COMS W4111: Introduction to Databases
Spring 2024, Sections 002/V02

Homework 2: Common

Introduction

This notebook contains HW2 Common. **Students on both tracks should complete this part.** To ensure everything runs as expected, work on this notebook in Jupyter.

Submission instructions:

- You will submit **a PDF** for this assignment
    - The most reliable way to save as PDF is to go to your browser's menu bar and click `File -> Print`. **Switch the orientation to landscape mode**, and hit save.
    - **MAKE SURE ALL YOUR WORK (CODE AND SCREENSHOTS) IS VISIBLE ON THE PDF. YOU WILL NOT GET CREDIT IF ANYTHING IS CUT OFF.** Reach out for troubleshooting.

Written Questions

W1

Explain Codd’s 3rd Rule.

- What are some interpretations of a NULL value?

- An alternative to using NULL is some other value for indicating missing data, e.g., using -1 for the value of a weight column. Explain the benefits of NULL relative to other approaches.

The rule states that NULL values in a table should be treated uniformly and systemically since there are multiple interpretations of a NULL value. For example, a NULL value could mean that the data was not available/known, or that the data is not applicable (height of a man in a table that includes women).

If say in a column of salaries, we don't know the salary of certain employees. If it is set to NULL, then average salary for those records does not take into account NULL values so the average is only the average of non-NULL records. However, if we set the missing data to be 0, then the average salary will be much lower than the true average salary since it is unlikely that anyone has 0 salary.

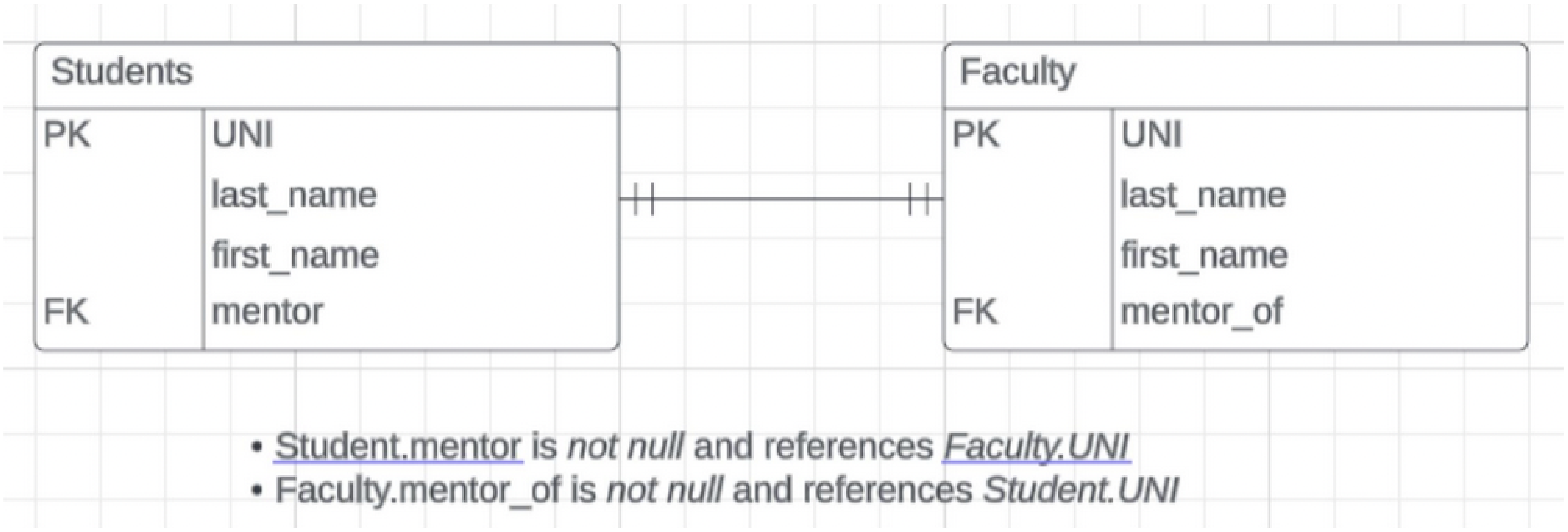
W2

Briefly explain the following concepts:

1. Primary key
2. Candidate key
3. Super key
4. Alternate key
5. Composite key
6. Unique key
7. Foreign key

1. It is (primary) one of the candidate keys (therefore one of the superkeys). It is non-NULL.
2. It is a minimal super key. That is removing any value from the super key results in it not being a superkey.
3. It is an attribute (or set of attributes) that are sufficient to uniquely identify a record in a table. It may or may not be minimal.
4. It is a secondary candidate key that is capable of identifying unique records in a table.
5. It is a key made up of multiple attributes that together identifies unique records.
6. These are similar to primary keys except they allow NULL values. Officially, these are attributes that uniquely identify rows. (https://www.gleek.io/blog/primary-vs-unique (https://www.gleek.io/blog/primary-vs-unique))
7. This is a constraint on an attribute of a table. Say table1 has a FK then any row in table1 with a non-null value of FK attribute must have a corresponding row in another table where the key is that FK attribute.

W3


Students
PK
UNI
last_name
first_name
FK
mentor
Faculty
PK
UNI
last_name
first_name
FK
mentor_of
• Student.mentor is *not null* and references *Faculty.UNI*
• Faculty.mentor_of is *not null* and references *Student.UNI*

Consider the logical data model above. The one-to-one relationship is modeled using two foreign keys, one in each table.

- Why does this make it difficult to insert data into the tables?
- What is a (simple) fix for this, i.e., how would you model a one-to-one relationship?

When adding a new student to "Students" we need to already have a mentor in the "Faculty" table (since "mentor" is FK) but that table already needs the new student to exist in "Students" since "mentor_of" is FK. This is circular and will cause a problem when adding a record to either table.

A quick fix would be to remove one or both of the FK constraints. If every student must have a mentor but not every faculty is a mentor then get rid of "mentor_of" FK or vice-versa.

W4

The relational model places restrictions on attributes. Many data scenarios have more complex types of attributes. Briefly explain the following types of attributes:

1. Simple attribute

2. Composite attribute
3. Derived attribute
4. Single-value attribute
5. Multi-value attribute

1. It is an atomic, indivisible attribute. That is, cannot be further subdivided into attributes. Eg: "Age"
2. It is made up of two or more simple attributes. Eg: "Address" can be made up of "City" + "State" + "Zipcode"
3. It is not stored in database but can be created using existing attributes. Eg: "Total" = "Price" * "Quantity"
4. It stores exactly one value per record. Eg: "DOB"
5. It can store multiple values per record: Eg: "Phone number(s)" Source: https://www.knowledgehut.com/blog/database/attributes-in-dbms (https://www.knowledgehut.com/blog/database/attributes-in-dbms)

W5

The slides associated with the recommended textbook list six basic relational operators:

1. select: σ
2. project: π
3. union: ∪
4. set difference: -
5. Cartesian product: ×
6. rename: ρ

The list does not include join: ⋈. This is because it is possible to derive join using more basic operators. Explain how to derive join from the basic operators.

We first get the *Cartesian product* (×) to get all combinations of records from two tables. Then use *select* (σ) to filter on matching attributes. (Can use *project* (π) to retain required columns as final step)

W6

Explain how using a natural join may produce an incorrect/unexpected answer.

Natural join uses columns of same name as join condition but if there isn't then there might be unexpected results. For example, if there are 2 tables with columns "ID" and "id" respectively and we intended to join on those columns but doing a natural join might result in an full outer join if there are no other identical column names.

W7

The UNION and JOIN operations both combine two tables. Describe their differences.

UNION aggregates the results of separate queries whereas JOIN creates a cartesian product and takes a specific subset of it. UNION puts tables on top of each other whereas JOIN puts them side-by-side.

W8

Briefly explain the importance of integrity constraints. Why do non-atomic attributes cause problems/difficulties for integrity constraints?

Integrity constraints are important in ensuring quality of data and maintaining certain set of rules that all data must adhere to. Without it, we would need to assume that people/programs that modify/use database never make an error and are always consistent with each other ensuring the same quality.

Non-composite attributes pose problems since the attributes as an aggregate are a "column" so if there is constraints on the aggregate, then any changes to the atomic attributes within the aggregate might break the integrity constraint of the aggregate. For example, if the aggregate is unique, this doesn't mean that the individual atomic attributes need to be unique so to ensure the integrity, all the atomic attributes need to be checked as a whole rather than a single column.

