Integralność bazy danych

Integralność (ang. *integrity*) łączy w sobie formalną poprawność bazy danych i procesów przetwarzania, poprawność fizycznej organizacji danych, zgodność ze schematem bazy danych, zgodność z ograniczeniami integralności oraz z regułami dostępu. Tak rozumiana integralność nie oznacza bynajmniej, że baza danych prawidłowo odwzorowuje sytuację i procesy opisanego przez nią świata zewnętrznego. Właściwszym terminem jest tu spójność (ang. *consistency*), a w szczególności:

- spójność fizyczna: operacje bazodanowe kończą się sukcesem
- spójność logiczna: baza danych jest spójna fizycznie, a jej zawartość odpowiada schematowi bazy danych i dodatkowym ograniczeniom.

Szeroko pojęta integralność danych obejmuje: logiczną spójność (spójność wewnętrzna) i zgodność ze stanem świata rzeczywistego opisywanego przez dane (spójność zewnętrzna). Integralność można rozpatrywać na wielu poziomach:

- Semantycznej zgodności danych przechowywanych w bazie z rzeczywistością, najczęściej w postaci ograniczeń. Ograniczenia te nazywamy więzami integralności (integrity constraints),
- Prawidłowego współużywania danych nie naruszających więzów integralności,
- Utrzymania integralności danych po zaistniałej awarii sprzętu lub oprogramowania,

Więzy integralności są warunkami, które powinny być spełnione przez określony podzbiór danych z bazy. Spełnienie tych warunków świadczy, że baza danych jest w stanie spójnym. Istnieją dwa sposoby sprawdzania reguł integralnościowych:

- deklaratywne,
- proceduralne.

Więzy integralności (większość z nich definiuje się w instrukcjach CREATE oraz ALTER TABLE języka SQL):

- Integralność encji (*entity integrity*): odnosi się do pojedynczej tabeli, w której powinien istnieć klucz pierwotny (PRIMARY KEY (PracownikID)). Jeżeli danej kolumnie nałożyliśmy warunek PRIMARY KEY, DBMS automatycznie nałoży jej warunki NOT NULL i UNIQUE.
- Integralność krotki: zakłada się, że każda krotka opisuje jeden obiekt świata rzeczywistego a wartość krotki powinna odpowiadać elementowi świata rzeczywistego. Na wartości przyjmowane przez krotki można nałożyć niezależne więzy, które muszą być spełnione przez wszystkie krotki niezależnie. Więzy te to:
    - zawężenie dziedziny atrybutu poprzez podanie przedziału wartości, listy możliwych wartości (np. płeć VARCHAR(1) NOT NULL CHECK (płeć IN ('M','F')))
    - podanie zależności pomiędzy wartościami różnych atrybutów w krotce,
    - podanie formatu wartości (imię VARCHAR(20))
    - zadeklarowanie konieczności występowania jakiejś wartości (NOT NULL)
    - zdefiniowanie niepowtarzalnych wartości atrybutu (UNIQUE)
- Więzy wewnętrzne relacji: sprawdzane są wartości występujących w krotkach w ramach tej samej relacji,

- Więzy zbioru krotek: sprawdzane są wartości atrybutów w różnych relacjach,

Integralność odwołań (*referential integrity*) dotyczy modelowania powiązań pomiędzy obiektami w rzeczywistości. W bazach relacyjnych realizowane jest to z wykorzystaniem klucza pierwotnego i odpowiadającego mu wartością klucza obcego. Klucz obcy - to jedna lub więcej kolumn tabeli odwołujących się do kolumny lub kolumn klucza pierwotnego (głównego) w innej tabeli. Tworząc klucz obcy, definiujemy związek między tabelą klucza pierwotnego i tabelą klucza obcego. Związek taki powstaje podczas połączenia kolumn tych samych typów danych z każdej tabeli. Łączenie tabel przy pomocy łączenie odpowiednich kolumn chroni dane z tabeli klucza obcego przez „osieroceniem", jakie mogłoby nastąpić w wyniku usunięcia odpowiadających im danych z tabeli klucza pierwotnego. Definiowanie kluczy obcych jest po prostu sposobem łączenia danych przechowywanych w różnych tabelach bazy danych.
W relacyjnych bazach danych integralność odwołań dotyczy sytuacji, kiedy tablica A zawiera klucz obcy (foreign key) będący równocześnie kluczem pierwotnym tablicy B. Warunek integralności odwołań ustala, że dla każdego wiersza tablicy A musi istnieć taki wiersz w tablicy B, że wartości kluczy obcego i pierwotnego są jednakowe. Np. dla każdej wartości kolumny „ISBN"(klucz obcy) w tablicy „Copies_of_the_book" musi istnieć taka sama wartość w kolumnie„ISBN"(klucz pierwotny) tablicy „Books".
Deklarować klucz obcy można tak:

```
CREATE TABLE  "Copies of the book" (
ISBN              varchar(12)              not null,
Num_inv           varchar(10)              not null,
Num_document      varchar(10),
Is_or_not         binary(1),
Capture_date      date,
Return_date       date,
PRIMARY KEY (ISBN, Num_inv)
);
ALTER TABLE "Copies of the book"
ADD FOREIGN KEY "FK_COPIES O_INCLUDES_BOOKS" (ISBN)
REFERENCES Books (ISBN)
ON UPDATE RESTRICT
ON DELETE RESTRICT;
```

Więzy integralności można podzielić ze względu na moment ich sprawdzania na więzy natychmiastowe lub odroczone.

Po utworzeniu tabeli za pomocą CREATE TABLE klucz obcy deklaruje się za pomocą klauzuli ALTER TABLE ADD FOREIGN KEY "FK_COPIES O_INCLUDES_BOOKS" (ISBN). Klucz ten jest połączony z kluczem pierwotnym tablicy „Books" za pomocą klauzuli REFERENCES Books (ISBN). Predykat ON UPDATE RESTRICT anuluje uaktualnienia wartości (ISBN) w kolumnie macierzystej, jeżeli na nią odwołają się rekordy tablic potomnych. Predykat ON DELETE RESTRICT anuluje skasowanie wartości (ISBN) w kolumnie macierzystej, jeżeli na nią odwołają się rekordy tablic potomnych.
Użycie: ON DELETE CASCADE, ON UPDATE CASCADE, spowodowałoby natomiast skasowani wszystkich połączone w tablicy rekordy (pierwszy przypadek) lub uaktualnione wszystkich połączonych w tablicach rekordów (drugi przypadek).

```
CREATE TABLE CLIENT (
ClientName CHARACTER (30) PRIMARY KEY,
Address1 CHARACTER (30),
Address2 CHARACTER (30),
City CHARACTER (25) NOT NULL,
State CHARACTER (2),
PostalCode CHARACTER (10),
Phone CHARACTER (13),
Fax CHARACTER (13),
ContactPerson CHARACTER (30)
) ;
CREATE TABLE TESTS (
TestName CHARACTER (30) PRIMARY KEY,
StandardCharge CHARACTER (30)
) ;
CREATE TABLE EMPLOYEE (
EmployeeName CHAR (30) PRIMARY KEY,
ADDRESS1 CHAR (30),
Address2 CHAR (30),
City CHAR (25),
State CHAR (2),
PostalCode CHAR (10),
HomePhone CHAR (13),
OfficeExtension CHAR (4),
HireDate DATE,
JobClassification CHAR (10),
HourSalComm CHAR (1)
) ;
CREATE TABLE ORDERS (
OrderNumber INTEGER PRIMARY KEY,
ClientName CHAR (30),
TestOrdered CHAR (30),
Salesperson CHAR (30),
OrderDate DATE,
CONSTRAINT NameFK FOREIGN KEY (ClientName)
REFERENCES CLIENT (ClientName)
ON DELETE CASCADE,
CONSTRAINT TestFK FOREIGN KEY (TestOrdered)
REFERENCES TESTS (TestName)
ON DELETE CASCADE,
CONSTRAINT SalesFK FOREIGN KEY (Salesperson)
REFERENCES EMPLOYEE (EmployeeName)
ON DELETE CASCADE
) ;

**********
```

```
CREATE TABLE ORDERS (
OrderNumber INTEGER PRIMARY KEY,
ClientName CHAR (30),
TestOrdered CHAR (30),
SalesPerson CHAR (30),
OrderDate DATE,
CONSTRAINT NameFK FOREIGN KEY (ClientName)
REFERENCES CLIENT (ClientName),
CONSTRAINT TestFK FOREIGN KEY (TestOrdered)
REFERENCES TESTS (TestName),
CONSTRAINT SalesFK FOREIGN KEY (Salesperson)
REFERENCES EMPLOYEE (EmployeeName)
ON DELETE SET NULL
) ;

**********
Klucze złożone
CREATE TABLE CLIENT (
ClientName CHAR (30) NOT NULL,
Address1 CHAR (30),
Address2 CHAR (30),
City CHAR (25) NOT NULL,
State CHAR (2),
PostalCode CHAR (10),
Phone CHAR (13),
Fax CHAR (13),
ContactPerson CHAR (30),
CONSTRAINT BranchPK PRIMARY KEY
(ClientName, City)
) ;

**********
klucze obce
CREATE TABLE ORDERS (
OrderNumber INTEGER PRIMARY KEY,
ClientName CHAR (30),
TestOrdered CHAR (30),
Salesperson CHAR (30),
OrderDate DATE,
CONSTRAINT BRANCHFK FOREIGN KEY (ClientName)
REFERENCES CLIENT (ClientName),
CONSTRAINT TestFK FOREIGN KEY (TestOrdered)
REFERENCES TESTS (TestName),
CONSTRAINT SalesFK FOREIGN KEY (Salesperson)
REFERENCES EMPLOYEE (EmployeeName)
) ;
```

Asercja

Asercja (*assertions*) to ograniczenia występujące w schemacie jako niezależne od tabel obiekty. Asercje służą do kontroli wartości wprowadzanych do tabel.

```
CREATE ASSERTION nazwa_ograniczenia
CHECK (predykat)
[atrybuty ograniczenia];
```

