Assignment 10

Question 1

For each of the following two schedules: S1: R1(x); R4(x); R3(z); R1(x); W3(z); R3(y); R2(z); W3(y); R1(z); R4(y); W4(y); R2(y); W4(x); W2(y); S2: R1(x); R1(y); W3(z); R3(y); W4(z); R2(z); R4(x); R3(x); W4(x); W2(z); W1(z); R2(y); W2(x); R3(x);

Answer the following questions:

Question 1a

Draw the precedence (serialization) graph for the schedule. Schedule 1:

Schedule 2:

Question 1b

Is the schedule conflict-serializable? If your answer is yes, list all of its equivalent serial schedules. If your answer is no, how can we abort the least number of transaction(s) so that the rest of the schedule becomes conflict-serializable?

S1 is conflict serializable because it is **acyclic**.

The equivalent serial schedules to S1 are: T3 T1 T4 T2

- Look at in-degree of vertices, T3 degree is 0, T1 is 1, T4 and T2 is 2.
- Also look at which vertices are pointing to each other to determine order.

S2 is **not** conflict serializable because it is cyclic.

The least number of transactions to abort to make the schedule conflict serializable is 2. We can make S2 conflict serializable by either:

- Aborting T2 and T4, or
- Aborting T1 and T3

Question 1c

Could the original or updated (by aborting some transactions) conflict serializable schedule be recoverable and/or cascadeless? Justify your answer.

S1 could be recoverable. Recoverable S1 example: R1(x); R4(x); R3(z); R1(x); W3(z); R3(y); R2(z); W3(y); R1(z); R4(y); W4(y); R2(y); W4(x); W2(y); E3; E1; E4; E2;

- The order of E1 and E4 doesn't matter
- If T3 decides to abort, all other transactions must abort as they all read dirty data from T3
- If T4 decides to abort, T2 has to abort because it read dirty data from T2

S1 **could not** be cascadeless because transaction 2 tries to read object z, which was written to by transaction 3, however transaction 3 has not committed since it has to write to object y after transaction 2 reads z, meaning transaction 2 is reading dirty data from an uncommitted transaction.

For S2, we have two updated schedules by aborting transactions: S2V1 could be recoverable. Recoverable S2V1, aborting T2 and T4: R1(x); R1(y); W3(z); R3(y); R3(x); W1(z); R3(x); E3; E1;

- This one is recoverable no matter the order of E1 and E3
- If either one aborts, the other one doesn't have to abort since dirty data is never read.

S2V2 could be recoverable. Recoverable S2V2, aborting T1 and T3: W4(z); R2(z); R4(x); W4(x); W2(z); R2(y); W2(x); E4; E2;

- E4 must be before E2
- If T4 aborts, T2 must abort as T2 reads dirty data from T4

S2V1 could be cascadeless. Cascadeless S2V1, aborting T2 and T4: R1(x); R1(y); W3(z); R3(y); R3(x); W1(z); E1; R3(x); E3;

- Since you never read dirty data in S2V1, it could be cascadeless.

S2V2 **could not** be cascadeless because T4 writes to z and T2 reads z after, but T4 can't commit before T2 reads x because it has operations to do

afterwards.

Question 2

Suppose the following log records (except the CHECKPOINT) are added into the memory copy of a log file, where T1 to T4 are transactions, and u, v, w, x are database objects.

Question 2a

If WAL protocol is adopted, which record(s) is/are guaranteed to cause the log file records be flushed out to the disk?

In WAL, the only operations that cause records to be flushed out to the disk are 'CHECKPOINT' and 'COMMIT'. Since 'CHECKPOINT' is ignored in this question, 'T3, COMMIT' is the only operation that guarantees the log records to be flushed out to the disk.

Question 2b

Suppose the above records are also the ones loaded into the database system after a system crash happened, and the recovery manager uses ARIES algorithm to recover from the crash, what would be the values stored in the database objects u, v, w, and x respectively after the recovery is successfully completed?

When system accidentally crashes and we need to restart the system: Let R = emptySet

- scan LOG to find all active and committed transactions
- scan LOG backwards for records [Ti, x, v, v'] and for each such record such that x not in R:
    - read x; if Ti was committed then write v' into x and put x to R; if Ti was active then write v into x;
- when done, write abort and end records for all active transactions
- restart completed

  Restart operation is idempotent (if system crashes during restart, we don't mind)

Using ARIES algorithm:

R = {} **Scan LOG to find all active and committed transactions:**

- T1, T3, and T4.

Scan LOG backwards for records [Ti, x, v, v'] and for each such record such that x not in R:

- **read x;
- **if Ti was committed then write v' into x and put x to R;**
- **if Ti was active then write v into x;****

  w = 15, u = 31.5, x = 35, v = 'FALSE' R = {x}

We don't care about the rest of the algorithm, we got the values.

