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

Homework 1
Introduction to Core Concepts, ER Modeling, Relational Algebra, SQL

Introduction

This notebook contains Homework 1. **Both Programming and Nonprogramming tracks should complete this homework.**

Submission Instructions

- You will submit **PDF and ZIP files** for this assignment. Gradescope will have two separate assignments for these.
- For the PDF:
  - 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.
  - **MAKE SURE YOU DON'T SUBMIT A SINGLE PAGE PDF.** Your PDF should have multiple pages.
- For the ZIP:
  - Zip a folder containing this notebook and any screenshots.
  - You may delete any unnecessary files, such as caches.

Add Student Information

In [1]:
```
# Print your name, uni, and track below

name = "Sparsh Binjrajka"
```

```
uni = "sb4835"
track = "Programming Track"

print(name)
print(uni)
print(track)
```

```
Sparsh Binjrajka
sb4835
Programming Track
```

Setup

SQL Magic

The `sql` extension was installed in HW0. Double check that if this cell doesn't work.

In [2]:
```
%load_ext sql
```

You may need to change the password below.

In [3]:
```
%sql mysql+pymysql://root:dbuserdbuser@localhost
```

In [4]:
```
%sql SELECT * FROM db_book.student WHERE ID = 12345
```

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

Out[4]:

ID	name	dept_name	tot_cred
12345	Shankar	Comp. Sci.	32

Python Libraries

In [5]:
```
from IPython.display import Image
import pandas
```

Written Questions

Chapter 1 from the recommended textbook Database System Concepts, Seventh Edition covers general information and concepts about databases and database management systems. Lecturing on the general and background information is not a good use of precious class time. To be more efficient with class time, the chapter 1 information is a reading assignment.

Answering the written questions in HW 1, Part 1 does not require purchasing the textbook and reading the chapter. The chapter 1 slides provided by the textbook authors provide the

necessary information. In some cases, students may also have to search the web or other sources to “read” the necessary information.

When answering the written questions, do not “bloviate”. The quantity of words does not correlate with the quality of the answer. We will deduct points if you are not succinct. The answers to the questions require less than five sentences or bullet points.

“If you can't explain something in a few words, try fewer.”

You may use external resources, but you should cite your sources.

W1

What is a database management system and how do relational databases organize data?

Database management system (DBMS) is a way to organise and store data in a systemic way. It allows users to edit, create, delete and store data in databases. (https://en.wikipedia.org/wiki/Database) Relational model is one of the ways of DBMS where data is stored as a collection of tables (or relations) where each row in a table represents a specific data point and the columns represent attributes of the data.

W2

Columbia University uses several applications that use databases to run the university. Examples are SSOL and CourseWorks. An alternate approach could be letting students, faculty, administrators, etc. use shared Google Sheets to create, retrieve, update, and delete information. What are some problems with the shared spread sheet approach and what functions do DMBS implement to solve the problems?

A shared Google Sheets allows anyone with access to the sheet to retrieve, edit, and delete anyone's data which is problematic since not everyone should be able to access let alone edit/delte privileged information and there is no way to ensure a hierarchical access system in a shared resource. DBMS can create several entity sets and ensure only certain sets of people have access to all information. For example: students should only have editing access to their own information but could be allowed to view faculty info, course info, etc.

W3

Explain the differences between SQL, MySQL Server and DataGrip.

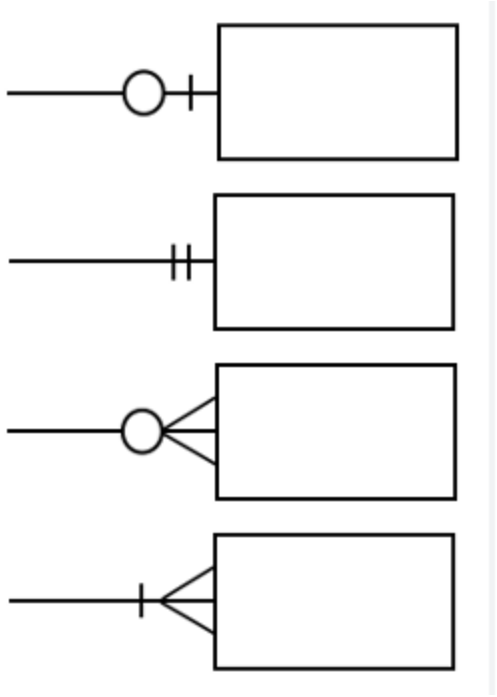
SQL is a language that is used to manage a DBMS. It provides functionality such as SELECT, INSERT, UPDATE, etc which can be used to create databases.