Przykład:

```
CREATE ASSERTION sprawdzenie_ceny
CHECK (towar.cena_jedn IS NOT NULL OR towar.cena_jedn >= 0);
```

```
CREATE TABLE ORDERS (
OrderNumber INTEGER NOT NULL,
ClientName CHAR (30),
TestOrdered CHAR (30),
Salesperson CHAR (30),
OrderDate DATE
) ;
CREATE TABLE RESULTS (
ResultNumber INTEGER NOT NULL,
OrderNumber INTEGER,
Result CHAR(50),
DateOrdered DATE,
PrelimFinal CHAR (1)
) ;
CREATE ASSERTION sprawdzanie_daty
CHECK (NOT EXISTS (SELECT * FROM ORDERS, RESULTS
WHERE ORDERS.OrderNumber = RESULTS.OrderNumber
AND ORDERS.OrderDate > RESULTS.DateReported)) ;
```

Domena

Domena (domain) tworzy obiekt w schemacie służący do definiowania kolumn jak alternatywa do typów danych. Domena określa typ danych, wartość domyślną, ograniczenia wartości i uporządkowanie.

```
CREATE DOMAIN nazwa_domeny [ AS ] typ_danych
[ DEAULT wartość_domyślna ]
[ definicja_ograniczenia ... ]
[ COLLATE nazwa_uporządkowania ];

gdzie
definicja_ograniczenia ::=
[ nazwa_ograniczenia ]
```

```
ograniczenie_typu_check

[ [NOT] DEFERRABLE ]
[ { INITIALLY IMMEDIATE } | { INITIALLY DEFERRED } ]
```

Przykład:

```
CREATE DOMAIN LeagueDom CHAR (8)
CHECK (LEAGUE IN ('American', 'National'));
CREATE TABLE TEAM (
TeamName CHAR (20) NOT NULL,
League LeagueDom NOT NULL
) ;
```

Transakcje i blokady

Transakcja to grupa rozkazów, która jest traktowana jako pojedyncza jednostka. Albo zostaną wykonane wszystkie rozkazy w transakcji albo żaden (zmiany wprowadzane przez nie do bazy danych są trwale zapisywane tylko wtedy, gdy wykonane zostaną wszystkie wchodzące w skład transakcji instrukcje). Transakcja służy do wykonania zmiany stanu bazy danych ze stanu spójnego w inny stan spójny. Celem systemu zarządzania transakcjami jest takie sterowanie operacjami w bazie danych, aby były one wykonane z możliwie wysokim współczynnikiem współbieżności i aby przeciwdziałać naruszeniu spójności bazy danych. Mówiąc o **współbieżnym** wykonywaniu operacji mamy na myśli wykonywanie operacji pochodzących z różnych transakcji i to w czasie, gdy transakcje te są aktywne. Transakcje, których operacje wykonywane są współbieżnie, nazywamy **transakcjami współbieżnymi**.

Przykład:
Rozważmy bazę danych zawierającą KONTO1 i KONTO2 - dane wskazujące stan dwóch różnych kont w banku. Niech będą dane dwa programy operujące na bazie danych:

Program zwracający stan konta:	Program dokonujący przelewu z konta na konto:

Info(konto) {
X:= read(konto);
Return(x);
}
	Przelew(konto_z, konto_na, kwota) {
	X:= read(konto_z);
	X:= x-kwota;
	Write(konto_z,x);
	X:= read(konto_na);
	X:= x+kwota;
	Write(konto_na,x);
	}

Wykonanie tych programów jako transakcji współbieżnych może być przyczyną zajścia przeplotów między ich operacjami. Abstrahując od konkretnych wartości, jak i od operacji wykonywanych poza bazą danych, transakcje powstałe w wyniku uruchomień programu info(KONTO1) mogą mieć postać:

Ti =(ri[KONTO1], ci), i=1,2,... - jeśli czytanie zakończyło się pomyślnie lub
Ti =(ri[KONTO1], ai), i=1,2,… - jeśli czytanie nie powiodło się (na przykład podmiot wydający to polecenie nie miał wystarczających uprawnień).

Podobnie uruchomienie programu przelew() z konkretnymi parametrami KONTO1 i KONTO2 może zaowocować powstaniem transakcji:

Ti =(ri[konto1],wi [konto1], ri [konto2], wi[konto2], ci) ,

z operacjami:

ri [KONTO1] – odczytuje stan konta KONTO1,
wi [KONTO1] – zapisuje nowy stan konta KONTO1,
ri [KONTO2] – odczytuje stan konta KONTO2,
wi [KONTO2] – zapisuje nowy stan konta KONTO2,
ci – oznacza pomyślne zakończenie transakcji(jej zatwierdzenie).
ai – oznacza odrzucenie transakcji

Wypisane schematy nie są jedynymi postaciami, jaką transakcja Ti może przyjąć. Może się zdarzyć, że dla którejś operacji system spowoduje jej odrzucenia z jakiegoś powodu (na przykład przez przerwania łączności z bazą danych, rozwiązywania problemu zakleszczenia, naruszenia warunku spójności (mówiącego na przykład, że stan konta nie może być niższy niż określona kwota), braku wystarczających uprawnień, itp.).
Wówczas transakcja może wyglądać następująco:

Ti =(ri [konto1],wi [konto1], ri [konto2] wi [konto2], ai),
Ti =(ri [konto1],wi [konto1], ri [konto2], ai),
Ti =(ri [konto1], wi [konto1], ai),
Ti =(ri [konto1], ai).

Z punktu widzenia stosowanych protokołów (algorytmów) zarządzania transakcjami istotnym jest przyjęcie pojęcia konfliktowości operacji, tzn. zdefiniowanie, jakie operacje są konfliktowe, a jakie nie. Z góry można określić, jakie operacje nigdy nie będą konfliktowe, a mianowicie operacje oi[x] oraz pj[y] nie są konfliktowe, jeśli:

a) i=j, (operacje z tej samej transakcji),

b) $x \neq y$, (operacje dotyczą rozłącznych danych)
c) żadna z nich nie jest operacją zapisu,
d) co najmniej jedna z nich pochodzi od transakcji, która w chwili wydania drugiej została już zakończona (zatwierdzona lub odrzucona).

W pozostałych przypadkach możemy dojść do konfliktu. Warunkami koniecznymi do konfliktu operacji oi[x] i pj[y], ale nie wystarczającymi, są:

a) $i \neq j$ (operacje pochodzą z dwóch różnych transakcji),
b) co najmniej jedna z tych operacji jest operacją zapisu,
c) x=y (operacje dotyczą tej samej danej lub przecinających się zbiorów danych),
d) obydwie transakcje, z których pochodzą rozważane operacje, są aktywne,
e) druga z operacji (pj[y]) powoduje zmianę zbioru danych x (wyznaczonego przez pewną formułę φ), na których działa pierwsza operacja (oi[x]).

Najprostszym sposobem zapobieżenia konfliktom w transakcjach jest szeregowa ich realizacja. Niestety, uniknięcie konfliktów za pomocą szeregowania transakcji okupione jest wtedy znacznym zmniejszeniem wydajność bazy danych (zabroniony jest wtedy wielodostęp do bazy danych). Dlatego częściej stosowanym rozwiązaniem jest określenie poziomu izolacji transakcji zgodnym z intencjami użytkowników bazy danych.

Standardem ISO wyróżniają się cztery poziomy izolacji. Im wyższy poziom izolacji transakcji (konfliktowości), tym niższa współbieżność (dłuższy czas wykonywania transakcji), ale jednocześnie większa niezawodność przetwarzania i jego bezpieczeństwo z punktu widzenia zachowania spójności bazy danych.

Transakcje i protokoły zarządzania transakcjami muszą spełniać postulat ASOT (atomowość, spójność, odizolowanie, trwałość), co z angielskiego brzmi ACID:

Atomicity (niepodzielność) – każda transakcja stanowi pojedynczą i niepodzielną jednostkę przetwarzania (a także odtwarzania), tj. w transakcji nie ma więc podtransakcji. Każda transakcja jest bądź wykonana w całości, bądź też żaden jej efekt nie jest widoczny w bazie danych.

Consistency (spójność) – transakcja rozpoczynając się w spójnym stanie bazy danych pozostawia bazę danych w stanie spójnym (tym samym lub innym). Jeśli transakcja narusza warunki spójności bazy danych, to SZBD powoduje jej odrzucenie.

Isolation (izolacja) – zmiany wykonywane przez transakcję jeśli nie są zatwierdzone, to nie są widziane przez inne transakcje (chyba, że przyjęty poziom izolacji na to zezwala).

Durability (trwałość) – zmiany dokonane przez transakcję zatwierdzone są trwale w bazie danych, tzn. nawet w przypadku awarii systemu musi istnieć możliwość ich odtworzenia.

Każda transakcja w momencie inicjowania otrzymuje jednoznaczny identyfikator. Identyfikator ten jest następne związany z każdą operacją składającą się na transakcję. Do operacji tych należą:

- ri[x] – czytanie danej x przez transakcję Ti
- wi[x] – zapisanie danej x przez transakcję Ti
- ai – odrzucenie (wycofanie) transakcji Ti (operacja ABORT)
- ci – zatwierdzenie transakcji Ti (operacja COMMIT)

przy czym x może nieść od wartości elementarnych danych aż po całe tabelę bazy danych.

Transakcje kończone są w następujących przypadkach:

- gdy zatwierdzone są instrukcją COMMIT (powoduje to trwałe zapisanie zmian)
- gdy wywołane zostanie ROLLBACK (co wycofuje wszystkie dokonane przez transakcję zmiany)
- gdy wykonana zostanie instrukcja DDL (efektem ubocznym wykonania ALTER, CREATE, COMMENT i DROP jest automatycznego zapamiętania zmian oraz definicji bazy)
- gdy sesja z rozpoczęta transakcją zostanie skutecznie zakończona (zmiany zostają wprowadzane jak po wywołaniu COMMIT)
- gdy nastapi nieplanowane odłączenie od bazy danych (zmiany zostają wycofane jak po instrukcji ROLLBACK)

Interactive SQL dostarcza dwóch opcji umożliwiających sterowanie sposobem zakończenia transakcji:

- ustawienie opcji AUTO_COMMIT na ON, wtedy ISQL automatycznie zatwierdza rezultaty transakcji kończących się sukcesem lub automatycznie wykonuje ROLLBACK po każdej transakcji kończącej się błędem
- ustawienie opcji COMMIT_ON_EXIT steruje zachowaniem nie zatwierdzonych transakcji kiedy zamykany jest ISQL. Kiedy opcja jest ustawiona na ON (ustawienie domyślne) ISQL wykonuje COMMIT w przeciwnym wypadku nie zatwierdzone zmiany są wycofywane (ROLLBACK)

Współbieżność

Współbieżność oznacza wykonywanie więcej niż jednej transakcji w tym samym czasie. Jeżeli nie istnieje specjalny mechanizm w serwerze bazy danych, współbieżne transakcje mogą wzajemnie na siebie wpływać powodując niespójność i błędność informacji. Mogą zaistnieć wtedy cztery przypadki:

Czytanie danych z transakcji nie zatwierdzonych

Czytanie danych z transakcji nie zatwierdzonych możliwe jest przy przyjęciu poziomu izolacji 0, tzn., gdy za konfliktowe uważa się tylko parę operacji zapisu, a dwie operacji, z których jedna jest operacja odczytu, nie są operacjami konfliktowymi. W standardzie SQL ten poziom izolacji nazywany jest także READ UNCOMMITED (popularne określany jako „brudne czytanie”). Można, więc czytać dane zmieniane przez transakcję, która nie została zatwierdzona. Reguły współbieżności dla tego poziomu izolacji przedstawiono w tabeli poniżej:

	Read	Write
Read	T	T
Write	T	N

gdzie operacje w lewej kolumnie traktowane są jako wcześniejsze od operacji w górnym wierszu, T oznacza, że operacje mogą być wykonywane współbieżnie, czyli nie są konfliktowe, N oznacza brak współbieżności, a więc konfliktowość.

wady	zalety

możliwość braku odtwarzalności, kaskady odrzuceń, anomalii powtórnego czytania oraz do pojawiania fantomów	wysoki współczynnik współbieżności transakcji.

