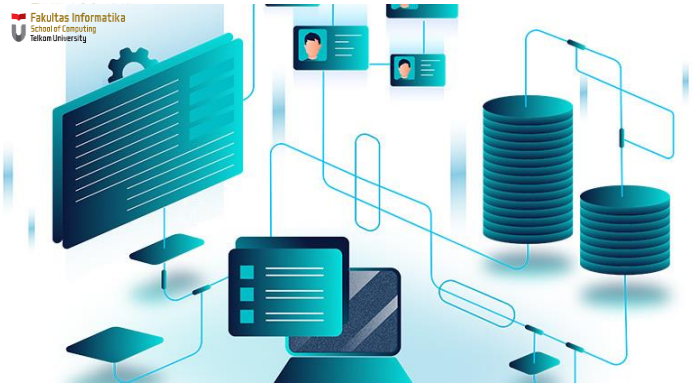
Slide 1

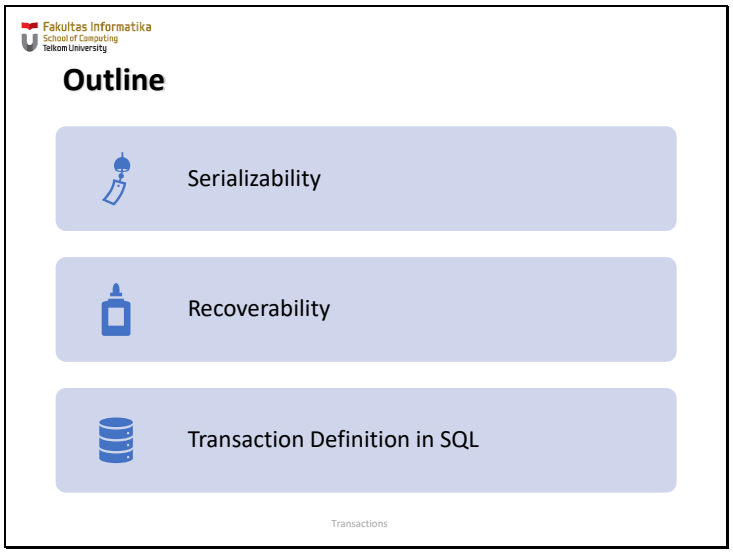
Fakultas Informatika
School of Computing
Telkom University



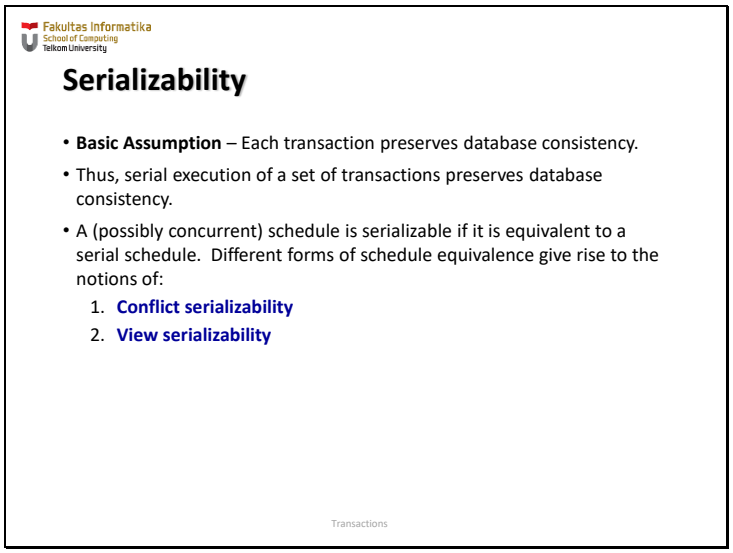
CSH2D3 - Database System

10 | Transactions (2)

Slide 2

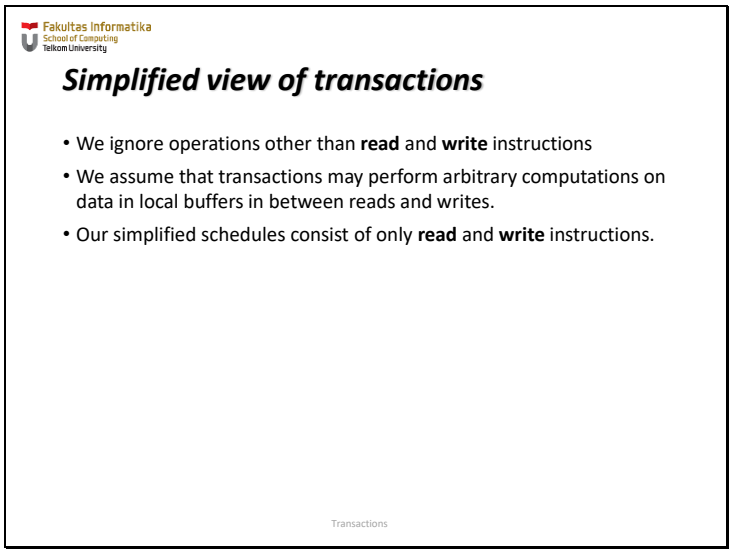

Fakultas Informatika
School of Computing
Telkom University
Outline
Serializability
Recoverability
Transaction Definition in SQL
Transactions