MySQL is a type of DBMS (relational) that uses SQL to manage database. It is open-source and one of the popular DBMS that is used to store data.

DataGrip is just an IDE (Integrated Development Environment) that provides an interface for users to interact with databases. It isn't a DBMS but just a tool that users can use to interact with a DBMS.

W4

Crow's Foot Notation has four endings for relationship lines. Briefly explain the meaning of each ending.



1. The entity on the left side (not pictured) is related to 0 or 1 entities of right side set.
2. Left entity is related to exactly one right side entity set.
3. Left Entity is related to 0, 1, or more entities on right side set.
4. Left entity is related to either 1 or more entities of right side set.

W5

What is a primary key and why is it important?

Primary key is a subset of attributes of an entity set that is used to uniquely identify an entity of that set. This is important since it helps in distinguishing entities of the same type and prevents duplicate entries from existing within an entity set.

W6

The relational algebra is closed under the operators. Explain what this means and give an example.

An operator takes one or two relations as inputs. Being "closed under the operator" means that the output of the operator is also a relation. For example, if there is a "Students" table with "uni" and "name" attributes and there exists a record with "name":"Sparsh" and "uni":"sb4835" then the operation: `select uni="sb4835"(Students)` selects those students with "uni" as "sb4835" and this output is also a relation which can be further used as input for another operator.

W7

Some of the Columbia University databases/applications represent the year/semester attribute of a section in the form "2023_2". The first four characters are the academic year, and the last character is the semester (1, 2, or 3). The data type for this attribute might be CHAR(6). Using this example, explain the concepts of domain and atomic domain. How is domain different from type?

The set of allowed values for each attribute is called the domain. The domain for the year/sem attribute is CHAR(6). That is, the possible values for this attribute are all strings of exactly 6 characters. However, this is not an atomic domain since it is composed of two separate domains (year and semester).

W8

Briefly explain the difference between a database schema and database instance.

Schema is the high-level definition of a relation. It specifies the name and attributes of that relation. Instance is the realization of the schema. That is, it is the actual data (or collection of rows) currently being stored in the database in that relation.

W9

Briefly explain the concepts of data definition language and data manipulation language.

Data definition language (DDL) is a way to define the structure of the relations whereas the data manipulation language (DML) is a way to manipulate the data through operations like select, insert, update, delete, etc.

W10

What is physical data independence?

There are three levels in the data abstraction hierarchy with conceptual at the top, next logical, and finally physical. The physical data independence is the ability to modify the physical layer without affecting the other two layers. Intuitively, since the physical layer specifies how the actual data is being stored, changing that should not affect the logical relations between the data itself.

Entity-Relationship Modeling

Overview

The ability to understand a general description of a requested data model and to transform into a more precise, specified *logical model* is one of the most important skills for using databases. SW and data engineers build applications and data models for end-users. The end-users, product managers and business managers are not SW or data modeling experts. They will express their *intent* in imprecise, text and words.

