Lecture 14
Transactions and Schedules

Chapter 17.4-17.5 : Elmasri & Navathe, 5ED
Chapter 16.2-16.3 and 17.1: Ramakrishnan & Gehrke, 3ED

Demetris Zeinalipour
http://www.cs.ucy.ac.cy/~dzeina/courses/epl446
Lecture Outline
Transactions and Schedules

• 16.2) **Transactions and Schedules** (Χρονοπρόγραμμα)
  – Serial Schedule (Σειριακό Χρονοπρόγραμμα) ... one after the other...
  – Complete Schedule (Πλήρες Χρονοπρόγραμμα) ... with Commit, Abort

• 17.1) **Serializability** (Σειριοποιησιμότητα)
  – “Correctness Measure” of some Schedule
  – Why is it useful? It answers the question: “Will an interleaved schedule execute correctly
  – i.e., a Serializable schedule will execute as **correctly as serial** schedule ... but in an **interleaved manner**!

• 17.1) **Recoverability** (Επαναφερσιμότητα)
  – “Recoverability Measure” of some Schedule.
  – Why is it useful? It answers the question: “Do we need to rollback a some (or all) transactions in an interleaved schedule after some Failure (e.g., ABORT)”
  – i.e., in a Recoverable schedule no transaction needs to be **rolled back** (διαδικασία επιστροφής) once committed!
Transactions and Schedules
(Δοσοληψίες και Χρονοπρόγραμμα)

• **Serial Schedule (Σειριακό Χρονοπρόγραμμα)**
  - A schedule in which the different transactions are **NOT interleaved** (i.e., transactions are executed from start to finish one-by-one)

<table>
<thead>
<tr>
<th>Serial Schedule</th>
<th>Serial Schedule</th>
</tr>
</thead>
<tbody>
<tr>
<td>T1</td>
<td>T2</td>
</tr>
<tr>
<td>R(B) W(B)</td>
<td>R(A) W(A)</td>
</tr>
<tr>
<td>R(B) W(B)</td>
<td>R(B) W(B)</td>
</tr>
<tr>
<td></td>
<td>R(A) W(A)</td>
</tr>
</tbody>
</table>

\[ N! \] Possible Serial Schedules, where \( N \) the number of Transactions
Transactions and Schedules
(Δοσοληψίες και Χρονοπρόγραμμα)

- **Complete Schedule (Πλήρες Χρονοπρόγραμμα)**
  - A schedule that contains either a **commit** (ολοκλήρωση δοσοληψίας) or an **abort** (ματαίωση δοσοληψίας) action for **EACH** transaction.*

<table>
<thead>
<tr>
<th>T1</th>
<th>T2</th>
</tr>
</thead>
<tbody>
<tr>
<td>R(A)</td>
<td>R(B)</td>
</tr>
<tr>
<td>W(A)</td>
<td>W(B)</td>
</tr>
<tr>
<td>Commit</td>
<td>Abort</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>T1</th>
<th>T2</th>
</tr>
</thead>
<tbody>
<tr>
<td>R(A)</td>
<td>R(B)</td>
</tr>
<tr>
<td>W(A)</td>
<td>W(B)</td>
</tr>
<tr>
<td>Commit</td>
<td>Abort</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>T1</th>
<th>T2</th>
</tr>
</thead>
<tbody>
<tr>
<td>R(A)</td>
<td>R(B)</td>
</tr>
<tr>
<td>W(A)</td>
<td>W(B)</td>
</tr>
<tr>
<td>Commit</td>
<td>Abort</td>
</tr>
</tbody>
</table>

*Note: consequently, a complete schedule will not contain any active transactions at the end of the schedule*
Transactions and Schedules
(Δοσοληψίες και Χρονοπρόγραμμα)

- **Interleaved Schedules** of transactions **improve performance**
  - Throughput (ρυθμαπόδοση): More Xacts per seconds; and
  - Response Time (χρόνος απόκρισης): A short transaction will not get stuck behind a long-running transaction

- Yet it might lead the DB to an **inconsistent state** as we have shown

- **Serial schedule** (σειριακό χρονοπρόγραμμα) is slower but guarantees consistency (correctness)

- We seek to identify schedules that are:
  - As fast as interleaved schedules.
  - As consistent as serial schedules
Transactions and Schedules (Δοσοληψίες και Χρονοπρόγραμμα)

- We shall now **characterize** different schedules based on the following two **properties**:

  **A. Based on Serializability (Σειριοποιησιμότητα)**
  - We shall ignore **Commits** and **Aborts** for this section
  - **Characterize** which schedules are **correct** when concurrent transactions are executing.
  - **Conflict Serializable Schedule** (Σειριοποιησιμότητα Συγκρούσεων)
  - **View Serializable Schedule** (Σειριοποιησιμότητα Όψεων)

**B. Based on Recoverability (Επαναθεπζιμόηηηα)**
- **Commits** and **Aborts** become important for this section!
- **Characterize** which schedules **can be recovered** and how easily.
- **Recoverable Schedule** (Επαναφέρσιμο Χρονοπρόγραμμα).
- **Cascadeless schedule** (Χρονοπρ. χωρίς διαδιδόμενη ανάκληση).
- **Strict Schedules** (Αυστηρό Χρονοπρόγραμμα).
Conflicting Actions
(Συγκρούμενες Πράξεις)

Two or more actions are said to be in conflict if:
- The actions belong to different transactions.
- At least one of the actions is a write operation.
- The actions access the same object (read or write).

- The following set of actions is conflicting:
  T1:R(X), T2:W(X), T3:W(X)

- While the following sets of actions are not:
  T1:R(X), T2:R(X), T3:R(X)  // No Write on same object
  T1:R(X), T2:W(Y), T3:R(X)  // No Write on same object
Conflict Equivalence
(Ισοδυναμία Συγκρούσεων)

- **Conflict Equivalence (Ισοδυναμία Συγκρούσεων)**
  The schedules $S_1$ and $S_2$ are said to be **conflict-equivalent** if the following conditions are satisfied:
  - Both schedules $S_1$ and $S_2$ involve the **same set** of transactions (including ordering of actions within each transaction).
  - The **order** (διάταξη) of each pair of conflicting actions in $S_1$ and $S_2$ are the same.

- **Why is the order of Conflicts important?** If two conflicting operations are **applied in different orders**, the **net effect** can be **different** on the database or on other transactions in the schedule. See example below:

  **Example:**
  
<table>
<thead>
<tr>
<th>T1</th>
<th>T2</th>
<th>NOT Conflict Equivalent to:</th>
</tr>
</thead>
<tbody>
<tr>
<td>$A=5$</td>
<td>R(A)</td>
<td>$A=5$</td>
</tr>
<tr>
<td>$A=A+1=6$</td>
<td>W(A)</td>
<td>$A=A+1=6$</td>
</tr>
<tr>
<td>$A=6$</td>
<td>R(A)</td>
<td>$A=6$</td>
</tr>
<tr>
<td>$A=A+1=7$</td>
<td>W(A)</td>
<td>$A=A+1=7$</td>
</tr>
<tr>
<td>(Alth. Result still same)</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

  **Order of conflict is different**

<table>
<thead>
<tr>
<th>T1</th>
<th>T2</th>
<th>NOT Conflict Equivalent to:</th>
</tr>
</thead>
<tbody>
<tr>
<td>$A=5$</td>
<td>R(A)</td>
<td>$A=5$</td>
</tr>
<tr>
<td>$A=A+1=6$</td>
<td>W(A)</td>
<td>$A=A+1=6$</td>
</tr>
<tr>
<td>$A=6$</td>
<td>R(A)</td>
<td>$A=6$</td>
</tr>
<tr>
<td>$A=A+1=7$</td>
<td>W(A)</td>
<td>$A=A+1=6$</td>
</tr>
</tbody>
</table>

  **Now Wrong Result!**

  **Order and conflicts are different**

  * Note that non-conflicting operations can arbitrary be **swapped around without compromising the order.**
Conflict Serializability

When the schedule is conflict-equivalent (ισοδύναμο συγκρούσεων) to some (any!) serial schedule.

 Serializable == Conflict Serializable

(that definition is in some textbooks different)

<table>
<thead>
<tr>
<th>Serial Schedule</th>
<th>Serializable Schedule A</th>
<th>Serializable Schedule B</th>
</tr>
</thead>
<tbody>
<tr>
<td>T1</td>
<td>T2</td>
<td>T1</td>
</tr>
<tr>
<td>R(A)</td>
<td>W(A)</td>
<td>R(A)</td>
</tr>
<tr>
<td>W(A)</td>
<td>R(B)</td>
<td>Order of conflicts same to T1; T2</td>
</tr>
<tr>
<td>R(B)</td>
<td>W(B)</td>
<td>R(B)</td>
</tr>
<tr>
<td>R(A)</td>
<td>W(A)</td>
<td>R(B)</td>
</tr>
<tr>
<td>W(A)</td>
<td>R(B)</td>
<td>R(B)</td>
</tr>
<tr>
<td>R(B)</td>
<td>W(B)</td>
<td>W(A)</td>
</tr>
</tbody>
</table>
Conflict Serializability
(Σειριοποιησιμότητα Συγκρούσεων)

- Why is Conflict Serializability important?
- We have already said that any serial schedule leaves the DB in a consistent (correct) state, but is inefficient—i.e., T1; T2 is as correct as T2; T1 (although they might have a different outcome).

