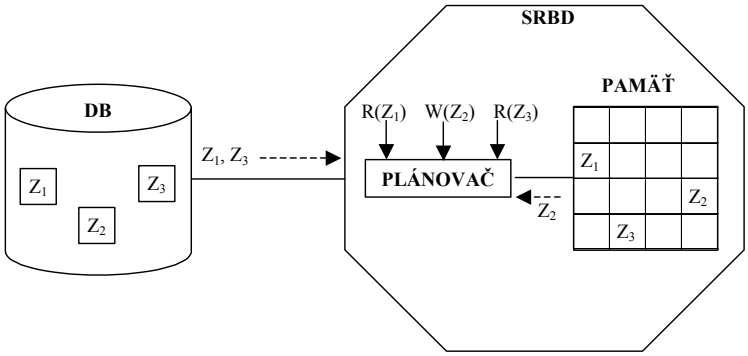
11. Paralelizmus v DBS



Z_i ……záznam, alebo viac záznamov uložených v stránke

Obr 11.1 Architektúra paralelizmu v SRBD

11.1 Rozvrhy

Definícia - Objekt dát (Granula dát)
Objekt dát je jednotka dát, ktorá je individuálne spracovávaná časťou SRBD, ktorá zabezpečuje paralelné spracovanie.

♦

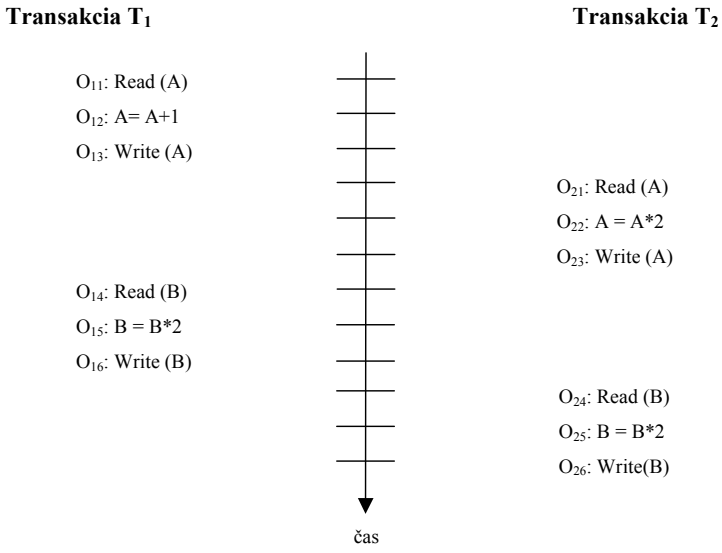
Definícia - Rozvrh
Rozvrh S je plán spracovania množiny transakcií $\{T_1, T_2, ..., T_n\}$, čo je definovaná postupnosť operácií získaných zmiešaním operácií O_{ij} z transakcií $T_1, T_2, ..., T_n$ v poradí umožňujúcom vykonania každej transakcie.

♦

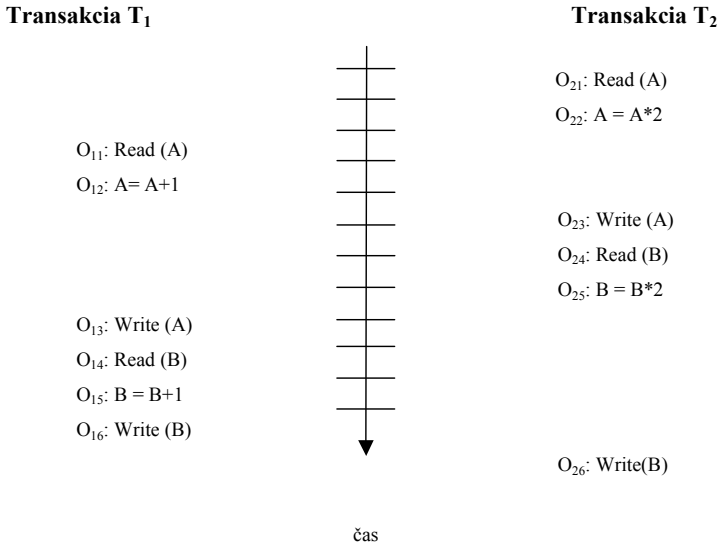
Príklad 11.1
Majme napríklad nasledovné dve transakcie T_1 a T_2:

