F24-W4111-03: Introduction to Databases: Homework 1, Part B

Submission Instructions

Note to TAs: Please complete this information, create GradeScope entries, etc.

Environment Setup

This section tests your environment for HW1B.

If you successfuly completed HW0, you should not have any problems.

Please make sure you set your MySQL user id and password correctly.

```
In [12]: # %pip install pandas
         import pandas
```

```
In [13]: import sqlalchemy
```

```
In [14]: import pymysql
```

```
In [15]: import json
```

```
In [16]: %load_ext sql
```

```
The sql extension is already loaded. To reload it, use:
  %reload_ext sql
```

```
In [17]: %sql mysql+pymysql://root:Tolly1221!@localhost
In [18]: engine = sqlalchemy.create_engine("mysql+pymysql://root:Tolly1221!@localhost")
```

Entity Relationship Modeling

Top-Down Modeling

The ability to prduce an ER diagram from a "human" description of the data model is an import skill. In this process, you may have to make and document assumptions or explain decisions. There is no single, correct answer. As long as your assumptions and decisions are reasonable, and your model accurately reflects requirements and decisions, your model answer is "correct."

In this scenario, there are four entity types/entity sets:

1. `Person(id, last_name, first_name, middle_name, created_timestamp, last_modified_timestamp)` : Basic information about a person. The type has properties/attributes:
   - `id` uniquely indentifies the `Person`
   - `last_name`
   - `first_name`
   - `middle_name`
   - `created_timestamp` : When the the entity was created for the first time.
   - `last_modified_timestamp` : The last time the entity's information changed.
2. `Contact_Information(contact_type, contact_value)` : Represents a mechanism for contacting a person.
   - `id` : A unique ID for the `Contact_Information` .
   - `contact_type` : Indicates the type of contact, e.g. "primary phone," "email," etc.
   - `contact_value:` The value for the contact. This is simply a text string for both types of contact. For example, "bilbo.baggins@shire.org (mailto:bilbo.baggins@shire.org)" or "+1 212-555-1212."
3. `Order(id, product_name, order_date, description)` : Represents someone having placed an order to purchase something. Order has the properties:
   - `id` : Uniquely identifies the `Order`
   - `product_name` : The name of the product, e.g. "Strawbery Poptarts," "Cross Pen."
   - `order_date` : The date the order was placed
   - `description` : A text description of the order
4. `Comment(id, comment, comment_timestamp)` : Represent a user's comment on an order. Comment has three properties:

- `id` : Uniquely identifies the `Comment`
- `comment` : Text of the comment
- `comment_timestamp` : Timestamp when the comment was made.

The model has the following relationships/entity sets:

- `Person-Comment` is a relationship the represents the fact that the `Person` made the `Comment`. A `Person` may make many `Comments`, but a `Comment` is made by exactly one user.
- `Order-Comment` associates the `Comment` with the `Order`. There may be many `Comments` on an `Order` but a `Comment` has one `Order`.
- `Person-Contact-Info` is between `Person` and `Contact-Info`. A `Person` may have multiple `Contact-Info` entries. A `Contact-Info` relates to exactly one `Person`.

The model must represent the fact that `Contact-Info` is valid between a start timestamp and end timestamp.

The system never deletes any information.

You must create a Crow's Foot Notation *logical model* that is your model that satisfies the requirements. You may have to add unspecified attributes to entity types. You can add comments and notes.

Show your diagram below. You can add notes to your diagram or add explanatory text. You can take a screenshot of your diagram and include below. The "Implement ER Diagram" question has an example of embedding an image in the notebook.

