# 2nd LAB SESSION ON TRANSACTIONS

**Given Name: ................................. Family name: ....................................................................**

**Given Name: ................................. Family name: ....................................................................**

1. (40%) Consider the No Steal / Force policy:

1. Provide the pseudo code of the *read*, *write*, *commit* and *abort* operations, so that we guarantee recoverability in **case of power failure**. Use as basis those in pages 19 and 20 for the steal / no force policy.
2. Under what circumstances that policy may be interesting (e.g., What are its **cons and pros**? **What kind of systems** can you think of that would suit it?)

.........................................................................................................................................................................................................................................................................................................................................................................................................................................................................................................................................................................................................................................................................................

1. (30%) Given a DBMS without any concurrency control mechanism, let’s suppose that we have the following history (actions have been numbered just to facilitate referencing them):

|  |  |  |  |
| --- | --- | --- | --- |
| **#Acc** | **T1** | **T2** | **T3** |
|  |  |  |  |
| 10 |  |  | BoT |
| 20 |  | BoT |  |
| 30 | BoT |  |  |
| 40 |  | R(E) |  |
| 50 | R(A) |  |  |
| 60 | W(A) |  |  |
| 70 |  |  | R(A) |
| 80 |  |  | W(A) |
| 90 | R(F) |  |  |
| 100 | R(D) |  |  |
| 110 | R(E) |  |  |
| 120 | W(E) |  |  |
| 130 |  | R(C) |  |
| 140 |  | W(C) |  |
| 150 |  | R(E) |  |
| 160 |  |  | R(F) |
| 170 |  |  | W(F) |
| 180 |  | COMMIT |  |
| 190 | COMMIT |  |  |
| 200 |  |  | COMMIT |

Let’s suppose now that the DBMS is based on an **optimistic technique** that validates readings at commit time. How would result the same history? **Is any transaction cancelled?**

1. (30%) Given a DBMS without any concurrency control mechanism, let’s suppose that we have the following history (actions have been numbered just to facilitate referencing them):

|  |  |  |  |
| --- | --- | --- | --- |
| **#Acc** | **T1** | **T2** | **T3** |
|  |  |  |  |
| 10 |  |  | BoT |
| 20 |  | BoT |  |
| 30 | BoT |  |  |
| 40 |  | R(E) |  |
| 50 | R(A) |  |  |
| 60 | W(A) |  |  |
| 70 |  |  | R(A) |
| 80 |  |  | W(A) |
| 90 | R(F) |  |  |
| 100 | R(D) |  |  |
| 110 | R(E) |  |  |
| 120 | W(E) |  |  |
| 130 |  | R(C) |  |
| 140 |  | W(C) |  |
| 150 |  | R(E) |  |
| 160 |  |  | R(F) |
| 170 |  |  | W(F) |
| 180 |  | COMMIT |  |
| 190 | COMMIT |  |  |
| 200 |  |  | COMMIT |

Let’s suppose now that the DBMS is based on a **dynamic timestamping** technique. How would result the same history? **Is any transaction cancelled?**