Serializable != Serial: NOT the same thing

- Being Serializable implies:
  A. That the schedule is a correct schedule.
     - It will leave the database in a consistent state.
     - The interleaving is appropriate and will result in a state as if the transactions were serially executed.
  B. That a schedule is a efficient (interleaved) schedule
     - That parameter makes it better than Serial 😊!

EPL446: Advanced Database Systems - Demetris Zeinalipour (University of Cyprus)
Testing for Serializability
(Ελεγχος Σειριοποιησιμότητας)

- How can we test if a schedule is Conflict Serializable?
  - There is a simple algorithm detailed next (that is founded on a Precedence Graph)

- Does the DBMS utilizes this algorithm? NO
  - We detail it only to gain a better understanding of the definitions.

- Why is the DBMS not using it?
  - Serializability is hard to check at runtime
    - Difficult to determine beforehand how the operations in a schedule will be interleaved (as it depends on the OS)

- In the next lecture, we will see that a DBMS utilizes a set of protocols (e.g., 2PL or other Concurrency Control techniques w/out locking) that guarantee that a schedule is always serializable.
Precedence Graph
(Γράφος Προτεραιότητας)

- **Why is it useful?** To find if a schedule is Conflict Serializable
- A **Precedence Graph** (Γράφος Προτεραιότητας) for a schedule S contains:
  - A node for each committed transaction in S
  - An arch from $T_i$ to $T_j$, if an action of $T_i$ precedes (προηγείται) and conflicts (συγκρούεται) with one of $T_j$'s actions.

A schedule $S$ is **conflict serializable** if and only if its **precedence graph** is **acyclic**.
- The above schedule is **not** Conflict Serializable!
Conflict Serializability Testing
(Έλεγχος Σειριοποιησιμότητας Συγκρούσεων)

Characterizing Schedules based on:
- Serializability
- Recoverability

<table>
<thead>
<tr>
<th>Transaction $T_1$</th>
<th>Transaction $T_2$</th>
<th>Transaction $T_3$</th>
</tr>
</thead>
<tbody>
<tr>
<td>read_item($X$); write_item($X$);</td>
<td>read_item($Z$); write_item($Z$);</td>
<td>read_item($Y$); write_item($Y$);</td>
</tr>
<tr>
<td>read_item($Y$); write_item($Y$);</td>
<td>read_item($Y$); write_item($Y$); read_item($X$); write_item($X$);</td>
<td></td>
</tr>
</tbody>
</table>

Schedule $F$

Above schedule is NOT Conflict Serializable!
Although efficient (interleaved) the above will NOT produce a correct result!

Precedence Graph

Cycles exist!

EPL446: Advanced Database Systems - Demetris Zeinalipour (University of Cyprus)
Let us now relax (χαλαρώσουμε) the definition of equivalence (and serializability) of schedules by introducing the notion of view equivalence serializability.

In View Equivalence, respective transactions in the two schedules read and write the same data values ("view" the same data values).

…. While in Conflict Equivalence, respective transactions in two schedules have the same order of conflicting actions.

Formal definitions and examples will follow…
• **View Serializable (Σειριοποιησιμότητα Όψεων):**
  – A Schedule is **View Serializable** if it is **view equivalent** to some **serial schedule**

**VENN Diagram**

- Conflict serializable => View serializable, but not vice versa.
- **Testing** for View Serializability is **NP-hard**, meaning that finding an efficient polynomial time algorithm to this problem is highly unlikely.
  - We will only work on small examples for which we can characterize the type of serializability by observation (so no algorithm is necessary).
View Equivalence (Formal Definition)

1. **Same WR order**: If in S1: \( w_j(A) \Rightarrow r_i(A) \); \( i,j \) are identifiers of Transactions then in S2: \( w_j(A) \Rightarrow r_i(A) \)

2. **First Read**: If in S1: \( r_i(A) \) reads initial DB value, then in S2: \( r_i(A) \) also reads initial DB value

3. **Last Write**: If in S1: \( w_i(A) \) does last write on A, then in S2: \( w_i(A) \) also does last write on A