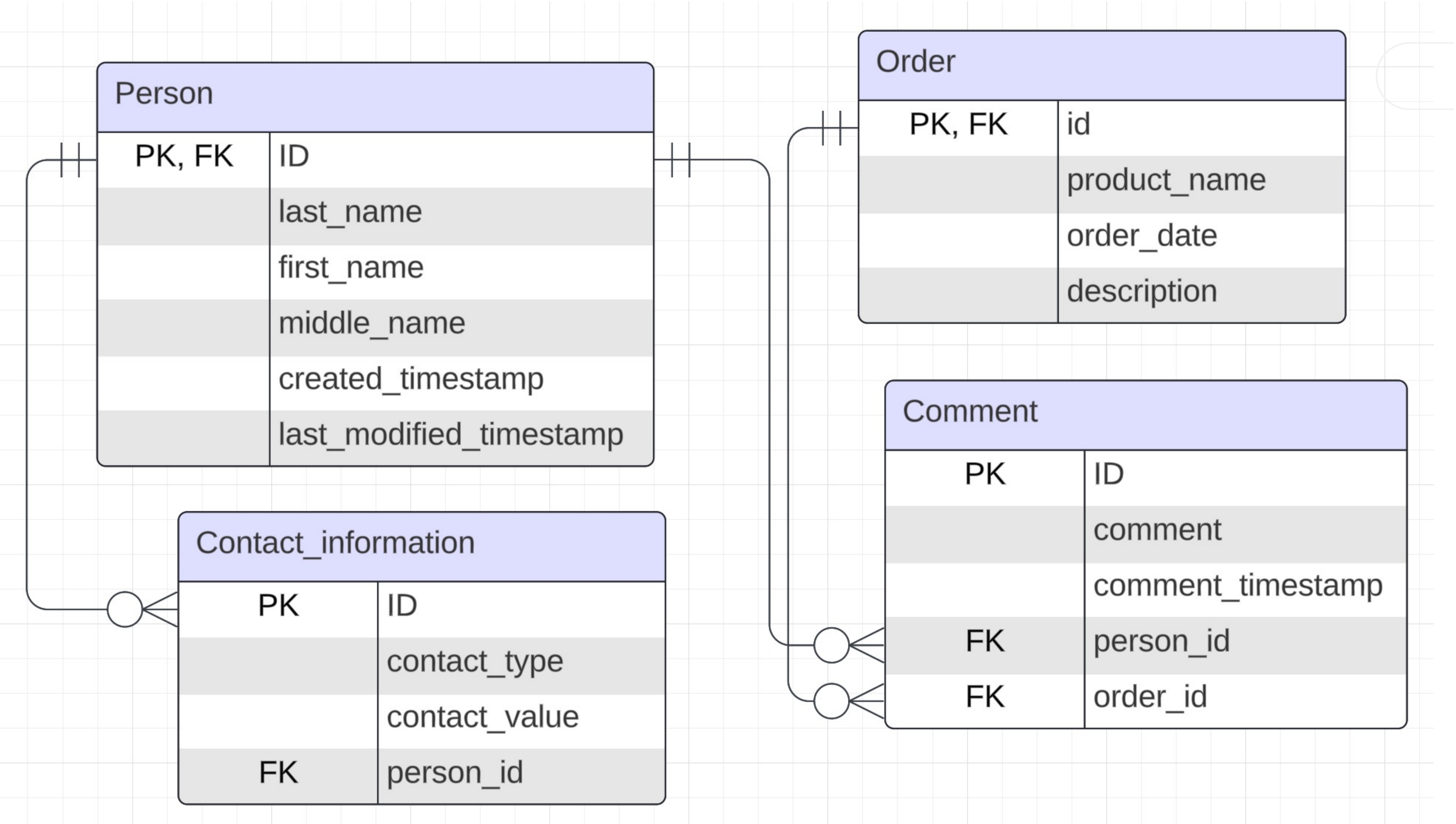
There is no single correct answer.

Digram:

In [39]:

```
from IPython.display import Image
img_path = '/Users/zhengfeichen/Desktop/Columbia/4111/W4111-Intro-to-Databases-Base/Homework-Assignme
Image(img_path)
```

Out[39]:



Person	
PK, FK	ID
	last_name
	first_name
	middle_name
	created_timestamp
	last_modified_timestamp

Order	
PK, FK	id
	product_name
	order_date
	description

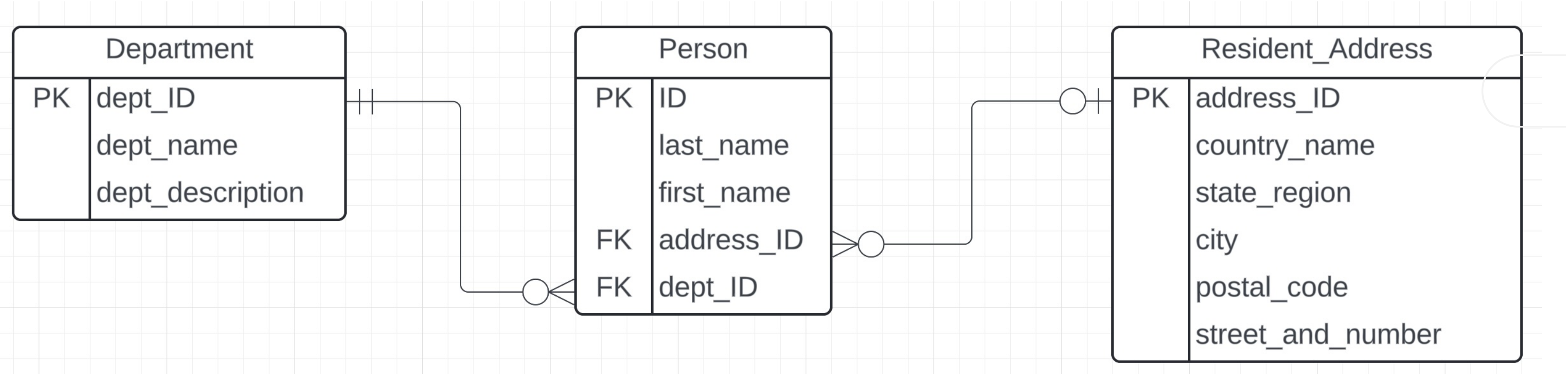
Contact_information	
PK	ID
	contact_type
	contact_value
FK	person_id

Comment	
PK	ID
	comment
	comment_timestamp
FK	person_id
FK	order_id

We create person_id in Contact_information and Comment table to connect between Person and Contact_information and between Person and Comment. Because they are all one-to-many relationship, person is one, comment and contact information are many, we create the line like that

Also, we create order_id in Comment to connect between Comment and Order. Because they are one-to-many relationship, Order is one, comments are many, we create the line like that

Implement ER Diagram



Write SQL DDL that creates the tables and relationships in the preceding diagram

You can pick `VARCHAR(32)` for the type of each column.

