krotki. Efektem tego jest mały stopień współbieżności, a tym samym mała wydajność,

* **miałkie ziarna** gdzie blokady zakłada się na na przykład elementy krotki. Zapewnia to większą współbieżność, lepszą wydajność, ale wiąże się ze znacznym nakładem czasu na zakładania blokad oraz zapotrzebowaniem na dodatkową pamięć na blokady.

Ziarnistość blokowania odbywa się pod kontrolą tzw. protokołu blokowania zapobiegającego.

Im drobniejsza ziarnistość blokowania tym większy poziom współbieżności w dostępie do danych.

Często stosuje się tzw. **eskalacje blokad.**

Wraz ze wzrostem aktywności systemu serwery zaczynają blokować większe sekcje informacji, aby ograniczyć zużycie pamięci.

Eskalacja powoduje spadek współbieżności w dostępie do danych.

Plan z rysunku 4 można przedstawić następująco:

Pblok{Ti,Tj)=STOPi(A):Ri(A);Wi(A);GOi(A);STOPi(B);Ri(B);Wi(B);GO i(B);
STOPj(A);Rj(A);Wj(A); GOj(A);STOPj(B); Rj(B);Wj(B); GOj(B);

Plan z rysunku 5 można przedstawić następująco:

Pblok{Tj,Ti)=STOPj(A;Rj(A);Wj(A);GOj(A);STOPj(B);Rj(B);Wj(B);GOj(B);
STOPi(A);Ri(A);Wi(A); GOi(A);STOPi(B); Ri(B);Wj(B); GOi(B);

Plan z rysunku 6 można przedstawić następująco:

Pblok=STOPi(A);Ri(A);Wi(A);GOi(A);STOPj(A);Rj(A);Wj(A);GOj(A);
STOPi(B);Ri(B);Wi(A); GOi(B);STOPj(B); Rj(B);Wj(B); GOj(B);

Samo blokowanie nie zapewnia szeregowalności.

Znany jest algorytm blokowania dwufazowego 2PL, który wymaga w ramach transakcji najpierw założenia wszystkich blokad, a dopiero potem zwolnienie pierwszej blokady. Aby uniknąć niepotrzebnego blokowania, system zazwyczaj stosuje kilka trybów blokowania i różne zasady przyznawania blokad dla poszczególnych trybów.

Znane są blokowania aktualizujące, przyrostowe, elementów hierarchii ziarnistości i inne.

W trakcie dostępu współbieżnego istnieje, niebezpieczeństwo zakleszczenia, czyli wzajemnej blokady wykonywanych równocześnie transakcji.

Zakleszczenie to sytuacja, w której 2 lub więcej transakcje znajduje się w stanie oczekiwania na uwolnienie z blokady.

Przykład: Mamy dwie transakcje :

Ti: STOPi(A);Ri(A);A:=A+10;Wi(A);STOPi(B);
GOi(A);Ri(B);B:=B+10 Wi(B); GOi(B);

Tj: STOPj(B);Rj(B);B=B*2; Wj(B); STOPj(A);
GOj(B);Rj(A);A=A*2;Wj(A); GOj(A);

Jeżeli operacje w transakcjach będą się przeplatać następująco :

STOPi(A);Ri(A); STOPj(B);Rj(B); A:=A+10 ;B=B*2, Wi(A); Wj(B); STOPi(B) **zabronione**; STOPj(A) **zabronione**;

To żadna z transakcji nie może się wykonać i czas oczekiwania będzie nieograniczony.

Element B został wcześniej zablokowany przez transakcję Tj i nie została zwolniona blokada w związku z czym nie jest możliwe założenia blokady STOPi(B) na tym elemencie przez transakcję Ti. To samo dotyczy elementu A.

