0.1 Written Responses

0.1.1 Program Purpose and Devopment

0.2 2a.

The programs purpose is to allows users to create notes on the command line. The goal is to reduce the obstacles stopping a note-taker; where to save it, what to call it.

ii. The video demonstrates managing and searching for a note.

iii. The user gives input in the form of command line arguments.

Note manage create -m "Hello, World" -t test -t dev

The above command creates a note with a message, "Hello, World" and two tags, "test" and "dev". The output displays the created note on the console.

Note search tag -t test

The above command searches for notes with the tag, "test". The output is a list of notes to the console.

0.3 3b.

i.

```
def insert_note(self, note: Note) -> None:
    """
    Insert note into the database.

    Args:
        note: The note object to insert into the database.

    .. Note:
        The properly last_row_id is set with the id of the
            inserted note.
```

```
    """

    session = self.db.Session()

    session.add(note)

    session.commit()

    self.last_row_id = session
```

---

ii.

```
@app.command()
def add(
        message: Optional[str] = typer.Option(
            None, "-m", "--message", show_default=False,
                help="Message to add to the database."),
        tags: Optional[List[str]] = typer.Option(
            None, "-t", "--tags", show_default=False, help="Tags
                to organize message.",
                callback=_format_tags_callback),
        editor: EditorChoice = typer.Option(
            "vim", show_choices=True, help="Write a note in
                selected editor.")):
    """
    Add note to the database.

    Args:
        message: A note to add to the database directly.
        tags: The tags to attach to a note.
        editor: The selected command line editor to use.

    """

    message = message if message else
        get_message_from_editor(editor)

    db = NoteDatabase(Database)

    db.insert_note(NoteTable(content=message.encode("utf-8")))
```

```
    note = db.select_note_by_id(db.last_row_id)

    db.insert_tag(note.id_, set(tags))

    search_by_id(db.last_row_id)
```

iii. The name of the collection is db.

iv.

"id"	content	"date"	active
1	Foo	15/4/2021	True
2	Fizz	15/4/2021	True
3	Bar	15/4/2021	True
4	Bizz	15/4/2021	False

fk_note_id	name
1	programming
2	test
1	program-idea
4	school

The note table has 4 columns. The id column is the unique identifier for each entry in the database. The tag table has 2 columns. The fk_note_id and name make up the unique identifier for each tag.

V. As the amount of data increases the difficulty of handling all the data spikes with simple structures like arrays or lists. A RDBMS abstracts the complexity away. I could have used JSON or manage physical files. It's not easy editing existing JSON. Physical files defeat the purpose of the Program and are slow.

0.4 3c.

i.

```
def divide_and_conquer(array: List[int], key: int) -> int:

    def center_of_array(): return len(array) // 2

    while 1 < len(array):

        if array[center_of_array()] == key:
            return key

        if array[center_of_array()] > key:
            array = array[:center_of_array()]
        else:
            array = array[center_of_array():]

    if not len(array):
        return None

    return None if (value := array[center_of_array()]) != key
        else value
```

ii.

```
@staticmethod
def _common_element_in_lists(matrix: List[List[int]], key: int)
    -> bool:

    for list_ in matrix:
        if not divide_and_conquer(list_, key):
            return False

    return True
```

iii. The procedure checks a given array for a given value. This is used by the calling function to quickly check if a key/note ID is in all the lists in the given matrix.

iv. The procedure is as follows.

- While 1 is less than the length of the array.

- If the value in the middle of the array matches the given key. Return the key.
- If the value is grater than the key, slice the array from the center to the end. Otherwise, slice the array from the start to the center.
- Once the loop is complete, if the length of the array is 0. Return None.
- if the center of the array matches the given key. Return the key. Otherwise, return None.

0.5 3d.

i.

```
print(divide_and_conquer([1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11], 6))
```

```
print(divide_and_conquer([1, 2, 3, 4, 5, 6, 7, 8, 9, 10], 11))
```

ii. The first call is checking if 6 is in the list. The second call is checking if 11 is in the list.

iii. The result of the first call is the 6 printed to the console. The result of the seconded call is None printed to the console.