INF / UFG

Disciplina

Banco de Dados

Conteúdo

Controle de Concorrência:

Timestamp

Preâmbulo

O **Controle de Concorrência Baseado em *Timestamp*** é uma outra abordagem que garante serialização (em adição protocolo 2PL).

A abordagem utiliza *timestamps* de transações para ordenar a execução de transações, em relação a um escalonamento serial equivalente.

Timestamp
É um identificador único criado pelo SGBD, para identificar cada transação.
Valores de *timestamp* denotam a ordem em que transações são submetidas:

Exemplo 1: contador mantido pelo sistema (pode ser zerado);
Exemplo 2: data e hora de início de cada transação.

Notação: **TS(T)** é o *timestamp* da Transação **T**.

Técnicas de controle de concorrência baseado em *timestamp* não usam bloqueios:

>> portanto, *deadlocks* **não** podem ocorrer.

Preâmbulo

Premissa.
A ordem em que itens de banco de dados são acessados pelas transações baseia-se na ordem de *timestamp* das transações.

Cada item de banco de dados possui dois valores de *timestamp*:

read_TS(X)
É o maior *timestamp* em relação a todos os *timestamps* das transações que leram **X** com sucesso:

>> **read_TS(X) = TS(T)**, onde T é a transação mais jovem a ler X com sucesso.

write_TS(X)
É o maior *timestamp* em relação a todos os *timestamps* das transações que escreveram **X** com sucesso:

>> **write_TS(X) = TS(T)**, onde T é a transação mais jovem a escrever X com sucesso.

Ordenação de Transações Baseada em *Timestamp*

Algoritmo básico – *Timestamp Ordering* (TO)

Sempre que uma transação **T** tentar emitir uma operação **read_item(X)** ou **write_item(X)**, o algoritmo compara o *timestamp* de **T** com **read_TS(X)** e **write_TS(X)**, para garantir que a ordem de *timestamps* não é violada.

A ordem de acesso ao item de dado *X* é a ordem de *timestamps* das transações que tentam acessar *X*.

Ordenação de Transações Baseada em *Timestamp*

Algoritmo básico – *Timestamp Ordering* (TO)

Sempre que uma transação **T** emitir uma operação **write_item(X)**:

>> checa se X foi lido ou escrito por uma transação mais jovem que T

```
se read_TS(X) > TS(T) ou write_TS(X) > TS(T) então

    abort e rollback T
    reinicia T com um novo timestamp

senão

    a Transação T executa write_item(X)
    write_TS(X) = TS(T)

endif
```


Ordenação de Transações Baseada em *Timestamp*

Algoritmo básico – *Timestamp Ordering* (TO)

Sempre que uma transação **T** emitir uma operação **read_item(X)**:

>> checa se X foi escrito por uma transação mais jovem que T

```
se write_TS(X) > TS(T) então

    abort e rollback T
    reinicia T com um novo timestamp

senão

    a Transação T executa read_item(X)
    read_TS(X) = max ( TS(T) , read_item(X) )

endif
```


Ordenação de Transações Baseada em *Timestamp*

Algoritmo básico – *Timestamp Ordering* (TO)

Se T for abortada e desfeita (*rolled back*):

>> qualquer transação T* que tenha usado um valor escrito por T deve também ser desfeita (*rolled back*);
>> qualquer transação T** que tenha usado um valor escrito por T* deve também ser desfeita (*rolled back*);
>> ocorre, então, *rollback* em cascata.

E se alguma T* ou T já tiver sido confirmada (*committed*) ?**

R – escalonamento não recuperável.

Apesar de não ocorrer *deadlock*, TO permite reinício cíclico (*starvation*), se uma transação for abortada e reiniciada.

Ordenação de Transações Baseada em *Timestamp*

Algoritmo modificado – *Strict Timestamp Ordering* (STO)

STO garante que os escalonamentos sejam estritos (para recuperação) e serializáveis.

Seja uma Transação **T** que:

executa uma operação **O** (seja ***read_item(X)*** ou ***write_item(X)***), e **TS(T) > write_TS(X)**

A operação **O** deverá ser atrasada até que a Transação **T*** que escreveu **X** (ou seja, **TS(T*) = write_TS(X)**) tenha sido confirmada ou abortada.

Para tal, STO precisa simular o bloqueio de um item **X** que tenha sido escrito por **T***, até que **T*** tenha sido confirmada ou abortada.

>> o algoritmo não causa *deadlock*, visto que
T espera por T* somente se TS(T) > TS(T*).