You must specify keys and foreign keys.

Create a new database that you name `hw1b_<uni>` and replace `<uni>` with your UNI. For example, mine would be `hw1b_dff9`.

You must enter and successfully execute your SQL in the code cell below.

In [9]:

```
%%sql

/* Your create and alter table statements. */

CREATE DATABASE hw1b_zc2735;
USE hw1b_zc2735;

CREATE TABLE Department (
    dept_ID VARCHAR(32) PRIMARY KEY,
    dept_name VARCHAR(32),
    dept_description VARCHAR(32)
);

# Create Resident_Address table
CREATE TABLE Resident_Address (
    address_ID VARCHAR(32) PRIMARY KEY,
    country_name VARCHAR(32),
    state_region VARCHAR(32),
    city VARCHAR(32),
    postal_code VARCHAR(32),
    street_and_number VARCHAR(32)
);

# Create Person table with foreign keys referencing Department and Resident_Address
CREATE TABLE Person (
    ID VARCHAR(32) PRIMARY KEY,
    last_name VARCHAR(32),
    first_name VARCHAR(32),
    address_ID VARCHAR(32),
    dept_ID VARCHAR(32),
    FOREIGN KEY (address_ID) REFERENCES Resident_Address(address_ID),
    FOREIGN KEY (dept_ID) REFERENCES Department(dept_ID)
);
```

```
 * mysql+pymysql://root:***@localhost
1 rows affected.
0 rows affected.
0 rows affected.
0 rows affected.
0 rows affected.
```

```
Out[9]: []
```

Relational Algebra

You will use the Relax calulator and the schema associated with the text book for this question.

https://dbis-uibk.github.io/relax/calc/gist/4f7866c17624ca9dfa85ed2482078be8/relax-silberschatz-english.txt/0 (https://dbis-uibk.github.io/relax/calc/gist/4f7866c17624ca9dfa85ed2482078be8/relax-silberschatz-english.txt/0)

Problem 1

Write a relational algebra expression that produces a result table with the following format:

`(student_id, student_name, course_title, course_id, sec_id, semester, year, instructor_id, instructor_name)`

- `student_id` is a student's ID (`student.ID`)
- `student_name` is a student's name (`student.name`)
- `course_title` (`course.title`)
- The following columns are common to `section, takes, teaches:`
  - `course_id`
  - `sec_id`
  - `semester`
  - `year`
- `instructor_id` is an instructor's ID (`instructor.ID`)
- `instructor_name` is an instructor's name (`instructor.name`)

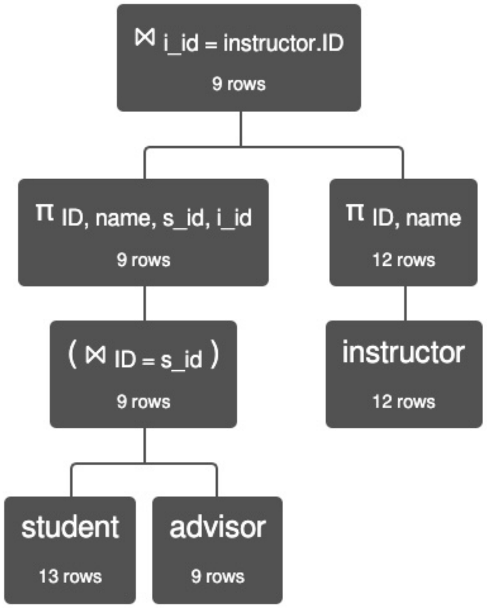
This derived relation represents student that took a section and the instructor taught the section.

Cut and paste your query in the markdown cell below.

Past relational algebra here. The following is an example of pasting a relational algebra expression. Replace the following with your expression.

```
/*
This query produces students and their advisors.
*/
ρ student_id←student.ID, student_name←student.name, course_title←course.title, instructor_id←
instructor.ID, instructor_name←instructor.name π student.ID, student.name, course.title, sect
ion.course_id, section.sec_id, section.semester, section.year, instructor.ID, instructor.name
( ( ( ( ( student ⨝ student.ID = takes.ID takes ) ⨝ takes.course_id = section.course_id and t
akes.sec_id = section.sec_id and takes.semester = section.semester and takes.year = section.y
ear section ) ⨝ section.course_id = teaches.course_id and section.sec_id = teaches.sec_id and
section.semester = teaches.semester and section.year = teaches.year teaches ) ⨝ teaches.ID =
instructor.ID instructor ) ⨝ section.course id = course.course id course )
```

Execute your query on the Relax calculator and show an image of the first page of your result below. The following shows an example of the format of the answer applied to the above example.



$$(\pi_{\text{ID, name, s\_id, i\_id}} (\text{student} \bowtie_{\text{ID = s\_id}} \text{advisor})) \bowtie_{\text{i\_id = instructor.ID}} \pi_{\text{ID, name}} (\text{instructor})$$