T_1:	T_2:
O_{11}:Read (A)	O_{21}:Read (A)
O_{12}:A = A + 1	O_{22}:A = A * 2
O_{13}:Write (A)	O_{23}:Write (A)
O_{14}:Read (B)	O_{24}:Read (B)
O_{15}:B = B + 1	O_{25}:B = B * 2
O_{16}:Write (B)	O_{26}:Write (B)

potom vykonanie jednotlivých operácií týchto transakcií môžeme usporiadať do rôznych rozvrhov, ale nie každý rozvrh može byť akceptovateľný:



Obr. 11.3 Rozvrh č. 1 – Vykonanie transakcií je akceptovateľné



Obr 11.4 Rozvrh č. 2 – Vykonanie transakcií je neakceptovateľné, pretože vedie ku strate dát

Definícia – Sériový rozvrh
Rozvrh S nad množinou transakcií $\{T_1, T_2, ..., T_n\}$ je sériovým rozvrhom, ak existuje permutácia π nad {1,2,...,n} takých, že $S=<T\pi_{(1)}, T\pi_{(2)}.., T\pi_{(n)}>$.

♦

Definícia – Serializovateľný rozvrh
Rozvrh **S** nad množinou transakcií **T** je serializovateľný, ak dáva rovnaké výsledky ako sériový rozvrh nad množinou transakcií **T**.

♦

Definícia – Ekvivalencia rozvrhov
Dva rozvrhy S_1 a S_2 nad množinou transakcií **T** sú ekvivalentné ak pre každú dvojicu operácií $\mathbf{O_{ij}}$ a $\mathbf{O_{kl}}$ **(i ≠k)** kedykoľvek platí $\mathbf{O_{ij}} \prec_1 O_{kl}$ potom $\mathbf{O_{ij}} \prec_2 \mathbf{O_{kl}}$.

♦

Definícia – Permutovateľné operácie
Dve operácie $\mathbf{O_{ij}}$ a $\mathbf{O_{kl}}$ sú permutovateľné (zameniteľné) ak ich vykonanie v poradí $\mathbf{O_{ij}} \mathbf{O_{kl}}$ dáva rovnaké výsledky ako poradie vykonania $\mathbf{O_{kl}} \prec \mathbf{O_{ij}}$.

♦

Poznámka

- *Akékoľvek dve operácie typu Read nad tým istým objektom sú permutovateľné.*
- *Akékoľvek dve operácie Read a Write nad tým istým objektom nie sú permutovateľné.*
- *Akékoľvek dve operácie nad rôznymi objektmi sú vždy permutovateľné pretože výsledok prvej nemá vplyv na vykonanie druhej a naopak*

Veta – dostatočná podmienka serializovateľnosti
Dostatočnou podmienkou, aby bol rozvrh serializovateľný je to, aby mohol byť transformovaný pomocou permutovateľných operácií na sériový rozvrh.

♦

Príklad 11.2
Majme nasledovný zjednodušený rozvrh transakcií T_1 a T_2 (predpokladáme, že sú vykonávané operácie Read(A), Write(A), Read(B), Write(B))

T_1: A = A + 1
T_2: A = A * 2
T_1: B = B + 1
T_2: B = B * 2.

Keďže operácie

A = A * 2
a
B = B + 1

sú vymeniteľné, potom môžeme operácie zameniť:

T_1: A = A + 1
T_1: B = B + 1
T_2: A = A * 2
T_2: B = B * 2

Tým sme dostali sériový rozvrh, a teda na základe vety o dostatočnej podmienke serializovateľnosti môžeme povedať, že pôvodný rozvrh je serializovateľný.