W9

What is the primary reason for creating indexes? What are the negative effects of creating unnecessary indexes?

Primary reason is to be more efficient and save time/resources. Counterintuitively, creating unnecessary indexes takes up unnecessary space and reduces performance.

W10

Consider the table `time_slot` from the sample database associated with the recommended textbook.

- The data type for the column `day` is `char(1)`. Given the data types MySQL supports, what is a better data type for `day` ?
- What is a scenario that would motivate creating an index on `day` ?

1. Best to use ENUM and assign single letter to correspond to each day. Eg ENUM('M', 'T', 'W', 'R', 'F', 'S', 'G') where 'R' corresponds to Thursday, and 'G' to Sunday.
2. Ticket reservation systems that frequently look up records based on day would benefit from creating an index on "day". Essentially any scenario that frequently uses "day" in queries for "joins" or "selects" would benefit from indexing on "day"

Relational Algebra

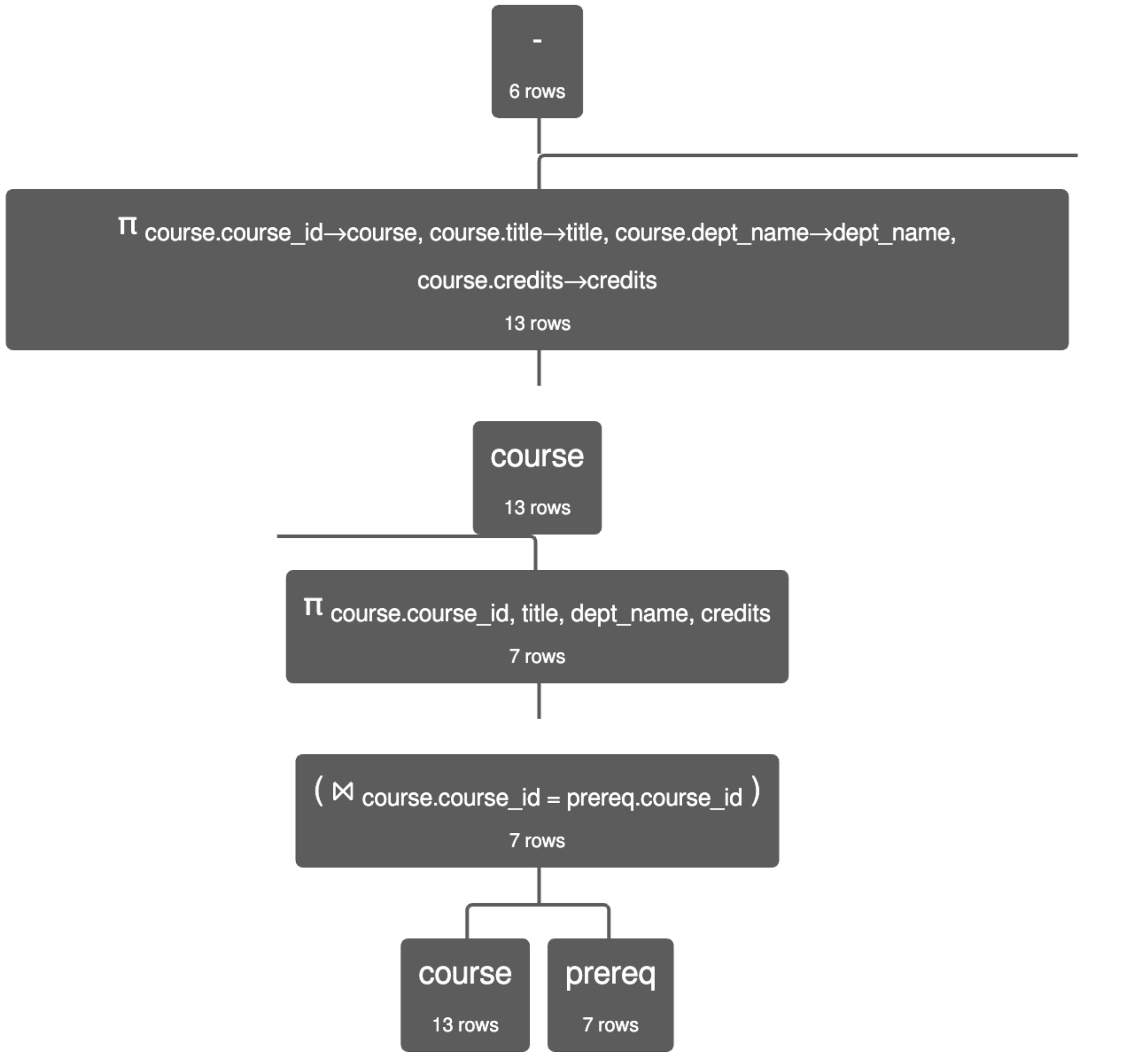
R1

- Write a relational algebra statement that produces a relation showing **courses that do not have a prereq**
- Your output should have the following columns (names should match exactly; there should be no prefixes):
  - `course_id`
  - `title`
  - `dept_name`
  - `credits`
- You may not use the anti-join: $\triangleright$ operator
- You should use the `course` and `prereq` tables

Algebra statement:

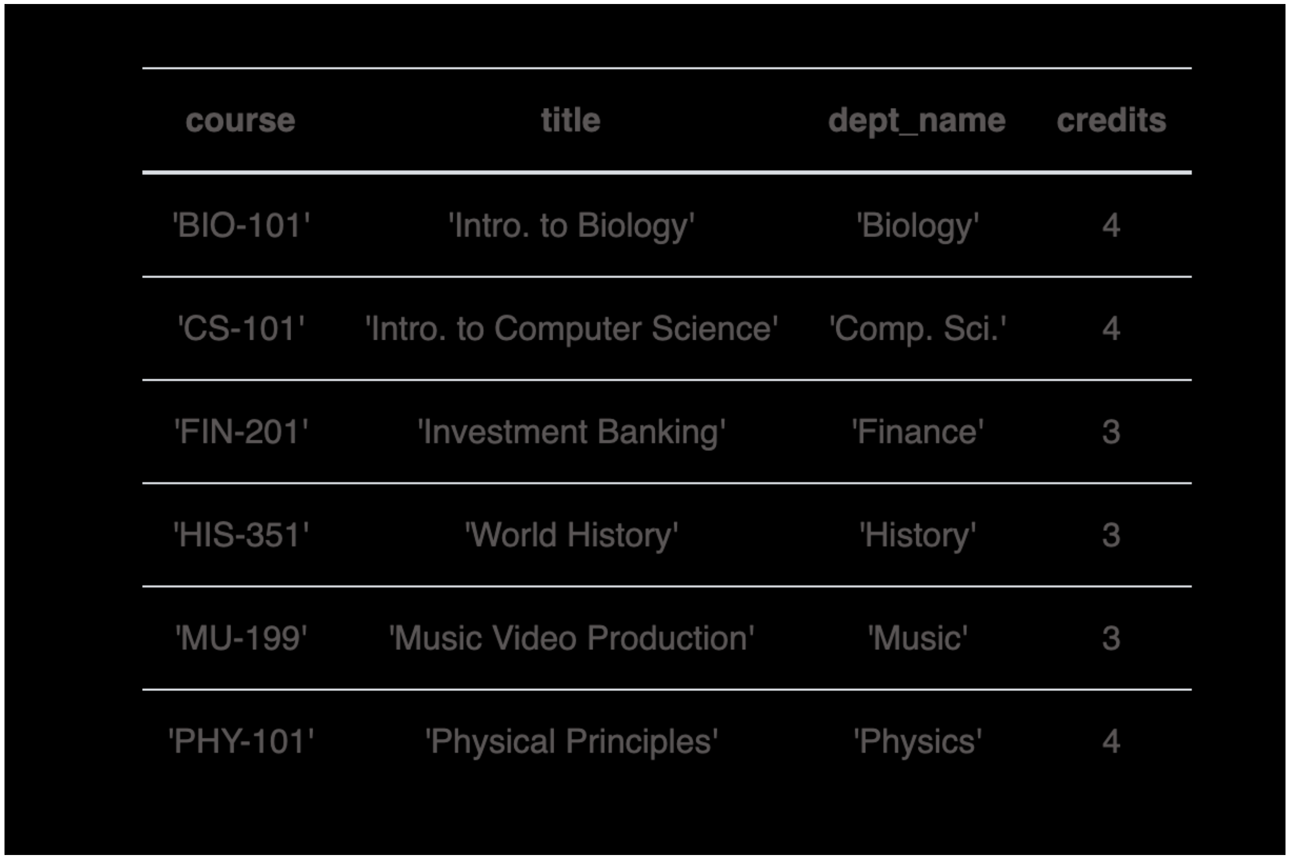
```
π course←course.course_id, title<-course.title, dept_name<-course.dept_name, credits<-cou
rse.credits (
course) -
π course.course id, title, dept name, credits (
```

