Lab 01 Part 02 Writeup

Team Members

Kellie Banzon, Cole Bemis, Tanner Larson

Initial Decisions *(Programming language, environment)*

We elected to use Python for this lab since we all have prior experience with the language and it supports simple Student objects and lists of objects.

We each selected our own coding environments. Cole prefers to use Vim, while Kellie and Tanner favor the PyCharm IDE.

Internal Architecture *(What data structures you used for what purposes)*

We created simple classes to wrap student and teacher information. We chose to organize the list of students into several dictionaries that group students by various keys. For example, we have a dictionary called "grouped_by_grade" that maps a grade number to a list of students in that grade. While these structures add initial time complexity and require more storage space, they allow for O(1) lookups when the user enters a query. Organizing the data in this way makes queries like "Given a bus route, find all students who take it" and "Find all students at a specified grade level" trivial to answer. Although some of our dictionaries hold redundant information, we reckoned that the "size" of this database renders these duplicates largely inconsequential. We chose to sacrifice storage efficiency to achieve fast responses to user input.

Modifications to Part 1 *(Decisions you made on how to modify your Part 1 code to accommodate new input data. Which parts of the code were affected?)*

For Part 2, we added a Teacher class (similar to our Student class in Part 1) and wrote a function to parse a file into a list of Teacher objects. We chose to continue with the approach we used for Part 1 and create several new dictionaries to look up students and teachers by different attributes.

For Part 2, we added the following dictionaries: students_by_classroom, teachers_by_lastname, teachers_by_classroom. Whenever we need to find a student's teacher, we lookup that student's classroom and use the teacher_by_classroom dictionary to find the correct teacher.

Query Updates *(Syntax and semantics of any additions to the query language, complete with examples.)*

For Part 2, we wanted to maintain support for single letter queries so users could type "G: 1" instead of "GRADE: 1". We chose names that didn't conflict with our previous system.

Queries added in Part 2 include:

- find_by_classroom_teacher → C[lassroom]: <number> T[eacher]
- find_by_grade_teachers → G[rade]: <number> T[eacher]
- enrollment → E[nrollment]
- raw → R[aw]: [grade=<number>] [bus=<number>] [teacher=<lastname>]

With regards to NR5, we wanted to provide data scientists with the flexibility to ethically filter specific data sets. We emulated the real functionality of public APIs closely. To maintain the privacy of the students, we assigned each student a unique ID rather than returning the student's name. Additionally, the RAW query supports both single and multiple filters so a user could request "students in 6th grade on bus 52 taking professor Fafard" by typing "R: grade=6 bus=52 teacher=FAFARD." Finally, all data will be returned when the user types "RAW."

Task Log

Part 1 tasks are color-coded in gray.

Task	Total Hours	Subtask / Description	Times	Assignee
Internal Architecture	2.0	Created Student class, parsed file into groups of student objects	2019/04/03 10:00-11:00	Cole Bemis
		Refactored to previous feature parity	2019/04/12 10:00-11:00	Cole Bemis
Parsing	00:35	Handle student, teacher, and info queries	2019/04/03 17:30-18:00	Kellie Banzon
		Handle grade, average, and bus queries	2019/04/05 10:10-11:00	Kellie Banzon
		Error handling	2019/04/07 23:20-23:30	Kellie Banzon
		Handle new queries	2019/04/15 17:30-18:00	Cole Bemis
		Support "HELP" command	2019/04/19 14:10-14:15	Kellie Banzon

Queries	04:10	R7	2019/04/03 17:40-17:50	Tanner Larson
		R4, R5, R6, R8	2019/04/03 17:50-18:00	Cole Bemis
		R9, R10, R11	2019/04/08 13:20-14:10	Tanner Larson
		Additional requirements design	2019/04/15 16:10-17:45	Kellie Banzon, Cole Bemis, Tanner Larson
		NR1, NR2	2019/04/15 18:35-18:40	Kellie Banzon
		NR5 support for single filter	2019/04/16 21:20-21:45, 2019/04/17 10:20-10:50	Kellie Banzon
		NR3, NR4	2019/04/18 16:00-17:00	Tanner Larson
		NR5 support multiple filters	2019/04/19 12:00-13:00	Tanner Larson
Testing	01:40	Writing test suite	2019/04/08 10:40-11:00	Kellie Banzon, Cole Bemis
		Writing R7 test case	2019/04/08 14:25-14:35	Tanner Larson
		Writing R9, R10, R11 test cases	2019/04/08 14:55-15:20	Tanner Larson
		Writing R8, E1 test cases	2019/04/08 20:05-20:20	Kellie Banzon
		Adding additional test cases, updating .in, .out, .expect files	2019/04/09 12:00-13:30	Kellie Banzon
		NR1, NR2	2019/04/15 18:40-19:00	Kellie Banzon
		NR5 single filter	2019/04/17 10:50-11:10	Kellie Banzon
		NR3, NR4	2019/04/19 02:15-02:30	Tanner Larson

		NR5 multiple filters, new help/error message, additional students and teachers	2019/04/19 14:15-15:00	Kellie Banzon
Documentation	01:35	Skeleton, organization, task log	2019/04/07 22:30-23:00	Kellie Banzon
		Internal Architecture	2019/04/08 10:15-10:30	Cole Bemis, Kellie Banzon
		README	2019/04/11 15:10-15:40	Tanner Larson
		Testing Part 1	2019/04/10 10:20-10:30	Cole Bemis, Kellie Banzon
		Writeup 1-1 Final Notes, edits & polishing	2019/04/10 10:20-10:45	Kellie Banzon
		Modifications to Part 1	2019/04/17 10:15-10:45	Cole Bemis
		Testing NR5	2019/04/17 11:30-11:40	Kellie Banzon
		Testing: Unique IDs solution	2019/04/19 02:30-02:40	Tanner Larson
		Query Updates, README updates	2019/04/19 14:15-15:00	Tanner Larson

Testing *(When, who, how long, how many bugs found, how long it took to fix them)*

We created two scripts to aid in the testing process. The first, called "gen-expect.sh," runs our program with all of the hand-written test input files (.in) and writes the output to .expect files. This convenience script made writing the expected output easier. We manually checked the generated output files and used these files as reference to diff against later. The second script, called "diff.sh," runs our program with all the test input files (.in) and diffs the output (.out) with the expected output (.expect).

Tanner and Kellie each wrote and ran test cases (see the Task Log for exact time spent). During Part 1, we found a bug in the parsing that assumed required second words would be present and a mistake in the output of certain queries that did not print according to the spec. Since both of these bugs are quite simple errors, each only took a few minutes to fix.

During Part 2, Kellie found a design flaw in the way we implemented IDs for each student. We tagged students by hashing the values associated with the object (i.e. first name, last

name, GPA, etc.) instead of by the order of the source file, because we wanted the IDs to be consistent every time the program is run. However, Python uses a random hash seed on each program execution, so hashes do not persist the way we want them to. In our final solution, Tanner used the hashlib library instead of the built-in hash function. This technique enabled us to utilize encoding algorithms (e.g. SHA-256) that produce consistent ids across runs.

Final Notes *(Anything else you want to share about your implementation)*

The program supports lowercase command words (i.e. “bus” is valid input). However, student names must be in all caps (i.e. “student: cookus” will fail).

For the RAW command to parse successfully, it must follow the format “<keyword>=<value>”. Filters can be called in any order, but any spaces in the filters will cause parsing to fail.