Slide 3



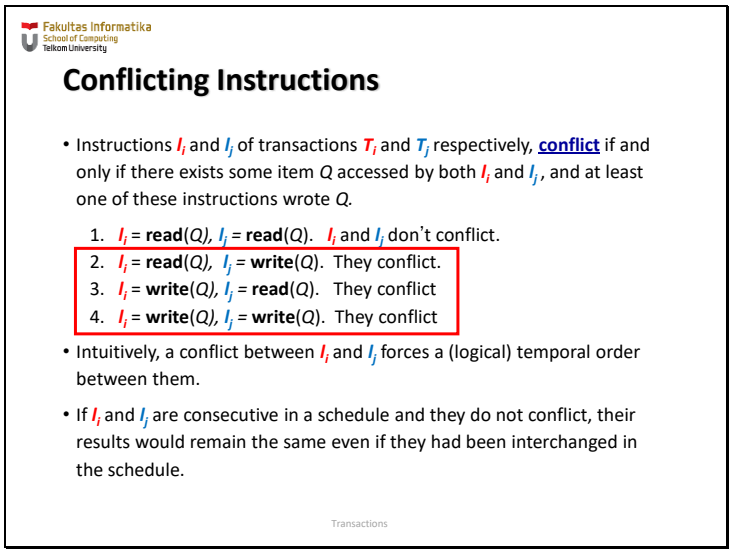
Masih ingat dengan serializable?
Suatu schedule dikatakan serializable, jika schedule tersebut equivalent terhadap serial schedule.
Disini kita akan bahas 2 bentuk equivalence schedule, yaitu : conflict serializability dan view serializability.

Slide 4



Pada pembahasan reializability ini, kita sederhanakan transakasi dengan hanya mempertimbangkan dua operasi, yaitu : READ dan WRITE,
Dan diasumsikan transaksi terjadi pada local buffer.

Slide 5

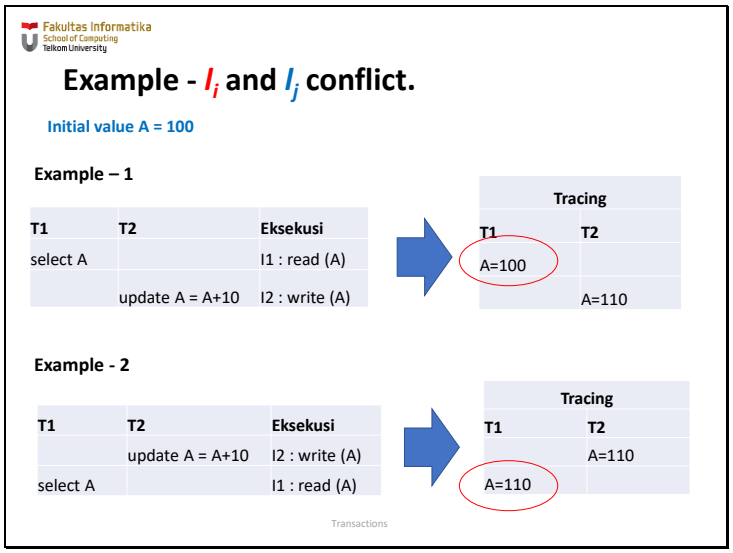


Definisi istilah yang digunakan :
Instruction = bagian dari suatu Transaction
Instruksi pada Transaction i, ditulis I_i
Instruksi pada Transaction j, ditulis I_j
Data, ditulis Q

Suatu instruksi pada transaction i dan suatu instruksi pada transaction j, dikatakan CONFLICT, Jika dan Hanya Jika ada sebuah item/data Q yang diakses oleh Ii dan Ij, DAN minimal salah satunya menulis/mengupdate data Q. Kondisi ini dapat terlihat pada kondisi No 2,3, dan 4.
Selain kondisi tersebut, dikatakan Ii dan Ij TIDAK CONFLIST, kondisi ini terlihat pada kondisi No1.

Kondisi CONFLICT ini perlu kita identifikasi, karena jika ada suatu transaksi yang mengupdate suatu data, maka data tersebut berubah nilainya.
Nah, jika terdapat 2 transaksi yang salah satunya atau bahkan keduanya mengupdate data yang sama, maka nilai akhir dari data tersebut akan sangat bergantung pada urutan schedule yang dibuat. Kita akan coba perhatikan contoh pada slide berikutnya.

