

122022010 Shrayank Mistry

Description: Simulate recovery using undo, redo and undo/redo logging.

Programming Language: Python

```
1  from tabulate import tabulate
2
3  /* GLOBAL VARIABLES
4  transactions_dict = {}
5  temp_values = {}
6  main_disk = {
7      'A': 0,
8      'B': 0,
9      'C': 0,
10     'D': 0,
11     'E': 0,
12     'F': 0,
13 }
14 cache_disk = {}
15 class transaction_data:
16     def __init__(self, transaction_id):
17         self.transaction_id = transaction_id
18         self.data_items = list()
19
20     def get_value(ins, a, b):
21         if (ins.find('+') > -1): return a + b
22         if (ins.find('-') > -1): return a - b
23         if (ins.find('*') > -1): return a * b
24         if (ins.find('/') > -1): return a / b
25
26     def load_data_to_cache(main_disk, cache_disk):
27         for key, value in main_disk.items():
28             cache_disk[key] = value
29
30     def read_transaction_file(filename):
31         file = open(filename, "r")
32         data = []
33
34         for line in file:
35             data.append(line.strip())
36
37         instrus_cnt = int(data[0])
38         instructions = data[1:]
39         return instrus_cnt, instructions
```



```
1  if __name__ == "__main__":
2      filename = "transaction-undo-redo.txt"
3      instrus_cnt, instructions = read_transaction_file(filename)
4
5      load_data_to_cache(main_disk, cache_disk)
6
7      /* Deferred Updation (0), Immediate Updation (1) */
8      type_of_updation = 1
9      create_log(instrus_cnt, instructions, type_of_updation)
10
11     cnt = 1
12     data = []
13
14     data_items = list(cache_disk.keys())
15     cache_values = list(cache_disk.values())
16     disk_bc = list(main_disk.values())
17
18     log_filename = "log-undo-redo.txt"
19     simulate_crash(log_filename)
20
21     disk_ac = list(main_disk.values())
22
23     while cnt <= len(data_items):
24         data.append([cnt, data_items[cnt - 1], cache_values[cnt - 1],
25                     disk_bc[cnt - 1], disk_ac[cnt - 1]])
26         cnt += 1
27
28     str = ''
29     if type_of_updation == 0:
30         str = "Defered Update"
31     if type_of_updation == 1:
32         str = "Immediate Update"
33
34     print("Type of Updation Policy = " + str)
35     print(tabulate(data, headers=["No.", "Data Item", "Cache", "Disk
36 [Before Crash]", "Disk[After Crash]"])))
```



```
1 def create_log(instrus_cnt, instructions, type):
2     log_file = open("log-undo-redo.txt", "w")
3     for i in range(instrus_cnt):
4         ins = instructions[i]
5
6         /* Normal Operations
7         if (ins.find('(') != -1):
8             operation = ins[0:ins.find('(')]
9             if (operation == "BEGIN"):
10                 transaction = ins[ins.find('(') + 1:ins.find(')')]
11                 log_file.write(f"<START {transaction}>\n")
12                 transactions_dict[transaction] = transaction_data(transaction)
13
14             if (operation == 'READ'):
15                 data_item = ins[ins.find('(') + 1:ins.find(',')]
16                 transaction = ins[ins.find(' ') + 1:ins.find(')')]
17                 # log_file.write(f"<{transaction} READ {data_item}>\n")
18                 temp_values[data_item] = cache_disk[data_item]
19
20             if (operation == 'WRITE'):
21                 data_item = ins[ins.find('(') + 1:ins.find(',')]
22                 transaction = ins[ins.find(' ') + 1:ins.find(')')]
23                 log_file.write(f"<{transaction} {data_item} {cache_disk[data_item]}
24 {temp_values[data_item]}>\n")
25                 cache_disk[data_item] = temp_values[data_item]
26                 if type == 1:
27                     main_disk[data_item] = cache_disk[data_item]
28                 transactions_dict[transaction].data_items.append(data_item)
29
30             if (operation == 'COMMIT'):
31                 transaction = ins[ins.find('(') + 1:ins.find(')')]
32                 log_file.write(f"<COMMIT {transaction}>\n")
33                 data_list = transactions_dict[transaction].data_items
34
35                 if type == 0:
36                     for d in data_list:
37                         main_disk[d] = cache_disk[d]
```



```
1  /* Arithmetic Operations
2      else:
3          updating_data = ins[0]
4          a = ins[ins.find('(') + 1]
5
6          /* Code to Refactor
7          b = ''
8          index = ins.find(')') - 1
9          while (ins[index] != ' '):
10             b += ins[index]
11             index -= 1
12             b = b[::-1]
13
14         try:
15             temp_b = int(b)
16             b = temp_b
17             a = cache_disk[a]
18         except:
19             /* GET BOTH a and b FROM MAIN DISK
20             a = cache_disk[a]
21             b = cache_disk[b]
22             current_value = cache_disk[updating_data]
23             updated_value = get_value(ins, a, b)
24             # log_file.write(f"<{ins[ins.find('_') + 1:]} {updating_data} {current_value} {updated_value}>\n")
25             temp_values[updating_data] = updated_value
26             continue
27     log_file.close()
```




```
1 def simulate_crash(log_file):
2     committed_transactions = dict()
3     uncommitted_transactions = dict()
4     started_transactions = dict()
5
6     commit_set = set()
7     uncommit_list = list()
8     started_set = set()
9
10    /* Get all started transactions
11    file = open(log_file, "r")
12    for line in file:
13        if line.find('START') == 1:
14            started_transactions[line[line.find(' ') + 1:line.find('>')]] = list()
15            started_set.add(line[line.find(' ') + 1:line.find('>')])
16    file.close()
17
18    /* Get all committed transactions
19    file = open(log_file, "r")
20    for line in file:
21        if line.find('COMMIT') == 1:
22            committed_transactions[line[line.find(' ') + 1:line.find('>')]] = list()
23            commit_set.add(line[line.find(' ') + 1:line.find('>')])
24    file.close()
25
26    uncommit_list = list(started_set - commit_set)
27
28    for t in uncommit_list:
29        uncommitted_transactions[t] = list()
30
31    #TODO For all Committed Transactions (Redo)
32    #TODO For all Uncommitted Transactions (Undo)
```



```
1 file = open(log_file, "r")
2     for line in file:
3         line_temp = line.replace('>', '').replace('<', '').replace('\n', '').split(' ')
4         if len(line_temp) == 4:
5             transaction_id = line_temp[0]
6             if committed_transactions.get(transaction_id) == None: /* Uncommitted Ts
7                 uncommitted_transactions[transaction_id].append(line.replace('\n', ''))
8             else:
9                 committed_transactions[transaction_id].append(line.replace('\n', '')) /*
Committed Ts
10
11     log_file = open(log_file, "a+")
12     /* Perform forward updates
13     for tid, operations in committed_transactions.items():
14         for op in operations:
15             op = op.replace('>', '').replace('<', '').split(' ')
16             transaction_id, data_item, old_value, new_value = op[0:]
17             main_disk[data_item] = new_value
18             log_file.write(f"<END {tid}>\n")
19
20     /* Perform Backward updates
21     for tid, operations in uncommitted_transactions.items():
22         operations = operations[::-1]
23         for op in operations:
24             op = op.replace('>', '').replace('<', '').split(' ')
25             transaction_id, data_item, old_value, new_value = op[0:]
26             main_disk[data_item] = old_value
27             log_file.write(f"<ABORT {tid}>\n")
28
29     file.close()
```

```

1  TODO ----- OUTPUT ----- #
2  * Type of Updation Policy = Deferred Update
3  * No.  Data Item      Cache    Disk[Before Crash]    Disk[After Crash]
4  * -----
5  *      1  A           60        60        60
6  *      2  B           50        50        50
7  *      3  C           50        50        50
8  *      4  D          100         0         0
9  *      5  E           20         0         0
10 *      6  F           10         0         0
11
12 ? Type of Updation Policy = Immediate Update
13 ? No.  Data Item      Cache    Disk[Before Crash]    Disk[After Crash]
14 ? -----
15 ?      1  A           60        60        60
16 ?      2  B           50        50        50
17 ?      3  C           50        50        50
18 ?      4  D          100       100         0
19 ?      5  E           20        20         0
20 ?      6  F           10        10         0
21
22 TODO ----- #

