

Transaksi

Budi Susanto

Teknik Informatika
Universitas Kristen Duta Wacana
Yogyakarta

Tujuan

- Memahami tentang konsep transaksi database.
- Memahami konsep serialisasi terhadap isolasi.

Konsep Transaksi

- Sebuah transaksi adalah unit eksekusi program yang mengakses/mengupdate berbagai item data.
 - Kumpulan operasi-operasi membentuk sebuah unit logika kerja.
 - Sistem database harus memastikan setiap eksekusi suatu transaksi walaupun terjadi kegagalan.
 - Oleh karena bisa terjadi banyak transaksi yang dieksekusi bersamaan, maka harus ada manajemen transaksi untuk mencegah inkonsistensi.

Konsep transaksi

- Suatu transaksi biasanya diawal dengan begin transaction; dan diakhiri dengan end transaction;.
- Semua perintah di dalamnya harus diperlakukan sebagai sebuah unit tunggal.
- Jika terjadi kegagalan terhadap satu perintah dalam transaksi, maka satu transaksi tersebut akan dibatalkan.

Konsep Transaksi

- Setiap sistem database harus memelihara properti transaksi berikut:
 - **Atomicity**
 - Menyatakan apakah semua operasi transaksi berhasil semua atau gagal sama sekali.
 - **Isolation**
 - Memastikan bahwa transaksi beroperasi secara benar tanpa adanya gangguan dari eksekusi bersamaan perintah database.
 - **Durability**
 - Suatu transaksi harus dapat bertahan terhadap kegagalan.
 - **Consistency**
 - Eksekusi suatu transaksi dalam isolasi, akan memelihara konsistensi database. Setelah berhasil, maka perubahan harus tersimpan secara tetap, walaupun terjadi kegagalan sistem.

Contoh

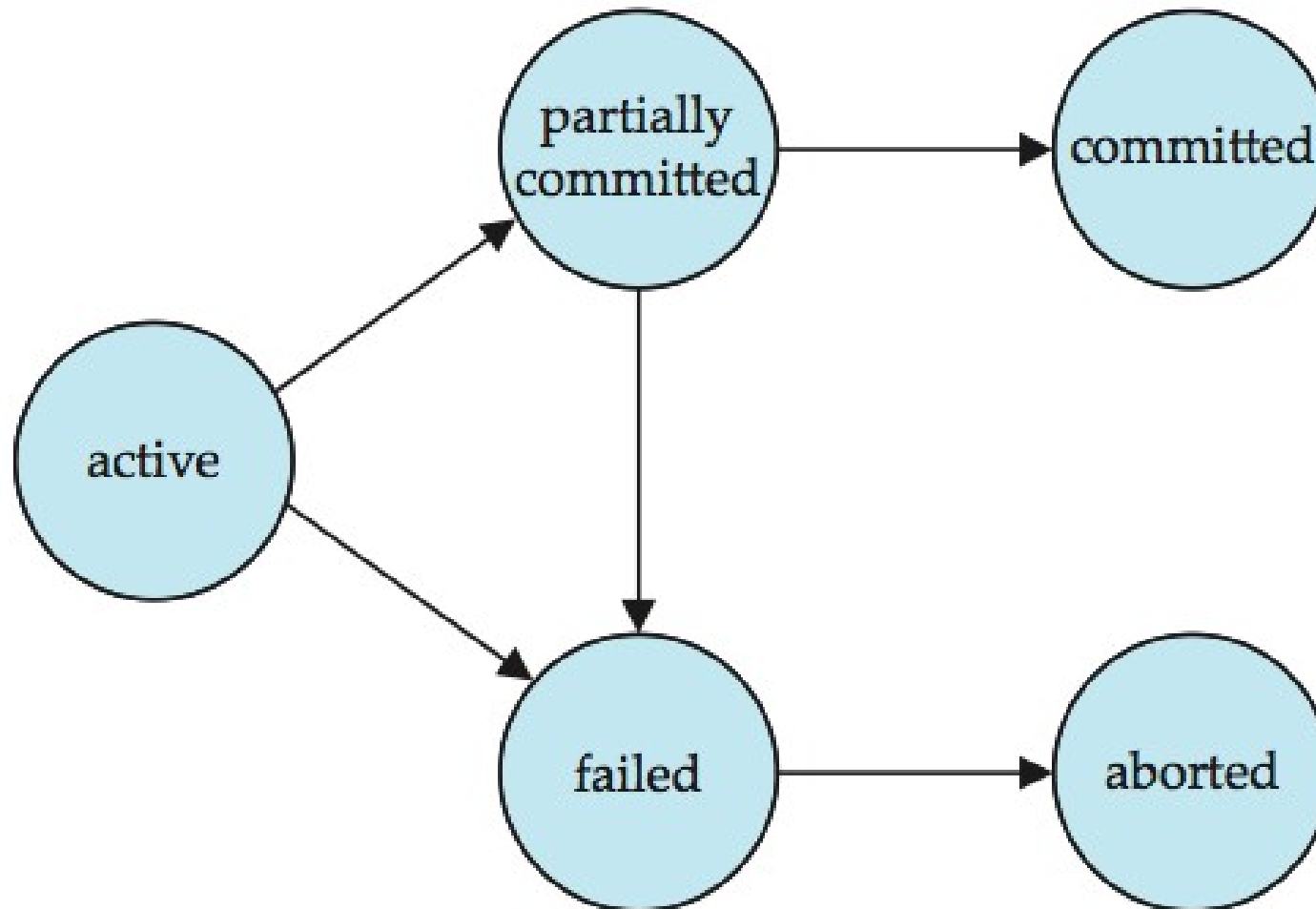
- T_i adalah transaksi transfer USD 500 dari account A ke account B.

```
 $T_i$ : read(A);  
      A := A - 500;  
      write(A);  
      read(B);  
      B := B + 500;  
      write(B).
```