Slide 6



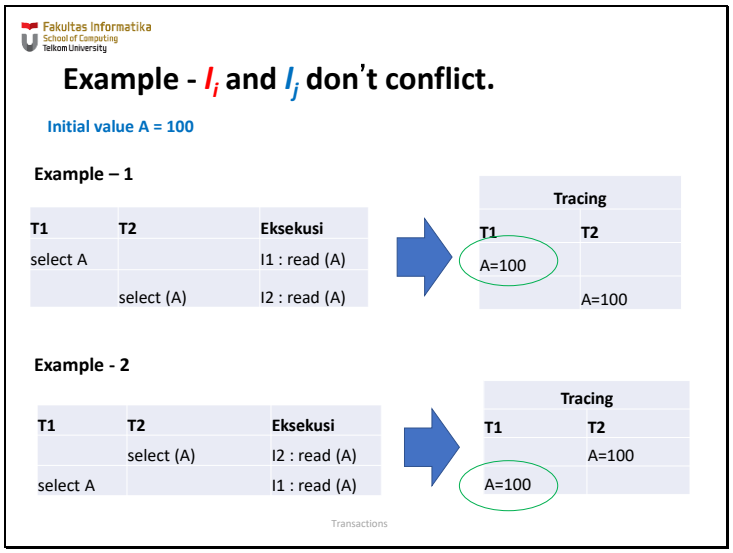
Perhatikan pada contoh 1 dan contoh 2 :
I1 : read (A) , I2 : write (A) , kondisi ini menunjukkan bahwa I1 dan I2 **conflict,** karena salah satunya melakukan perubahan data.

Nah jika pada schedule nya kita ubah urutan eksekusinya,
Pada contoh 1 : urutan eksekusi I1 – I2
Pada contoh 2 : urutan eksekusi I2 – I1
Maka, T1 pada contoh 1, akan membaca bahwa nilai data A = 100,
T1 pada contoh 2, akan membaca nilai data A = 110

Kondisi ini yang menyebabkan perubahan urutan eksekusi , akan menghasilkan hasil data yang berbeda pada Transaksi 1
Jika hasilnya berbeda, maka schedule pada contoh 1 dan contoh 2 tidak equivalent.

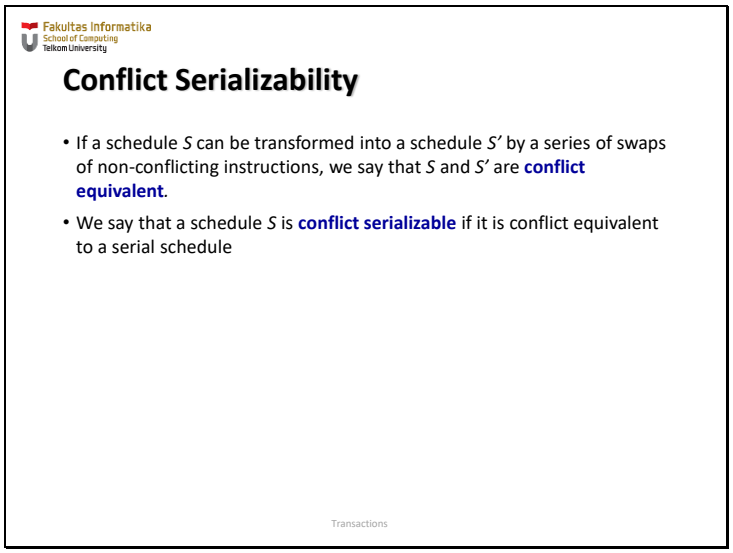
Nah, coba lakukan simulasi yang sama pada kondisi 3 dan kondisi 4

Slide 7



Perhatikan pada contoh 1 dan contoh 2 :
I1 : read (A) , I2 : read (A) , kondisi ini menunjukkan bahwa I1 dan I2 **tidak conflict**, karena tidak ada yang melakukan perubahan data.
Pada kondisi ini, jika kita ubah urutan eksekusinya, maka Transaksi 1 baik pada contoh 1 dan contoh 2, akan membaca A dengan nilai yang sama.
Maka schedule pada contoh 1 dan contoh 2 dikatakan equivalent.

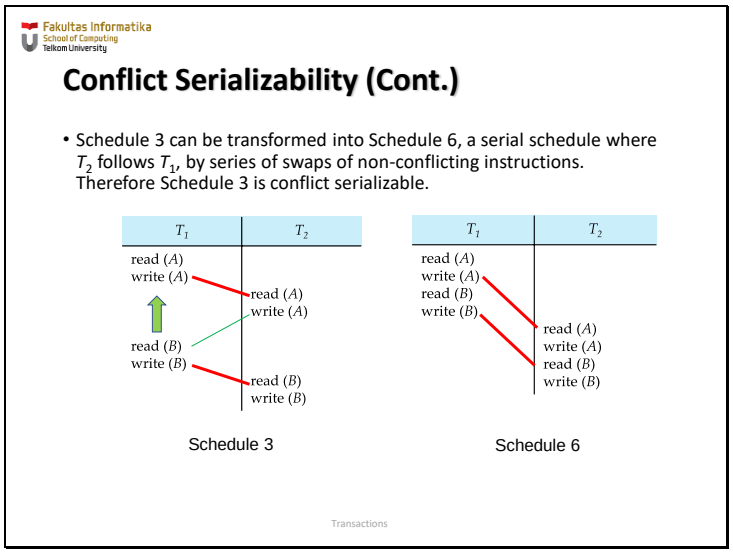
Slide 8



Jika suatu schedule S dapat diubah menjadi schedule S' dengan mengubah urutan eksekusi (swap) dari instruksi yang **tidak conflict,** maka dikatakan schedule S dan S' : **CONFLICT EQUIVALENT**

