Problem Set 5, Part I

Problem 1: Data models for a NoSQL document database

1-1)

Author Document
```
{
    _id: <44777>,
    name: “Ina Garten”
    dob: “1948-02-02”
}
```

Book Document
```
{
    _id: <9781984822789>,
    author_id: <44777>,
    title: “Go-To Dinners: A Barefoot Contessa Cookbook”,
    publisher: “Clarkson Potter”,
    num_pages: 256,
    genre: “cookbook”,
    numInStock: 30
}
```

Sales Document
```
{
    id: <ObjectId1>,
    book_id: <9781984822789>,
    date: “2022-11-22”,
    time: “11:00”,
    numSold: 1
}
```

1-2)

Need more documents, but less references

1-3)

Author document
```
{
    _id: <44777>,
    name: “Ina Garten”
    dob: “1948-02-02”
}
```

Book Document
```
{
    _id: <9781984822789>,
    author_id: <44777>,
```

```
        title: “Go-To Dinners: A Barefoot Contessa Cookbook”,
        publisher: “Clarkson Potter”,
        num_pages: 256,
        genre: “cookbook”,
        numInStock: 30,
        sales:
        {
                date: “2022-11-22”,
                time: “11:00”,
                numSold: 1
        }
}
```

1-4)
Need less documents, but more references

1-5)
When the bookseller attempts to sell one or more copies of a given book, there would be multiple documents needed for each copy sold of a given book, which means we need to create a new document each time. If an update makes a document bigger than the space allocated for it on disk, it may need to be relocated. Therefore, we use references if embedded documents could lead to significant growth in the size of the document over time. In this particular use of the database, the sales embedded documents could lead to significant growth in the size of the document each time the bookseller attempts to update the sales data, so the reference-based approach from part 1 is preferred

Problem 2: Logging and recovery

2-1) undo-redo

	possible on-disk
A	100, 110
B	200, 210, 220
C	300, 310
D	400, 410, 420

2-2) redo-only

	possible on-disk
A	100
B	200, 210
C	300
D	400

2-3) undo-only

	possible on-disk
A	100, 110
B	210, 220
C	300, 310
D	400, 410, 420

2-4) undo-redo **without** logical logging
(recovery using undo-redo logging)

LSN	backward pass	forward pass
0	skip	skip
10	undo: A = 100	skip
20	skip	skip
30	skip	redo: B = 210
40	undo: C = 300	skip
50	undo: D = 400	skip
60	add commit list	skip
70	undo: B = 210	skip
80	undo: D = 410	skip

2-5) undo-redo **with** logical logging
(recovery using LSNs)
on-disk datum LSNs: A: ~~10~~ 0, B: ~~70~~ 30, C: 0, D: 0

LSN	**backward pass**	**forward pass**
0	skip	skip
10	10 == 10 undo: A = 100	skip
20	skip	skip
30	skip	30 != 0 don't redo
40	0 != 40 don't undo	skip
50	0 != 50 don't undo	skip
60	add commit list	skip
70	70 == 70 undo: B = 210 datum LSN = 30	skip
80	0 != 80 don't undo	skip

2-6)

a. At the checkpoint, active txns are txn 1 and 2. We go back until we've seen the start records of all uncommitted txns in the most recent checkpoint record. Since txn 1 is uncommitted, we go back until we've seen the start of it, where still LSN = 0.
b. We begin from the log record that comes after the most recent checkpoint record, where LSN = 40.

__**Problem 3: Two-phase commit**__

beginning with the receipt of the prepare message from the coordinator and ending with the sending of the ready message to the coordinator

• if a site is ready, it:

1. prepares its subtxn – putting it in the ready state

Preparing a subtxn means ensuring it can be either committed or rolled back – even after a failure:

• need to at least force dirty log records to disk

• some logging schemes need additional steps:

For redo-only:

  • At transaction commit:
  1. write the commit log record
  2. force all dirty log records to disk
  (changed database pages are allowed to go to disk anytime after this)

• If a transaction aborts, none of its changes can be on disk.

2. tells the coordinator it's ready