Sistem Basis Data

MANAJEMEN TRANSAKSI 2

Alif Finandhita, S.Kom

Eksekusi Konkurensi

- Pada eksekusi konkurensi, banyak transaksi dimungkinkan untuk diproses secara bersama – sama dalam suatu sistem.
- Memperbolehkan banyak transaksi untuk mengupdate data secara konkuren akan menyebabkan beberapa komplikasi dengan konsistensi data.
- Untuk memastikan bahwa konsistensi tetap terjaga dengan baik pada eksekusi konkuren, membutuhkan usaha yang lebih kuat. Akan lebih mudah jika transaksi berjalan secara serial.

Eksekusi Konkurensi (2)

- Keuntungan konkurensi :
  - Meningkatkan utilitas disk dan prosesor
  - Mengurangi rata – rata waktu respon transaksi
- Dalam konkurensi dikenal dengan konsep penjadwalan *(schedule)* yang akan membantu mengidentifikasi eksekusi agar konsistensi datanya tetap terjaga.
- Sistem Basis Data harus mengontrol interaksi diantara transaksi konkuren supaya transaksi – transaksi tersebut tidak menghancurkan konsistensi basis datanya (data menjadi tidak konsisten).

Eksekusi Konkurensi - Schedule

- Penjadwalan / Schedule merupakan urutan yang mengindikasikan urutan kronologis instruksi yang mana dari transaksi konkuren yang akan dieksekusi.
- Sebuah jadwal merupakan bagian dari suatu transaksi yang harus terdiri dari semua instruksi yang ada di dalamnya.
- Sebuah jadwal harus menjaga urutan instruksi yang muncul di setiap transaksi.

Eksekusi Konkurensi – Schedule (2)

- Misal T_1 adalah transaksi transfer \$50 dari A ke B, dan T_2 adalah transfer 10% dari jumlah rekening A ke B. Penjadwalan berikut adalah pada transaksi serial, dimana T_1 dulu yang diselesaikan baru diikuti oleh T_2

T_1	T_2
read(A)	
$A := A - 50$	
write (A)	
read(B)	
$B := B + 50$	
write(B)	
	read(A)
	$temp := A * 0.1$
	$A := A - temp$
	write(A)
	read(B)
	$B := B + temp$
	write(B)

Penjadwalan 1 di text book

Eksekusi Konkurensi – Schedule (3)

- Contoh penjadwalan selanjutnya, dengan transaksi yang sama, tetapi dalam bentuk transaksi konkurensi, tapi ekivalen dengan penjadwalan sebelumnya.

T_1	T_2
read(A)	
$A := A - 50$	
write(A)	
	read(A)
	$temp := A * 0.1$
	$A := A - temp$
	write(A)
read(B)	
$B := B + 50$	
write(B)	
	read(B)
	$B := B + temp$
	write(B)

Penjadwalan 3 pada Text book

Eksekusi Konkurensi – Schedule (3)

- Pada contoh penjadwalan yang pertama dan yang kedua, jumlah rekening A + B sebelum dan sesudah transaksi sama, tapi tidak dengan contoh penjawalan konkurensi berikut :

T_1	T_2
read(A)	
$A := A - 50$	
	read(A)
	$temp := A * 0.1$
	$A := A - temp$
	write(A)
	read(B)
write(A)	
read(B)	
$B := B + 50$	
write(B)	
	$B := B + temp$
	write(B)

Penjadwalan 4 pada text book

Serializability

- Sistem basis data harus dapat mengontrol eksekusi konkurensi dari suatu transaksi untuk memastikan database tetap terjaga konsistensinya.
- Asumsi dasar, bahwa setiap transaksi harus tetap menjaga konsistensi database.
- Eksekusi secara serial pada suatu transaksi dapat menjaga database tetap konsisten.
- Sebuah jadwal (yang mungkin konkuren), serializable jika setara dengan penjadwalan secara serial. Pada bagian ini ada dua bentuk ekivalensi penjadwalan *(Schedule Equivalence) : Conflict* dan *View Serializability.*
- Pada serializability kita abaikan operasi selain dari instruksi **read** dan **write**, serta diasumsikan bahwa transaksi dapat melakukan perhitungan terhadap data di dalam buffer lokal diantara instruksi **read** dan **write**.

Conflict Serializability

- Instruksi I_i dan I_j masing – masing dari transaksi T_i dan T_j , konflik jika dan hanya jika ada beberapa item data yang sama (Q) diakses secara bersama – sama baik oleh I_i dan I_j , serta setidaknya salah satu dari instruksi melakukan operasi *write (Q)*.
  1. Jika I_i = **read(**Q**)** dan I_j = **read(**Q**)**, maka I_i dan I_j tidak konflik
  2. Jika I_i = **read(**Q**)** dan I_j = **write(**Q**)**, maka I_i dan I_j konflik
  3. Jika I_i = **write(**Q**)** dan I_j = **read(**Q**)**, maka I_i dan I_j konflik
  4. Jika I_i = **write(**Q**)** dan I_j = **write(**Q**)**, maka I_i dan I_j konflik

- Secara intuitif, konflik diantara I_i dan I_j memaksakan perintah logika temporal diantara keduanya.