Jika suatu schedule S **CONFLICT EQUIVALENT** dengan suatu **SERIAL SCHEDULE**, maka dikatakan schedule S : **CONFLICT SERIALIZABLE**

Slide 9



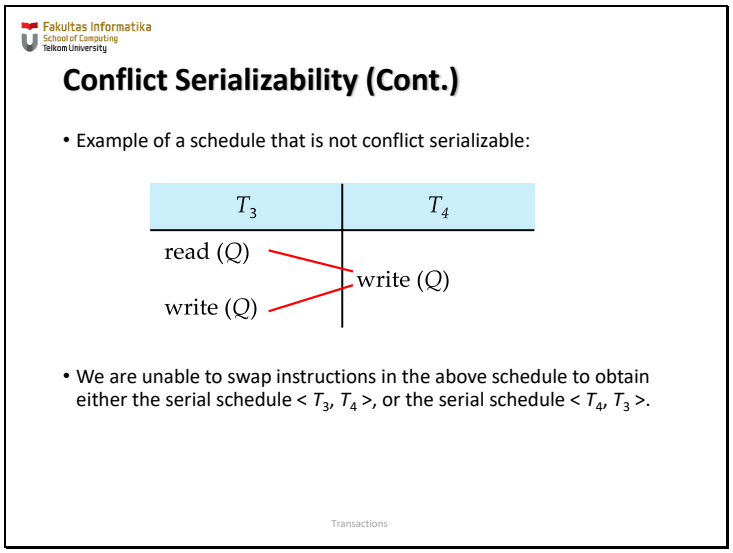
Pada gambar di slide, schedule 6 merupakan SERIAL SCHEDULE, dimana seluruh instruksi pada T1 diselesaikan terlebih dahulu, baru diikuti oleh instruksi pada T2.
Schedule 3 dapat ditransform menjadi Schedule 6 (serial schedule), dengan mengubah urutan eksekusi instruksi yang tidak conflict.
Kita coba identifikasi urutan eksekusi mana yang tida
Write (A) pada T1 – read (A) pada T2 ➔ conflict : tidak bisa diubah urutannya.
Read (B) pada T1 – write (A) pada T2 □ tidak conflict, karena mengakses data yang berbeda, sehingga bisa di swap
Jika Read (B) di swap , maka urutan eksekusinya berubah, read (B) pada T1 bisa dikerjakan terlebih dahulu sebelum write (A) pada T2
Read (B) pada T1 – read (A) pada T2 □ tidak conflict , karena tidak ada perubahan data, dan juga megakses data yang berbeda, sehingga bisa di swap lagi.
Maka read (B) pada T1 bisa dikerjakan terlebih dahulu sebelum read (A) pada T2
Write (B) pada T1 – write (A) pada T2 □ tidak conflict, karena mengakses data yang berbeda, sehingga bisa di swap
Write (B) pada T1 – read (A) pada T2 □ tidak conflict, sehingga bisa di swap lagi.

Kita lihat pada schedule 6, posisi urutan eksekusi sudah berubah dari hasil **swap instruksi** yang **TIDAK CONFLICT.**
Instruksi yang CONFLICT, urutan ekseskusi nya tidak berubah, perhatikan posisi :

Write (A) pada T1 masih dieksekusi terlebih dahulu sebelumn read(A) pada T2 dan
Write (B) pada T1 dieksekusi terlebih dahulu sebelum read(B) pada T2

Perlu diingat, bahwa urutan instruksi pada transaksi yang sama **TIDAK BOLEH** di swap