The values stored in the database after recovery: u = 31.5, v = 'FALSE', w = 15, x = 35

Question 3

A database table, Employees, has many columns. Two of these columns are Emp_ID and Last_Name. The following 28 tuples are inserted into this table:

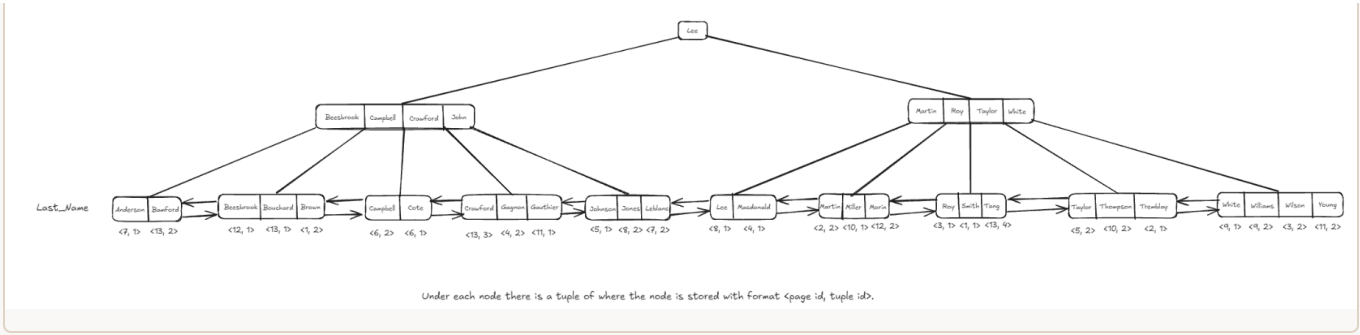
Emp_ID	Last_Name	other columns
1001	Smith	...
1002	Brown	...
1003	Tremblay	...
1004	Martin	...
1005	Roy	...
1006	Wil	

You are asked to construct two B+ tree indices on the table Employees. One is a primary/sparse/clustered index using Emp_ID as the search key, and the other a secondary/dense/unclustered index using Last_Name as the search key.

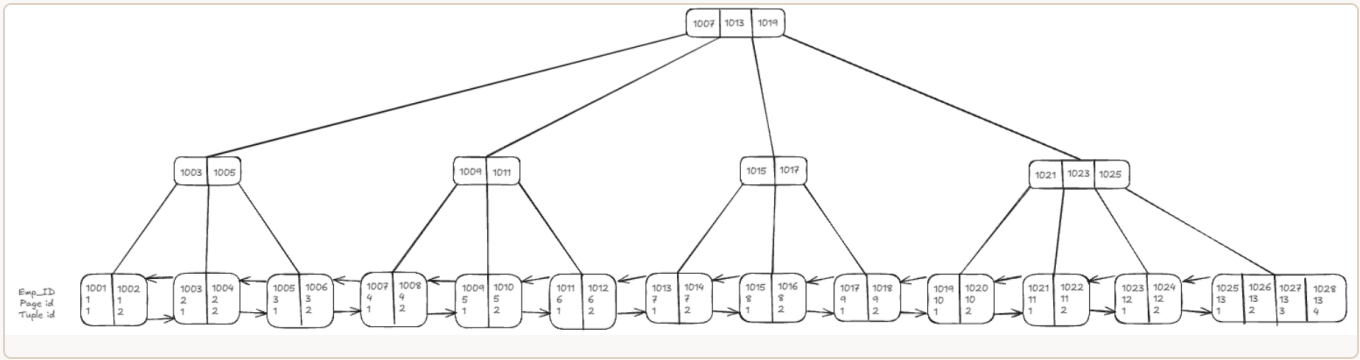
Suppose that, no matter what type of data to be stored in each page, for both trees, each leaf node can hold at most 4 data items and 2 page pointers, and each internal node can hold at most 4 search keys and 5 page pointers.

Question 3a

Draw these two above mentioned B+ tree indices. Primary index:



Secondary index:



Question 3b

Identify the nodes (in your index trees and data pages) that may be loaded into memory when the database executes the following query

efficiently by using the appropriate index:

```sql
select *
from Employees
where Last_Name = 'Moss';
```

Nodes:

- Root node containing 'Lee'.
- Internal node containing 'Martin', 'Roy', 'Taylor', and 'White'
- Leaf node containing 'Martin', 'Miller', and 'Morin' Data pages:
- No data pages are accessed as 'Moss' is not present in the tree.

Question 3c

Identify the nodes (in your index trees and data pages) that may be loaded into memory when the database executes the following query efficiently by using the appropriate index:

```sql
select *
from Employees
where Emp_ID = '1006';
```

Nodes:

- Root node containing 1007, 1013, 1019
- Internal node containing 1003, 1005
- Leaf node containing 1005, 1006 Data pages:

- Since 1006 was found in the 3rd leaf node, data page 3 is accessed.

Question 3d

Describe what would happen (in the data pages and indices) if the following SQL statement is executed:

```sql
insert into Employees (Emp_ID, Last_Name)
values ('1029', 'O\'Brian');
```

First, we must update the primary index:

- Here is a picture of the updated tree, only the bottom right section.

- Here is a picture of the updated secondary index tree, notice that the tuples on the leaf nodes linking to page and tuple ids are updated to reflect the changes in the primary index tree.