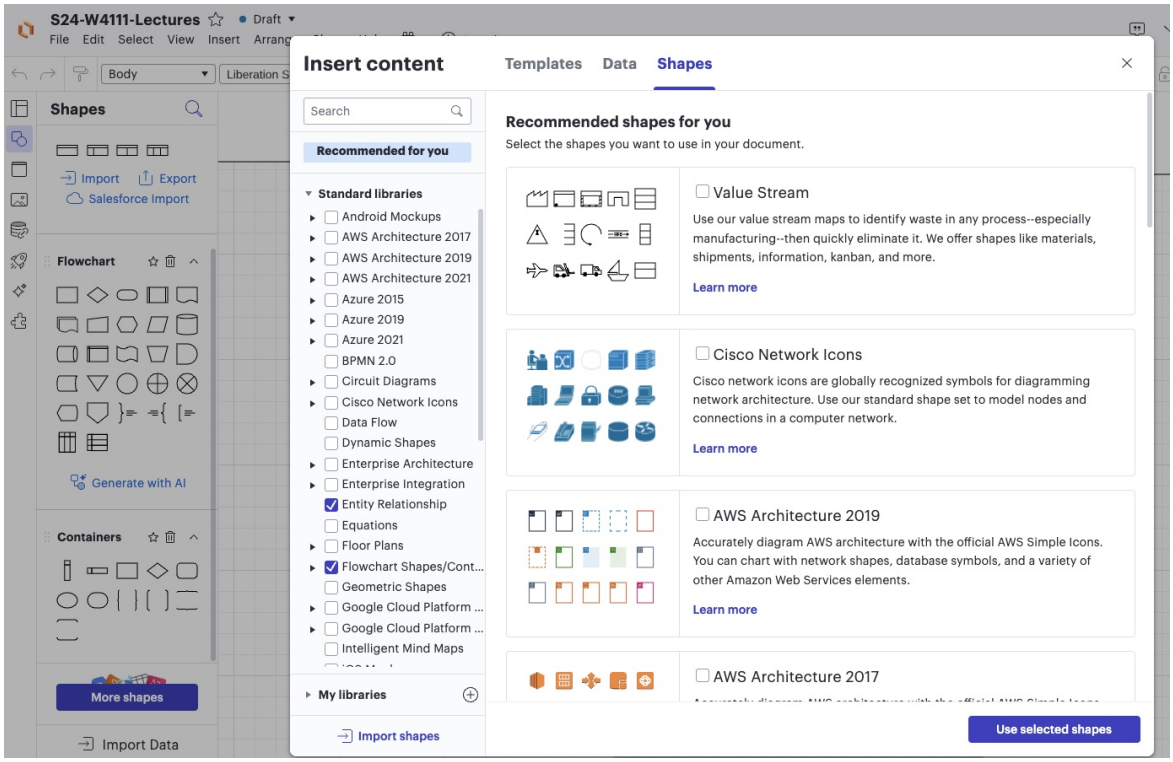
The users and business stakeholder often can understand and interact using a *conceptual model* but details like keys, foreign keys, ... are outside their scope.

In this problem, you will:

- Understand a short written description of a requested data model.
- Produce a *conceptual data model diagram* using Lucidchart.
- Produce a *logical data model diagram* using Lucidchart.

You can sign up for a free Lucidchart account. The free account provides the capabilities you will need for this course.

To draw the diagrams, you need to add the *entity relationship* shapes. Lecture 2 demonstrated how to add the shapes.



Adding Entity Relationship Shapes

We provide a simple Lucidchart document from Lecture 2 that helps you get started. You need a Lucidchart account to access the document and diagrams.

Data Model Description

The data model represents banks, customers. employees and accouts. The model has the following entity types/sets:

1. *Customer*
2. *Employee* of the banking company
3. *Branch,* which is a location of one of the banks offices
4. *Savings Account*
5. *Checking Account*
6. *Loan*
7. *Portfolio*

Customer has the following properties:

- *customerID*
- *lastName*
- *firstName*
- *email*
- *dateOfBirth*

Employee has the following properties:

- *employeeID*
- *lastName*
- *firstName*
- *jobTitle*

Branch has the following properties:

- *branchID*
- *zipCode*

Savings Account has the following properties:

- *accountID*
- *balance*
- *interestRate*

Checking Account has the following properties:

- *accountID*
- *balance*

Loan has the following properties.

