

Slide 1



Fakultas Informatika
Sekolah of Computing
Telkom University

CSH2D3 - Database System
10 | Transactions (2)

Transactions

**Fakultas Informatika**
School of Computing
Telkom University

Outline

**Serializability**

**Recoverability**

**Transaction Definition in SQL**

Transactions



Serializability

- **Basic Assumption** – Each transaction preserves database consistency.
- Thus, serial execution of a set of transactions preserves database consistency.
- A (possibly concurrent) schedule is serializable if it is equivalent to a serial schedule. Different forms of schedule equivalence give rise to the notions of:
 1. **Conflict serializability**
 2. **View serializability**

Transactions

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.



Simplified view of transactions

- We ignore operations other than **read** and **write** instructions
- We assume that transactions may perform arbitrary computations on data in local buffers in between reads and writes.
- Our simplified schedules consist of only **read** and **write** instructions.

Transactions

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



Conflicting Instructions

- Instructions I_i and I_j of transactions T_i and T_j respectively, **conflict** if and only if there exists some item Q accessed by both I_i and I_j , and at least one of these instructions wrote Q .
 - $I_i = \text{read}(Q)$, $I_j = \text{read}(Q)$. I_i and I_j don't conflict.
 - $I_i = \text{read}(Q)$, $I_j = \text{write}(Q)$. They conflict.
 - $I_i = \text{write}(Q)$, $I_j = \text{read}(Q)$. They conflict.
 - $I_i = \text{write}(Q)$, $I_j = \text{write}(Q)$. They conflict.
- Intuitively, a conflict between I_i and I_j forces a (logical) temporal order between them.
- If I_i and I_j are consecutive in a schedule and they do not conflict, their results would remain the same even if they had been interchanged in the schedule.

Transactions

Definisi istilah yang digunakan :

Instruction = bagian dari suatu Transaction

Instruksi pada Transaction i, ditulis I_i

Instruksi pada Transaction j, ditulis I_j

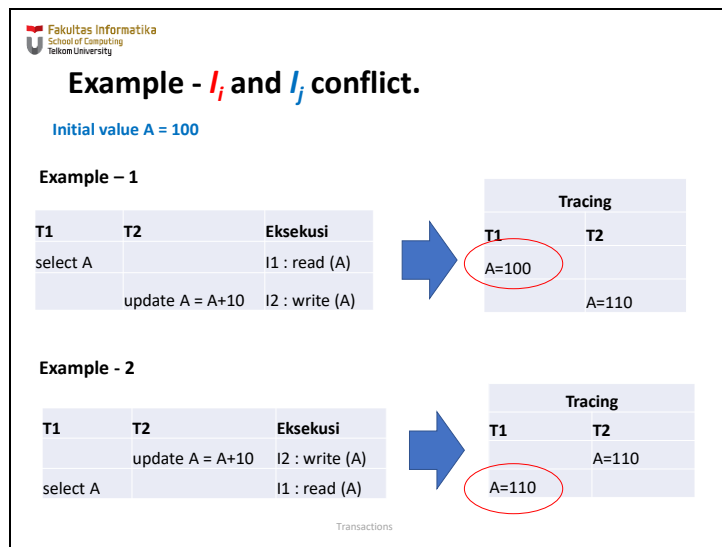
Data, ditulis Q

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 I_i dan I_j , DAN minimal salah satunya menulis/mengupdate data Q . Kondisi ini dapat terlihat pada kondisi No 2,3, dan 4.

Selain kondisi tersebut, dikatakan I_i dan I_j TIDAK CONFLICT, 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.



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

Fakultas Informatika
School of Computing
Relkom University

Example - I_i and I_j don't conflict.

Initial value $A = 100$

Example - 1

T1	T2	Eksekusi
select A		I1 : read (A)
	select (A)	I2 : read (A)

→

Tracing	
T1	T2
A=100	
	A=100

Example - 2

T1	T2	Eksekusi
	select (A)	I2 : read (A)
select A		I1 : read (A)

→

Tracing	
T1	T2
	A=100
A=100	

Transactions

Perhatikan pada contoh 1 dan contoh 2 :

$I1 : \text{read}(A)$, $I2 : \text{read}(A)$, 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.