Przykłady anomalii

Brudne czytanie. Transakcja A zmienia wartość wierszy, ale ich nie zatwierdza lub wycofuje zmiany. Transakcja B czyta zmodyfikowane wiersze a transakcja A dalej modyfikuje wiersze bez zatwierdzania lub wycofuje operacje, B czyta dane, których wartość nigdy nie zastanie zatwierdzona

Niepowtarzalne czytanie. Transakcja A czyta wiersz, wtedy transakcja B go modyfikuje lub usuwa i zatwierdza zmiany. Transakcja A ponownie nie odczyta wiersza danych lub odczyta inną wartość

Wiersze widma. Transakcja A czyta zbiór wierszy spełniających podany warunek, wtedy transakcja B wykonuje rozkaz INSERT lub UPDATE na wierszach, które nie spełniały warunku transakcji A. Transakcja B zatwierdza zmiany, które spowodują spełnienie warunków transakcji A. Transakcja A powtarza czytanie i uzyskuje inny zestaw wierszy

Przykład

Sprzedaż:
```
SELECT id, name, unit_price
FROM product;
UPDATE PRODUCT
SET unit_price = unit_price + 95
WHERE NAME = 'Tee Shirt';
```

Księgowy:
```
SELECT SUM( quantity * unit_price ) AS inventory
FROM product;
```

Sprzedaż:
```
ROLLBACK;
UPDATE product
SET unit_price = unit_price + 0.95
WHERE NAME = 'Tee Shirt';
```

Księgowy:
```
SELECT SUM( quantity * unit_price ) AS inventory
FROM product;
```

Zakaz czytania danych z transakcji nie zatwierdzonych.

Zakaz czytania danych z transakcji nie zatwierdzonych wprowadza poziom izolacji 1. Poziom ten w standardzie SQL określany jest także jako READ COMMITTED. Przy tym poziomie izolacji dopuszczalne jest jednak zapisywanie danych w transakcjach nie zatwierdzonych. Za konfliktowe uważa się wówczas takie pary operacji, gdzie pierwsza jest operacją zapisu, a druga czytania lub obydwie są operacjami zapisu. Dwie operacje, z których pierwsza jest operacją czytania, a druga operacją zapisu nie są więc konfliktowe. Można, zatem zapisywać dane, które zostały przeczytane przez transakcję jeszcze nie zatwierdzoną. Reguły współbieżności dla poziomu READ COMMITED:

	Read	Write
Read	T	T
Write	N	N

wady	zalety
nie chroni przed anomalią związaną z powtórnym czytaniem ani przed pojawianiem się fantomów	eliminacja anomali związanych z brakiem odtwarzalności i z kaskadą odrzuceń

Zakaz czytania i zapisywania danych w transakcjach nie zatwierdzonych

Zakaz czytania w transakcjach nie zatwierdzonych i zakaz zapisywania w nich związany jest z przyjęciem konfliktowości na poziomie 2, gdy za konfliktowe uważa się takie pary operacji, gdzie, co najmniej jedna jest operacją zapisu. W standardzie SQL określa się go także jako REPEATABLE READ. Za niekonfliktowe uważa się tylko operacje czytania. Jeśli więc transakcja nie zatwierdzona przeczytała jakąś daną, to dana ta może być tylko czytana przez inną transakcję. Jeśli natomiast transakcja nie zatwierdzona zapisała jakąś daną, to nie można jej ani odczytać, ani tym bardziej zapisać dopóty, dopóki transakcja ta nie zostanie zatwierdzona. Reguły współbieżności operacji mają wówczas postać:

	Read	Write
Read	T	N
Write	N	N

wady	zalety
eliminuje anomalie powtórnego czytania	Nie eliminuje natomiast problemu fantomów.

Historie szeregowalne

Rozwiązanie problemu fantomów wymaga poszerzenia rozważanych dotychczas pojęć współbieżności i konfliktowości w kierunku uwzględnienia formuł (predykatów) definiujących zbiory danych, na których działają rozważane transakcje.
Niech dane będą operacje $o(\varphi)$ i $p(\psi)$pochodzące z dwóch różnych i aktywnych transakcji (φ i ψ są formułami określającymi zbiory danych, na których działają operacje) oraz niech $o(\varphi) < p(\psi)$.
Przyjmijmy oznaczenia:

a) $X=\{x| \varphi(x)\}$ - zbiór danych spełniających warunek φ bezpośrednio przed wykonaniem operacji $o(\varphi)$,

b) $Y=\{y|\ \psi(y)\}$ - zbiór danych spełniających warunek ψ bezpośrednio przed wykonaniem operacji $p(\psi)$,
c) $X^*=\{x|\ \varphi(x)\}$ - zbiór danych spełniających warunek φ bezpośrednio po wykonaniu operacji $p(\psi)$,

Pojęcie współbieżności operacji rozszerzamy obecnie następująco:

a) Dwie operacji READ[φ] i READ[ψ] są zawsze współbieżne.
b) Dwie operacje o(φ), i WRITE[ψ] są współbieżne, jeśli zbiór, na którym działa druga z tych operacji, jest rozłączny ze zbiorem związanym z wykonaniem pierwszej z nich oraz wykonanie drugiej operacji nie zmieni zbioru związanego z wykonaniem pierwszej. Formalnie : $X \cap Y = 0$ oraz $X = X^*$aaslda
c) Operacje WRITE[φ] oraz READ[ψ] są współbieżne, jeśli zbiór, na którym działa druga z tych operacji, jest rozłączony ze zbiorem związanym z wykonaniem pierwszej z nich. Formalnie: $X \cap Y = 0$.

Reguły współbieżności przedstawiono w tablicy poniżej

	READ[ψ]		WRITE[ψ]	
	$X \cap Y = 0$	$X \cap Y \neq 0$	$(X \cap Y = 0) \cap (X=X^*)$	$(X \cap Y = 0) \cap (X \neq X^*)$
READ[φ]	T	T	T	N
WRITE[φ]	T	N	T	N

Przyjęcie tego rodzaju współbieżności eliminuje wszystkie problemy w tym również problem fantomów. Ten poziom izolacji określa się w standardzie SQL jako SERIALIZABLE. Według standardu jest to domyślny poziom izolacji.

Porównanie poziomów izolacji:

poziom izolacji	brudne czytanie	czytanie bez powtórzeń	fantom
READ UNCOMMITED	TAK	TAK	TAK
READ COMMITED	NIE	TAK	TAK
REPEATABLE READ	NIE	NIE	TAK
SERIALIZABLE	NIE	NIE	NIE

Anomalne historii przetwarzania transakcji

Z punktu widzenia analizy poprawności protokołów (algorytmów) zarządzania transakcjami istotnie jest analizowanie historii przetwarzania transakcji. Historia taka znana jest po wykonaniu wyznaczonego zbioru transakcji.

Definicja 1. Niech Σ={T1, T2, .., Tn} będzie zbiorem transakcji. Ciąg H=(o1, o2, ..., om) operacji pochodzących z transakcji należących do zbioru Σ **nazywamy historią przetwarzania transakcji za zbioru** Σ. Jeśli operacja o poprzedza operację o* w historii H, to stosować będziemy zapis o<o*.

Definicja 2. Mówimy, że transakcja T* **czyta z transakcji** T daną x, jeśli T jest ostatnią transakcją aktywną, która zapisała x. Mówimy, że transakcja T* **zapisuje w transakcji** T daną x, jeśli Todczytała x i pozostaje transakcją aktywną w momencie zapisu x przez T*.

Nieodtwarzalne historie przetwarzania.

Przypuśćmy, że transakcja T1 zmieniła wartość danej x, a następnie transakcja T2 wczytała x i na podstawie jej wartości zmieniła wartość danej y w bazie danych. Przypuśćmy dalej, że transakcja T2 została zatwierdzona, a po tym zdarzeniu powstała konieczność odrzucenia transakcji T1. Należałoby więc wycofać wszystkie zmiany, jakie wprowadziła w bazie danych transakcja T2, a także wszystkie konsekwencje tych zmian – w szczególności więc zmianę wartości danej y. Ta ostatnia operacja jest jednak niemożliwa, gdyż transakcja T2, która tę zmianę wykonała jest już zatwierdzona. Zaistniała więc sytuacja, w której baza danych jest **nieodtwarzalna.**

Rozważmy historię przetwarzania

H1:w1[x] r2[x] w2[y] c2 w1[z] a1.

H1 opisuje przetwarzanie nieodtwarzalne. Transakcja T2 czyta z transakcji T1, w1[x]<r2[x], i T2jest zatwierdzana przed odrzuceniem T1, c2<a1. Operacja w2[y] może oznaczać zapis wartości danej y wyznaczonej na podstawie wartości danej x. Zmiany tej jednak nie można wycofać podczas wycofywania konsekwencji transakcji T1, gdyś transakcja T2 została wcześniej zatwierdzona.