Definícia – Precedencia transakcií

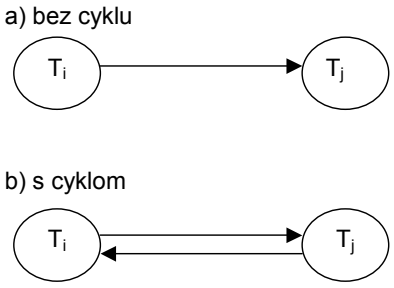
Precedencia transakcií znamená, že transakcia $\mathbf{T_i}$ predchádza transakciu $\mathbf{T_j}$ v rozvrhu $\mathbf{S=\{ T_1,T_2\ldots, T_n\}}$, ak v ňom existujú dve nepermutovateľné operácie $\mathbf{O_{ik}}$ a $\mathbf{O_{jl}}$, kde platí $\mathbf{O_{ik}} \prec \mathbf{O_{jl}}$ teda operácia $\mathbf{O_{ik}}$ sa vykoná v transakcii $\mathbf{T_i}$ pred vykonaním operácie $\mathbf{O_{jl}}$ v transakcii $\mathbf{T_j}$.

♦

Definícia – Graf precedencie transakcií

Graf precedencie transakcií predstavuje orientovaný graf G=(T,V), ktorého množina vrcholov T je množinou transakcií a množina hrán V je množina orientovaných hrán $v_{ij} = [T_i, T_j]$ takých, že transakcia T_i predchádza transakciu T_j a $T_i \in T$, $T_j \in T$.

♦



Obr. 11.5 Ukážka grafu precedencie

Veta – dostatočná podmienka serializovateľnosti 2

Dostatočnou podmienkou serializovateľnosti rozvrhu je, ak v grafe precedencie transakcií neexistuje cyklus.

Veľmi dôležitým a špeciálnym prípadom je prípad, keď nad daným objektom databázy X existujú len dve operácie – Read a Write. Vlastne nám reprezentujú dve atomické operácie v dvoch nezávislých transakciách. Čiže ak máme dve transakcie T_i a T_j a ich dve operácie O_{ik} a O_{jl}, potom máme tri možnosti:

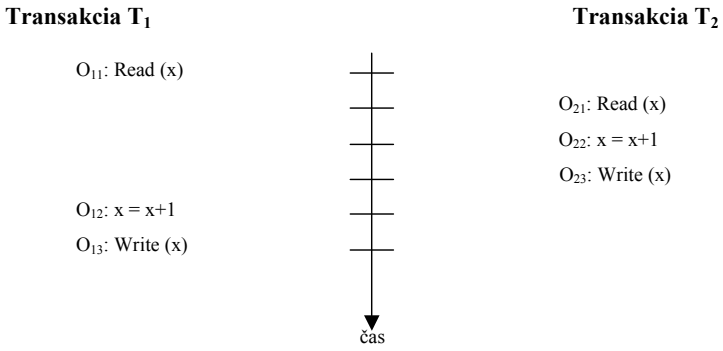
- O_{ik}=Read(X) a operácia O_{ik} = Read(X), potom obe sú permutovateľné operácie, takže T_i môže predchádzať T_j a tak isto T_j môže predchádzať T_j
- O_{ik}=Read(X) a operácia O_{ik} = Write(X), potom sú to nepermutovateľné operácie, takže transakcia T_i musí predchádzať transakciu T_j
- O_{ik}=Write(X) a operácia O_{ik} = Write(X), potom sú to nepermutovateľné operácie, takže transakcia T_i musí predchádzať transakciu T_j

11.2 Problémy súvisiace s paralelizmom

Pri paralelnom spracovaní transakcií je nutné riešiť poradie spracovania jednotlivých operácií pomocou rozvrhov. V prípade, že by sme sa tomuto problému nevenovali, mohlo by dôjsť k nasledovným štyrom typom chýb, a tým aj k poškodeniu databázy.

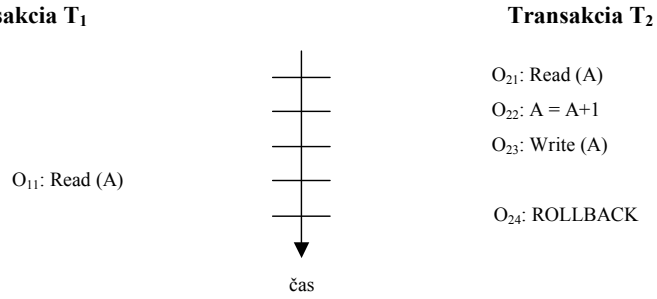
1. problém straty operácií
2. problém nekomitovaných dát
3. prípad nekonzistentných dát
4. fantóm