- *loanID*
- *balance*
- *interestRate*

Portfolio has the following properties:

- *portfolioID*
- *createdDate*

The data model has the following relationships:

- *Customer Branch* connects a customer and a branch. A *Customer* is connected to exactly one *Branch*. A *Branch* may have 0, 1 or many customers.
- *Employee Branch* connects an employee and a branch. An *Employee* is connected to exactly one *Branch*. A *Branch* may have 0, 1 or many associated employees.
- *Savings Account Branch*, *Checking Account Branch*, and *Loan Branch* all have the same pattern.
  - An account/loan has exactly one branch.
  - A *Branch* many have 0, 1 or many accounts/loans.
- *Savings Customer*, *Checking Customer*, *Loan Customer*, and *Portfolio Customer* follow the same pattern.
  - The account/loan has exactly one customer.
  - The customer may have 0 or 1 of each type of account.

- A *Portfolio* is related to exactly one *Customer*, exactly one *Savings Account*, exactly one *Checking Account*, and exactly one *Loan*.
- *Portfolio Advisor* relates a *Portfolio* and *Employee*. An *Employee* may be the advisor for 0, 1 or many *Portfolios*. A *Portfolio* may have at most one *Employee* advisor.

Answer

1. Place your Logical Model diagram below.
2. You *may* have to add attributes to entities to implement the model.
3. You *may* make reasonable assumptions. Please document your assumptions below. You may add comments/notes to your diagram for clarity.

Assumptions:

1. Savings, Checking, Loan, and Porfolio are all related to exactly one branch and one customer. In including customerID as a FK in each of the above entities, we also model the branchID since customer entity contains branchID.

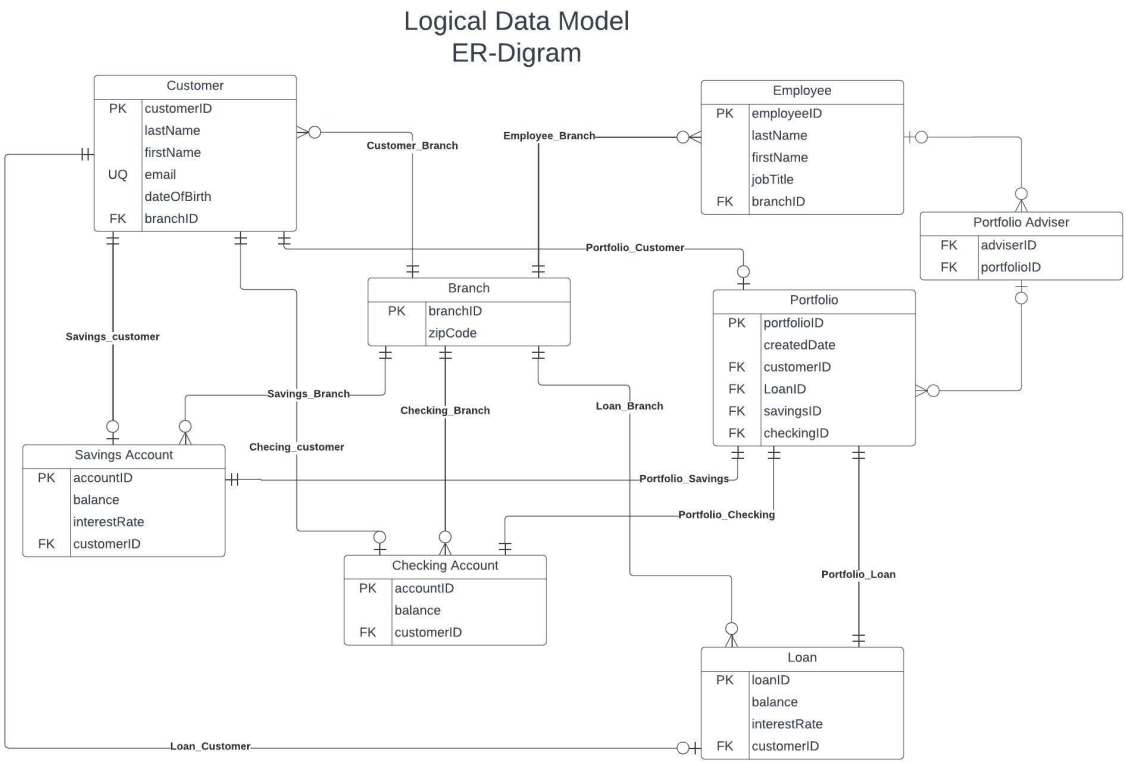
2. If a customer has a portfolio then they do not have an additional savings/checking/loan since by conditions every portfolio is related to exactly one of each type of customer, savings, checking, and loan. Further, every customer can have at most 1 account of each type.