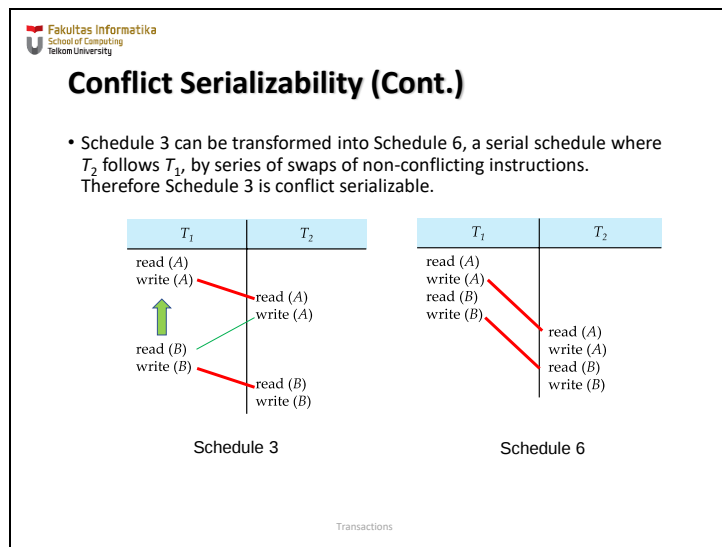
Conflict Serializability

- If a schedule S can be transformed into a schedule S' by a series of swaps of non-conflicting instructions, we say that S and S' are **conflict equivalent**.
- We say that a schedule S is **conflict serializable** if it is conflict equivalent to a serial schedule

Transactions

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**



Pada gambar di slide, schedule 6 merupakan SERIAL SCHEDULE, dimana seluruh instruksi pada T_1 diselesaikan terlebih dahulu, baru diikuti oleh instruksi pada T_2 .

Schedule 3 dapat ditransform menjadi Schedule 6 (serial schedule), dengan mengubah urutan eksekusi instruksi yang tidak conflict.

Kita coba identifikasi urutan eksekusi mana yang tidak

Write (A) pada T_1 – read (A) pada T_2 ➔ conflict : tidak bisa diubah urutannya.

Read (B) pada T_1 – write (A) pada T_2 □ tidak conflict, karena mengakses data yang berbeda, sehingga bisa di swap

Jika Read (B) di swap, maka urutan eksekusinya berubah, read (B) pada T_1 bisa dikerjakan terlebih dahulu sebelum write (A) pada T_2

Read (B) pada T_1 – read (A) pada T_2 □ tidak conflict, karena tidak ada perubahan data, dan juga mengakses data yang berbeda, sehingga bisa di swap lagi.

Maka read (B) pada T_1 bisa dikerjakan terlebih dahulu sebelum read (A) pada T_2

Write (B) pada T_1 – write (A) pada T_2 □ tidak conflict, karena mengakses data yang berbeda, sehingga bisa di swap

Write (B) pada T_1 – read (A) pada T_2 □ 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 eksekusi nya tidak berubah, perhatikan posisi :

Write (A) pada T_1 masih dieksekusi terlebih dahulu sebelum read (A) pada T_2 dan

Write (B) pada T_1 dieksekusi terlebih dahulu sebelum read (B) pada T_2

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

Fakultas Informatika
School of Computing
Telkom University

Conflict Serializability (Cont.)

- Example of a schedule that is not conflict serializable:

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

- We are unable to swap instructions in the above schedule to obtain either the serial schedule $\langle T_3, T_4 \rangle$, or the serial schedule $\langle T_4, T_3 \rangle$.

Transactions

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



View Serializability

- Let S and S' be two schedules with the same set of transactions. S and S' are **view equivalent** if the following three conditions are met, for each data item Q ,
 - If in schedule S , transaction T_i reads the initial value of Q , then in schedule S' also transaction T_i must read the initial value of Q .
 - If in schedule S transaction T_i executes **read**(Q), and that value was produced by transaction T_j (if any), then in schedule S' also transaction T_i must read the value of Q that was produced by the same **write**(Q) operation of transaction T_j .
 - The transaction (if any) that performs the final **write**(Q) operation in schedule S must also perform the final **write**(Q) operation in schedule S' .
- As can be seen, view equivalence is also based purely on **reads** and **writes** alone.

Transactions

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

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

 **View Serializability (Cont.)**

- A schedule S is **view serializable** if it is view equivalent to a serial schedule.
- Every conflict serializable schedule is also view serializable.
- Below is a schedule which is view-serializable but *not* conflict serializable.

T_{27}	T_{28}	T_{29}
read (Q)	write (Q)	
write (Q)		write (Q)

- What serial schedule is above equivalent to?
- Every view serializable schedule that is not conflict serializable has **blind writes**.

Transactions

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!

Fakultas Informatika
School of Computing
Relikom University

Other Notions of Serializability

- The schedule below produces same outcome as the serial schedule $\langle T_1, T_5 \rangle$, yet is not conflict equivalent or view equivalent to it.

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)

- Determining such equivalence requires analysis of operations other than read and write.

Transactions

Ada juga bentuk Serializability yang lain.