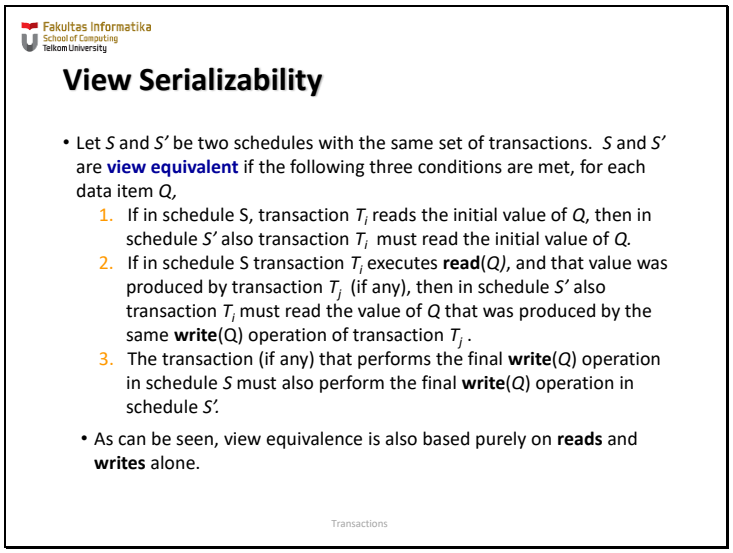
Slide 10



Pada contoh slide ini. Schedule ini tidak bisa diubah menjadi serial schedule, sehingga schedule ini dikatakan **TIDAK CONFLICT SERIALIZABLE.**
Serial schedule, mengharuskan T3 diselesaikan terlebih dahulu baru T4, **ATAU** T4 diselesaikan baru T3. Kedua serial schedule tersebut tidak dapat dicapai, karena tidak ada instruksi baik dari T3 maupun T4 yang bisa di swap.

Trace :
Read(Q) pada T3 CONFLICT dengan write(Q) pada T4 → tidak bisa di swap
Write(Q) pada T3 CONFLICT dengan write(Q) pada T4 □ tidak bisa di swap

Slide 11

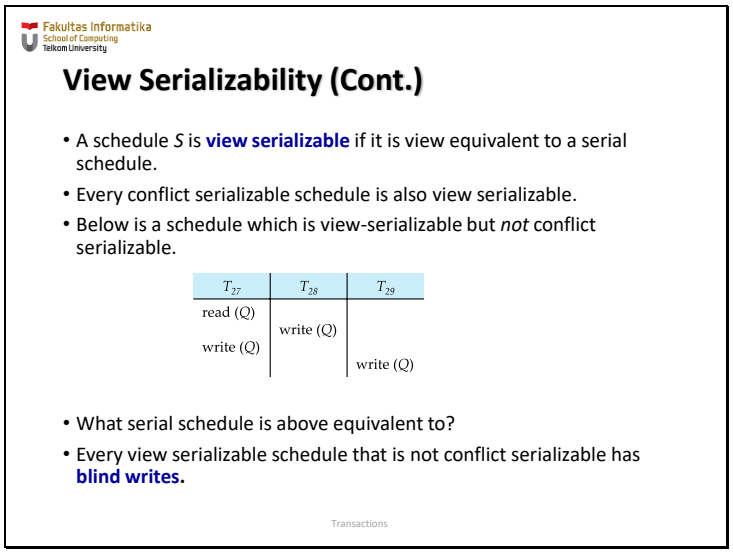


Bentuk equivalent schedule yang lain adalah View Serializability.
Suatu schedule S dan S' dikatakan VIEW EQUIVALENT jika memenuhi 3 syarat untuk setiap data item Q :

- Jika pada schedule S, Ti read nilai awal dari Q, maka pada schedule S', Ti juga membaca nilai awal Q.
- Jika pada schedule S, Ti read nilai Q yang dihasilkan/diubah/write oleh Tj (jika ada), maka pada schedule S': Ti juga membaca nilai Q yang dihasilkan/diubah/write oleh Tj
- Jika pada schedule S Ti merupakan Transaki yang terakhir menghasilkan/mengubah/ write Q , maka pada schedule S', Ti juga melakukan instruksi write terakhir terhadap Q

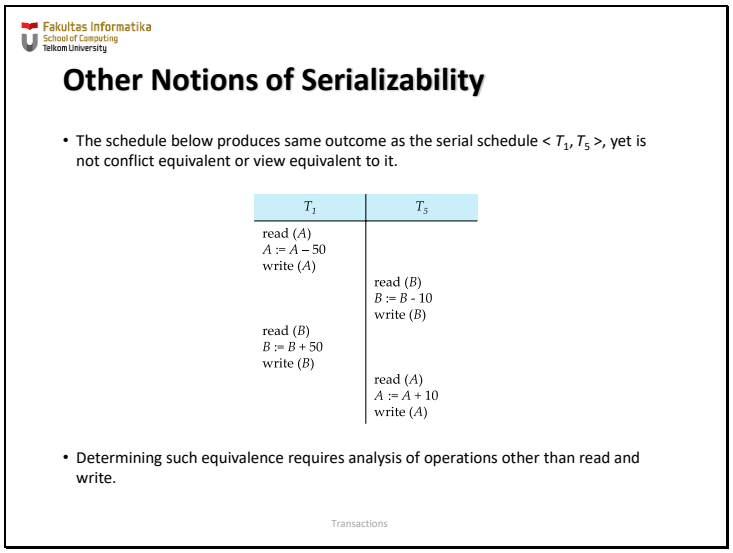
Pada kasus ini, kita masih berpatokan pada instruksi READ dan WRITE saja.