Struktur Penyimpanan

- Volatile storage
 - Main memory dan cache memory
- Nonvolatile storage
 - Secodary storage device
- Stable storage
 - Replication of nonvolatile storage

State Diagram Transaction



Transaction Isolation

- Sangat mudah dan aman jika dapat menjalankan beberapa transaksi yang berjalan bersamaan secara serial.
- Tapi dengan adanya eksekusi bersamaan, akan ada beberapa keuntungan:
 - Improved throughput and resource utilization.
 - Reduced waiting time.

Transaction Isolation

- Sistem database harus mengontrol interaksi di antara transaksi untuk mencegah mereka menghancurkan konsistensi database.
 - Mekanismenya disebut *concurrency-control schemes*.
- Dalam eksekusi satu atau lebih transaksi secara bersamaan, urutan eksekusi transaksi disebut sebagai **penjadwalan (*schedule*)**.

Transaction Isolation

- T1 adalah transaksi transfer USD 500 dari A ke B.
- T2 adalah transaksi transfer 10% saldo A ke account B.

```
T1: read(A);  
      A := A - 500;  
      write(A);  
      read(B);  
      B := B + 500;  
      write(B).
```

```
T2: read(A);  
      temp := A * 0.1;  
      A := A - temp;  
      write(A);  
      read(B);  
      B := B + temp;  
      write(B).
```

Contoh Penjadwalan Serial

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

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

Concurrency Control

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) commit	read(B) $B := B + temp$ write(B) commit

Consistent

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) commit	$B := B + temp$ write(B) commit

Inconsistent

Concurrency Control

- Agar dalam penjadwalan concurrent transaction setara dengan penjadwalan yang serial, maka perlu adanya mekanisme pendeteksian serialisasi terhadap penjadwalannya.
- Untuk mendeteksi ini, kita dapat mempertimbangkan operasi **read** dan **write**.
- Dari bentuk yang serial akan dapat dijadwalkan secara concurrent, maka perlu mengikuti *conflict serializability*.

Conflict Serializability

- Terdapat dua instruksi berurutan, I dan J , dari transaksi T_i dan T_j , dimana ($i \neq j$).
- Jika I dan J merujuk pada item data yang berbeda, maka kita dapat menukarkan I dan J tanpa mempengaruhi hasil sembarang perintah dalam jadwal.
- Namun jika I dan J merujuk pada item data yang sama, misal Q , maka pertukaran keduanya perlu diperhatikan.

