Database Systems – George Mason University
Home Assignment 2

Consider the following University Database schemas. Primary keys are underlined.

Relational Schemas	**Meaning and additional information**
department *(dcode, dname, chair)*	Department identified by *dcode* is named *dname* and has department chair with ssn *chair* (this ssn must appear in the table *faculty* below)
course *(dcode, cno, title, units)* not a weak entity	Course identified by *(dcode,cno)* has *title* and *units* (e.g., 3 units/credits) . (*dcode* must appear in the table *department)*
prereq *(dcode, cno, pcode, pno)* recursive function	Course (*dcode, cno)* has a prerequisite course (*pcode, pno).* Both pairs of courses must appear in the table *course.*
class has its own key, so its not dependent on the course and dept ***class*** *(class, dcode, cno, instr)* on delete cascade not present therefore not a weak entity	Class identified by *class* id is offered by department *dcode*, has course number *cno*, and is taught by instructor with ssn *instr* (this ssn must appear in the table *faculty* below; *dcode* and *cno* pair must appear in the table *course*)
faculty *(ssn, name, dcode, rank)* connected to dept	Faculty identified by *ssn* has a *name* and *rank*, and belongs to department *dcode* (which must appear in the table *department*)
student *(ssn, name, major, status)* connected to course via transcript relation	Student identified by *ssn* has a *name*, *major* and *status*.
enrollment *(class, ssn)* relation between class and student	Student identified by *ssn* is enrolled in the class identified by *class* no (*ssn* must appear in the table *student*, and *class* must appear in the table *class*)
transcript *(dcode, cno, ssn, grade)* connected to course, class	Student identified by *ssn* took the course identified by (*dcode*, *cno*) and received the *grade.* Assume that the only grades available are A, B, C and F. (*ssn* must appear in the table *student; dcode* and *cno* must appear in the table *course.*

Implement the following queries using:

A. Relational calculus, in the file **ha2lib_calculus.py** in the folder **solution_calculus.** Create initial template of **ha2lib_calculus.py** by duplicating **ha2lib_calculus_template.py**

B. Relational algebra, in the file **ha2lib_algebra.py** in the folder **solution_algebra.** Create initial template of **ha2lib_algebra.py** by duplicating **ha2lib_algebra_template.py**

C. SQL, in the file **sql_views.sql** in the folder **solution_sql.** Create initial template of **sql_views.sql** by duplicating **sql_views_template.sql**

a. Find students (*ssn, name, major, status*) who have taken the course "cs530" (must be in *transcripts*). Order the result by *ssn.*

b. Find students (*ssn, name, major, status*) named “John” (i.e., name = "John" in student) who have taken the course “CS 530” (must be in *transcripts*). Order the result by *ssn*.

c. Find students (*ssn, name, major, status*) who satisfied all prerequisites of each class they are enrolled in. Order the result by *ssn*.

d. Find students (*ssn, name, major, status*) who are enrolled in a class for which they have not satisfied all its prerequisites. To satisfy the prerequisite, the student needs to have obtained grade “B” or higher. Order the result by *ssn*.

e. Find students (*ssn, name, major, status*) named “John” who are enrolled in a class for which they have not satisfied all its prerequisites. To satisfy the prerequisite, the student needs to have obtained the grade “B” or higher. Order the result by *ssn*. this should give an empty set

f. Find courses (*dcode, cno*) that do not have prerequisites. Order the result by *dcode, cno*.

g. Find courses (*dcode, cno*) that do have some prerequisites. Order the result by *dcode, cno*.

h. Find classes (*class, dcode, cno, instr*) that are offered this semester and have prerequisites. Order the result by *class*.

i. Find students (*ssn, name, major, status*) who received only the grades “A” or “B” in every course they have taken (must appear in Transcripts). Order the results by *ssn.*

j. Find students (*ssn, name, major, status*) who are currently enrolled in a class taught by professor Brodsky (name = “Brodsky” in faculty). Order the result by *ssn*.

k. Find students (ssn) from the enrollment table who are enrolled in all classes. Order the result by *ssn*.

l. Find CS students (ssn) from the enrollment table who are enrolled in all math classes (dcode = “MTH”). Order the result by *ssn*.