Execution:


-
6 rows
π course.course_id→course, course.title→title, course.dept_name→dept_name, course.credits→credits
13 rows
course
13 rows
π course.course_id, title, dept_name, credits
7 rows
(⋈ course.course_id = prereq.course_id)
7 rows
course
13 rows
prereq
7 rows

R1 Execution Result (part1)

Execution:



course	title	dept_name	credits
'BIO-101'	'Intro. to Biology'	'Biology'	4
'CS-101'	'Intro. to Computer Science'	'Comp. Sci.'	4
'FIN-201'	'Investment Banking'	'Finance'	3
'HIS-351'	'World History'	'History'	3
'MU-199'	'Music Video Production'	'Music'	3
'PHY-101'	'Physical Principles'	'Physics'	4

R1 Execution Result (part2)

R2

- Write a relational algebra query that produces a relation showing **students who have taken sections taught by their advisors**
  - A section is identified by `(course_id, sec_id, semester, year)`
- Your output should have the following columns (names should match exactly; there should be no prefixes):
  - `student_name`
  - `instructor_name`

        - `course_id`
        - `sec_id`
        - `semester`
        - `year`
        - `grade`
    - You should use the `takes` , `teaches` , `advisor` , `student` , and `instructor` tables
    - As an example, one row you should get is

student_name	instructor_name	course_id	sec_id	semester	year	grade
'Shankar'	'Srinivasan'	'CS-101'	1	'Fall'	2009	'C'

    - Shankar took CS-101, section 1 in Fall of 2009, which was taught by Srinivasan. Additionally, Srinivasan advises Shankar

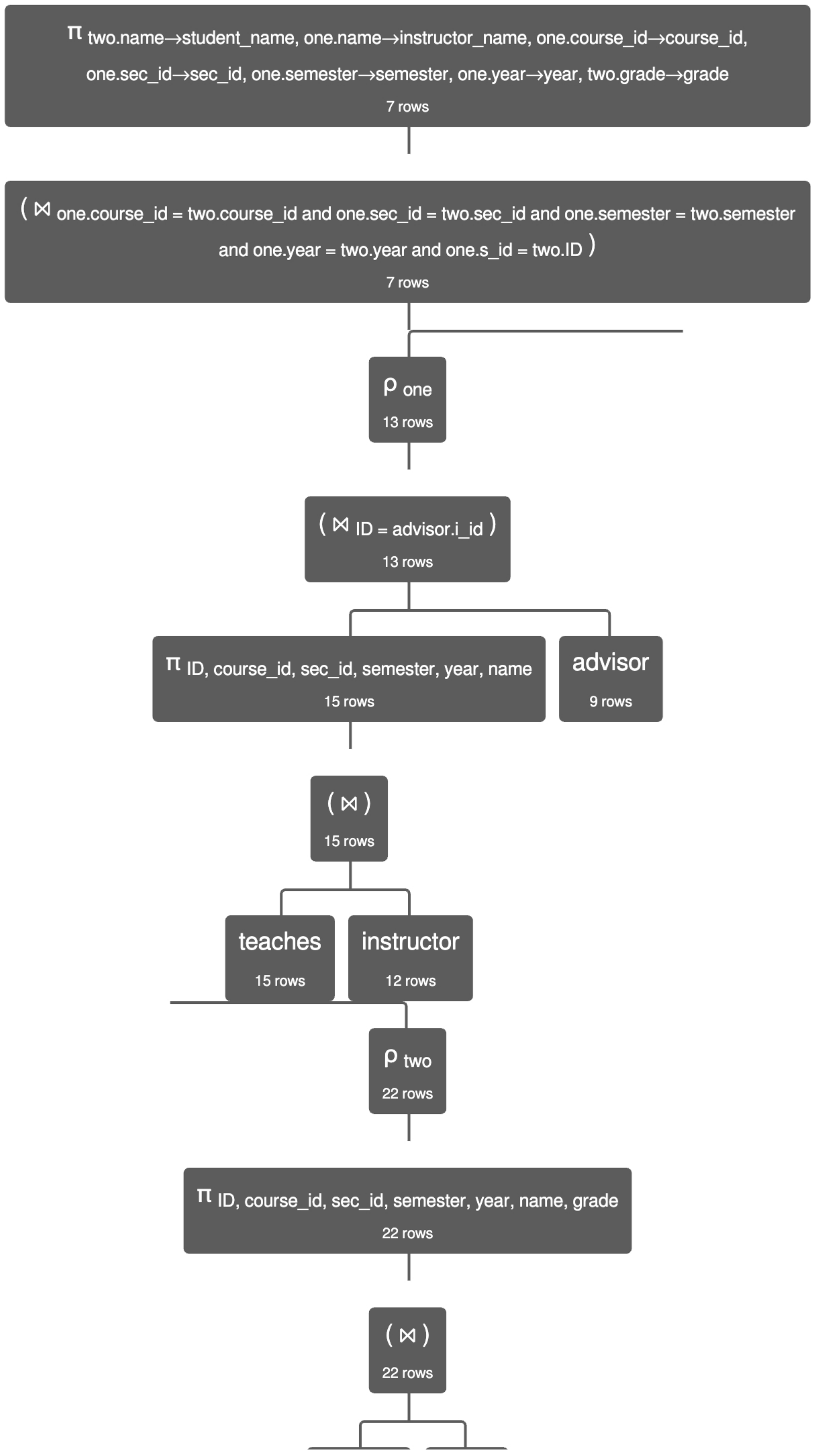
Algebra statement:

```
π student_name ← two.name, instructor_name ← one.name, course_id ← one.course_id, sec_id
← one.sec_id, semester ← one.semester, year ← one.year , grade ← two.grade
(ρ one ((π ID, course_id, sec_id, semester, year, name (teaches ⋈ instructor))
⋈
ID = advisor.i_id
advisor)
⋈ one.course_id = two.course_id ∧ one.sec_id = two.sec_id ∧ one.semester = two.semester ∧
one.year = two.year ∧ one.s_id = two.ID
(ρ two (π ID, course_id, sec_id, semester, year, name, grade (student ⋈ takes))))
```