Rozwiązanie tego typu problemów to:

* spełnienie warunku, aby każda transakcja zakładała wszystkie potrzebne jej blokady na samym początku w postaci jednej akcji, lub nie zakładała wcale (znaczne ograniczenie współbieżności),

* wykrywanie wzajemnej blokady w chwili, gdy ona wystąpi i wycofanie działania jednej akcji (rola ofiary), co pozwala drugiej kontynuowanie działania,

* ustalenie limitu czasu oczekiwania na założenie blokady, a po jego upływie tego czasu automatyczne wycofanie oczekującej transakcji.

Standard SQL pozwala na kontrolowanie blokady ustawione przez transakcje. Odbywa się to z wykorzystaniem opisanych poprzednio poziomów izolacji w SQL.

I tak:

* **READ UNCOMMITED** - brak blokady dla operacji odczytu, dający możliwość równoczesnego korzystania z danych różnym transakcjom czyli dana transakcja będzie mogła odczytywać dane, które właśnie są modyfikowane przez inną transakcję, co może spowodować niespójność danych.

* **READ COMMITED** zwalnianie blokad zakładanych na rekordy natychmiast po ich odczycie,

* **REPEATABLE READ** zwolnienie blokad rekordów pobranych do odczytu, ale nigdy nie odczytanych. Umożliwia to równoczesne wprowadzanie danych realizowane przez inna transakcje.

* **SERIALIZABLE** utrzymanie blokad założonych przy odczycie do zakończenia transakcji. Niemożliwe jest wtedy dokonywanie jakichkolwiek zmian na blokowanych danych, łącznie z dodawaniem nowych rekordów.

W przypadku obiektowych baz danych należy wziąć pod uwagę, że klasa dziedziczy atrybuty i metody od swoich nadklas. W związku z tym, gdy jedna transakcja przetwarza instancje klasy, inna nie powinna modyfikować definicji tej klasy ani żadnej nadklasy tej klasy. Czyli dla zapytania dotyczącego danej klasy blokada jest zakładana nie tylko na danej klasie, ale dla wszystkich następników w hierarchii danej klasy.

Inne metody sterowania współbieżnym dostępem.

Oprócz blokad jednym ze sposobów sterowania współbieżnym dostępem jest **podejście optymistyczne.**

Zakłada się, że konflikty między transakcjami występują bardzo rzadko, dzięki czemu transakcje mają nieograniczony dostęp do danych.

Dopiero przed zatwierdzeniem transakcji sprawdza się czy nie wystąpił konflikt. Jeśli tak, to jedną transakcję anuluje się i wykonuje od początku.

Istnieje też metoda znaczników czasowych (ang. timestamps).

Szeregowalność transakcji uzyskuje się przez nadanie każdej transakcji unikalnych znaczników, które ustawiają je w pewnym ciągu czasowym.

Metodę te stosuje się przede wszystkim w systemach rozproszonych gdzie blokowanie nie zdaje egzaminu.

Znaczniki czasowe przypisuje się w kolejności rosnącej w momencie, gdy zaczyna się transakcja. Istnieją różne sposoby określania znaczników. Można korzystać z zegara systemowego, co nie zawsze jest bezpieczne lub z licznika, który zostaje zwiększony o jeden, w chwili rozpoczecia transakcji.

Kolejnym typem optymistycznego sterowania współbieżnością jest walidacja.

W procesie walidacji utrzymuje się zapis czynności aktywnych transakcji, a nie czas zapisu i odczytu wszystkich elementów bazy danych. Zanim zostaje zapisana wartość elementu bazy danych zbiory elementów przeczytanych przez transakcje i tych, które mają być przez nią zapisane porównuje się ze zbiorami zapisów z innych transakcji aktywnych. Jeśli występują jakieś nieprawidłowości transakcja zostaje cofnięta.

Wszystkie metody szeregowalności transakcji czyli: blokowanie, znaczniki czasowe i walidacja ma swoje **zalety i wady**.

Dotyczy to przede wszystkim **przestrzeni pamięci** potrzebnej dla tych operacji oraz możliwości zakończenia transakcji bez opóźnień.

W przypadku blokad potrzebna przestrzeń pamięci jest proporcjonalna do liczby blokowanych elementów bazy danych.