• The premise behind view equivalence:
  – "The view": the read operations are said to see the same view in both schedules.
  – **Rule**: As long as each read operation of a transaction reads the result of the same write operation in both schedules, the write operations of each transaction must produce the same results.
View Serializability
(Σειριοποιησιμότητα Όψεων)

- **View Serializability Summary:**
  
  I. Same Transaction **Reads Data First.**
  II. Same Transaction **Writes Data Last.**
  III. Same **WR Order** of actions.

b) **Conflict & View** Serializable Schedule (also serial)

<table>
<thead>
<tr>
<th>T1</th>
<th>T2</th>
<th>T3</th>
</tr>
</thead>
<tbody>
<tr>
<td>R(A)</td>
<td>W(A)</td>
<td>W(A)</td>
</tr>
<tr>
<td>W(A)</td>
<td>W(A)</td>
<td></td>
</tr>
</tbody>
</table>

Why is the second schedule View Serializable?
In both schedules (a) and (b): First Read by T1 and Final Write performed by T3 and WR order does not change (actually we have no reads in T2 and T3)

Order of conflicts not same with any serial schedule

c) **View** Serializable but **NOT Conflict** Serializable

<table>
<thead>
<tr>
<th>T1</th>
<th>T2</th>
<th>T3</th>
</tr>
</thead>
<tbody>
<tr>
<td>R(A)</td>
<td>W(A)</td>
<td></td>
</tr>
<tr>
<td>W(A)</td>
<td>W(A)</td>
<td></td>
</tr>
</tbody>
</table>
• So far we have characterized schedules based on serializability (σειριοποιησιμότητα), i.e., correctness.

• Now it is time to characterize schedules based on recoverability (επαναφερσιμότητα)

• Why is this important?
  – For some schedules it is easier to recover from transaction failures than others.

• In summary, a Recoverable Schedule (Επαναφέρσιμο Χρονοπρόγραμμα) is a schedule where no transaction needs to be rolled back (διαδικασία επιστροφής) once committed.

• Commit/Abort points now become quite important!
Recoverable Schedule
(Επαναφέρσιμο Χρονοπρόγραμμα)

• Recoverable Schedule (Επαναφέρσιμο Χρονοπρόγραμμα)
  – A schedule $S$ is recoverable if no transaction $T$ in $S$ commits until all transactions $T'$, that have written an item that $T$ reads, have committed.

• Rule: In other words, the parents of dirty reads need to commit before their children can commit.

Consider the Following schedule:

<table>
<thead>
<tr>
<th></th>
<th>T1</th>
<th>T2</th>
</tr>
</thead>
<tbody>
<tr>
<td>R(X)</td>
<td>W(X)</td>
<td>R(X)</td>
</tr>
<tr>
<td>W(X)</td>
<td></td>
<td>W(X)</td>
</tr>
<tr>
<td>R(Y)</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Is this schedule recoverable? Answer: NO

Why NOT recoverable?
• Because $T2$ made a dirty read and committed before $T1$ … next slide explains why this is a problem …
Recoverable Schedule

(Επαναφέρσιμο Χρονοπρόγραμμα)

• But why is the schedule **Nonrecoverable (Μη-επαναφέρσιμο)**?
  • Because when the **recovery manager** rolls back (step a) **T1** then A gets its initial value.
  • But T2 has already utilized this wrong value and committed something to the DB
  • The DB is consequently in an inconsistent state!

```plaintext
T1  | T2
---|---
R(X) W(X) |commit
R(Y) W(X) |Abort
```

B) Now DB is inconsistent! (because X was committed based on T1’s aborted transaction)
Recoverable Schedule

(Επαναφέρσιμο Χρονοπρόγραμμα)

- How can we make the Schedule Recoverable?

<table>
<thead>
<tr>
<th>Recoverable Schedule A</th>
<th>Recoverable Schedule B</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>T1</strong></td>
<td><strong>T2</strong></td>
</tr>
<tr>
<td>R(X)</td>
<td>W(X)</td>
</tr>
<tr>
<td>W(X)</td>
<td>R(Y)</td>
</tr>
<tr>
<td>R(X)</td>
<td>Commit</td>
</tr>
<tr>
<td>R(Y)</td>
<td>(order of conflicting actions has changed)</td>
</tr>
</tbody>
</table>

Initial Unrecoverable Schedule

<table>
<thead>
<tr>
<th>T1</th>
<th>T2</th>
</tr>
</thead>
<tbody>
<tr>
<td>R(X)</td>
<td>dirty read</td>
</tr>
<tr>
<td>W(X)</td>
<td>R(X)</td>
</tr>
<tr>
<td>R(Y)</td>
<td>W(X)</td>
</tr>
<tr>
<td>Abort</td>
<td>Commit</td>
</tr>
</tbody>
</table>