In [ ]:

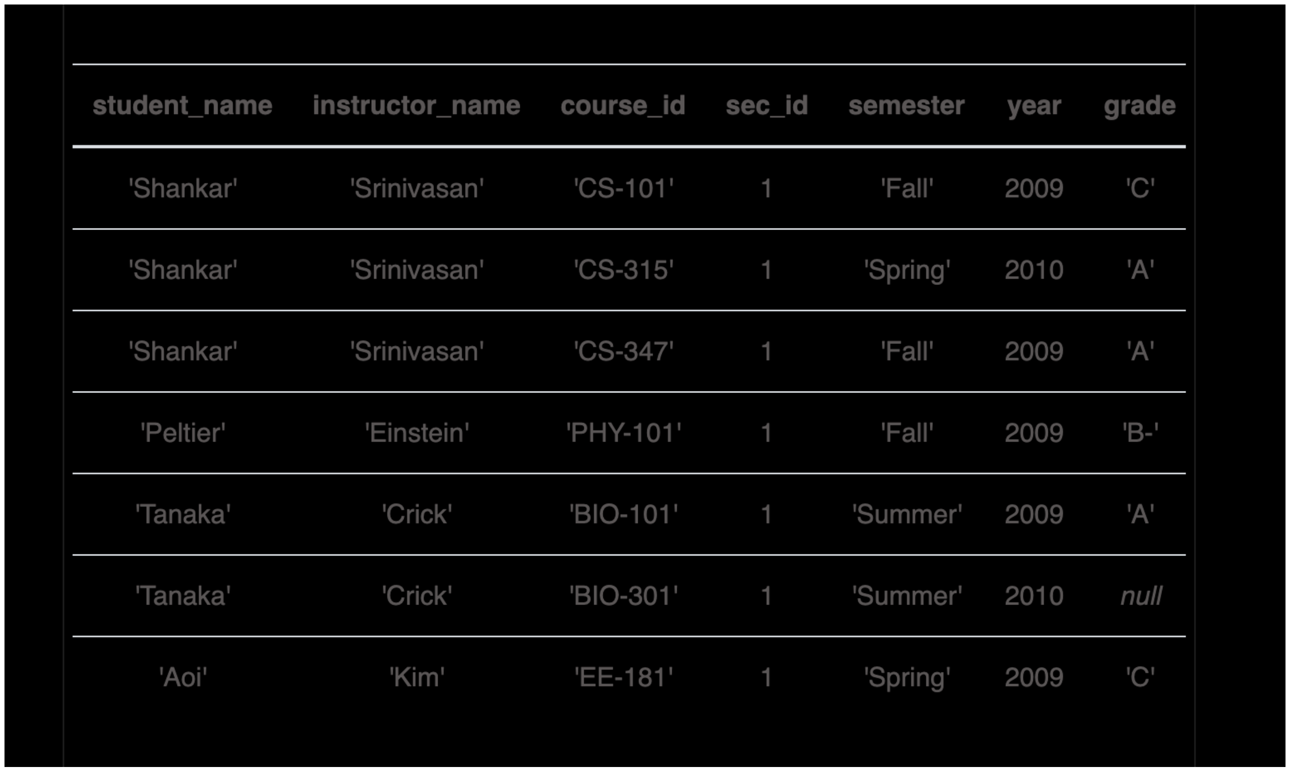
In [ ]:

Execution:


π two.name→student_name, one.name→instructor_name, one.course_id→course_id, one.sec_id→sec_id, one.semester→semester, one.year→year, two.grade→grade
7 rows
(⋈ one.course_id = two.course_id and one.sec_id = two.sec_id and one.semester = two.semester and one.year = two.year and one.s_id = two.ID)
7 rows
ρ one
13 rows
(⋈ ID = advisor.i_id)
13 rows
π ID, course_id, sec_id, semester, year, name
15 rows
advisor
9 rows
(⋈)
15 rows
teaches
15 rows
instructor
12 rows
ρ two
22 rows
π ID, course_id, sec_id, semester, year, name, grade
22 rows
(⋈)
22 rows

```
In [ ]:
```

Execution:



student_name	instructor_name	course_id	sec_id	semester	year	grade
'Shankar'	'Srinivasan'	'CS-101'	1	'Fall'	2009	'C'
'Shankar'	'Srinivasan'	'CS-315'	1	'Spring'	2010	'A'
'Shankar'	'Srinivasan'	'CS-347'	1	'Fall'	2009	'A'
'Peltier'	'Einstein'	'PHY-101'	1	'Fall'	2009	'B-'
'Tanaka'	'Crick'	'BIO-101'	1	'Summer'	2009	'A'
'Tanaka'	'Crick'	'BIO-301'	1	'Summer'	2010	*null*
'Aoi'	'Kim'	'EE-181'	1	'Spring'	2009	'C'

R2 Execution Result (part2)

R3

- Write a relational algebra query that produces a relation showing **sections that occur on Friday and start after 10 AM**
- Your output should have the following columns (names should match exactly; there should be no prefixes):
  - `course_title`
  - `sec_id`
  - `semester`
  - `year`

    - `day`
    - `start_hr`
- You should use the `course` , `section` , and `time_slot` tables

Algebra statement:

```
π course_title ← one.title,
sec_id ← one.sec_id,
semester ← one.semester,
year ← one.year,
day ← two.day,
start_hr ← two.start_hr
(ρ one (course ⋈ section)
⋈ one.time_slot_id = two.time_slot_id
ρ two (π time_slot_id, start_hr, day (σ start_hr > 10 ∧ day = 'F' (time_slot))))
```