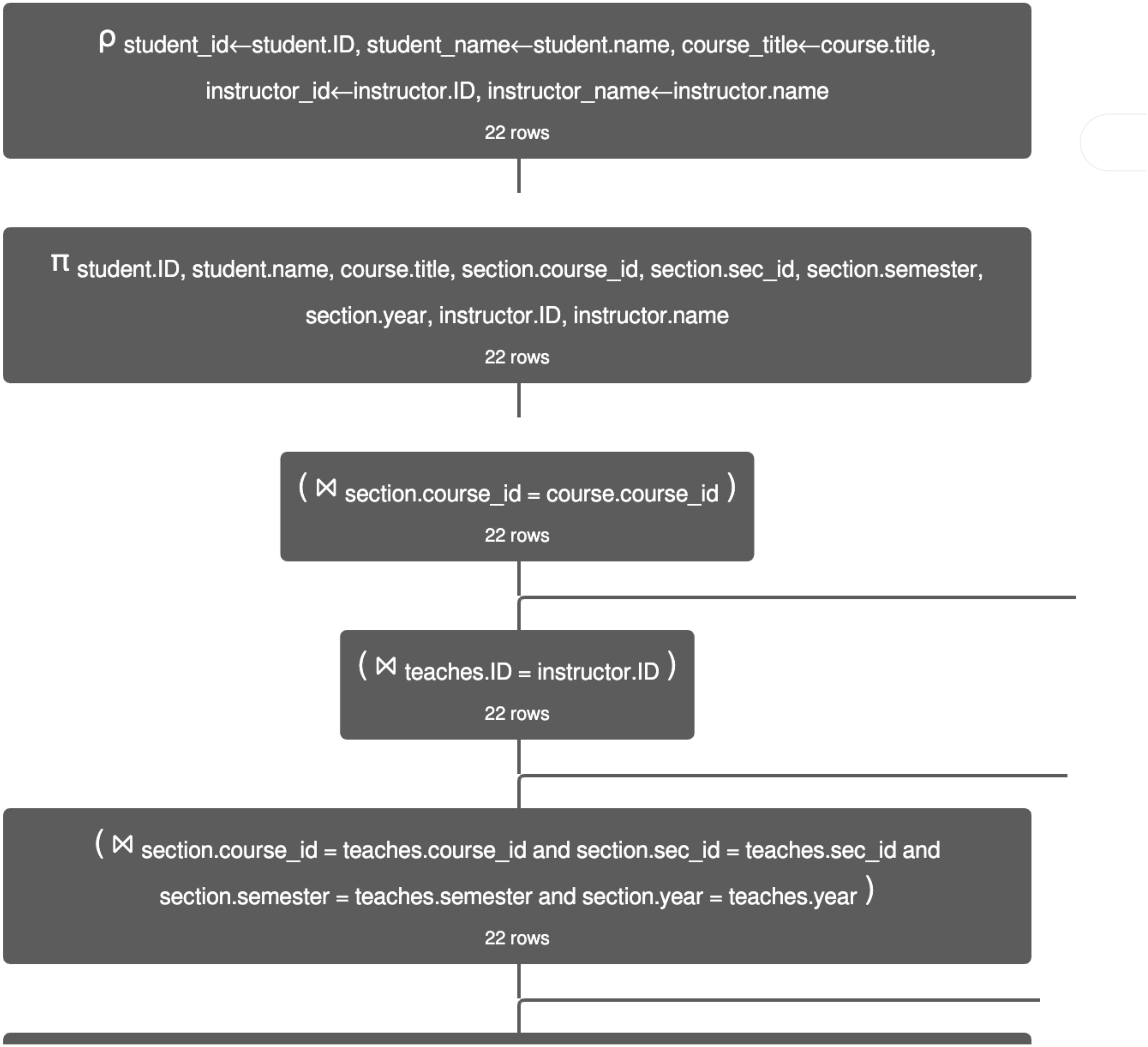
Execution time: 2 ms

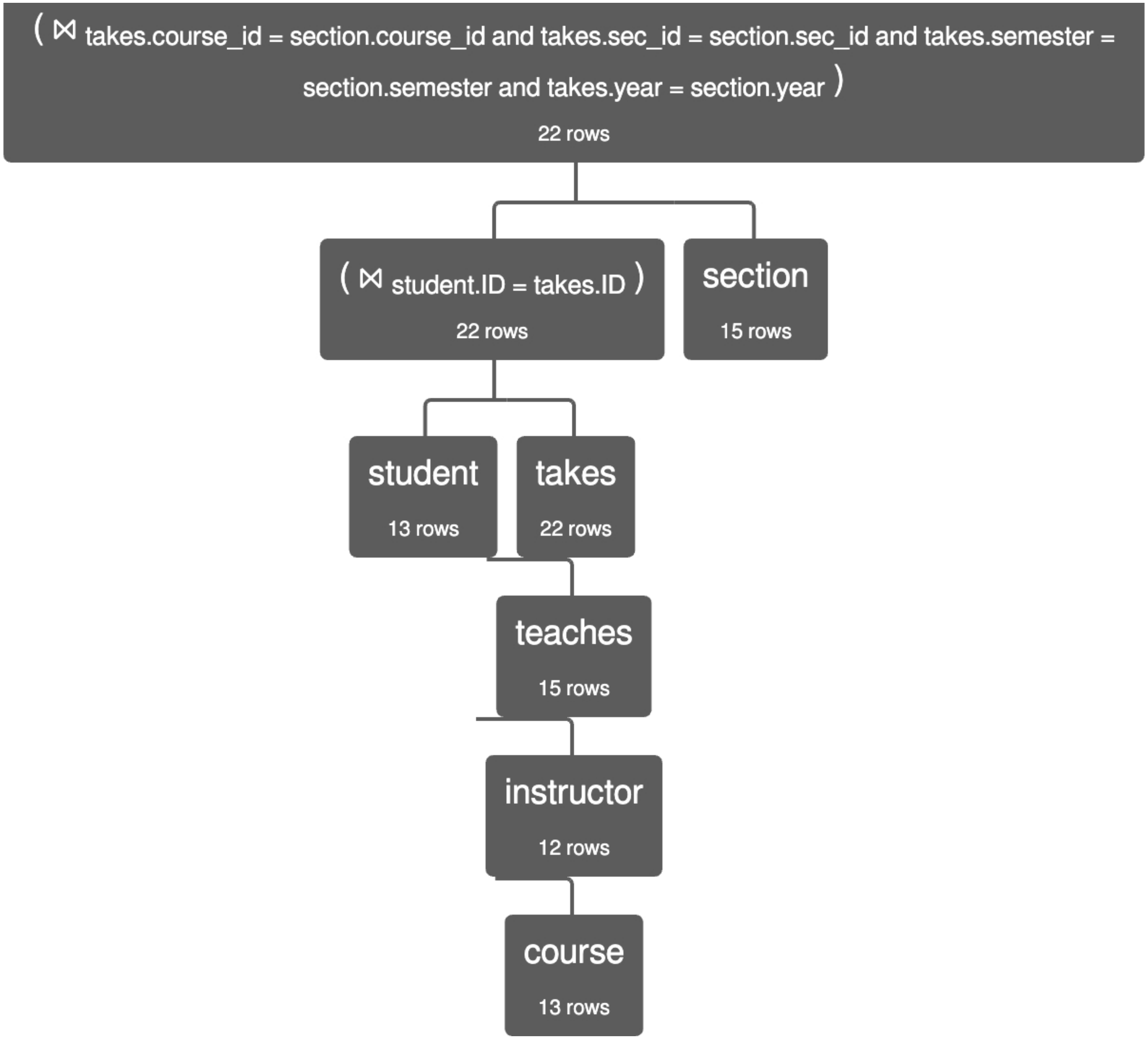
student.ID	student.name	advisor.s_id	advisor.i_id	instructor.ID	instructor.name
128	'Zhang'	128	45565	45565	'Katz'

