INFOMDWR – Lab 2: Relational Algebra and SQL

In this lab, you will work on querying databases using relational algebra and SQL. The exercises in this lab are similar to the examples that you studied during the lecture. The purpose of the lab sessions is to get the hands-on experience that is required for future carrier.

The exercises in this lab are based on the university database from the Database System Concepts.

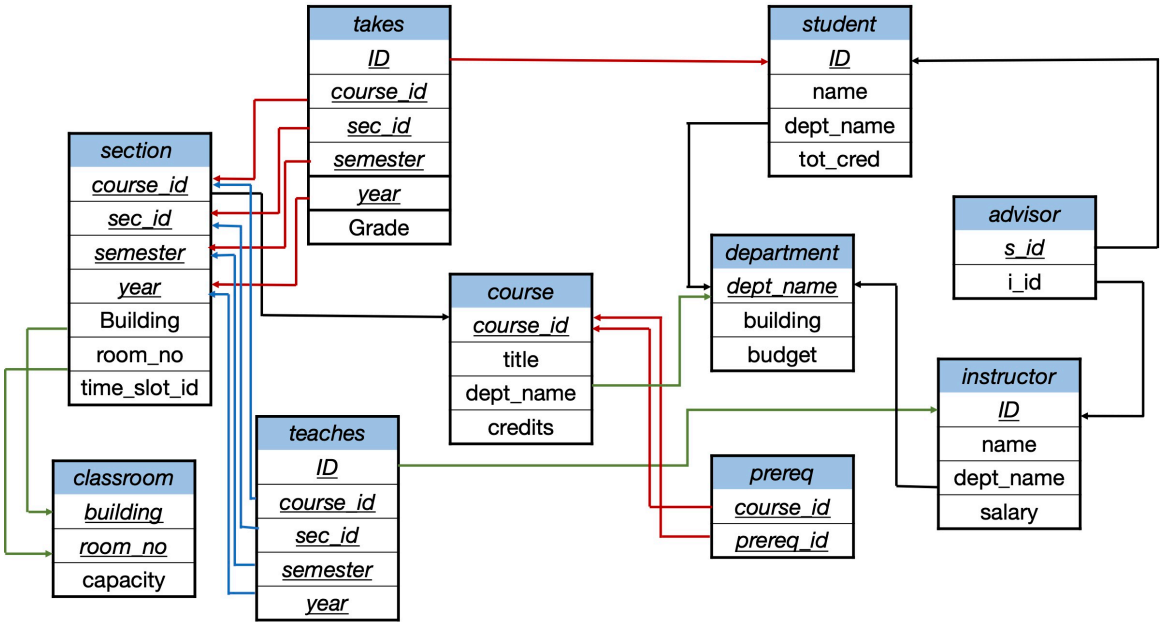

Figure 1: University database

1 Relational algebra

> **Note**
>
> Write the following queries in relational algebra:
>
> a. Find the titles of courses in the `Comp. Sci.` department that have 3 credits.
> b. Find the IDs of all students who were taught by an instructor named `Einstein`.
> c. Find the highest salary of any instructor.
> d. Find all instructors earning the highest salary (there may be more than one with the same salary).
> e. Find the enrollment of each section that was offered in Fall 2009.

2 SQL

For `SQL`, we will have two types of exercises. Type1, you need to interpret and understand the outcomes of an `SQL` Query. Type2, You will need to write your `SQL` query to extract specific piece of data.

Running SQL queries

> **Note**
>
> To run these queries, download the university database `univdb-sqlite.db` from here. The database contains a table `time_slot` that is not included in the schema diagram. After that, execute these queries and describe the task that they are supposed to perform and comment on the query outputs.
> Q1. SELECT-FROM-WHERE
>
> ```
> SELECT ID, name, dept_name
>     FROM student
>     WHERE dept_name < "Finance"
> ```
>
> Q2. Using DISTINCT
>
> ```
> SELECT DISTINCT dept_name
>     FROM student
> ```
>
> Q3. Using ALL
>
> ```
> SELECT ALL dept_name
>         FROM student
> ```
>
> Q4. Using named literal attribute

```
SELECT 'MyName' as 'V1'
```

Q5. Using literal attribute with `FROM`

```
SELECT 'A'  FROM student
```

Q6. Performing calculations on the Query output

```
SELECT dept_name, building, budget, (budget + (budget * 0.15)) as new_budget
FROM department;
```

Q7. Join

```
SELECT name, course_id, sec_id
    FROM studnt, takes
    WHERE student.ID = takes.ID
            and sec_id = 2;
```

Q8. Self-join

```
SELECT * FROM student as S1, student as S2
    WHERE S1.dept_name = S2.dept_name
        AND S1.name <> S2.name;
```

Q9. Left-outer-join

```
SELECT * FROM student
JOIN takes
ON student.ID = takes.ID
```

- What will be the difference when using LEFT OUTER JOIN instead of JOIN

Writing SQL queries

Note

In this part, you will write your own `SQL` queries to perform the following tasks.

1. Write a query that returns the information of the students and the courses they have taken.
2. SQLite supports LEFT OUTER JOIN only. How do we rewrite the following query on SQLite.

```
SELECT * FROM studen
Right outer JOIN takes
ON student.ID = takes.ID
```

3. SQLite supports only LEFT OUTER JOIN. How can rewrite the following query on SQLite.

```
SELECT * FROM student
FULL outer JOIN takes
ON student.ID = takes.ID
```

4. Write a query that finds the names of all students that contain the substring "an" in their names
5. Write a query that returns the names and tot_cred of students with total credits between 32 and 80 (inclusive)
6. Write a query that returns the information of the students in the department Biology and Physics using set operations
7. Write a query that returns the information of all students except the students in the computer science department using set operations
8. Given the query:

```
SELECT dept_name, avg(salary) avg_salary
FROM instructor
Group by dept_name
HAVING avg_salary > 70000;
```

- What does this query do?
- Why we used HAVING?
- What about replacing HAVING by WHERE?

9. Write a query to return the names of students with the largest and smallest number of `tot_cred`.
10. Write a query to return the number of departments that have more than one student.
11. Write a query to return the number of students in the Comp. Sci., Physics, Finance and History.