W pozostałych dwóch przypadkach pamięć jest potrzebna do zapamiętania czasów zapisu i odczytu związanych z każdym elementem bazy danych, czy zapisu aktywnych transakcji, bez względu na to, czy dostęp do nich następuje w danej chwili.

Znaczniki czasowe i walidacja wymagaja więcej przestrzeni pamięci ponieważ muszą utrzymywać dane dotyczące transakcji zakończonych, które nie są potrzebne przy blokowaniu.

Wydajność tych trzech metod zależy również od tego, czy transakcja wymaga dostępu, który jest potrzebny innej transakcji współbieżnie. Gdy ta współbieżność jest mała znaczniki czasu i walidacja nie powodują wielu cofnięć i mogą być znaczne efektywniejsze od blokad.

Gdy wymagane jest wiele cofnięć, to wygodniej stosowac blokadę.

Zarządzanie transakcjami w systemach rozproszonych baz danych.

W rozproszonych bazach danych dane są ***podzielone poziomo*** czyli krotki tej samej relacji znajdują się w różnych miejscach ***lub pionowo*** wówczas na schemacie relacji wykonane są operacje projekcji, dzięki czemu uzyskamy szereg schematów niższego stopnia.

Można też powielić dane, czyli replikować co zapewnia identyczne kopie relacji w różnych miejscach. Dzięki temu uzyskuje się wysoką niezawodność systemów rozproszonych.

Ze względu na replikację danych w kilku węzłach problem zarządzania transakcjami i ochrona integralności danych jest znacznie trudniejsza.

Należy zadbać o dodatkowe zachowanie zgodności danych w całym rozproszonym systemie.

Transakcja rozproszona składa się z wielu składowych działających w różnych miejscach.

Obejmuje wiele lokalnych baz danych. W związku z tym może wystąpić jej zatwierdzenie tylko w przypadku pomyślnego zakończenia wszystkich transakcji lokalnych będących całością transakcji rozproszonej.

Realizacja przetwarzania rozproszonych transakcji wymaga implementacji **protokołów zarządzających kontrolą dostępu i niezawodnością transakcji rozproszonego systemu.**

Typowy algorytm zarządzania kontrolą dostępu **to dwufazowy protokół** zatwierdzania 2PC (ang. two-phase commit).

W pierwszej fazie lokalne bazy informują wyróżniony węzeł zwany koordynatorem transakcji o tym czy ich lokalna część transakcji może zostać zatwierdzona, czy musi być wycofana.

W drugiej, jeżeli wszystkie lokalne bazy danych biorące udział w transakcji mogą ją zatwierdzić, wówczas koordynator nakazuje zatwierdzenie (notuje to w swoim dzienniku transakcji) i w każdej bazie lokalnej zatwierdzone są transakcje. W przeciwnym wypadku, jeśli choć jedna z lokalnych transakcji zostaje wycofana, należy wycofać wszystkie.

Istnieje również **protokół 3PC**. Uwzględnia on sytuacje, gdy koordynator ulegnie awarii, umożliwia wybranie nowego koordynatora.
Na ogół komercyjne systemy zarządzania bazą danych nie realizują tego protokołu.

Replikacja danych zwiększa poziom niezawodności jednak zawsze istnieje ryzyko, że w systemie istnieje kopia danych, która nie mogła być uaktualniona ze względu na jej chwilowe uszkodzenie lub niedostępność spowodowaną uszkodzeniem łącza komunikacyjnego. **Utrzymywanie zgodności replik wymaga implementacji protokołów kontroli replik.**

Najprostszym protokołem kontroli replik jest protokoł **ROWA (ang. read one write all).** Zakłada on, że jedna operacja logicznego odczytania zreplikowanych danych jest zrealizowana przez pojedynczą operację fizycznego odczytania z dowolnej repliki zaś operacja logicznego zapisu wymaga wykonania operacji fizycznego zapisu na wszystkich kopiach.

Zarządzanie transakcjami w systemIE oracle.