11.2.1 Problém straty operácií

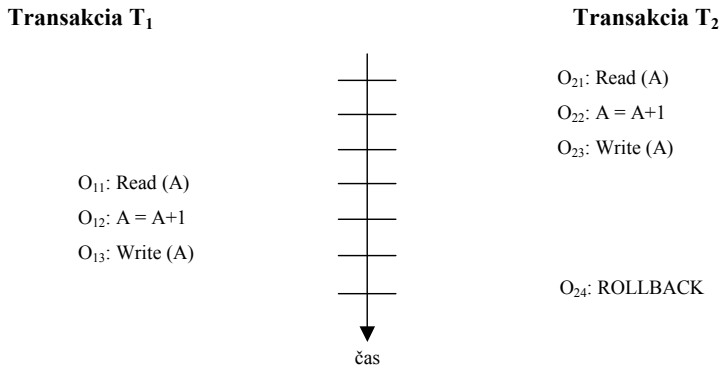


Obr. 11.6 Prípad stratených zmien

11.2.2 Problém nekomitovaných dát

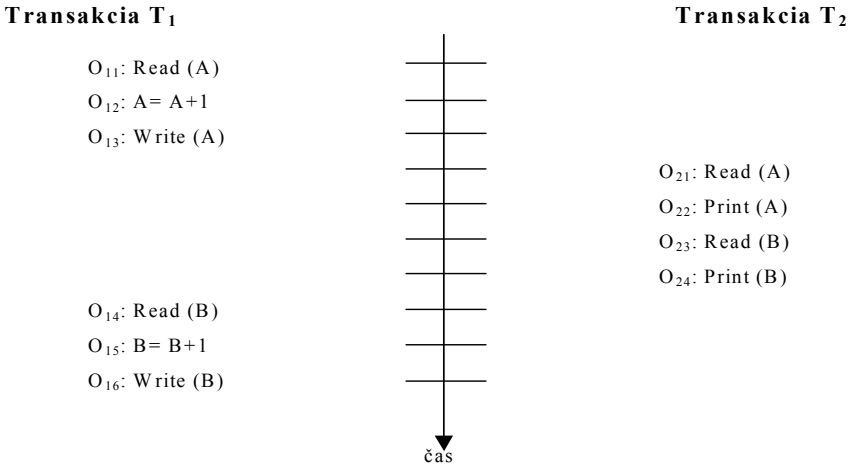


Obr. 11.7 Problém nekomitovaných dát



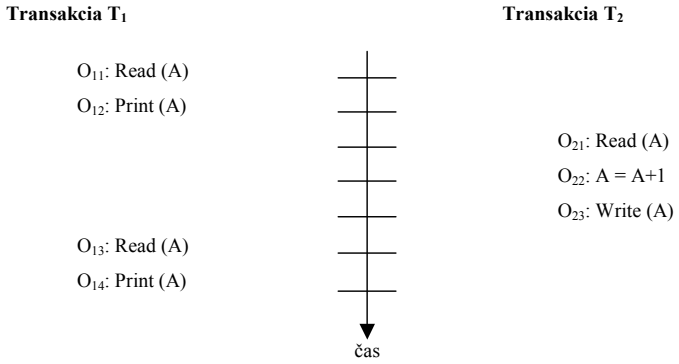
Obr. 11.8 Problém nekomitovaných dát pri neúspešnej transakcii

11.2.3 Problém nekonzistentných dát



Obr. 11.9 Prípad nekonzistentných výstupov

11.2.4 Fantóm



Obr. 11.10 Fantóm

11.3 Techniky paralelného spracovania transakcií

Rozoznávame dva protokoly

- zamykanie
- časové pečiatky

11.3.1 Zamykanie

Pri tejto technike rozpoznávame dva druhy zámkov:

- zdieľaný zámok (S-lock)
- exkluzívny zámok (X-lock).

Zdieľaný zámok (S-LOCK)

Umožňuje prístup k záznamom pre čítanie a zabezpečuje nemennosť záznamov. Je možné, aby niekoľko transakcií súčasne získalo zdieľaný zámok, ale žiadna z nich nemôže získať X-LOCK pokiaľ nie sú všetky S-LOCK uvoľnené.