Slide 12



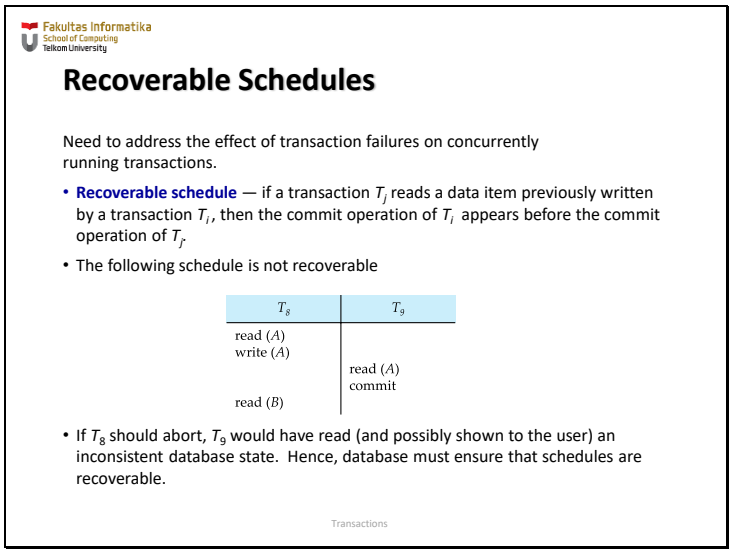
Suatu schedule S dikatakan VIEW SERIALIZABLE , jika schedule tersebut VIEW EQUIVALENT terhadap serial schedule.
Setiap schedule yang conflict serializable PASTI view serializable, tapi **tidak berlaku sebaliknya**
Contoh pada slide ini, merupakan VIEW-SERIALIZABLE namun TIDAK CONFLICT SERIALIZABLE.
Setiap schedule yang view serializable dan Tidak Conflict Serializable memiliki blind write.
Pengertian blind writes disini adalah adanya perubahan data tanpa memperhatikan nilai data yang sebelumnya (tanpa melakukan read data).
Tentukan serial schedule dari schedule pada contoh tersebut!

Slide 13



Ada juga bentuk Serializability yang lain.
Contoh schedule pada slide ini menghasilkan nilai akhir yang sama seperti serial schedule <T1, T5>, meskipun tidak conflict equivalent dan juga tidak view equivalent terhadap serial schedule.
Pada bentuk ini, juga dianalisis instruksi selain instruksi READ dan WRITE.

Slide 14



Recoverable Schedule
Schedule yang recoverable diperlukan untuk dapat menghandle adanya kegagalan transakasi pada concurrent transaction.
Suatu schedule dikatakan recoverable schedule, jika sebuah transaksi Tj membaca data yang dihasilkan oleh Ti, dan Ti melakukan instruksi commit terlebih dahulu sebelum Tj melakukan commit.
Schedule pada contoh dikatakan tidak recoverable, kenapa?
Pada schedule tersebut, Jika T8 kemudian melakukan abort, maka T9 akan membaca data yang tidak konsisten.

Pada kasus ini, penggunaan Commit diperhatikan.

Slide 15

Fakultas Informatika
School of Computing
Telkom University

Cascading Rollbacks

- **Cascading rollback** – a single transaction failure leads to a series of transaction rollbacks. Consider the following schedule where none of the transactions has yet committed (so the schedule is recoverable)

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

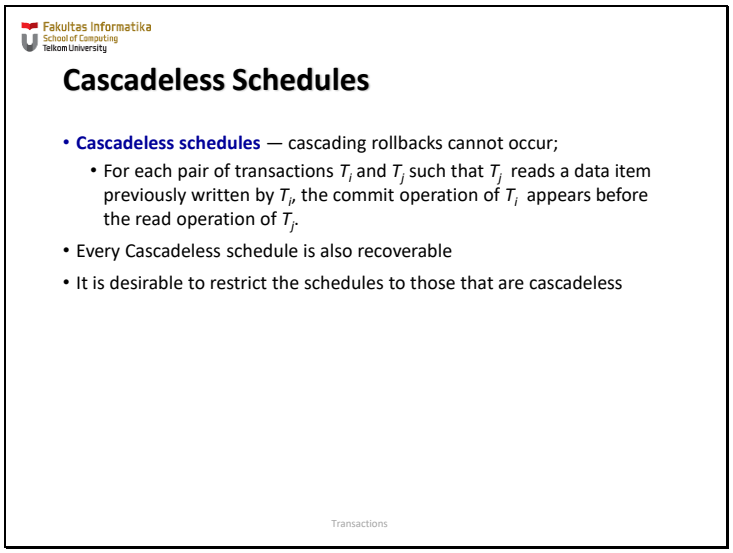
If T_{10} fails, T_{11} and T_{12} must also be rolled back.