Powodem opisanej anomalii jest to, że transakcja czytająca dane z innej transakcji została zatwierdzona w czasie aktywności transakcji, z której czytała. Aby sytuacji takiej uniknąć, należałoby czekać z zatwierdzeniem transakcji T2 do czasu, aż zostanie zatwierdzona transakcja T1. Przyjmujemy więc, że **historia H opisuje przetwarzanie odtwarzalne, jeśli każda transakcja jest zatwierdzana po zatwierdzeniu wszystkich transakcji, z których czyta**. To znaczy, że c2 musi być później niż a1 (lub c1).

Historie przetwarzania z kaskadą odrzuceń.

Przestrzeganie zasady odtwarzalności nie jest wystarczające. Mimo jej przestrzegania może dojść do sytuacji, gdy odrzucenie jednej transakcji pociągnie za sobą konieczność odrzucenia zależnej od niej (w jakimś sensie) innej transakcji, odrzucenie tej drugiej może spowodować konieczność odrzucenia trzeciej itd., co może prowadzić do **kaskady odrzuceń**.

Niech na przykład transakcja T2 wczyta dane zmienione przez nie zatwierdzoną jeszcze transakcję T1. Przypuśćmy, że transakcja T1 zostaje po tym zdarzeniu odrzucona. Konsekwencją tego jest także konieczność odrzucenia transakcji T2. Ale T2 już wpisała dane do innych pół

tabel bazy danych (w2(u)). Może to spowodować konieczność kaskadowego odrzucania wielu transakcji.

Rozważmy następującą historie H2 powstałą z historii H1:

H2: w1[x] r2[x] w2[u] w1[z] a1

H2 opisuje przetwarzanie odtwarzalne. Jednak wykonanie operacji a1 powoduje odrzucenie (wycofanie) transakcji i, w konsekwencji, kaskadowe odrzucenie transakcji T2. Sytuacji tej można uniknąć, jeśli czytanie danych zmienionych przez transakcje jest dopuszczalne dopiero wtedy, gdy transakcje te zostały już zatwierdzone. **Historia H opisuje przetwarzanie bez kaskady odrzuceń, jeśli transakcji czyta dane zapisane przez transakcje już zatwierdzone**. To znaczy, że r2[x] musi być później niż a1 (lub c1)

Historie przetwarzania z anomalią powtórnego czytania.

Przypuśćmy, że transakcja T2 czyta daną y, a następnie transakcja T1 zapisuje nową wartość danej y jest zatwierdzana (transakcja T1 zapisuje w transakcji T2). Jeśli teraz transakcja T2 ponownie przeczyta daną y, to może się okazać, że dana ta ma inną wartość. Transakcja T2 dysponuje więc dwiema różnymi wartościami tej samej danej. Może zdarzyć się też sytuacja, że transakcja T1 usunie daną y. Wówczas przy próbie ponownego czytania, transakcja T1 ma informację, że danej y nie ma w bazie danych. Opisana anomalię nazywa się *anomalią powtórnego czytania*.

Rozważmy historię przetwarzania transakcji:

H3: w1[x] r2[y] w1[y] w1[z] c1 r2[y] c2

W H3 występuje anomalia powtórnego czytania, gdy między dwoma wystąpieniami operacji czytania, r2[y], wystąpiła operacja zapisu w1[y], czyli r2[y] < w1[y] <r2[y]. **Historie nazywa się historią bez anomalii powtórnego czytania, jeśli transakcja nie może zapisywać danych czytanych przez transakcje jeszcze nie zatwierdzone.**

Historie przetwarzania z fantomami.

Przypuśćmy, że transakcja T2 wczytała z tabeli R zbiór rekordów spełniających warunek φ. Następne inna transakcja, T1, dołączyła do R nowy rekord r spełniający warunek φ i została zatwierdzona. Jeśli T2 ponownie odwoła się do rekordów tabeli R spełniających warunek φ, to okaże się, że tym razem zbiór ten jest inny. Podobna sytuacja wystąpi, jeśli transakcja T1 dokona takiej modyfikacji rekordu r* nie spełniającego warunku φ, że po jej wykonaniu rekord **z** warunek ten będzie spełniał. Ten nowy rekord pojawiający się w obszarze zainteresowań transakcji T nazywany jest **fantomem** lub **zjawą**.

Rozważmy historię przetwarzania

H4: r2[u] w1[z] c1 r2[u] c2.

W historii H4 może wystąpić zjawisko fantomów. Jeśli bowiem operacja r2[u] wczytuje zbiór rekordów spełniających warunek φ, operacja w1[z] spowoduje, że zbiór takich rekordów ulegnie zmianie (na przykład tak zostaną zmienione pola rekordu **z**, że po zmianie rekord **z** będzie spełniał warunek φ), to powtórne wykonywanie operacji r2[u] zwróci inny zbiór rekordów.

Problem fantomów jest nieco podobny do anomalii powtórnego czytania. Jednak tym razem brak jest bezpośredniego konfliktu między wykonywanymi operacjami. Konflikt ten zauważalny jest

dopiero wtedy, gdy uwzględnione są warunki, jakie spełniają zbiory danych, na których wykonywane są operacje.

Przetwarzanie transakcji na różnych poziomach izolacji

Przyjęcie konkretnego poziomu izolacji wiąże się z określonymi problemami – zbyt niski poziom zapewni zwiększenie współczynnika współbieżności, ale może doprowadzić do niekorzystnych cech związanych z zachowaniem spójności bazy danych. Poziom zbyt wysoki może powodować nieuzasadnione opóźnianie transakcji.

Szeregowalność transakcji

Celem zarządzania transakcjami jest uzyskanie poprawnych historii przetwarzania. Pojęcie poprawności rozpatrywane jest na przyjętym poziomie izolacji.

Definicja 3. Niech Σ={T1, T2, .., Tn} będzie zbiorem transakcji a H=(o1, o2, ..., om) historią przetwarzania transakcji ze zbioru Σ. Historię H nazywamy sekwencyjną, jeżeli dla każdych dwóch transakcji, wszystkie operacje jednej z nich poprzedzają wszystkie operacje drugiej. W przeciwnym wypadku historia jest współbieżna.

Definicja 4. Niech Σ={T1, T2, .., Tn} będzie zbiorem transakcji a H=(o1, o2, ..., om) historią przetwarzania transakcji ze zbioru Σ. Grafem **szeregowalności** historii H nazywamy graf G(H)= (V,E), gdzie:

V – zbiór wierzchołków równy zbiorowi Σ,

$E \subseteq V \times V$ - zbiór krawędzi, przy czym krawędź $Ti \rightarrow Tj \in E$ wtedy i tylko wtedy, gdy istnieją konfliktowe operacje oi oraz oj pochodzące z transakcji odpowiednio Ti oraz Tj, takie, że oi< oj.

Wierzchołkami w grafie szeregowalności są więc transakcje ze zbioru Σ, a krawędź $Ti \rightarrow Tj$ oznacza, że istnieją konfliktowe operacje oi oraz oj pochodzące z transakcji odpowiednio Ti oraz Tj, gdzie oi poprzedza oj.

Jeśli kolejność operacji konfliktowych w H jest taka, że określana przez nią relacja nie jest relacją częściowego porządku $\leq$, to taka historia nie jest poprawna oraz graf szeregowalności jest grafem acyklicznym (niepoprawnym).

Szeregowalność jest poprawna, kiedy rezultat operacji jej historii jest taki samy jako rezultat operacji historii sekwencyjnej, a graf szeregowalności jest grafem acyklicznym.

Zgodnie z powyższą definicją, sekwencję operacji jest wyznaczona na podstawie kolejności konfliktowych operacji występujących w historii przetwarzania H. Do badania poprawności historii przetwarzania wykorzystuje się analizę grafów szeregowalności.

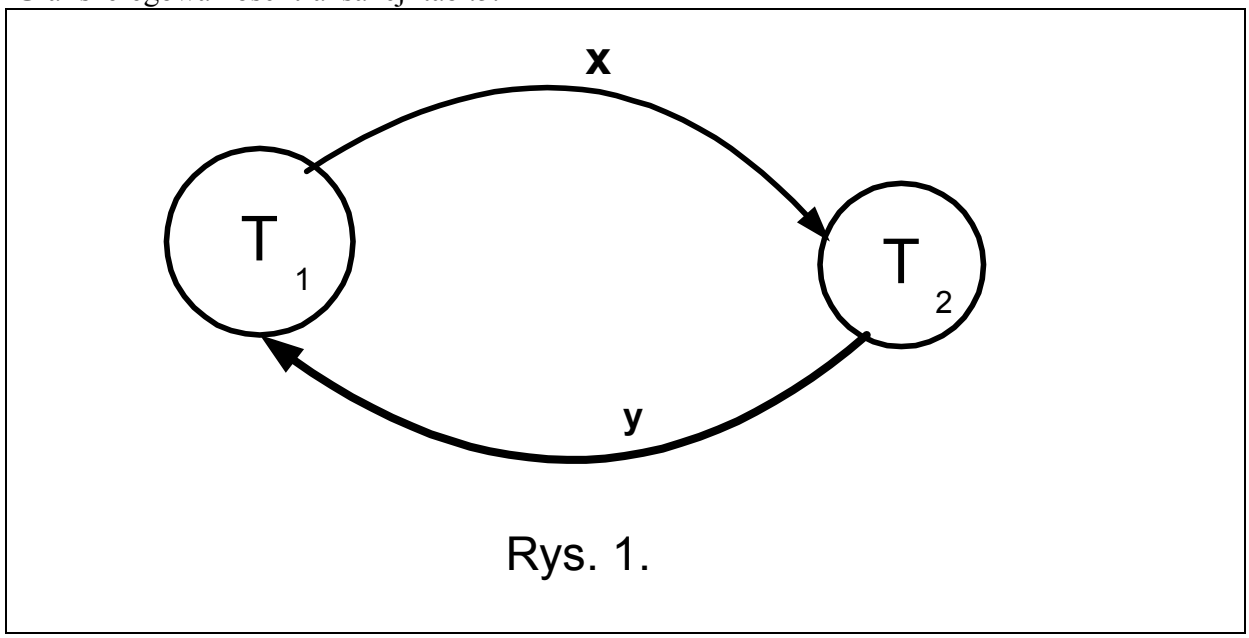
Tablica 5. Przykład grafu, który nie jest Szeregowanym

Transakcja T1 realizuje operacje przelewu kont. Transakcja T2 realizuje operacje kapitalizacji kont. Dla początkowego stanu kont X0 =100, Y0=400 przy poprawnej szeregowalności : a)Xk=220; Yk=330, jeżeli (T1->T2) lub b)Xk =210; Yk= 340, jeżeli (T2->T1).