Exkluzívny zámok (X-LOCK)

Môže ho v danom momente vlastniť len jediná transakcia. Iná transakcia nemôže získať X-LOCK a ani S-LOCK ak už existuje na daný objekt X-LOCK.

Samotné prideľovanie zámkov je riadené tromi základnými operáciami:

- R_LOCK – sa vykonáva vždy pred operáciou Read a vyžaduje pridelenie zdieľaného zámku na dátový objekt.
- W_LOCK – táto operácia predchádza operáciu Write dátového objektu, alebo Read dátového objektu, pri ktorom sa predpokladá následná operácia Write. Pri tejto operácii sa vždy vyžaduje pridelenie exkluzívneho zámku na dátový objekt
- UNLOCK – táto operácia zabezpečí uvoľnenie zámku, ktorý je pridelený dátovému objektu.

Pre transakciu platí, že je dobre definovaná z hľadiska zamykania, ak pre ňu platia nasledovné podmienky:

1. každá operácia Read je predchádzaná operáciou R_LOCK, alebo W_LOCK, za ktorou nasleduje operácia UNLOCK
2. každá operácia Write musí predchádzať operáciu W_LOCK a byť nasledovaná operáciou UNLOCK

	S	X	-
S	Y	N	Y
X	N	N	Y
-	Y	Y	Y

Obr. 11.11 Matica zamykania

V prípade, že pri zamykaní využívame zdieľané, alebo exkluzívne zámky, potom SRBD musí zabezpečiť nasledovné podmienky:

1. Transakcia T pred každou operáciou Read(X) musí spustiť operáciu R_LOCK(X), alebo W_LOCK(X)
2. Pred každou operáciou Write(X) v transakcii T je nutné spustiť operáciu W_LOCK(X)
3. Transakcia T musí zabezpečiť spustenie operácie UNLOCK(X) po vykonaní všetkých operácií Read(X) a Write(X)
4. Transakcia T nesmie pripustiť vykonanie operácie R_LOCK(X) v prípade, že na dátový objekt X je pridelený exkluzívny zámok inou transakciou
5. Transakcia T nesmie dovoliť vykonanie operácie W_LOCK(X), ak na objekt je daný zdieľaný, alebo exkluzívny zámok inou transakciou
6. Transakcia T nemôže vykonať operáciu UNLOCK(X) na objekty X, ktoré sú zamknuté inou transakciou

V prípade, že pri zamykaní využívame len binárne zámky, potom SRBD používa len dve operácie – LOCK a UNLOCK a je potrebné zabezpečiť nasledovné podmienky:

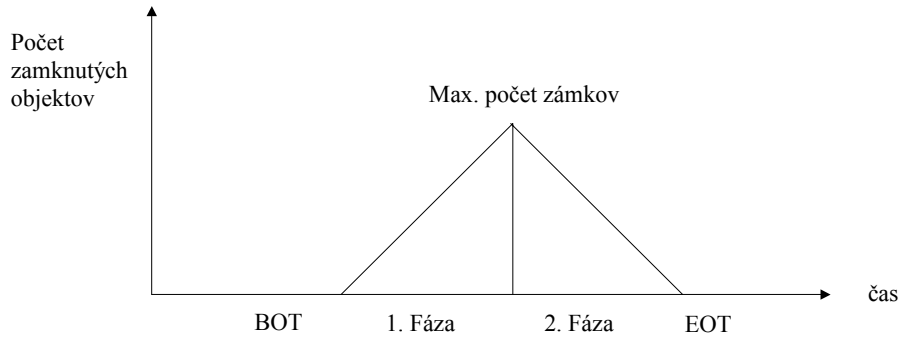
1. Transakcia T pred každou operáciou Read(X), alebo Write(X) musí spustiť operáciu LOCK(X)
2. Transakcia T musí zabezpečiť spustenie operácie UNLOCK(X) po vykonaní všetkých operácií Read(X) a Write(X)
3. Transakcia T nesmie pripustiť vykonanie operácie LOCK(X) v prípade, že na dátový objekt X je pridelený zámok inou transakciou
4. Transakcia T nemôže vykonať operáciu UNLOCK(X) na objekty X, ktoré sú zamknuté inou transakciou

11.3.1.1 Dvojfázové zamykanie

Definícia - Dvojfázový uzamykací protokol :
V transakcii T všetky zamykacie operácie R_LOCK, W_LOCK predchádzajú prvú operáciu UNLOCK.