Contoh schedule pada slide ini menghasilkan nilai akhir yang sama seperti serial schedule $\langle T_1, T_5 \rangle$, meskipun tidak conflict equivalent dan juga tidak view equivalent terhadap serial schedule.

Pada bentuk ini, juga dianalisis instruksi selain instruksi READ dan WRITE.

 **Fakultas Informatika**
School of Computing
Telkom University

Recoverable Schedules

Need to address the effect of transaction failures on concurrently running transactions.

- **Recoverable schedule** — if a transaction T_j reads a data item previously written by a transaction T_i , then the commit operation of T_i appears before the commit operation of T_j .
- The following schedule is not recoverable

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

- If T_8 should abort, T_9 would have read (and possibly shown to the user) an inconsistent database state. Hence, database must ensure that schedules are recoverable.

Transactions

Recoverable Schedule

Schedule yang recoverable diperlukan untuk dapat menghandle adanya kegagalan transaksi pada concurrent transaction.

Suatu schedule dikatakan recoverable schedule, jika sebuah transaksi T_j membaca data yang dihasilkan oleh T_i , dan T_i melakukan instruksi commit terlebih dahulu sebelum T_j melakukan commit.

Schedule pada contoh dikatakan tidak recoverable, kenapa?

Pada schedule tersebut, Jika T_8 kemudian melakukan abort, maka T_9 akan membaca data yang tidak konsisten.

Pada kasus ini, penggunaan Commit diperhatikan.

 **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 transaksi akan menyebabkan terjadinya rollback pada transaksi sebelumnya.

Perhatikan schedule pada contoh, dimana tidak ada satupun transaksi yang melakukan commit.

Jika T_{10} gagal/digagalkan, maka T_{11} dan T_{12} harus di roll back.

Kondisi seperti ini mungkin saja terjadi pada sejumlah besar transaksi.



Cascadeless Schedules

- **Cascadeless schedules** — cascading rollbacks cannot occur;
 - For each pair of transactions T_i and T_j such that T_j reads a data item previously written by T_i , the commit operation of T_i appears before the read operation of T_j .
 - Every Cascadeless schedule is also recoverable
 - It is desirable to restrict the schedules to those that are cascadeless

Transactions

Untuk menghindari terjadinya cascading rollback, maka perlu dipertimbangkan schedule yang cascadeless.

Syarat cascadeless schedule yaitu :

Untuk setiap pasangan transaksi T_i dan T_j , dimana T_j membaca item data yang dihasilkan dan di commit terlebih dahulu oleh T_i .

Setiap cascadeless schedule juga merupakan recoverable schedule.



Concurrency Control

- A database must provide a mechanism that will ensure that all possible schedules are
 - either conflict or view serializable, and
 - are recoverable and preferably cascadeless
- A policy in which only one transaction can execute at a time generates serial schedules, but provides a poor degree of concurrency
 - Are serial schedules recoverable/cascadeless?
- Testing a schedule for serializability *after* it has executed is a little too late!
- **Goal** – to develop concurrency control protocols that will assure serializability.

Transactions

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



Concurrency Control (Cont.)

- Concurrency-control schemes tradeoff between the amount of concurrency they allow and the amount of overhead that they incur.
- Some schemes allow only conflict-serializable schedules to be generated, while others allow view-serializable schedules that are not conflict-serializable.

Transactions

Skema concurrency control akan saling Tarik ulur antara jumlah concurrency yang diijinkan dan jumlah overhead yang dapat terjadi.

Beberapa skema mengijinkan untuk hanya mengenerate schedule yang conflict serializable, sementara skema lain mengenerate schedule view-serializable yang tidak conflict serializable.



Weak Levels of Consistency

- Some applications are willing to live with weak levels of consistency, allowing schedules that are not serializable
 - E.g., a read-only transaction that wants to get an approximate total balance of all accounts
 - E.g., database statistics computed for query optimization can be approximate (why?)
 - Such transactions need not be serializable with respect to other transactions
- Tradeoff accuracy for performance

Transactions

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



Levels of Consistency in SQL-92

(also known as Isolation Level)

- **Serializable** — default
- **Repeatable read** — only committed records to be read.
 - Repeated reads of same record must return same value.
 - However, a transaction may not be serializable – it may find some records inserted by a transaction but not find others.
- **Read committed** — only committed records can be read.
 - Successive reads of record may return different (but committed) values.
- **Read uncommitted** — even uncommitted records may be read.

Transactions

Ada beberapa level consistency yang perlu diketahui. Level ini juga sering disebut isolation level.

Urutan isolation level tertinggi adalah serializable, kemudian repeatable read, dst

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)

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)`

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**"

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)		Write(X)
		Read(X)
Read(X)		Write(X)
Write(X)		Read(X)
	Write(X)	

Transactions



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>

Transactions

25