Conflict Serializability

- $I = \text{read}(Q), J = \text{read}(Q)$.
- $I = \text{read}(Q), J = \text{write}(Q)$
- $I = \text{write}(Q), J = \text{read}(Q)$
- $I = \text{write}(Q), J = \text{write}(Q)$
- Kita mengatakan I dan J adalah *conflict*, jika keduanya beroperasi pada data yang sama, dan setidaknya satu instruksi adalah operasi **write**.

Contoh Conflict Serializability

- Perhatikan penjadwalan berikut:

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

Contoh Conflict Serializability

- Dapat diserialisasi menjadi:

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

Conflict Serializability

- Jika suatu jadwal S dapat diubah menjadi S' dengan sederetan pertukaran instruksi yang tidak konflik, dikatakan S dan S' *conflict equivalent*.
- Jadwal S dikatakan conflict serializability jika memiliki persamaan konflik dengan jadwal serial.

Contoh Tidak Serializability

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

Mengapa?

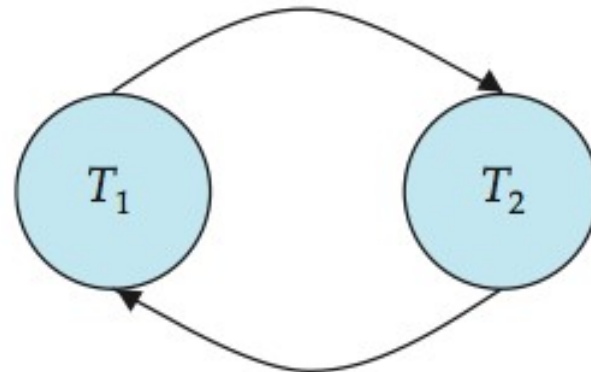
Precedence Graph

- Precedence graph terdiri dari pasangan $G=(V,E)$
 - V = himpunan transaksi yang terlibat dalam jadwal
 - E = himpunan semua edge $T_i \rightarrow T_j$ untuk 3 kondisi:
 - T_i menjalankan *write*(Q) sebelum T_j menjalankan *read*(Q)
 - T_i menjalankan *read*(Q) sebelum T_j menjalankan *write*(Q)
 - T_i menjalankan *write*(Q) sebelum T_j menjalankan *write*(Q)
- Precedence graph dapat digunakan untuk menentukan conflict serializability.

Precedence Graph

Jika sebuah edge $T_i \rightarrow T_j$ muncul dalam graf, maka dalam sembarang jadwal serial S' yang equivalent dengan S , T_i harus muncul sebelum T_j .

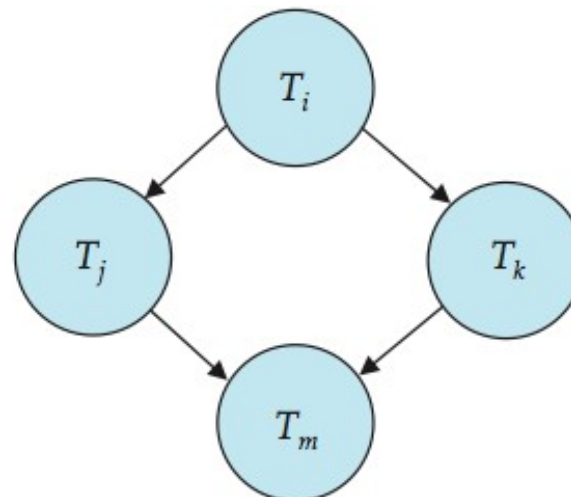
Perhatikan jadwal pada slide 13 kanan. Jika dibuat graf:



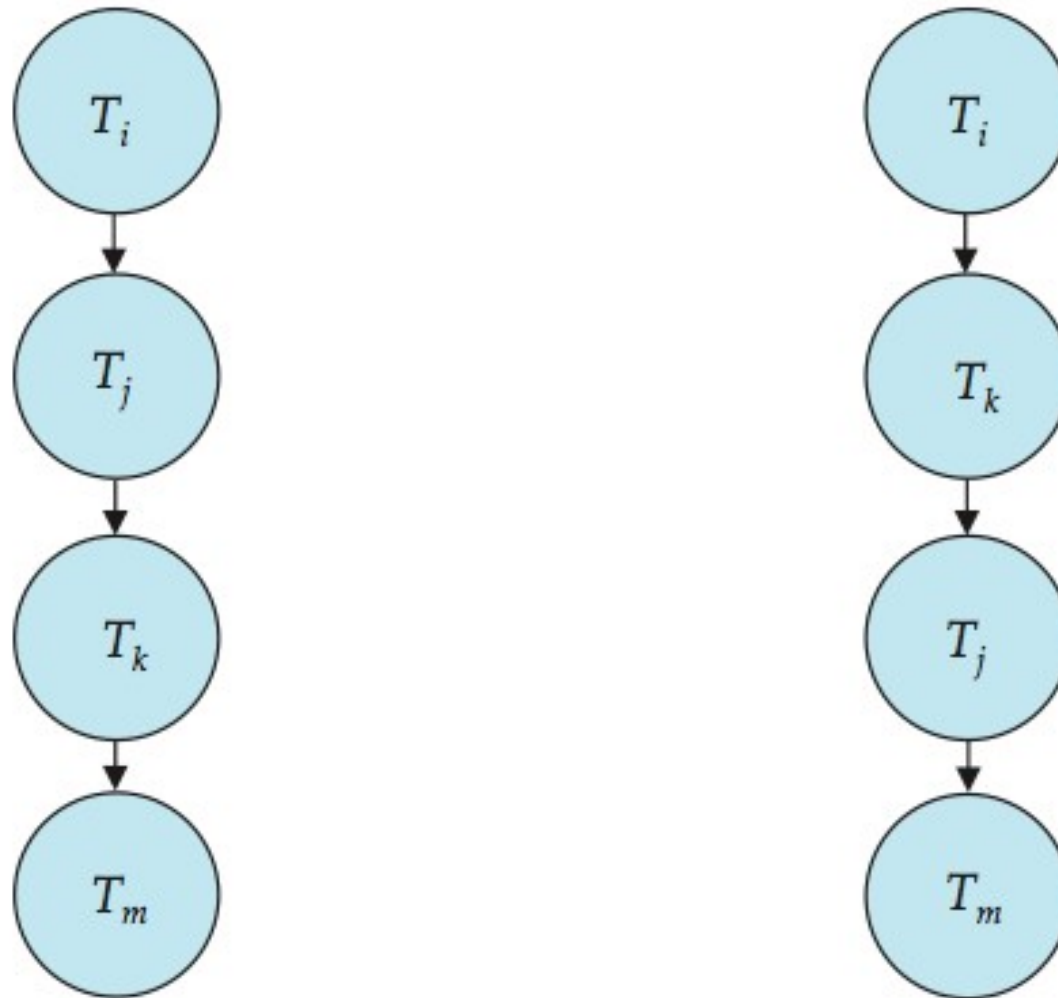
Maka dapat dikatakan bahwa jadwal tersebut tidak conflict serializable.

Serializability Order

- Sebuah serializability order dapat dicapai dengan menemukan linear order yang konsisten dengan order sebagian dalam precedence graph.
- Proses ini disebut *topological sorting*.



Serializability Order



Bagaimana dengan jadwal ini?

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)

Transaction Isolation and Atomicity

- Jika suatu transaksi T_i gagal, untuk alasan apapun, kita perlu membatalkan pengaruh dari transaksi T_i untuk menjamin properti atomicity.
- Jika ada transaksi T_j yang tergantung pada T_i , maka transaksi T_j juga harus dibatalkan.
- Jadwal yang dapat di-recover jika terjadi kegagalan:
 - Recoverable Schedule
 - Cascadeless Schedule

Recoverable Schedule

- Perhatikan jadwal berikut :

T_6	T_7
read(A)	
write(A)	
	read(A)
	commit
read(B)	

- Disebut *partial schedule*, sebab T_6 tidak terdapat commit/abort.
- Jadwal tersebut tidak recoverable. Sebab?
- Agar recoverable, sebaiknya ?