♦

Veta
Ak sú všetky transakcie zabezpečené dvojfázovým uzamykacím protokolom, tak všetky možné rozvrhy sú sérializovateľné.



Obr. 11.12 Dvojfázový zamykací protokol

V praxi sa môžeme stretnúť s viacerými variantmi dvojfázového zamykania. Spomenutá technika je známa aj ako **základné dvojfázové zamykanie.** Variáciou je tzv. **konzervatívne dvojfázové zamykanie**, ktoré vyžaduje pridelenie zámkov na všetky objekty, s ktorými transakcia pracuje, na začiatku vykonania transakcie. Tieto objekty získame pri vytváraní základných množín používaných objektov (ReadSet a WriteSet). Ďalším variantom, v praxi veľmi často používaným, je tzv. **striktné dvojfázové zamykanie**, ktoré zabezpečí, že zámky budú uvoľnené, až pri vykonávaní operácie COMMIT, čiže žiadne zámky nebudú uvoľňované skôr, ako transakcia nebude potvrdená. Ďalšou, ešte reštriktívnejšou metódou je tzv. **rigorózne dvojfázové zamykanie**, ktoré uvoľní všetky zámky až po ukončení transakcie.

Nevýhodou používania techniky zamykania je situácia, ktorá spôsobí **uviaznutie** systému (DEADLOCK).V ďalšom texte sa budeme zaoberať riešením tohto problému.

11.3.2 Časové pečiatky

Definícia – Časová pečiatka transakcie
Časová pečiatka transakcie je numerická hodnota priradená transakcii, umožňujúca usporiadať bežiace transakcie.

♦

V prípade, že máme dve transakcie T_i , T_j, a i<j, potom hovoríme, že transakcia T_i je staršia transakcia a T_j je mladšia transakcia.

Definícia – Časová pečiatka objektu
Časová pečiatka objektu je numerická hodnota priradená objektu DB, vyjadrujúca číslo transakcie, ktorá s ňou naposledy manipulovala.

♦

11.3.2.1 Časová pečiatka objektu

Tieto môžu byť:

- Univerzálna časová pečiatka – objektu je pridelená hodnota transakcie, ktorá posledná spracovávala daný objekt bez ohľadu na to, či to bola operácia Read, alebo Write
- Časová pečiatka Read – objektu je pridelená hodnota transakcie, ktorá posledná čítala daný objekt
- Časová pečiatka Write – objektu je pridelená hodnota transakcie, ktorá posledná zapisovala daný objekt

11.3.2.2 Úplné pečiatkovanie

```
Procedure READ (Ti: transakcia, X: objekt_DB);
      IF TimeStamp_Object(X) ≤ i THEN
            BEGIN
                  Vykonaj Read(X);
                  TimeStamp_Object(X):=i;
            END
            ELSE
                  ABORT;
      END IF;
End READ;

Procedure WRITE (Ti: transakcia, X: objekt_DB);
      IF TimeStamp_Object(X) ≤ i THEN
            BEGIN
                  Vykonaj Write(X);
                  TimeStamp_Object(X):=i;
            END
            ELSE
                  ABORT;
      END IF;
End WRITE;
```