Instructions for coding algebra, calculus and SQL queries:

1. Download and unpack archive **cs450_550_ha2_univ_db_template.zip**. It has a number of files and folders. If you use ATOM studio (which I recommend), under "File", choose "Add Project Folder" and select folder **cs450_550_ha2_univ_db_template** (the root folder). If you don't use ATOM studio, you can use any IDE that has syntax binding for SQL, JSON (Java Script Object Notation) and Python.
2. As described in "ha2_instructions.txt", install:
   a. Python 3.5 or higher
   b. cx_oracle python module
   c. cx_oracle client
   d. recommended: ATOM studio (alternatively you can use any other IDE which has syntax binding for SQL, JSON and Python
3. Assume a JSON database of the form as given in the file "testDBs/sampleUnivDB.json" (see file in the root folder). The meaning of the stored info is self-explanatory. For the purpose of queries below, assume that the possible grades are A, B, C and F; and that to satisfy a prerequisite for a class/course means to have taken the prerequisite courses (in transcript) with the grade of B or better.
4. Create the file "credentials.py" by duplicating the file "credentials_template.py which is in the solution_sql folder. Fill it with your Oracle DBMS credentials.
5. Implement the queries in by filling out the templates in the following files (note: see examples described in **8)**
   a. Relational algebra: solution_algebra/ha2lib_algebra.py
   b. Tuple relational calculus: solution_calculus/ha2lib_calculus.py
   c. SQL: solution_sql/sql_views.sql
6. To check your queries, you need to use your command line and change **your current working directory to the respective folder**. For example: change your working directory to *solution_algebra* for algebra queries.
7. To check the syntax of your queries and make it for the sample database, use the following (note: it only prints the output):
   a. Relational algebra: ha2_test_algebra.py
   b. Tuple relational calculus: ha2_test_calculus.py
   c. SQL: ha2_test_sql.py

   → E.g., for algebra go to folder *solution_algebra* in command line.
   → Run in command line:

   ```
   >>> python3 ha2_test_algebra.py > out.json
   ```
8. Example of the queries discussed in class has been provided for all 3 types of queries in their respective folder. The query files have the format

class_example_<type>.py To see the outputs these files produce, you can use *class_example_main_<type>.py*. E.g., for algebra, run

```
>>> python class_example_main_algebra.py > out.json
```

and see the results in *out.json* file.

9. Note that the file *testDBs/correct_answers.json* contains the correct answer to queries. You can use it for debugging your queries.

10. The folder *testDBs* also contains JSON files *db1.json, db2.json*, ... which are the databases against which your queries are being tested at the end. So, you can also view these (do not edit anything inside these) to debug your quries.

11. **To check your queries**, run

```
>>> python3 ha2_produce_answers_main_<type>.py
```

in the respective folder. This will print some status about what is being executed and finally at the end the output as well. Finally, this will also save your output in *answers.json* in the same folder. You can also view this for debugging purposes as this contains exactly what output your script is generating for the corresponding databases.

→ E.g., for sql go to folder *solution_sql* in command line.

→ Run in command line:

```
>>> python3 ha2_produce_answers_main_sql.py
```

12. Finally, to **get the report** of your generated answer, run

```
>>> python3 report_unordered.py
```

in the command line of the respective folder. This will generate (or update) the file **report.json** which contains the report.

13. Open ***report.json*** in Atom (or your other preferred IDE): you can see how many correct queries you have out of total queries, and it gives you a per query report, including for which test databases it produced correct vs. incorrect answer. It is convenient to prettify report.json, and collapse it before you open the relevant parts.

14. Also, do not forget to go through the ha2_instructions.txt thoroughly before you start to write your queries.