In [35]:

```
img_path = '/Users/zhengfeichen/Desktop/Columbia/4111/W4111-Intro-to-Databases-Base/Homework-Assignme
Image(img_path)
```

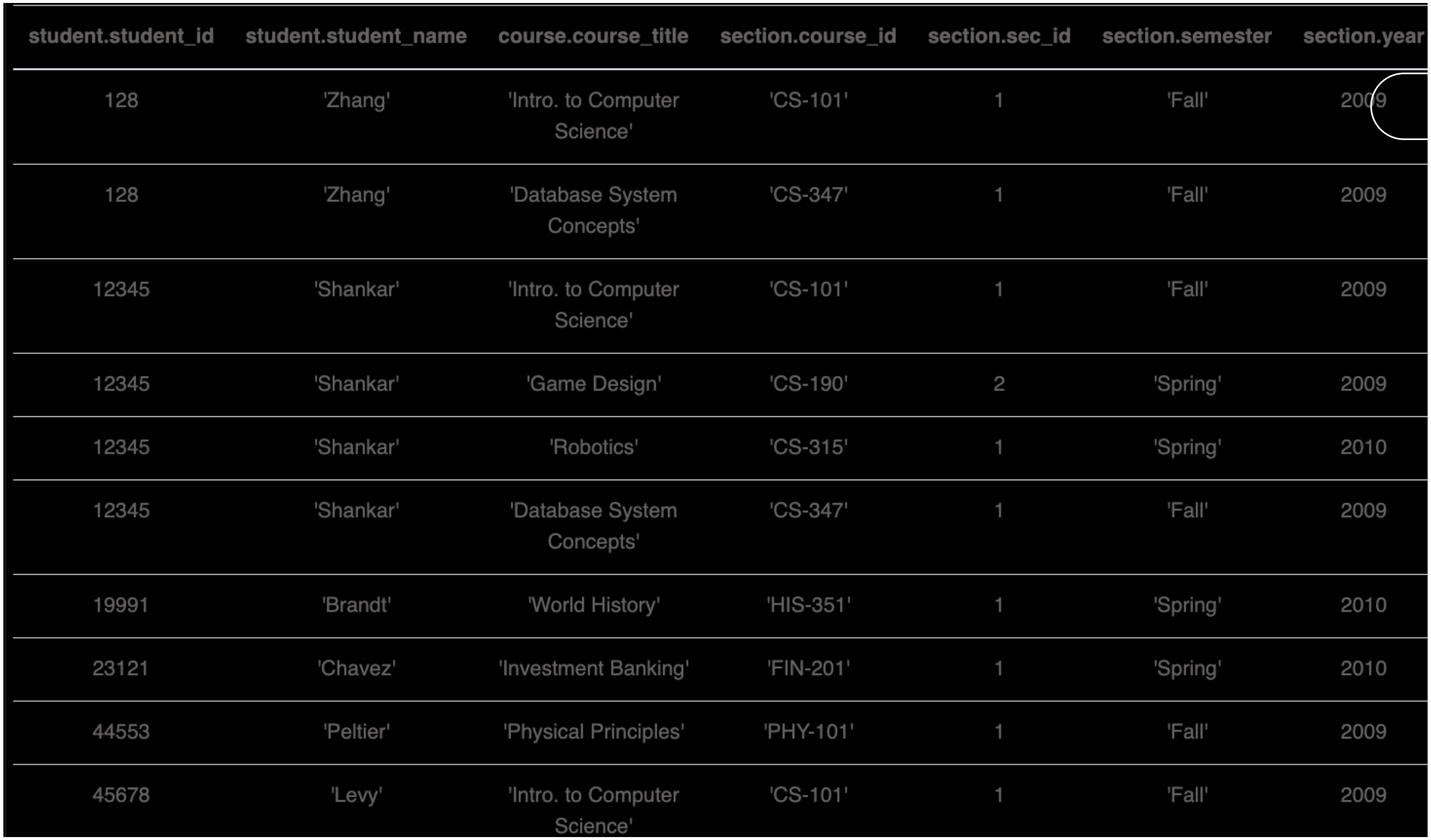
Out[35]:


ρ student_id←student.ID, student_name←student.name, course_title←course.title, instructor_id←instructor.ID, instructor_name←instructor.name
22 rows
π student.ID, student.name, course.title, section.course_id, section.sec_id, section.semester, section.year, instructor.ID, instructor.name
22 rows
(⋈ section.course_id = course.course_id)
22 rows
(⋈ teaches.ID = instructor.ID)
22 rows
(⋈ section.course_id = teaches.course_id and section.sec_id = teaches.sec_id and section.semester = teaches.semester and section.year = teaches.year)
22 rows


(⋈ takes.course_id = section.course_id and takes.sec_id = section.sec_id and takes.semester = section.semester and takes.year = section.year)
22 rows
(⋈ student.ID = takes.ID)
22 rows
section
15 rows
student
13 rows
takes
22 rows
teaches
15 rows
instructor
12 rows
course
13 rows

In [36]:
```
img_path = '/Users/zhengfeichen/Desktop/Columbia/4111/W4111-Intro-to-Databases-Base/Homework-Assignme
Image(img_path)
```

Out[36]:



student.student_id	student.student_name	course.course_title	section.course_id	section.sec_id	section.semester	section.year
128	'Zhang'	'Intro. to Computer Science'	'CS-101'	1	'Fall'	2009
128	'Zhang'	'Database System Concepts'	'CS-347'	1	'Fall'	2009
12345	'Shankar'	'Intro. to Computer Science'	'CS-101'	1	'Fall'	2009
12345	'Shankar'	'Game Design'	'CS-190'	2	'Spring'	2009
12345	'Shankar'	'Robotics'	'CS-315'	1	'Spring'	2010
12345	'Shankar'	'Database System Concepts'	'CS-347'	1	'Fall'	2009
19991	'Brandt'	'World History'	'HIS-351'	1	'Spring'	2010
23121	'Chavez'	'Investment Banking'	'FIN-201'	1	'Spring'	2010
44553	'Peltier'	'Physical Principles'	'PHY-101'	1	'Fall'	2009
45678	'Levy'	'Intro. to Computer Science'	'CS-101'	1	'Fall'	2009

Problem 2

Write a relational algebra expression that produces a result table with the following format:

`(dept_name, building, classroom, capacity)`

This contains tuples where:

- The department is in the building, e.g. there is a tuple in `department` that has the `dept_name` and `building`.
- The `classroom` is in the `building`.
- The result ONLY contains entries for which the department's `budget` is greather than 100,000 or the classroom's `capacity` is greater than 100.