11.3.2.3 Čiastočné pečiatkovanie

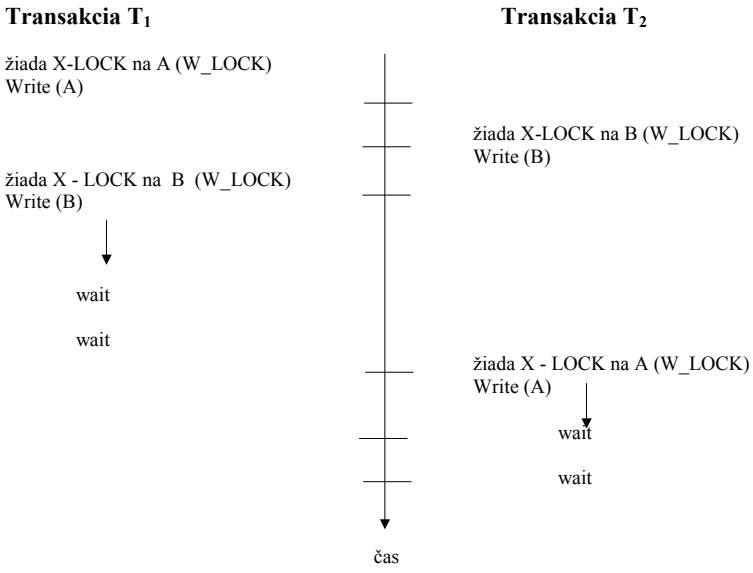
```
Procedure READ (Ti: transakcia, X: objekt_DB);
      IF WriteTimeStamp_Object(X) ≤ i THEN
            BEGIN
                  Vykonaj Read(X);
                  ReadTimeStamp_Object(X):=MAX(ReadTimeStamp_Object(X),i);
            END
      ELSE
            ABORT;
      END IF;
End READ;

Procedure WRITE (Ti: transakcia, X: objekt_DB);
      IF (WriteTimeStamp_Object(X) ≤ i ) AND
         (ReadTimeStamp_Object(X) ≤ i) THEN
            BEGIN
                  Vykonaj Write(X);
                  WriteTimeStamp_Object(X):=i;
            END
      ELSE
            ABORT;
      END IF;
End WRITE;
```

Príklad 11.3

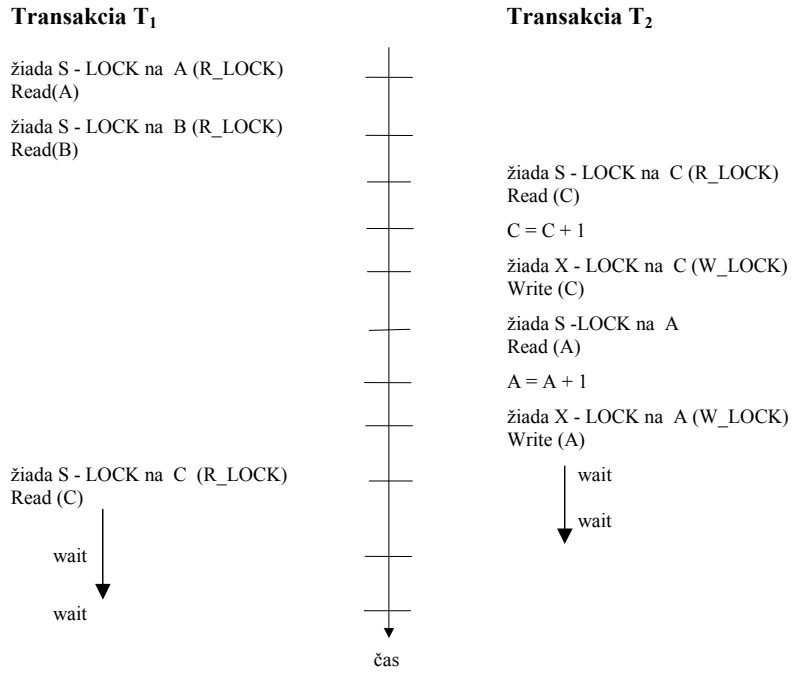
Majme objekt DB zmenený transakciu T_2 s časovou pečiatkou
WriteTimeStamp_Object =2.
Ak bude tento objekt čítaný mladšou transakciu napr. T_7 tak časová pečiatka sa zmení nasledovne –
ReadTimeStamp_Object =7.

Z toho vyplýva, že akákoľvek transakcia mladšia ako T_2 môže tento objekt čítať.
V prípade zmeny objektu v DB sa kontroluje, kedy bola vykonaná posledná operácia Read aj Write nad objektom. Oprava je teda možná len ak časová pečiatka transakcie je väčšia (transakcia je mladšia) ako časové pečiatky Read a Write tohto DB objektu. Z toho vyplýva, že ak WriteTimeStamp_Object =2 a ReadTimeStamp_Object =7, tak len transakcia s časovou pečiatkou väčšou, alebo rovnou 7 môže zmeniť tento objekt.