Cascadeless Schedule

- Perhatikan jadwal berikut:

T_8	T_9	T_{10}
read(A) read(B) write(A)	read(A) write(A)	
abort		read(A)

- $T_8 \rightarrow T_9 \rightarrow T_{10}$.
- Jika T_8 gagal, maka T_9 dibatalkan, dan T_{10} juga dibatalkan.

Latihan!

- Buatlah jadwal yang tidak conflict serializable dari jadwal berikut:

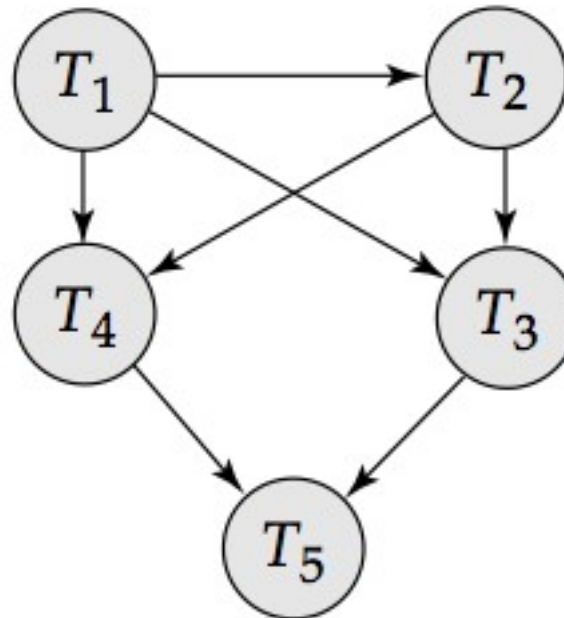
```

T1: read(A);
      read(B);
      if A = 0 then B := B + 1;
      write(B).
T2: read(B);
      read(A);
      if B = 0 then A := A + 1;
      write(A).
  
```

- Syarat konsistensi: $A=0 \wedge B=0$, dengan $A=B=0$

Latihan

- Apakah dari precedence graph ini mewakili jadwal yang conflict serializable?



Latihan

- Gambarkan presedence graph untuk jadwal berikut:

T_1	T_2	T_1	T_2
<pre>read_item(X); X:=X-N; write_item(X); read_item(Y); Y:=Y+N; write_item(Y);</pre>	<pre>read_item(X); X:=X+M; write_item(X);</pre>	<pre>read_item(X); X:=X-N; write_item(X); read_item(Y); Y:=Y+N; write_item(Y);</pre>	<pre>read_item(X); X:=X+M; write_item(X);</pre>

Latihan

- Gambarkan presedence graph untuk jadwal berikut:

T_1	T_2	T_1	T_2
read_item(X); $X:=X-N$;		read_item(X); $X:=X-N$;	
	read_item(X); $X:=X+M$;	write_item(X);	
write_item(X); read_item(Y);			read_item(X); $X:=X+M$;
	write_item(X);	read_item(Y); $Y:=Y+N$;	write_item(X);
$Y:=Y+N$; write_item(Y);		write_item(Y);	

Latihan

- Apakah termasuk jadwal conflict serializable?

transaction T_1	transaction T_2	transaction T_3
	read_item (Z); read_item (Y); write_item (Y);	read_item (Y); read_item (Z);
read_item (X); write_item (X);	read_item (X);	write_item (Y); write_item (Z);
read_item (Y); write_item (Y);	write_item (X);	

Latihan

- Apakah termasuk jadwal conflict serializable?

transaction T_1	transaction T_2	transaction T_3
		read_item (Y); read_item (Z);
read_item (X); write_item (X);		write_item (Y); write_item (Z);
	read_item (Z);	
read_item (Y); write_item (Y);	read_item (Y); write_item (Y); read_item (X); write_item (X);	