Transakcja T_1	Transakcja T_2
begin transaction	

Read (x)	
x := x+100	
Write (x) X=200	begin transaction
	Read (x)
	x := x * 1.1
	Write (x) X=220(!)
	Read (y)
	y := y * 1.1
	Write (y) Y=440
Read (y)	commit
y := y - 100	
Write (y) Y=340(!)	
commit	

Graf szeregowalności transakcji tabl.5.

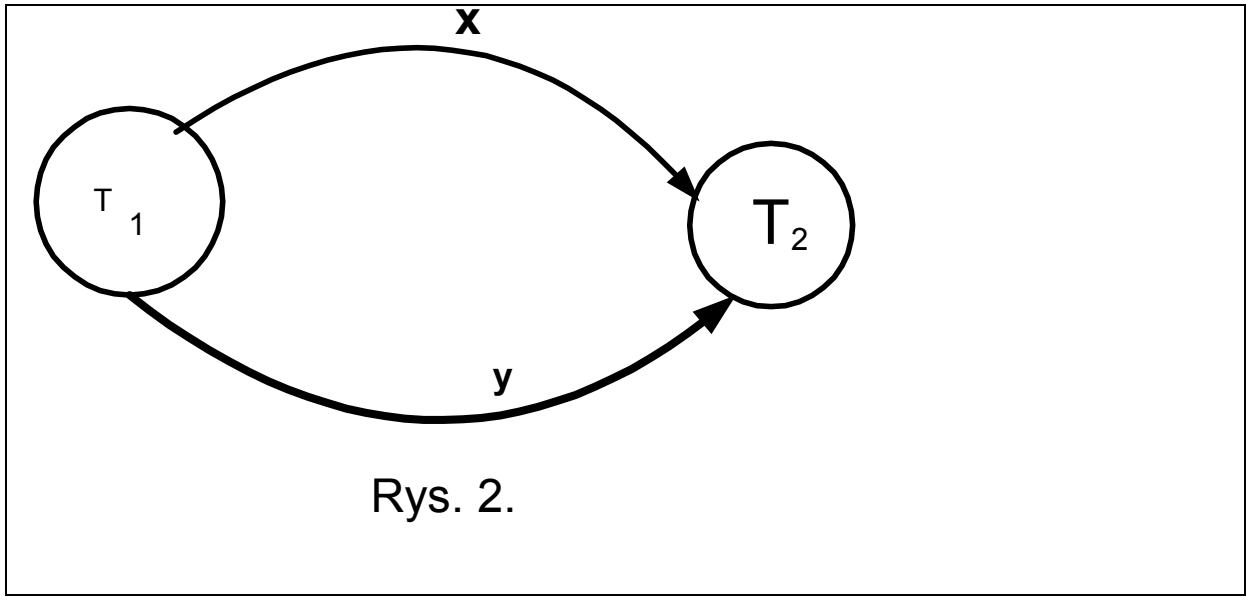


Rys. 1.

Tablica 6. Przykład grafu, który jest Szeregowanym
Transakcja T1 realizuje operacje przelewu kont. Transakcja T2 realizuje operacje kapitalizacji kont. Dla początkowego stanu kont X0 =100, Y0 =400 przy poprawnej szeregowalności : a)Xk=220; Yk=330, jeżeli (T1->T2) lub b)Xk =210; Yk= 340, jeżeli (T2->T1).

Transakcja T1	Transakcja T2
begin transaction	
Read (x)	
x := x+100	
Write (x) X=200	begin transaction
	Read (x)
	x := x * 1.1

	Write (x) X=220
Read (y)	
y := y - 100	
Write (y) Y=300	
commit	Read (y)
	y := y * 1.1
	Write (y) Y=330
	commit



Rys. 2.

Graf szeregowalności transakcji tabl.6.

Łatwo wykazać (Bernstein, 1987), że jeśli graf szeregowalności jest acykliczny, to odpowiadająca mu historia przetwarzania jest poprawna – zorientowany graf acykliczny jest bowiem graficzną formą reprezentacji zbioru częściowo uporządkowanego. Jeśli natomiast w grafie szeregowalności istnieje jakikolwiek cykl, to odpowiadająca mu historia przetwarzania jest na pewno niepoprawna.
Szeregowalność oznacza, że zbiór operacji występujących w historii H możemy ułożyć w ciąg, w którym operacje poszczególnych transakcji nie przeplatają się, ale jednocześnie zachowana jest kolejność wszystkich operacji konfliktowych. Taka historia przetwarzania odpowiada szeregowemu wykonywaniu transakcji, stąd historie generujące częściowy porządek w zbiorze transakcji nazywamy *historiami szeregowalnymi*. Zauważmy, że w przypadku wystąpienia pętli lub cyklu w grafie szeregowalności odpowiadającym historii H nie dałoby się przekształcić H w historie przetwarzania szeregowego.

Zarządzanie transakcjami w języku SQL

Transakcji mają właściwość ASOT. Transakcja rozpoczyna się w chwili wydania polecenia inicjującego transakcję (begin transaction...). Charakterystyki transakcji określa się za pomocą komend SET TRANSACTION i SET CONSTRAINTS o składni:

```
COMMIT [ WORK ];

ROLLBACK [ WORK ];

SET TRANSACTION
{ ISOLATION LEVEL
{ READ UNCOMMITED
| READ COMMITED
| REPEATABLE READ
| SERIALIZABLE
| { READ ONLY | READ WRITE }
| { DIAGNOSTICS SIZE } } .,.. ;

gdzie opcje:
        tryb dostępu:                   READ ONLY, READ WRITE
        rozmiar obszaru diagnostyk:     DIAGNOSTICS SIZE ilość-warunków
        Poziom izolacji:                ISOLATION LEVEL izolacja
    izolacja:   SERIALIZABLE  (domyślna)
                REPEATABLE READ,
                READ COMMITED,
                READ UNCOMMITED.

SET CONSTRAINTS { lista-warunków/ ALL}
                { DEFERRED / IMMEDIATE }
```

SET CONSTRAINTS ustała tryb sprawdzania warunków spójności na natychmiastowy (IMMEDIATE) lub opóźniony (DEFERRED). Przyjęcie określonego poziomu izolacji może być źródłem problemów omówionych wcześniej. W tablice 7 jest pokazany związek poziomów izolacji z problemami przetwarzania transakcji.

Przykłady przetwarzania bazy danych na różnych poziomach izolacji.

Przypuśćmy, że w bazie danych istnieje tabela Towar o postaci:

Nazwa	Cena	Stan
200MMX	320	20
233MMX	370	50

Poziom izolacji 0. Przy poziomie izolacji 0 możliwe jest czytanie danych zmienionych przez transakcje jeszcze nie zatwierdzone. Mówi się wówczas o „brudnym czytaniu". Dopuszczenie takiego czytania bardzo zwiększa współbieżność przetwarzania, ale jednocześnie może

doprowadzić do udostępniania nieprawdziwych danych z bazy danych, co ilustruje przykład następującej historii przetwarzania:

Transakcja T1	Transakcja T2
set transaction isolation level 0	set transaction isolation level 0
begin transaction update Towar set Cena = 300 Where Nazwa = '200MMX' T1 zmienia cenę	
	begin transaction select Cena from Towar where nazwa = '200MMX' T2 czyta zmienioną cenę
Rollback T1 wycofuje zmianę	
	T2 posiada niepoprawną informację o cenie

Zerowy poziom izolacji może być stosowany tylko w takich transakcjach, o których wiemy, że nawet w przypadku błędnych danych nie spowodują poważnych negatywnych konsekwencji. Można go stosować na przykład dla transakcji, których zadaniem jest tylko udzielanie informacji z bazy danych.

Poziom izolacji 1. Cechą charakterystyczną tego poziomu izolacji jest to, że możliwe jest aktualizowanie przez transakcję T2 danych wczytanych przez nie zakończoną jeszcze transakcję T1. Po powtórnym odwołaniu się do tych samych danych transakcji T1 można uzyskać sprzeczne informacje. Ilustruje to historia przetwarzania transakcji (odczyt-zapis)

Transakcja T1	Transakcja T2
set transaction isolation level 1	set transaction isolation level 1
begin transaction select Cena, Stan from Towar where Nazwa = '200MMX' T1 czyta cenę i stan towaru	
	begin transaction update Towar set Cena = 310 where Nazwa = '200MMX' T2 zmienia cenę wczytaną przez T1
select sum(Cena*Stan) from Towar where Nazwa = '200MMX' T1 czeka na zakończenie T2	
	Commit
Wykonanie oczekującej operacji 'select' *dla T1. Wynik jest sprzeczny z poprzednią operacją* 'select'	

Poziom izolacji 2. W poziomie izolacji 2 mamy zagwarantowanie, że przy ponownym odwołaniu się do tych samych danych dostajemy identyczne informacje. Historia przetwarzania transakcji na poziomie izolacji 2 (odczyt-zapis-odczyt)

Transakcja T1	Transakcja T2
set transaction isolation level 2	set transaction isolation level 2
begin transaction select Cena, Stan from Towar where Nazwa = '200MMX' T1 czyta cenę i stan towaru	
	begin transaction update Towar set Cena = 310 where Nazwa = '200MMX' T2 czeka na zakończenie T1
select sum (Cena*Stan) from Towar where Nazwa = '200MMX' T1 oblicza wartość towaru	
Commit	
	Wykonanie oczekującej operacji 'update' *dla T2.*

Poziom izolacji 2 zabezpiecza przed modyfikacją wczytanych danych, ale nie przed dołączeniem nowych wierszy. Sytuację te ilustruje historia przetwarzania podana poniżej (odczyt-dołączenie-odczyt)

Transakcja T1	Transakcja T2
set transaction isolation level 2	set transaction isolation level 2
begin transaction select Cena, Stan from Towar where Nazwa = '200MMX' T1 czyta cenę i stan towaru	
	begin transaction insert into Towar values ('200MMX', 250,10) T2 dołącza nowy wiersz „fantom"
	Commit
Select sum (Cena*Stan) from Towar where Nazwa ='200MMX' *T1 oblicza wartość towaru. Wynik jest sprzeczny z poprzednią operacją select*	

Poziom izolacji 3. Przed pojawieniem się fantomów chroni poziom izolacji 3. Przy tym poziomie izolacji przetwarzanie z poprzedniego przykładu (odczyt-dołączenie-odczyt) miałoby następującą historię:

Transakcja T1	Transakcja T2

set transaction isolation level 3	set transaction isolation level 3
begin transaction select Cena, Stan from Towar where Nazwa = ‘200MMX’ T1 czyta cenę i stan towaru	
	begin transaction insert into Towar values (‘200MMX’, 250, 10) T2 czeka na zakończenie T1
Select sum (Cena*Stan) from Towar where Nazwa =’200MMX’ *T1 oblicza wartość towaru.*	
Commit	
	Wykonanie „insert” *przez T2*

Metody sterowania współbieżnością transakcji na różnych poziomach izolacji.