11.4 Uviaznutie (DEADLOCK)



Obr. 11.13 Príklad uviaznutia



Obr. 11.14 Príklad uviaznutia

Definícia - Uviaznutie

Uviaznutie je situácia, pri ktorej každá transakcia zo skupiny transakcií spĺňa nasledovné podmienky:

- Každá transakcia zo skupiny transakcií je blokovaná čakaním na objekt DB
- Ukončenie každej z transakcií, ktorá nepatrí do skupiny blokovaných neumožní odblokovanie žiadnej z nich

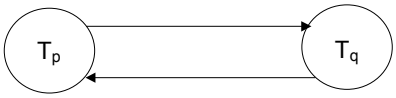
♦

11.4.1 Detekcia uviaznutia

V praktických aplikáciách je výhodné monitorovať priebeh transakcií a zamykania tak, aby bolo možné analyzovať uviaznutie. S výhodou je možné využívať teóriu grafov a pomocou nej konštruovať tzv. **čakacie grafy**.

Definícia – Čakací graf
Čakací graf je graf G=(T,W), kde T je množina vrcholov reprezentovaná paralelne bežiacimi transakciami { T_1, T_2, …,T_n} zdieľajúcimi objekty DB { OBJ_1, OBJ_2, …,OBJ_m} a W je množina orientovaných hrán vyjadrujúcich čakanie takých, že orientovaná hrana $v_{pq} = [T_p, T_q]$ znamená, že transakcia T_p čaká na T_q , ak T_p žiada zamknutie objektu OBJ_i, ktorý je zamknutý transakciou T_q ♦

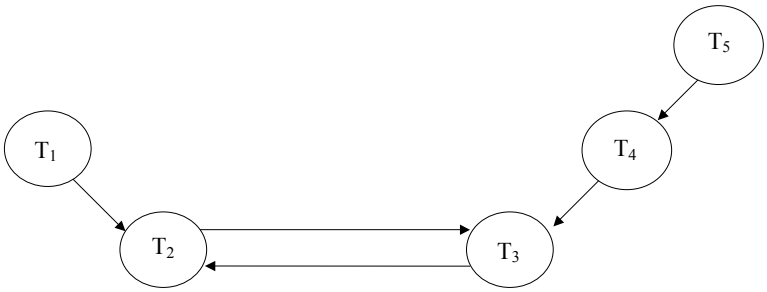
Obr. 11.15 Čakací graf bez cyklu



Obr. 11.16 Čakací graf s cyklom

Na obrázku 11.15 máme zobrazenú situáciu, kedy transakcia T_p čaká na uvoľnenie objektu, ktorý je zamknutý transakciou T_q. Obrázok 11.16 zobrazuje situáciu, kedy transakcia T_p čaká na uvoľnenie objektu, ktorý je zamknutý transakciou T_q , zároveň aj transakcia T_q čaká na uvoľnenie nejakého objektu, ktorý je zamknutý transakciou T_p.

V prípade, že jedna z transakcií uvoľní objekt v systéme, ktorý vytvára čakací graf, uvoľnenie objektu spôsobí odstránenie hrany z čakacieho grafu.



Obr. 11.17 Príklad zložitejšieho čakacieho grafu

Veta
Uviaznutie existuje vtedy, ak sa v čakacom grafe vyskytuje cyklus.

11.4.2 Predchádzanie uviaznutiu

11.4.2.1 Metóda WAIT- DIE

Algoritmus metódy:

```
IF TimeStamp(Ti) < TimeStamp(Tj ) THEN
      BEGIN
            Ti WAITS;
      END
      ELSE BEGIN
            Ti DIES;
           END;
END IF;
```

11.4.2.2 Metóda WOUND – WAIT

Algoritmus metódy:

```
IF TimeStamp(Ti) < TimeStamp(Tj ) THEN
      BEGIN
            Tj IS WOUNDED;
      END
      ELSE BEGIN
            Ti WAITS;
            END;
END IF;
```