- Can lead to the undoing of a significant amount of work

Transactions

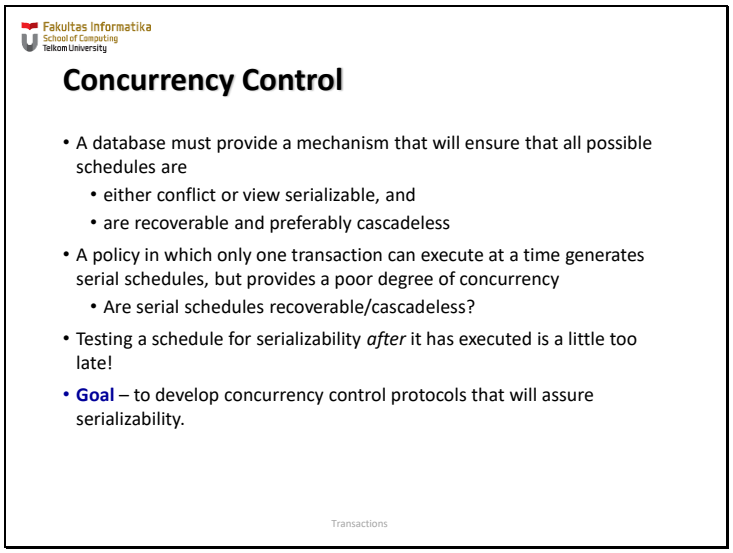
Dikatakan cascading rollback jika terjadi suatu kondisi dimana kegagalan satu transaski akan menyebabkan terjadinya rollback pada transaksi sebelumnya.
Perhatikan schedule pada contoh, dimana tidak ada satupun transaksi yang melakukan commit.
Jika T10 gagal/digagalkan, maka T11 dan T12 harus di roll back.
Kondisi seperti ini mungkin saja terjadi pada sejumlah besar transaksi.

Slide 16



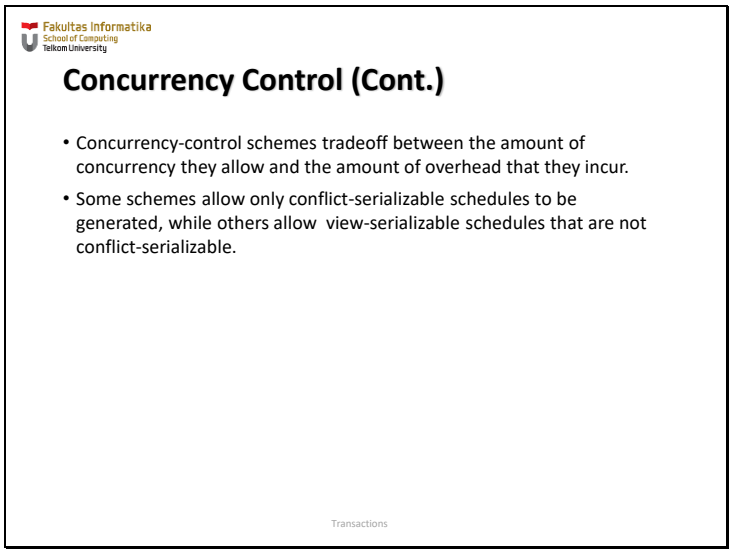
Untuk menghindari terjadinya cascading rollback, maka perlu dipertimbangkan schedule yang cascadeless.
Syarat cascadeless schedule yaitu :
Untuk setiap pasangan transaksi Ti dan Tj, dimana Tj membaca item data yang dihasilkan dan di commit terlebih dahulu oleh Ti.
Setiap cascadeless schedule juga merupakan recoverable schedule.

Slide 17



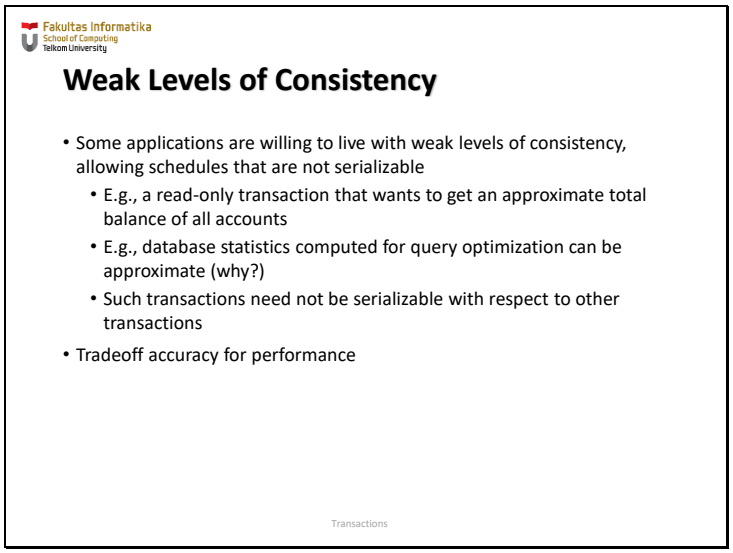
Sebuah database harus menyediakan mekanisme untuk memastikan schedule yang disusun merupakan conflict atau view serializable dan juga recoverable. Akan lebih baik jika bisa cascadeless.