Nonrecoverable

Recoverable

EPL446: Advanced Database Systems - Demetris Zeinalipour (University of Cyprus) 14-22
Other Schedules based on Recoverability
(Άλλα χρονοπρογράμματα βάση Επαναφερσιμότητας)

- There are **more strict types** of Schedules (based on the **Recoverability properties**)
  - **Cascadeless schedule** (Χρονοπρόγραμμα χωρίς διαδιδόμενη ανάκληση):
    (or Schedule that **Avoids Cascading Rollbacks**)
    - Refers to cases where we have aborts.
  - **Strict Schedules** (Αυστηρό Χρονοπρόγραμμα)
    - These schedules are very simple to be recovered!
    - Thus, the DBMS prefers this class of Schedules.

![Diagram of schedule types]

- All schedules
  - Recoverable
    - Avoid Cascading Aborts
      - Strict
      - Serial
Cascadeless Schedule (Χρονοπρόγραμμα χωρίς διαδιδόμενη ανάκληση):
• Cascadeless schedule (Χρονοπρόγραμμα χωρίς
diadiđómeno anákληση): or Schedule that Avoids
Cascading Rollbacks)
  – One where a rollback does not cascade to other Xacts
  – Why is this necessary? Rollbacks are Costly!
  – How can we achieve it? Every transaction reads only
    the items that are written by committed transactions.

T1
R(X)  W(X)
R(Y)  W(Y)
Abort

T2
W(X)
R(X)
Commit

a) dirty read
b) Roll back
c) (Cascade)
We need to Roll back T2 as well!

NOT Cascadeless (but Recoverable)
Let us turn the previous example into a Cascadeless Schedule:

- Recall, in order to get a Cascadeless Schedule, every transaction **must read only committed data**.

#### Cascadeless Schedule

<table>
<thead>
<tr>
<th>T1</th>
<th>T2</th>
</tr>
</thead>
<tbody>
<tr>
<td>R(X)</td>
<td>R(X)</td>
</tr>
<tr>
<td>W(X)</td>
<td>W(X)</td>
</tr>
<tr>
<td>R(Y)</td>
<td>R(Y)</td>
</tr>
<tr>
<td>W(Y)</td>
<td>W(Y)</td>
</tr>
<tr>
<td>Abort</td>
<td></td>
</tr>
</tbody>
</table>

- **a) Roll back**
- **b) Now T2 reads the X value that existed before T1 started.**

#### Another Cascadeless Schedule

<table>
<thead>
<tr>
<th>T1</th>
<th>T2</th>
</tr>
</thead>
<tbody>
<tr>
<td>R(X)</td>
<td>R(X)</td>
</tr>
<tr>
<td>W(X)</td>
<td>W(X)</td>
</tr>
<tr>
<td>R(Y)</td>
<td>R(Y)</td>
</tr>
<tr>
<td>W(Y)</td>
<td>W(Y)</td>
</tr>
<tr>
<td>Abort</td>
<td>W(Y)</td>
</tr>
<tr>
<td></td>
<td>Commit</td>
</tr>
</tbody>
</table>

- **… but not strict**
Strict Schedule
(Aυστηρό Χρονοπρόγραμμα)

- **Strict Schedule (Αυστηρό Χρονοπρόγραμμα):**
  - A schedule is strict if overriding of uncommitted data is not allowed.
  - Formally, if it satisfies the following conditions:
    - Ti reads a data item X after Ti has terminated (aborted or committed)
    - Tj writes a data item X after Ti has terminated (aborted or committed)

- Why is this necessary? **Eliminates Rollbacks!**
  - If a schedule is strict, a rollback can be achieved simply by resetting the Xact variables to the value before its start value, e.g.,

---

<table>
<thead>
<tr>
<th></th>
<th>T1</th>
<th>T2</th>
</tr>
</thead>
<tbody>
<tr>
<td>// X=9</td>
<td></td>
<td>W(X,5)</td>
</tr>
<tr>
<td>W(X,8)</td>
<td>Commit</td>
<td></td>
</tr>
</tbody>
</table>

b) Rollback
X=9 now

---

a) Changing an item that has not been committed yet (X=8 now)

---

c) Why is this a problem?
Because X now became again 9 rather than X=8 or X=5!
Characterizing Schedules based on:
Serializability
Recoverability

- **Venn Diagram** Illustrating the Different ways to characterize a Schedule based on Serializability and Recoverability