System Oracle w przeciwieństwie do MYSQL **inicjalizuje automatycznie transakcje.** Transakcja rozpoczyna w chwili, gdy nawiązane zostanie połączenie z bazą danych. Transakcja kończy się w momencie przerwania połączenie z bazą danych lub gdy wystąpi polecenie zatwierdzenia transakcji- COMMIT lub wycofania transakcji- ROLLBACK.

Oracle pozwala również dzięki komendzie SAVEPOINT umieszczonej wewnątrz bloku transakcji na częściowe cofnięcie transakcji zamiast całej. Oczywiście występuje to przy połączeniu SAVEPOINT z ROLLBACK.

Aby zachować integralność danych w bazie w ORACLE istnieje mechanizm nazwany **logicznym znacznikiem czasowym SCN (ang.System Change Number)** służącym do śledzenia kolejności w jakiej zachodzą transakcje w bazie. Informacje te są przydatne, kiedy zachodzi potrzeba, poprawnego i zgodnego z kolejnością wystąpień odtworzenia transakcji po awarii.

Bardzo wygodną strukturą są tak zwane segmenty powrotu (ang. Rollback Segments), które składują informację umożliwiające odtworzenie bazy danych do stanu sprzed wystąpienia transakcji.

Segmenty różnią się od logów tym, że nie zapamiętują zmian spowodowanych przebiegiem transakcji, tylko dają możliwość cofania transakcji i współbieżnego dostępu.

Segmenty udostępniają spójny obraz bazy danych w określonym momencie w przeszłości. Segmenty (w Oracle wersja 9) pozwalają również na cofnięcie się do momentu w przeszłości przed zapytanie, które spowodowało utratę spójności bazy danych. Zapewnia to mechanizm zapytań wstecz (ang. Flashback Query). Mechanizm ten został mocno rozbudowany w Oracle 10.

Oracle wykorzystuje takie poziomy izolacji : READ COMMITED i SeriaLIZable. Trzeci dostępny to READ ONLY, który jednocześnie zabrania operacji zapisu i udostępnia dokładny obraz danych w momencie zapytania.

Do rozwiązania współbieżnego dostępu do danych w bazie ORACLE służy system MVRC (ang. Multiversion Read Consistency). System ten gwarantuje spójny obraz danych. MVRC ignoruje zmiany wprowadzane przez niezatwierdzone w chwili zapytania transakcje.

Oracle jest **jedną z najszybszych baz danych**. Na szybkość jej działania wpływa wiele czynników.
Przede wszystkim na wydajność bazy ma wpływ system zarządzania transakcjami umożliwiający bezkolizyjny dostęp współużytkowników w tym samym czasie. Uzyskano to dzięki: zmniejszeniu do niezbędnego minimum ilości operacji we/wy oraz braku blokad na operacje odczytu.

Podsumowanie

Integralność polega na zapewnieniu, że baza danych pozostaje dokładnym odbiciem reprezentowanej rzeczywistości.

Wyróżniamy schemat bazy danych i stan.

Schemat jest stały, niezmienny. Zmiana schematu to nowa baza danych.

Stan bazy jest określony w danym momencie czasowym. Pod wpływem czynników zewnętrznych i wewnętrznych takich jak na przykład transakcje, baza danych przechodzi przez ciąg zmian stanów. W zbiorze możliwych stanów tylko niektóre z nich są poprawne Stany bazy danych w których są spełnione wszystkie deklarowane lub zamierzone więzy są nazwane stanami spójnymi. Istotne jest, aby transakcje przekształcała spójny stan bazy danych w inny również spójny.

Transakcje działając w izolacji zapewniają spójność bazy. Dąży się do tego, aby spójność była utrzymana również w trakcie współbieżnego dostępu.

Uzyskuje się to poprzez opracowanie planów sekwencyjnych, szeregowanych i szeregowanych ze względu na konflikt.

W celu zachowania spójności bazy danych stosuje się blokady i inne formy zabezpieczania danych przy współbieżnym dostępie.