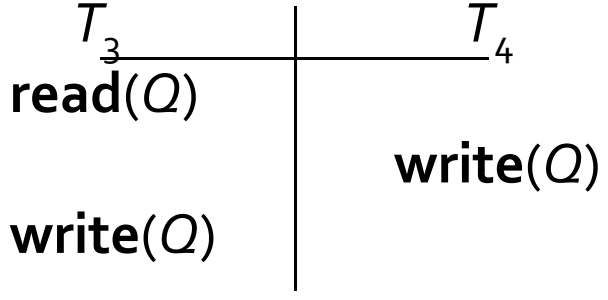
- Jika I_i dan I_j merupakan instruksi dari transaksi berbeda dan tidak konflik, maka urutan instruksi I_i dan I_j dapat ditukar sehingga menghasilkan jadwal baru dengan hasil yang tetap sama.

Conflict Serializability (2)

- Jika jadwal S dapat ditransformasikan menjadi jadwalan S'oleh serangkaian pertukaran instruksi yang non-konflik, maka bisa dikatakan S dan S' adalah **conflict equivalent**.

- Bisa dikatakan bahwa jadwal *S* adalah **conflict serializable** jika conflict equivalent terhadap penjadwalan serial.

- Contoh penjadwalan yang tidak **conflict serializable**



Conflict Serializability (3)

- Penjadwalan 3 di bawah bisa ditransformasikan ke penjadwalan 1, penjadwalan serial dimana T_2 diikuti T_1, dengan serangkaian penukaran instruksi yang tidak konflik. Oleh karena itu penjadwalan di bawah ini **conflict serializability**

T_1	T_2
read(A)	
write(A)	
	read(A)
	write(A)
read(B)	
write(B)	
	read(B)
	write(B)

View Serializability

- Misalkan *S* dan *S'* adalah dua jadwal dengan serangkaian transaksi yang sama. *S* dan *S'* adalah **view equivalent**.

- *S* dan *S'* adalah **view equivalent** jika memenuhi kondisi sebagai berikut :
  1. Untuk masing – masing data item *Q* , jika transaksi T_i membaca inisialisasi nilai pada *Q* di dalam jadwal *S*, maka transaksi T_i yang ada di jadwal *S'* juga harus membaca inisialisasi nilai pada *Q.*
  2. Untuk masing – masing data item *Q*, jika transaksi T_i mengeksekusi *read(Q)* di dalam jadwal *S*, dan jika nilai yang dihasilkan oleh operasi *write(Q)* dieksekusi oleh transaksi T_j, maka operasi *read(Q)* pada transaksi T_i yang harus ada di jadwal *S'* juga membaca nilai Q yang dihasilkan oleh operasi *write(Q)* yang sama pada transaksi T_j
  3. Untuk masing – masing data item *Q*, transaksi yang sampai ke tahap akhir operasi *write(Q)* di dalam jadwal *S*

- Seperti yang terlihat, view serializability juga dilihat berdasarkan pada operasi read dan write saja

View Serializability (2)

- Sebuah jadwal S adalah **view seralizable ,** setara dengan penjadwalan serial.
- Setiap penjadwalan **conflict serializable** juga **view serializable**
- Penjadwalan di bawah ini (penjadwalan 9 di text book) adalah penjadwalan yang view serializable tapi tidak conflict serializable

T_3	T_4	T_6
read(Q)		
	write(Q)	
write(Q)		
		write(Q)

View Serializability (3)

- Penjadwalan di bawah (penjadwalan 9 di text book) menghasilkan outcome yang sama sebagai jadwal serial (T_1, T_2)

T_1	T_5
read(A)	
$A := A - 50$	
write(A)	
	read(B)
	$B := B - 10$
	write(B)
read(B)	
$B := B + 50$	
write(B)	
	read(A)
	$A := A + 10$
	write(A)

Recoverability

- **Recoverable Schedule** – Jika transaksi T_j membaca sebuah item data yang sebelumnya ditulis oleh transaksi T_i , maka operasi commit T_i muncul sebelum operasi commit T_j .
- Jadwal berikut (jadwal 11 di text book) tidak recoverable jika T_9 commit segera setelah read

T_8	T_9
read(A)	
write(A)	
	read(A)
read(B)	

- Jika T_8 dibatalkan, maka T_9 bisa jadi akan membaca state database inkonsisten. Maka dengan demikian database harus memastikan bahwa jadwal dapat dipulihkan jika terjadi sesuatu hal.

Recoverability (2)

- **Cascading Rollback –** kesalahan pada satu transaksi yang yang dapat berpengaruh pada serangkaian transaksi lainnya sehingga keseluruhan transaksi akan di-rollback.

