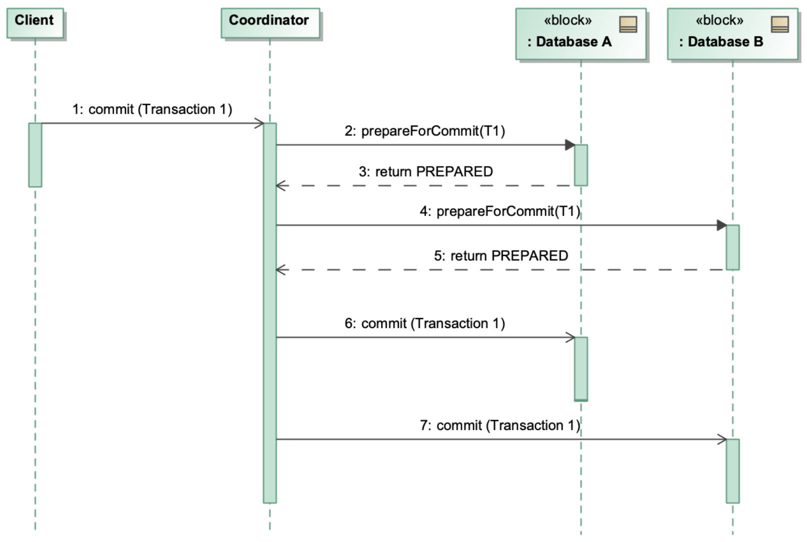
2-Phase Commit



2.1 Overview

A **transaction** can be defined as a very small unit or task of a program which cannot be further divided in to smaller sub-tasks. A transaction must maintain Atomicity, Consistency, Isolation, and Durability – commonly known as **ACID** properties – in order to ensure accuracy, completeness, and data integrity. **Distributed transactions** are transactions in which two or more network hosts are involved. We will refer to these hosts as *COHORTS*.

The **Two-phase commit protocol (2PC)** is a distributed algorithm that coordinates all the *COHORTS* that participate in a distributed transaction on whether to **commit** or **abort** the transaction. The protocol is fault-tolerant i.e, the protocol achieves its goal even in cases of temporary system failure.

2.2 Intended Service Informally Stated

The 2 Phase Commit service has four functions that can be called by local users in the environment

- `start_process()` : `COORDINATOR` object attempts to prepare all the transaction's participating cohorts
  - returns `COMMIT` or `ABORT`
- `commit():` `COMMIT` to the database
  - can only be called if `start_process()` returns `COMMIT`
  - returns `true` or `false` on completion
- `abort():` Undo all the work done in the database
  - can only be called if `start_process()` returns `ABORT`
  - returns `true` or `false` on completion

- `end_process()` : `COHORTS` object attempts to end the process
  - can only be called if `put()` or `rollback()` returns `true`
  - returns `COMPLETE` or `ERROR` on execution

2.3 Implementation

Global variables

`Thread` will simulate transcation-like behavior

`NUM_COHORTS` will define the number of *COHORTS* (>=2)

The two classes for this service are defined as follows:

`COORDINATOR:`

- `__init__()`
  - initializes the `COORDINATOR` object

- `start_voting_process()`
  - begins with the voting phase of the process

- `request_prepare_COHORTS()`
  - command `COHORTS` to start preparing for transaction
  - returns `YES/COMMIT` or `NO/ABORT` in case of a local failure

- `request_commit_COHORTS()`
  - command `COHORTS` to begin the `COMMIT` process
  - returns `SUCCESS` or `FAILURE` in case of a local failure

- `request_rollback()`
  - command `COHORTS` to begin the `ABORT` process
  - returns `SUCCESS` or `FAILURE` in case of a local failure

`COHORT:`

- `__init__()`
  - initializes the `COHORT` object

- `prepare()`
  - prepare *COHORT* object to start preparing for the transaction
  - return `YES/COMMIT` or `NO/ABORT`

- `commit_COHORT()`
  - commit *COHORT* object
  - return `SUCCESS` or `FAILURE`

- `rollback_COHORT()`
  - rollback each commit to its previous state i.e. undo the work for the *COHORT*

    - rollback each commit to its previous state i.e., undo the work for the COHORT
    - return `SUCCESS` or `FAILURE`
- `end_self`
    - end the Thread running this *COHORT*

```
from threading import get_ident

class Service():
    def __init__(self):
        Thread.__init__(self)
        for i in NUM_COHORTS
            self.COHORTS = COHORTS(i) # initialize COHORTS

        coord = COORDINATOR # coordinator that assists with the service
        votes = False

    def start_process(self):
        CIC: i in self.COHORTS    # input part
        CU:  self.COHORTS(i).prepare()
        ROC: all(self.votes == True) # output part
        RU:  pass

    def commit(self):
        CIC: self.votes == True
        CU:  request_commit_cohorts(self.COHORTS)
        ROC: all(self.COHORTS.commit() == True) # output part
        RU:  pass

    def abort(self):
        CIC: self.votes == False
        CU:  request_rollback_cohorts(self.COHORTS)
        ROC: all(self.COHORTS.rollback() == True) # output part
        RU:  pass

    def end_process(self):
        CIC: rollback() == True || commit == True
        CU:  pass
        ROC: pass
        RU:  self.COHORTS(i).end()
```

The class has four non-init functions: `start_process()` , `commit()` , `abort()` and `end_process()` , corresponding to the four functions called by users of the service. The **init** function defines variables adequate to express when a non-init function can be called and when it can return.

Every non-init function has four components: CIC, short for call input condition; CU, short for call update; ROC, short for return output condition; and RU, short for return update.

In [ ]:

```
In [ ]:
```