Sterowanie współbieżnością transakcji realizuje się przez przetwarzanie historii niepoprawnych do historii szeregowalnych. Szeregowalność może być realizowana za dopomogą blokowania danych oraz metody znaczników czasowych.

Istnieją cztery typy blokad:

- **blokada czytania** (współdzielona)
- **blokada widmowa (**współdzielona)
- **blokada zapisu (** wyłączna)
- **blokada nie-widmowa (**współdzielona)

Metody blokowania danych.

Blokowanie to jest protokół, który jest wykorzystywany podczas równoległego dostępu do danych przez różne transakcje. Z każdym używanym obiektem w bazie danych jest związana blokada (lock). Kiedy jakaś transakcja otrzyma dostęp do danych, mechanizm blokowania nie dopuści do tych samych danych innych transakcji. Wyróżniają się dwa podstawowe typy blokad:

- blokadę współdzieloną (shared lock),
- blokadę wyłączną (exclusive lock).

Operacje na danej nie powodujące jej uaktualnienia powinny być poprzedzone założeniem blokady współdzielonej. Operacje uaktualniające daną powinny być poprzedzone założeniem na niej blokady wyłącznej. Ze względu na proces blokowania, dane w bazie danych mogą występować w jednym z trzech stanów:

- dana nie zablokowana 0
- dana zablokowana dla odczytu Read (współdzieloną - shared lock)
- dana zablokowana dla zapisu Write (wyłączną - exclusive lock).

Blokada może być ustalona dla różnych poziomach detalizacji danych:

- wartość kolumny tabeli
- wiersz tabeli
- tabela, widok

- baza danych.

Blokady implementowane są za pomocą oddzielnych bitów w polu danych. Wartość tych bitów odpowiadają typowi blokady. Mechanizm blokowania zawiera zasoby sterowania kolejkami dla blokowania danych. Główne reguły protokołu blokowania danych:

- Transakcja która ustawiła blokadę danej „dla odczytu" (Read) może tylko czytać, zaś nie może tej danej uaktualniać.
- Transakcja, która ustawiła blokadę danej „dla zapisu" (Write) może czytać oraz uaktualniać tą daną.
- Transakcja realizuje się zgodnie z protokółem blokowania dwu-fazowego(two-phase locking): wszystkie operacje blokowania poprzedzają pierwszą operację odblokowania.

Operacja czytania ri[x] transakcji i nie jest operacją konfliktową, dlatego blokada „dla odczytu" **(Read)** jednej danej x może być ustawiona jednocześnie przez wiele transakcji. Natomiast blokada „dla zapisu" (**Write)** operacji wi[x] transakcji i blokuje dostęp do danej x przez inne transakcje.

Protokół blokowania danych przez transakcje składa się z następujących czynności:

- Jakakolwiek transakcja i, która potrzebuje dostępu do obiektu x musi na początek ustawić blokadę tego obiektu . Blokada może być ustawiona „dla odczytu" (Read) lub „dla zapisu" (Write). W ostatnim przypadku dostęp do czytania oraz do zapisu obiektu x będzie miała tylko transakcja, która ustaliła blokadę Write.
- Blokada będzie ustalona skuteczne, kiedy obiekt x nie ma żadnej blokady.
- W tych przypadkach, kiedy obiekt x jest już zablokowany przez inną transakcję, menedżer SŻBD musi analizować, czy typ blokady nowej jest kompatybilnym z typem blokady ustawionej wcześniej. Kiedy transakcja Tj chce ustawić dla obiektu x typ blokady Read oraz obiekt ten został już zablokowany blokadą Read przez inną transakcją Ti, to transakcja Tj będzie miała dostęp dla odczytu obiektu x równolegle z transakcją Ti. W innych przypadkach transakcja Tj będzie w stanie oczekiwania (Wait) dopóki, dopóty blokada obiektu x nie zostanie zwolniona przez transakcją Ti.
- Transakcja Ti utrzyma blokadę obiektu x dopóki, dopóty nie odblokuje go w sposób jawny. Odblokowanie może być spowodowane skutecznym zatwierdzenia transakcji (Commit) lub w razie je wycofania (Rollback). Po odblokowaniu obiektu x przez Ti inne transakcji będą mogły sięgnąć do obiektu x.
- Transakcja Ti może rozszerzyć swoje blokadę „dla odczytu" (Read) obiektu x do poziomu blokady „dla zapisy" (Write), kiedy inne transakcje nie ustawiły blokad tego obiektu.

Algorytm blokowania dwufazowego

Najszerzej stosowanym w praktyce jest algorytm blokowania dwufazowego (two-phase locking) oznaczony przez 2PL. Istotą tego algorytmu są następujące założenia:

- Każda transakcja zawiera dwie fazy: fazę blokowania (ekspansji) oraz fazę odblokowania (zwijania).
- W fasie blokowania transakcja musi uzyskać blokady wszystkich danych, do których będzie dokonywać dostępu. Moment założenia wszystkich żądanych blokad, równoznacznych z zakończeniem fazy blokowania, jest nazywany punktem akceptacji (commit point).
- W fazie odblokowania (po operacji **commit** lub **rollback)**, następuje zdejmowanie wszystkich nałożonych blokad. Ponadto w fazie tej nie można zakładać nowych blokad.

Diagram czasowy fazę blokowania (ekspansji) oraz fazę odblokowania (zwijania) jest pokazany na rysunku poniżej.

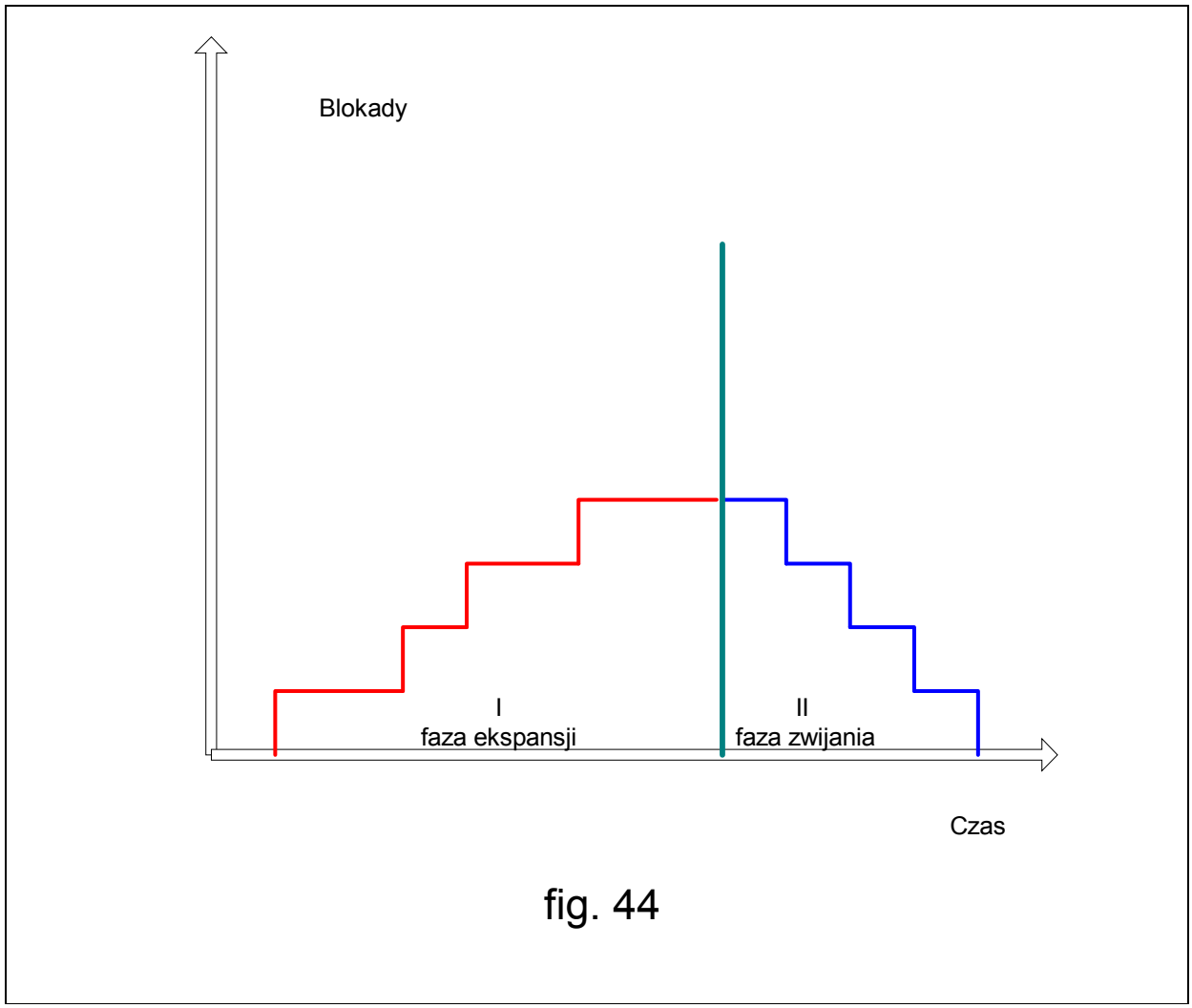


fig. 44

W algorytmie 2PL odczyt danej jest możliwy natychmiast po nałożeniu blokady tej danej, a więc w fazie blokowania, natomiast zapis jest możliwy dopiero w po osiągnięciu przez transakcję punktu akceptacji, a więc w fazie odblokowania.
Operacja zapisu jest wykonywana następująco. Założenie blokady wyłącznej („dla zapisu” (Write)) jest równoznaczne z wykonaniem zapisu wstępnego w obszarze roboczym związanym z zapisywaną daną. Zapis właściwy jest realizowany dopiero w fazie odblokowania, w momencie zdejmowania blokady tej danej na podstawie zawartości obszaru roboczego.