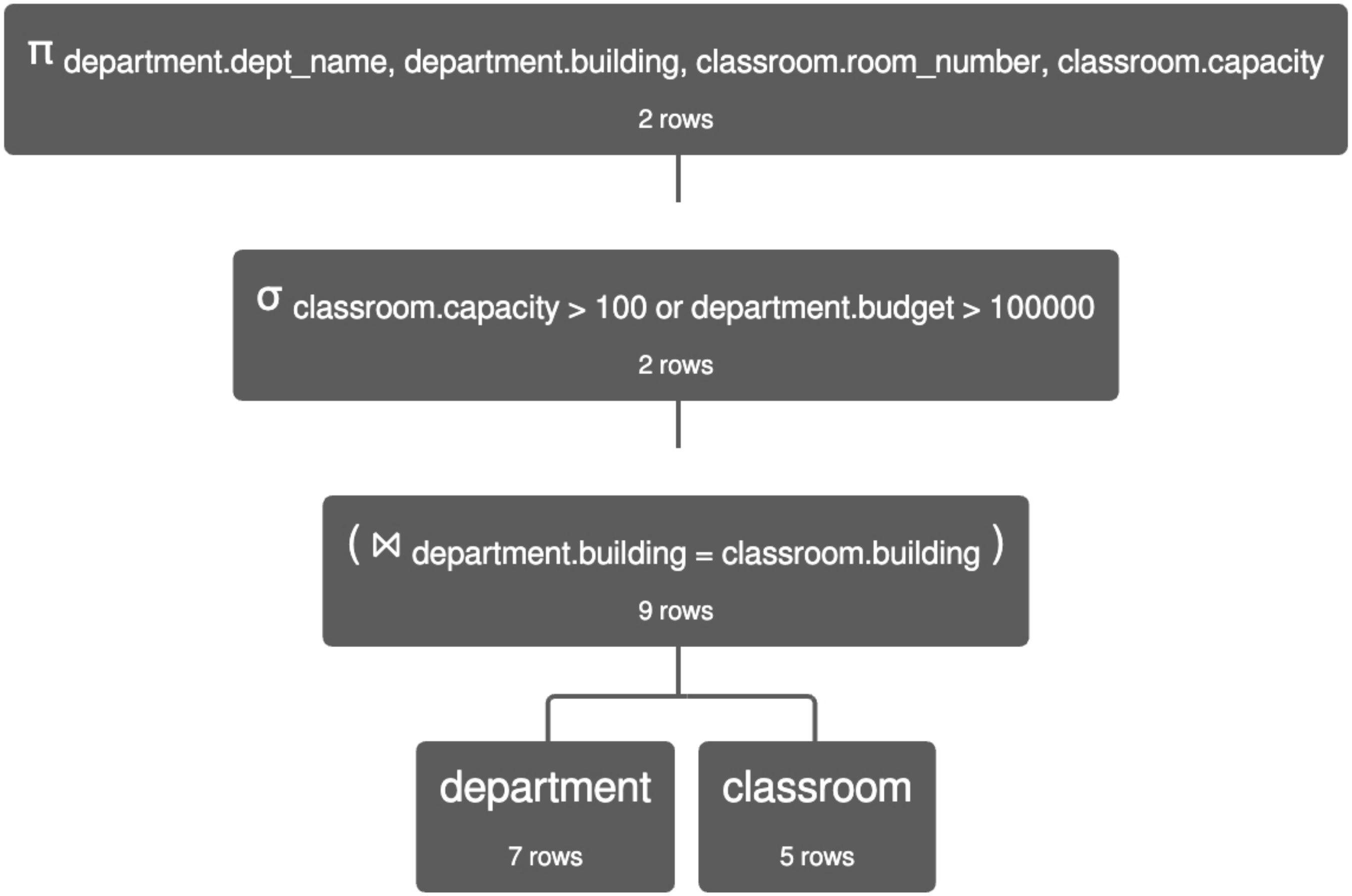
Past relational algebra here.

π department.dept_name, department.building, classroom.room_number, classroom.capacity σ classroom.capacity > 100 ∨ department.budget > 100000 (department ⋈ department.building = classroom.building classroom)

In [37]:

```
img_path = '/Users/zhengfeichen/Desktop/Columbia/4111/W4111-Intro-to-Databases-Base/Homework-Assignme
Image(img_path)
```

Out[37]:

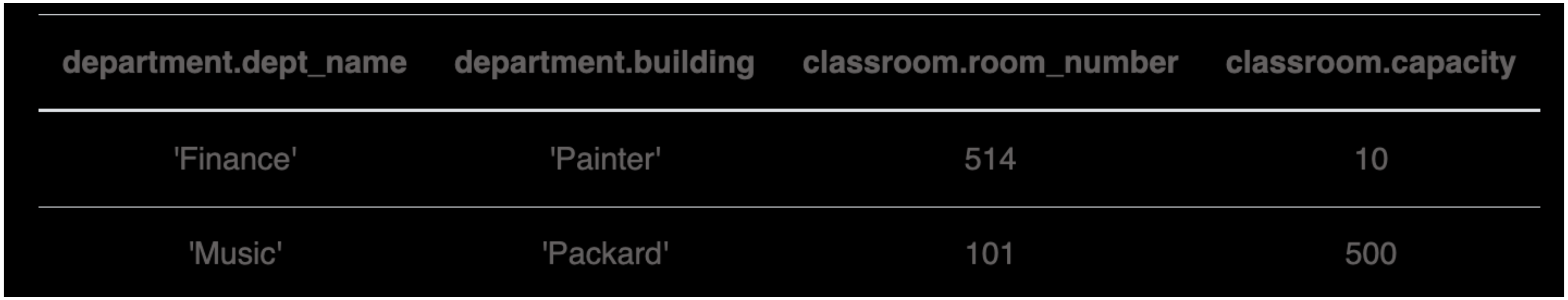

π department.dept_name, department.building, classroom.room_number, classroom.capacity
2 rows
σ classroom.capacity > 100 or department.budget > 100000
2 rows
(⋈ department.building = classroom.building)
9 rows
department
7 rows
classroom
5 rows

Execute your query on the Relax calculator and show an image of the first page of your result below.