Slide 18



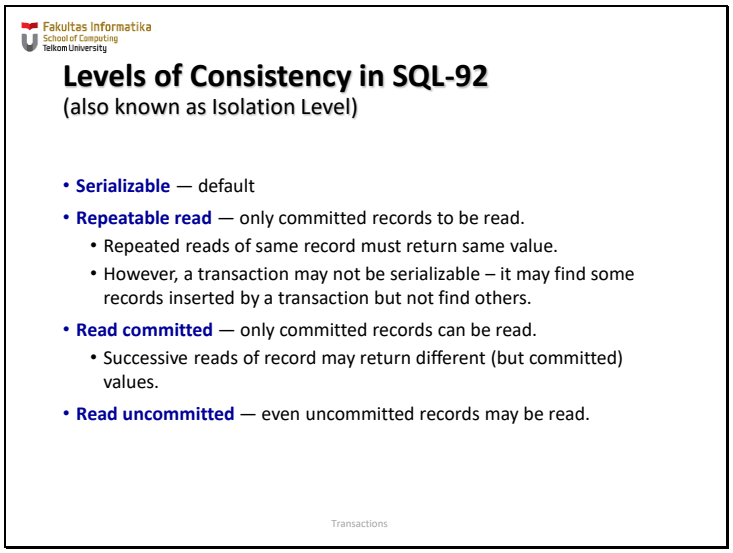
Skema concurrency control akan saling Tarik ulur antara jumlah concurrency yang diijinkan dan jumlah overhead yang dapat terjadi.
Beberpa skema mengijinkan untuk hanya mengenerate schedule yang conflict serializable, sementara skema lain mengenerate schedule view-serializable yang tidak conflict serializable.

Slide 19



Beberapa aplikasi mungkin mengimplementasikan level constistency yang rendah, sehingga dapat mengijinkan adanya schedule yang tidak serializable.
Misal : transaksi read-only, dll.

Slide 20



Ada beberapa level consistency yang perlu diketahui. Level ini juga sering disebut isolation level.
Urutan isolation level tertinggi adalah serializable, kemudian repeatable read, dst

Slide 21

Fakultas Informatika
School of Computing
Telkom University

Levels of Consistency

- Lower degrees of consistency useful for gathering approximate information about the database
- Warning: some database systems do not ensure serializable schedules by default
- E.g., Oracle (and PostgreSQL prior to version 9) by default support a level of consistency called snapshot isolation (not part of the SQL standard)

Slide 22

Fakultas Informatika
School of Computing
Telkom University

Transaction Definition in SQL

- In SQL, a transaction begins implicitly.
- A transaction in SQL ends by:
  - **Commit work** commits current transaction and begins a new one.
  - **Rollback work** causes current transaction to abort.
- In almost all database systems, by default, every SQL statement also commits implicitly if it executes successfully
  - Implicit commit can be turned off by a database directive
    - E.g., in JDBC -- connection.setAutoCommit(false);
- Isolation level can be set at database level
- Isolation level can be changed at start of transaction
    - E.g. In SQL **set transaction isolation level serializable**
    - E.g. in JDBC -- connection.setTransactionIsolation(Connection.TRANSACTION_SERIALIZABLE)

Fakultas Informatika
School of Computing
Telkom University

Transactions as SQL Statements

- E.g., Transaction 1:
  select *ID, name* **from** *instructor* **where** *salary* > 90000
- E.g., Transaction 2:
  insert into *instructor* **values** ('11111', 'James', 'Marketing', 100000)
- Suppose
  - T1 starts, finds tuples salary > 90000 using index and locks them
  - And then T2 executes.
  - Do T1 and T2 conflict? Does tuple level locking detect the conflict?
  - Instance of the **phantom phenomenon**
- Also consider T3 below, with Wu's salary = 90000
  update *instructor*
  set *salary* = *salary* * 1.1
  where *name* = 'Wu'
- Key idea: Detect "**predicate**" conflicts, and use some form of "**predicate locking**"

Slide 24

Fakultas Informatika
School of Computing
Telkom University

Exercise

Consider the following schedule 1 and 2.
Assume that the consistency level of the transactions is **read uncommitted.**

1. Examine whether the schedules is conflict serializable or view serializable or non of them!
2. If the schedule is serializable, write down the serial schedule(s)!

Schedule 1

T_0	T_1
Read(A)	
	Read(B)
Read(B)	
	Read(A)
	Write(A)
Write(A)	
	Write(B)
Write(B)	

Schedule 2

T_0	T_1	T_2
		Read(X)
	Read(X)	
Read(X)		
	Write(X)	
	Read(X)	
		Write(X)
		Read(X)
Write(X)		
		Write(X)
		Read(X)
Read(X)		
Write(X)		
	Write(X)	

Slide 25

Fakultas Informatika
School of Computing
Telkom University

References

Silberschatz, Korth, and Sudarshan. *Database System Concepts* – 7th Edition. McGraw-Hill. 2019.

Slides adapted from Database System Concepts Slide.

Source: https://www.db-book.com/db7/slides-dir/index.html