Tablica 14 . Przykład protokołu dwufazowego .
Transakcja T1 realizuje operacje przelewu kont. Transakcja T2 realizuje operacje kapitalizacji kont. Dla początkowego stanu kont X0 =100, Y0 =400 przy poprawnej szeregowalności : a)Xk=220; Yk=330, jeżeli (T1->T2) lub b)Xk =210; Yk= 340, jeżeli (T2->T1).

Transakcja T_1	Transakcja T_2
begin transaction	begin transaction
Write_Lock (x)	Write_Lock (x)
Wait	Read (x)
Wait	x := x * 1.1
Wait	Write (x) X=110
Wait	Write_Lock (y)
Wait	Read (y)
Wait	y := y * 1.1
Wait	Write (y) Y=440
Wait	commit / Unlock(x),Unlock(y)
Read (x)	
x := x+100	
Write (x) X=210 (+)	
Write_Lock (y)	
Read (y)	
y := y - 100	
Write (y) Y=340 (+)	
commit/ Unlock(x),Unlock(y)	

Tablica 15. Przykład niepoprawnego grafu szeregowalności , realizowanego przez protokół, który nie jest dwufazowym.
Transakcja T1 realizuje operacje przelewu kont. Transakcja T2 realizuje operacje kapitalizacji kont. Dla początkowego stanu kont X0 =100, Y0 =400 przy poprawnej szeregowalności : a)Xk=220; Yk=330, jeżeli (T1->T2) lub b)Xk =210; Yk= 340, jeżeli (T2->T1).

Transakcja T_1	Transakcja T_2
begin transaction	
Write_Lock (x)	begin transaction
Read (x)	Write_Lock (x)
x := x+100	Wait
Write (x) /Unlock(x) X=200	Wait
	Read (x)
	x := x * 1.1
	Write (x), Unlock(x) X=220(!)
Write_Lock (y)	Write_Lock (y)
Wait	Read (y)
Wait	y := y * 1.1
Wait	Write (y) Y=440
Wait	commit / Unlock(y)
Read (y)	
y := y - 100	
Write (y) Y=340(!)	

commit/ Unlock(y)	

Zakleszczenia transakcji.

Zakleszczenie (deadlock) może wystąpić przy dwu-fazowym blokowaniu transakcji. Sytuacja ta powstaje wtedy, gdy transakcja T1 blokuje daną X i żąda dostępu do danej Y, podczas gdy transakcja T2 blokuje daną Y i żąda dostępu do danej X; żadna z tych transakcji nie może kontynuować swojego przebiegu. Możliwe są zakleszczenia, w których uczestniczy wiele transakcji.

Tablica 16. Przykład zakleszczenia transakcji

Transakcja T_1	Transakcja T_2
begin transaction	begin transaction
Write_Lock (y)	Write_Lock (x)
Read (y)	Read (x)
y := y - 100	x := x * 1.1
Write (y)	Write (x)
Write_Lock (x)	Write_Lock (y)
Wait...	Wait...

Jedynym sposobem walki z zakleszczeniem transakcji jest wycofanie jednej z zakleszczonych transakcji. Główne strategii walki z zakleszczeniem transakcji:

- Wykrywanie zakleszczeń
- Zapobieganie zakleszczeń.

Wykrywanie zakleszczeń może być realizowane przez graf oczekiwania transakcji (wait-for-graf). Ten graf odwzorowuje zależność jednej transakcji od drugiej. Przykład grafu jest pokazany na fig 45. Każdy wierzchołek odpowiada transakcji. Krawędź T1->T2 oznacza, że transakcja T1 czeka na odblokowanie danej X przez transakcje T2.

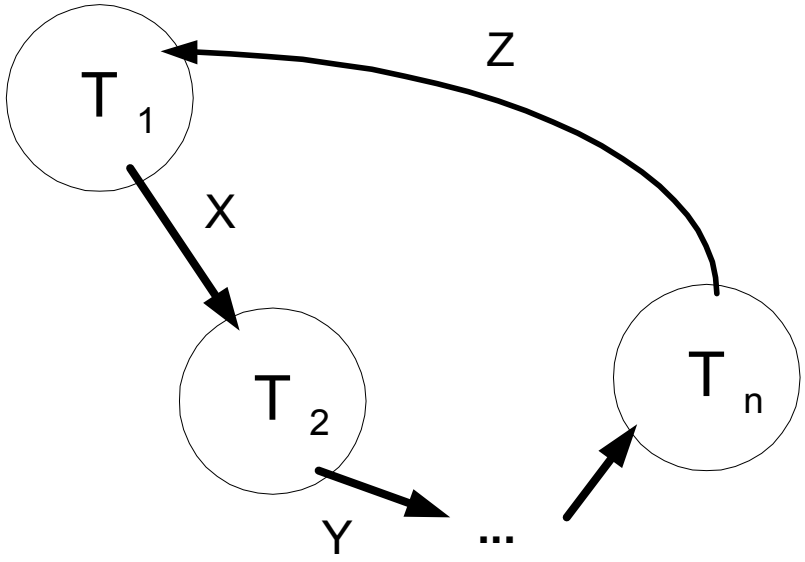


Fig.45

Pętla w grafu oczekiwań jest warunkiem koniecznym oraz dostatecznym istnienia zakleszczeń transakcji. Algorytm wykrywania zakleszczeń zawiera następnie kroki:

a) Wyznaczenia początkowej wartości interwału T pomiędzy dwoma kolejnymi generacjami grafu oczekiwań transakcji. Przy małym T częste wykrywanie zakleszczeń powoduje obciążenie procesora oraz małą wydajność SZBD. Przy dużym T transakcje, które są zakleszczone, mogą być nie wyznaczone w ciągu interwału T.
b) Generacja grafu zakleszczeń po zakończeniu interwału T.
c) Analiza grafu zakleszczeń:

```
If zakleszczenia istnieją  Then
    wycofanie transakcji, która jest w pętle grafu oczekiwań;
   T:=T/2;
   GOTO 2;
Else
   T:=2T;
   GOTO 2;
EndIf
```

Metody znaczników czasowych.

Metody znaczników czasowych(Timestamp ordering) są alternatywą do metod szeregowania historii przetwarzania transakcji przez blokowania danych. Te metody są wykorzystywane w przypadkach, kiedy konflikty pomiędzy transakcjami są rzadkie. Dla usunięcia konfliktów nie jest potrzebny graf oczekiwania transakcji.

Definicja. Znacznik czasowy (Timestamp) transakcji **T**, czyli **TS(T)**, jest jej unikalnym identyfikatorem, który wyznaczy się czas zjawy transakcji **T** w SZBD. Znaczniki są przydzielone transakcjom w kolejności, w której transakcje pojawiają się w SZBD.

Również z transakcjami, w bazie danych z każdą daną (X) są związane następne dwie wartości znaczników czasowych:

- Read_TS(X) –największy (najstarszy) znacznik czasowy spośród wszystkich transakcji, które pomyślnie odczytały daną X.
- Wtrite_TS(X) - największy (najstarszy) znacznik czasowy spośród wszystkich transakcji, które pomyślnie zapisały daną X.

Implementacja algorytmu znaczników czasowych dla operacji odczytywania danych:

```
Read (Tj,X)  begin
        If  (Write_TS (X)= =TRUE)  Then
< abort Tj  and restart it with a new Timestamp>;
        Else begin
                < Read X>;
                Read_TS (X) := max (Read_TS(X), TS(Tj ));
        End;
End Read;
```

Dla realizacji operacji odczytywania danej X transakcja Tj czyta typ znacznika czasowego danej X. Kiedy ten typ ma wartość Write_TS(X) ustaloną przez inną transakcję Ti, transakcja Tj będzie wycofana oraz startowana z nowym znacznikiem czasowym. Kiedy typ znacznika czasowego danej X jest Read_TS (X) ustalony przez inną transakcję Ti, ten typ zmieni się na największy znacznik (najstarszy) spośród transakcji Ti oraz Tj.

Implementacja algorytmu znaczników czasowych dla operacji zapisywania danych:

```
Write(T,X) begin
        If (TS (Tj) <Read_TS(X) or TS(Tj)<Write_TS(X)) Then
        < abort Tj and restart it with a new Timestamp>;
        Else begin
        <Write X>
                Write_TS(X) :=  TS(Tj);
        End;
End Write;
```

Dla realizacji operacji zapisywania daną X transakcja czyta z początku typ znacznika czasowego daną X ustalonego wcześniej przed inną transakcją. Kiedy ten znacznik jest starszy czym znacznik Tj, transakcja Tj będzie wycofana już z nowym znacznikiem czasowym. Kiedy transakcja Tjjest starsza od etykiety znacznika czasowego daną (X), dana (X) będzie miała nową wartość: Write_TS(X) := TS(Tj).

Zaleta metody znaczników czasowych:	Wada metody znaczników czasowych:
Wykorzystanie metody nie powoduje zakleszczeń transakcji.	Dość często jest wykorzystywane wycofanie transakcji.

Automatyczne zatwierdzanie transakcji

Większość DBMS posiada specjalny tryb pracy (***ang Autocommit mode***), w którym każde wykonanie zapytania powoduje automatyczne zatwierdzenie transakcji. Do włączenia trybu ***Autocommit*** w serwerze SQL SYBASE ANYWHERE służy następujące polecenie:

```
SET OPTION auto_commit = 'on' ;
```

Dla wyłączenia tego trybu służy następujące polecenie:

```
SET OPTION auto_commit = 'off' ;
```

Rozpoczęcie transakcji

W niektórych implementacjach do rozpoczęcia transakcji służy instrukcja:

```
BEGIN TRANSACTION
```

pozwalająca jawnie określić początek transakcji (choć zdarza się, że instrukcją rozpoczynającą jest BEGIN lub BEGIN TRANS). W SQL:2003 nie ma wyróżnionej takiej procedury - transakcje rozpoczynają instrukcje takie jak: CREATE TABLE, SELECT, UPDATE.

Zakończenie transakcji

Jawne zakończenie trasnakcji zapewnia wykonanie instrukcji:

```
COMMIT;
```

Powoduje ono:

- Zakończenie transakcji,
- Zatwierdzenie zmian w bazie danych,
- Usunięcie wszystkich założonych blokad i punktów zachowania,
- Udostępnienie zmian innym użytkownikom bazy.

Instrukcja ROLLBACK

Wykonanie instrukcji

```
ROLLBACK;
```

powoduje:

- Zakończenie transakcji,
- Wycofanie wszystkich zmian, które byli dokonane od rozpoczęcia transakcji,
- Usunięcie wszystkich założonych blokad i punktów zachowania.