In [ ]:

```

```

In [ ]:

```

```

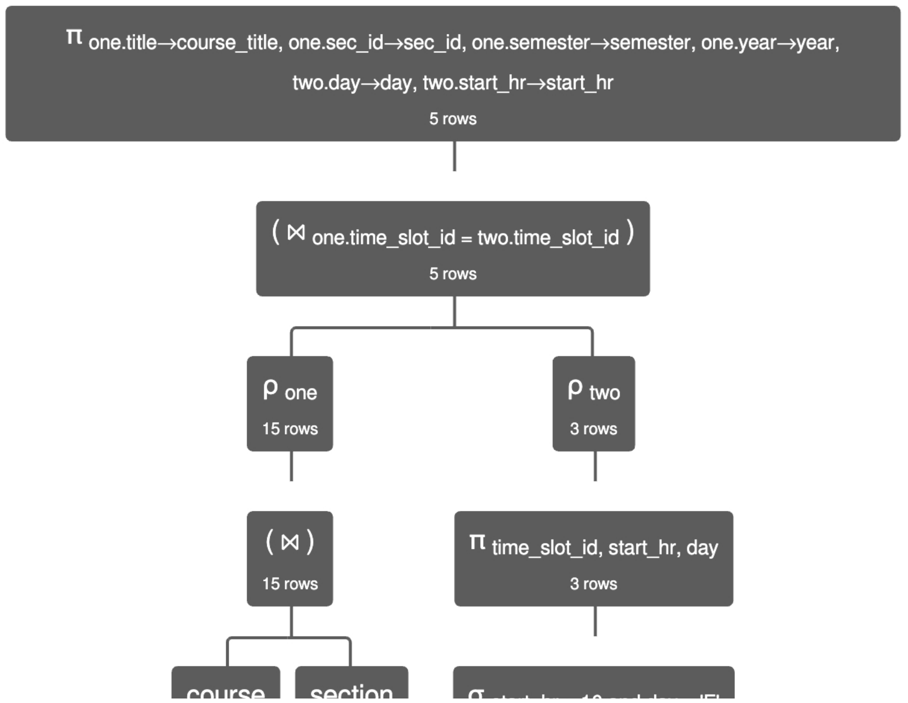
In [ ]:

```

```

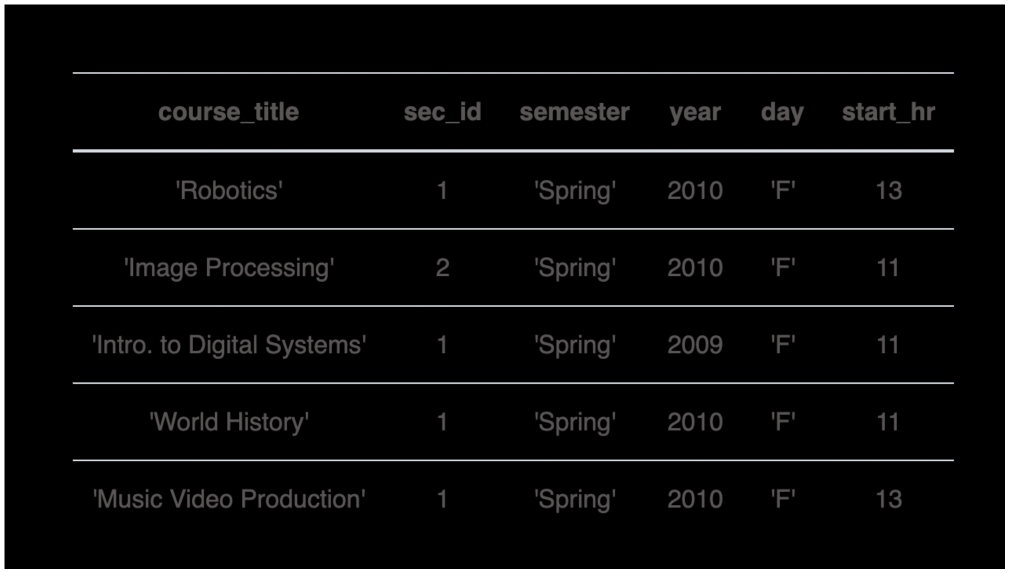
In [ ]:

```

```

Execution:


π one.title→course_title, one.sec_id→sec_id, one.semester→semester, one.year→year, two.day→day, two.start_hr→start_hr
5 rows
(⋈ one.time_slot_id = two.time_slot_id)
5 rows
ρ one
15 rows
ρ two
3 rows
(⋈)
15 rows
π time_slot_id, start_hr, day
3 rows
course
section

Execution:



course_title	sec_id	semester	year	day	start_hr
'Robotics'	1	'Spring'	2010	'F'	13
'Image Processing'	2	'Spring'	2010	'F'	11
'Intro. to Digital Systems'	1	'Spring'	2009	'F'	11
'World History'	1	'Spring'	2010	'F'	11
'Music Video Production'	1	'Spring'	2010	'F'	13