- Misalkan pada penjadwalan berikut dimana belum ada transaksi yang di-commit, sehingga jika terjadi masalah bisa dipulihkan

T_{10}	T_{11}	T_{12}
read(A)		
read(B)		
write(A)		
	read(A)	
	write(A)	
		read(A)

- Jika T_{10} transaksinya fail, maka T_{11} dan T_{12} juga harus di rollback. Jika tidak demikian, maka akan menyebabkan banyaknya pekerjaan yang tidak akan terselesaikan.

Recoverability (3)

- **Cascadeless Schedule** – cascading rollback tidak dapat terjadi, untuk setiap pasang transaksi T_i dan T_j dimana T_j membaca sebuah item data yang sebelumnya ditulis oleh T_i dan operasi commit pada T_i muncul sebelum operasi read pada T_j.

- Setiap cascadeless schedule juga dapat dipulihkan (recoverable);

Implementasi Isolasi

- Jadwal harus conflict atau view serializable, dan recoverable, demi terjaganya konsistensi database dan juga sebaiknya cascadeless.
- Sebuah kebijakan dimana hanya satu transaksi yang dapat dieksekusi dalam satu waktu yang menghasilkan penjadwalan serial, tapi minim konkurensi.
- Beberapa skema hanya mengijinkan jadwal conflict-serializable untuk di-generate, dimana yang skema lainnya ada yang memungkinkan jadwal view-serializable yang tidak conflict-serializable.

Definisi Transaksi di SQL

- DML (Data Manipulation Language) harus menyertakan sebuah konstruksi untuk menspesifikasikan serangkaian aksi yang meliputi suatu transaksi.
- Di dalam SQL, transaksi dimulai secara implisit.
- Transaksi di SQL diakhiri oleh salah satu dari statement sebagai berikut :
  - **commit work -** commit transaksi yang sedang berlangsung dan memulai transaksi yang baru
  - **rollback work –** pembatalan terhadap transaksi yang sedang berlangsung

Level Konsistensi di SQL 92

- **Serializable –** default
- **Repeatable read –** hanya record yang dicommit yang dibaca, pembacaan secara berulang untuk record yang sama harus dapat menghasilkan nilai yang sama. Bagaimanapun, transaksi mungkin tidak serializable - bisa menemukan beberapa record yang ditambahkan oleh suatu transaksi tapi tidak menemukan record lainnya.
- **Read commited –** hanya record yang dicommit yang akan dibaca , tetap commit untuk pembacaan record secara berulang yang menghasilkan nilai yang berbeda.
- **Read uncommited –** record yang tidak dicommit dibaca

Testing Serializability

- Pada saat mendesain skema kontrol konkurensi, kita harus tunjukan bahwa jadwal yang dibuat oleh skema tersebut adalah serializable.
- Terdapat metode simpel dan efisien untuk menentukan conflict serializability dari suatu jadwal.
- Misalkan sebuah jadwal S. Kita dapat membuat suatu grafik langsung yang diberi nama grafik preseden (**presedence graph**).

Testing Serializability (2)

- Grafik preseden terdiri dari sepasang G = (V,E), dimana V adalah serangkaian simpul dan E adalah serangkaian tepian / busur.

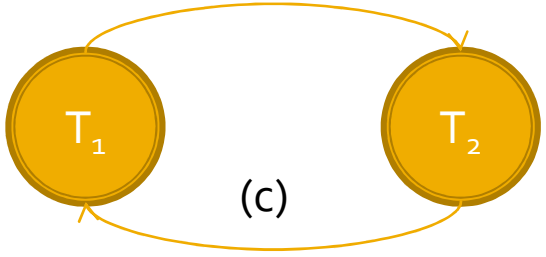
- Serangkaian simpul terdiri dari semua transaksi yang berperan serta di dalam penjadwalan. Serangkaian tepian / busur terdiri dari semua bentuk T_i -> T_j untuk masing – masing dari ketiga kondisi berikut :
  1. T_i eksekusi write(Q) sebelum T_j eksekusi read(Q)
  2. T_i eksekusi read(Q) sebelum T_j eksekusi write(Q)
  3. T_i eksekusi write(Q) sebelum T_j eksekusi write(Q)

- Jika bentuk T_i -> T_j ada di dalam grafik preseden, maka di setiap jadwal S' serial yang ekivalen ke jadwal S, T_i harus muncul sebelum T_j

Testing Serializability (3)

- Sebagai contoh, grafik preseden untuk penjadwalan 1 digambarkan di (a), terdiri dari bentuk dasar T_1 -> T_2 , dimana semua instruksi T_1 dieksekusi sebelum instruksi pertama pada T_2 dieksekusi. Begitu juga sebaliknya untuk contoh yang digambarkan di (b).

Testing Serializability (3)



- Grafik preseden untuk jadwal 4, terdiri dari bentuk T_1 -> T_2 , karena T_1 mengeksekusi read(A) sebelum T_2 mengeksekusi write(A). Grafik ini jg terdiri dari bentuk T_2 -> T_1 , karena T_2 mengeksekusi read(B) sebelum T_1 eksekusi write(B).

- Jika grafik preseden untuk S membentuk suatu lingkaran, maka jadwal S tidak conflict serializable. Jika sebaliknya, maka jadwal S conflict serializable