Instrukcje SAVEPOINT, ROLLBACK TO SAVEPOINT

Transakcje składające się z dużej liczby poleceń lub modyfikujące dużą liczbę wierszy warto podzielić na kilka mniejszych części. Począwszy od SQL:1999 transakcje mogą być podzielone na subtransakcje za pomocą wyrażenia SAVEPOINT:

```
SAVEPOINT savepoint-name;
```

Wyrażenie to definiujących położenie tzw. punktu kontrolnego (większość SZBD pozwala na definiowanie punktów kontrolnych). Wprowadzenie punktu kontrolnego umożliwia częściowe wycofanie rozpoczętej transakcji. Dzięki temu zmiany wprowadzone przed punktem kontrolnym nie zostają utracone. O ile zdefiniowaliśmy punkt kontrolny, możemy wycofać część zmian wprowadzonych w ramach transakcji. W tym celu należy wykonać instrukcję:

```
ROLLBACK TO SAVEPOINT savepoint-name;
```

Przykład z wycofaniem transakcji:
Utworzenie tabeli tymczasowej, w którą wpisywane będą sumy zamówień klientów z tabeli Zamówienia

```
CREATE TABLE Temp
        (KlientID CHAR(4) PRIMARY KEY,
        SumaZamówienia INTEGER);
```

Utworzenie transakcji wpisującej dane do tabeli

```
BEGIN TRANSACTION
INSERT INTO Zamówienie (KlientID, SumaZamówienia)
        SELECT KlientID, SUM(SumaZamówienia)
        FROM Zamówienia
        GROUP BY KlientID;
```

Wyświetlenie danych z tabeli tymczasowej (widać wprowadzone dane)

```
SELECT * FROM Temp;
```

Wycofanie transakcji:

```
ROLLBACK;
```

Wyświetlenie danych z tabeli tymczasowej po wycofaniu transakcji (tabela jest pusta)

```
SELECT * FROM Temp;
```

Przykład z SAVEPOINT

Usunięcie wierszy z tabeli

```
SAVEPOINT SP1;
DELETE FROM Temp WHERE CUSTID='klient1';
SAVEPOINT SP2;
DELETE FROM Temp WHERE CUSTID=' klient2';
SAVEPOINT SP3;
DELETE FROM Temp WHERE CUSTID=' klient3';
SAVEPOINT SP4;
```

Wyświetlenie danych z tabeli tymczasowej (widać brak klient1, klient2, klient3)

```
SELECT * FROM Temp;
```

Wycofanie transakcji do SP2

```
ROLLBACK TO SAVEPOINT SP2
```

Wyświetlenie danych z tabeli tymczasowej (widać brak klient1)

```
SELECT * FROM Temp;
```

Konsekwencją wprowadzania ograniczeń na wartości tabel (np. NOT NULL) jest konieczność wstawiania do tabel wierszy, które te ograniczenia spełniają. Tak też dzieje się przy okazji realizacji transakcji. Czasem jednak wygodnym byłoby wstawienie najpierw jakiegoś pustego wiersza do tabeli, a później, pod koniec transakcji, wypełnienie go właściwymi wartościami. Aby było to możliwe w SQL:2003 można określać ograniczenia jako:

```
DEFERRABLE lub NOT DEFERRABLE
```

Ograniczenia NOT DEFERRABLE są stosowane natychmiast. Można jednak sprawić, aby te ograniczenia były początkowo DEFERRED lub IMMEDIATE.
Jeśli ograniczenie DEFERRABLE jest ustawione na IMMEDIATE, działa ono jak ograniczenie NOT DEFERRABLE (tzn. natychmiast). Jeśli ograniczenie DEFERRABLE jest ustawione na DEFERRED, nie jest ono wymuszane.

Dlatego, aby wstawić w tabelę wiersz z pustymi wartościami (albo wykonać inna operację naruszająca ograniczenia DEFERRABLE, można użyć wyrażenia jak niżej:

```
SET CONSTRAINTS ALL DEFERRED ;
```

Spowoduje ono, że wszystkie ograniczenia DEFERRABLE staną się DEFERRED. Działaniem tym nie zostaną objęte ograniczenia NOT DEFERRABLE. Po wykonaniu operacji, po których wartości w tabeli nie naruszają ograniczeń, można przywrócić pierwotne ustawienia ograniczeń:

```
SET CONSTRAINTS ALL IMMEDIATE ;
```

W przypadku, gdy zapomni się o wykonaniu powrotnego polecenia, wykona się ono automatycznie przy zatwierdzaniu transakcji za pomocą COMMIT. Jeśli wtedy będzie naruszone jakieś ograniczenie, zgłoszony zostanie błąd.

Przykład:
Niech tabela EMPLOYEE ma kolumny EmpNo, EmpName, DeptNo, Salary. Niech DeptNo będzie kluczem obcym odwołującym się do tabeli DEPT o kolumnach DeptNo, DeptName, Payroll, przy czym DeptNo jest kluczem prywatnym. Kolumna PayRoll zawiera sumę wartości Salary osobno dla każdego z departamentów. Można więc utworzyć widok:

```
CREATE VIEW DEPT2 AS
SELECT D.*, SUM(E.Salary) AS Payroll
FROM DEPT D, EMPLOYEE E
WHERE D.DeptNo = E.DeptNo
GROUP BY D.DeptNo ;
```

Podobnie można utworzyć równoważną postać tej tabeli za pomocą następującego widoku:

```
CREATE VIEW DEPT3 AS
SELECT D.*,
(SELECT SUM(E.Salary)
FROM EMPLOYEE E
WHERE D.DeptNo = E.DeptNo) AS Payroll
FROM DEPT D ;
```

Przypuśćmy, że nie chcemy obliczać SUM za każdym razem, gdy odwołujemy się do DEPT.Payroll. Zamiast tego zależy nam na zapisaniu aktualnej wartości w kolumnie Payroll tabeli DEPT table. Dlatego należałoby za każdym razem, kiedy zmienia się Salary, zmieniać również Payroll.

Aby mieć pewność, że Salary jest właściwa, można dołożyć ograniczenie CONSTRAINT do definicji tabeli:

```
CREATE TABLE DEPT
(DeptNo CHAR(5),
DeptName CHAR(20),
Payroll DECIMAL(15,2),
CHECK (Payroll = (SELECT SUM(Salary)
FROM EMPLOYEE E WHERE E.DeptNo= DEPT.DeptNo)));
```

Przypuśćmy, że chcemy zwiększyć teraz Salary pracownika 123 o wartość 100.
Można to wykonać za pomocą wyrażenia:

```
UPDATE EMPLOYEE
SET Salary = Salary + 100
WHERE EmpNo = '123' ;
```

Przy czym nie wolno zapomnieć o wykonaniu jednocześnie:

```
UPDATE DEPT D
SET Payroll = Payroll + 100
WHERE D.DeptNo = (SELECT E.DeptNo
FROM EMPLOYEE E
WHERE E.EmpNo = '123') ;
```

Pojawia się problem: wszystkie ograniczenia powinny być sprawdzane po każdym z wyrażeń. W praktyce implementacja sprawdza tylko ograniczenia, które związane są z modyfikowanymi wartościami.

Tak więc po pierwszym poprzedzającym UPDATE wyrażeniem, implementacja sprawdza wszystkie ograniczenia, które związane są z wartościami modyfikowanymi przez wyrażenie. Są to ograniczenia zdefiniowane w tabeli DEPT (bo odnoszą się one do kolumny Salary tabeli EMPLOYEE, a wyrażenie UPDATE modyfikuje tą kolumnę). Po wykonaniu pierwszego UPDATE ograniczenia są przekroczone. Zakładamy, że przed wykonaniem UPDATE baza danych jest poprawna i każda wartość Payroll w tabeli DEPT równa się sumie wartości Salary w odpowiadających kolumnach tabeli EMPLOYEE. Kiedy pierwsze UPDATE zwiększa Salary, równość ta staje się fałszywa. Drugie wyrażenie UPDATE koryguje to i znowu baza danych jest w stanie, w którym wszystkie ograniczenia są spełnione. Pomiędzy dwoma stanami ograniczenia są niespełnione.

Używając SET CONSTRAINTS DEFERRED można tymczasowo zablokować ograniczenia lub je zawiesić (wszystkie lub tyko ich część). Ograniczenia są odłożone aż do wykonania SET CONSTRAINTS IMMEDIATE lub COMMIT lub ROLLBACK.

```
SET CONSTRAINTS DEFERRED ;
```

```
UPDATE EMPLOYEE
SET Salary = Salary + 100
WHERE EmpNo = ‘123’ ;
UPDATE DEPT D
SET Payroll = Payroll + 100
WHERE D.DeptNo = (SELECT E.DeptNo
FROM EMPLOYEE E
WHERE E.EmpNo = ‘123’) ;
SET CONSTRAINTS IMMEDIATE ;
```

Powyższa procedura blokuje wszystkie ograniczenia. Można jednak zablokować tylko część ograniczeń (aby np. zachować sprawdzanie wartości kluczy głównych w DEPT):

```
CREATE TABLE DEPT
(DeptNo CHAR(5),
DeptName CHAR(20),
Payroll DECIMAL(15,2),
CONSTRAINT PayEqSumsal
CHECK (Payroll = SELECT SUM(Salary)
FROM EMPLOYEE E WHERE E.DeptNo = DEPT.DeptNo)) ;
```

Można też blokować ograniczenia indywidualnie:

```
SET CONSTRAINTS PayEqSumsal DEFERRED;
UPDATE EMPLOYEE
SET Salary = Salary + 100
WHERE EmpNo = ‘123’ ;
UPDATE DEPT D
SET Payroll = Payroll + 100
WHERE D.DeptNo = (SELECT E.DeptNo
FROM EMPLOYEE E
WHERE E.EmpNo = ‘123’) ;
SET CONSTRAINTS PayEqSumsal IMMEDIATE;
```

Jeśli podczas drugiego UPDATE inkrement błędnie zostałby zadeklarowany jako wartość 1000, to wywołanie SET CONSTRAINTS . . . IMMEDIATE spowodowałoby sprawdzenie ograniczeń, i w efekcie, zgłoszenie wyjątku.
Jeśli zamiast SET CONSTRAINTS . . . IMMEDIATE wykonane byłoby COMMIT, to przy niespełnieniu ograniczeń, COMMIT spowodowałoby ROLLBACK.