```

```

1  29
2  BEGIN(T1)
3  READ(A, T1)
4  READ(B, T1)
5  A = [A + 50]_T1
6  BEGIN(T2)
7  WRITE(A, T1)
8  B = [B + A]_T1
9  WRITE(B, T1)
10 COMMIT(T1)
11 READ(A, T2)
12 A = [A + 10]_T2
13 READ(C, T2)
14 C = [C + 50]_T2
15 WRITE(A, T2)
16 WRITE(C, T2)
17 BEGIN(T3)
18 READ(D, T3)
19 COMMIT(T2)
20 D = [D + 100]_T3
21 WRITE(D, T3)
22 BEGIN(T4)
23 READ(E, T4)
24 E = [E + 10]_T4
25 WRITE(E, T4)
26 READ(F, T4)
27 F = [F + E]_T4
28 WRITE(F, T4)
29 E = [E + F]_T4
30 WRITE(E, T4)

```

```

1  <START T1>
2  <START T2>
3  <T1 A 0 50>
4  <T1 B 0 50>
5  <COMMIT T1>
6  <T2 A 50 60>
7  <T2 C 0 50>
8  <START T3>
9  <COMMIT T2>
10 <T3 D 0 100>
11 <START T4>
12 <T4 E 0 10>
13 <T4 F 0 10>
14 <T4 E 10 20>
15 <END T1>
16 <END T2>
17 <ABORT T4>
18 <ABORT T3>

```