Fakultas Informatika
School of Computing
Telkom University

Database System

07 | Transaction 02

Tahun Ajar Ganjil 2024/2025

Oleh:
Tim Dosen

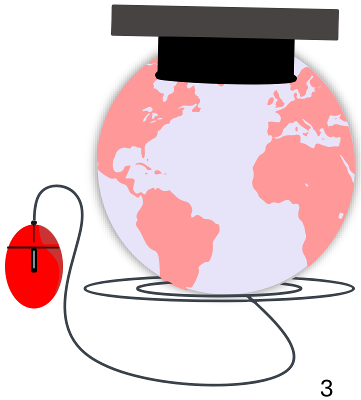
Fakultas Informatika
School of Computing
Telkom University

Goals of the Meeting

Students are able to explain transaction management

OUTLINES

- Serializability
- Recoverability
- Transaction Definition in SQL



SERIALIZABILITY

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**

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.

CONFLICTING INSTRUCTIONS

- Instructions l_i and l_j of transactions T_i and T_j respectively, **conflict** if and only if there exists some item Q accessed by both l_i and l_j, and at least one of these instructions wrote Q.
  1. l_i = **read**(Q), l_j = **read**(Q). l_i and l_j don't conflict.
  2. l_i = **read**(Q), l_j = **write**(Q). They conflict.
  3. l_i = **write**(Q), l_j = **read**(Q). They conflict
  4. l_i = **write**(Q), l_j = **write**(Q). They conflict
- Intuitively, a conflict between l_i and l_j forces a (logical) temporal order between them.
- If l_i and l_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.

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

CONFLICT SERIALIZABILITY (CONT.)

- Schedule 3 can be transformed into Schedule 6, a serial schedule where T_2 follows T_1, by series of swaps of non-conflicting instructions. Therefore Schedule 3 is conflict serializable.

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

Schedule 3

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

Schedule 6

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 < T_3, T_4 >, or the serial schedule < T_4, T_3 >.

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*,
  1. 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*.
  2. 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 .
  3. 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.

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**.

OTHER NOTIONS OF SERIALIZABILITY

- The schedule below produces same outcome as the serial schedule < T_1, T_5 >, 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.

RECOVERABILITY

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.

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

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

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.

CONCURRENCY CONTROL (CONT.)

- Schedules must be conflict or view serializable, and recoverable, for the sake of database consistency, 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.
- 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.

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

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.

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

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

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

ANY QUESTIONS?

Fakultas Informatika
School of Computing
Telkom University

THANK YOU

Database System