In [38]:

```
img_path = '/Users/zhengfeichen/Desktop/Columbia/4111/W4111-Intro-to-Databases-Base/Homework-Assignme
Image(img_path)
```

Out[38]:



department.dept_name	department.building	classroom.room_number	classroom.capacity
'Finance'	'Painter'	514	10
'Music'	'Packard'	101	500

SQL

Use the database that is associated with the recommended textbook for these questions. You loaded this in HW0.

Problem 1

Write a SQL query that produces a table of the form `(student_id, student_name, advisor_id, advisor_name)` that shows the ID and name of a student combined with their advisor. Only include rows where both the student and the advisor are in the `Comp. Sci.` and the student has at least 50 total credits.

Execute your SQL below.

In [20]:
```
%%sql
use db_book
```

```
 * mysql+pymysql://root:***@localhost
0 rows affected.
```

Out[20]:
```
[]
```

In [21]:
```
%%sql
select student.ID, student.name, student.tot_cred, instructor.ID as advisor_id, instructor.name as ad
from student join advisor on student.ID = advisor.s_ID join instructor on advisor.i_ID = instructor.I
where student.tot_cred >= 50 and instructor.dept_name = 'Comp. Sci.'
```

```
 * mysql+pymysql://root:***@localhost
2 rows affected.
```

Out[21]:

ID	name	tot_cred	advisor_id	advisor_name
00128	Zhang	102	45565	Katz
76543	Brown	58	45565	Katz

Problem 2

Consider the following query.

In [22]:
```
%%sql
select * from db_book.student where dept_name='Comp. Sci.'
```

```
 * mysql+pymysql://root:***@localhost
4 rows affected.
```

Out[22]:

ID	name	dept_name	tot_cred
00128	Zhang	Comp. Sci.	102
12345	Shankar	Comp. Sci.	32
54321	Williams	Comp. Sci.	54
76543	Brown	Comp. Sci.	58

The following table makes a copy of the student table.

```
In [25]: %sql create table student_hw1b as select * from student
```

```
 * mysql+pymysql://root:***@localhost
13 rows affected.
```

```
Out[25]: []
```

```
In [26]: %sql select * from student_hw1b where dept_name='Comp. Sci.'
```

```
 * mysql+pymysql://root:***@localhost
4 rows affected.
```

Out[26]:

ID	name	dept_name	tot_cred
00128	Zhang	Comp. Sci.	102
12345	Shankar	Comp. Sci.	32
54321	Williams	Comp. Sci.	54
76543	Brown	Comp. Sci.	58

We are now going to make some changes to `student_hw1b`

Write and execute a SQL statement that changes Willliams tot_cred to 75.

```
In [27]: %sql update student_hw1b set tot_cred = 75 where name = 'Williams';
```

```
 * mysql+pymysql://root:***@localhost
1 rows affected.
```

```
Out[27]: []
```

Show the result.

In [28]:
```
%sql select * from student_hw1b where dept_name='Comp. Sci.'
```

```
 * mysql+pymysql://root:***@localhost
4 rows affected.
```

Out[28]:

ID	name	dept_name	tot_cred
00128	Zhang	Comp. Sci.	102
12345	Shankar	Comp. Sci.	32
54321	Williams	Comp. Sci.	75
76543	Brown	Comp. Sci.	58

Write a SQL statement that deletes Williams from the `student_hw1b` table and execute in the cell below.

In [30]:
```
%%sql
DELETE FROM student_hw1b WHERE name = 'Williams';
```

```
 * mysql+pymysql://root:***@localhost
1 rows affected.
```

Out[30]:
```
[]
```

Show the resulting table.

In [31]:
```
%sql select * from student_hw1b where dept_name='Comp. Sci.'
```

```
 * mysql+pymysql://root:***@localhost
3 rows affected.
```

Out[31]:

ID	name	dept_name	tot_cred
00128	Zhang	Comp. Sci.	102
12345	Shankar	Comp. Sci.	32
76543	Brown	Comp. Sci.	58

Write and execute SQL statement that puts the original data for Williams back in the table.

In [32]:

```
%%sql INSERT INTO student_hw1b (ID, name, dept_name, tot_cred)
SELECT ID, name, dept_name, tot_cred
FROM student
WHERE name = 'Williams'
```

```
 * mysql+pymysql://root:***@localhost
1 rows affected.
```

Out[32]:

```
[]
```

Show the table.

In [33]:

```
%sql select * from student_hw1b where dept_name='Comp. Sci.'
```

```
 * mysql+pymysql://root:***@localhost
4 rows affected.
```

Out[33]:

ID	name	dept_name	tot_cred
00128	Zhang	Comp. Sci.	102
12345	Shankar	Comp. Sci.	32
76543	Brown	Comp. Sci.	58
54321	Williams	Comp. Sci.	54