3. In Portfolio entity, we include savingsID, checkingID, loanID as FK to model the fact that every portfolio has exactly one account of each type.

4. We create an associateive entity called Portfolio Adviser that stores the employeeID of the advisor of portfolio corresponding to portfolioID. We can make a composite PK since every portfolio can have at most one advisor. For now, I've just let it both be FKs.

ER Diagram:

Save your diagram to an image, place in the same directory as your notebook and change the file name in the HTML `img` tag in this Markdown cell.



Logical ER Diagram

Relational Algebra

R-1

The following is the SQL DDL for the `db_book.classroom` table.

```
CREATE TABLE IF NOT EXISTS db_book.classroom
(
    building    VARCHAR(15) NOT NULL,
    room_number VARCHAR(7)  NOT NULL,
    capacity    DECIMAL(4)  NULL,
    PRIMARY KEY (building, room_number)
);
```

Using the notation from the lecture slides, provide the corresponding relation schema definition.

db_book.classroom(building:varchar, room_number:varchar, capacity:decimal)

Answer Format

For the answers to the relational algebra questions, you will use the RelaX calculator with the schema associated with the book. You answer should include the algebra statement in as text and a screenshot of the execution result. Question **R0** below shows a sample of that the answer will look like.

R0

Write a relational algebra statement that produces a table of the following form:

- ID is the instructor ID
- name is the instructor name
- course_id, sec_id, semester, year of a section
- building, room_number

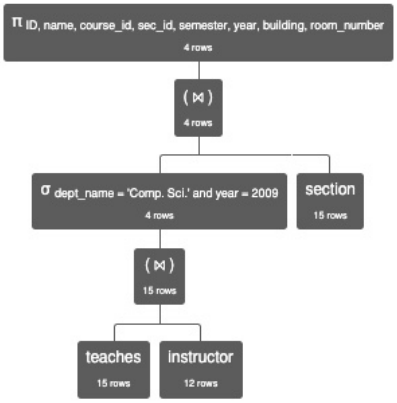
Note:

1. You will have to use the instructor, teaches and section relations
2. Your answer should only include sections taught in `Comp. Sci.` in `2009`

Algebra statement:

```
π ID, name, course_id, sec_id, semester, year, building,
room_number(
    (σ dept_name='Comp. Sci.' ∧ year=2009
        (teaches ⋈ instructor)
        )
⋈ section)
```

Execution:



π ID, name, course_id, sec_id, semester, year, building, room_number ((σ dept_name = 'Comp. Sci.' and year = 2009 (teaches ⋈ instructor)) ⋈ section)

Execution time: 1 ms

teaches.ID	instructor.name	teaches.course_id	teaches.sec_id	teaches.semester	teaches.year	section.building	section.room_number
10101	'Srinivasan'	'CS-101'	1	'Fall'	2009	'Packard'	101
10101	'Srinivasan'	'CS-347'	1	'Fall'	2009	'Taylor'	3128
83821	'Brandt'	'CS-190'	1	'Spring'	2009	'Taylor'	3128
83821	'Brandt'	'CS-190'	2	'Spring'	2009	'Taylor'	3128

‹ 1 ›

RO Execution Result

R1

Write a relational algebra statement that produces a relation with the columns:

- `student.name`
- `student.dept_name`
- `student.tot_cred`
- `instructor.name` (the instructor that advises the student)
- `instructor.dept_name`

Only keep students who have earned more than 90 credits.

Note:

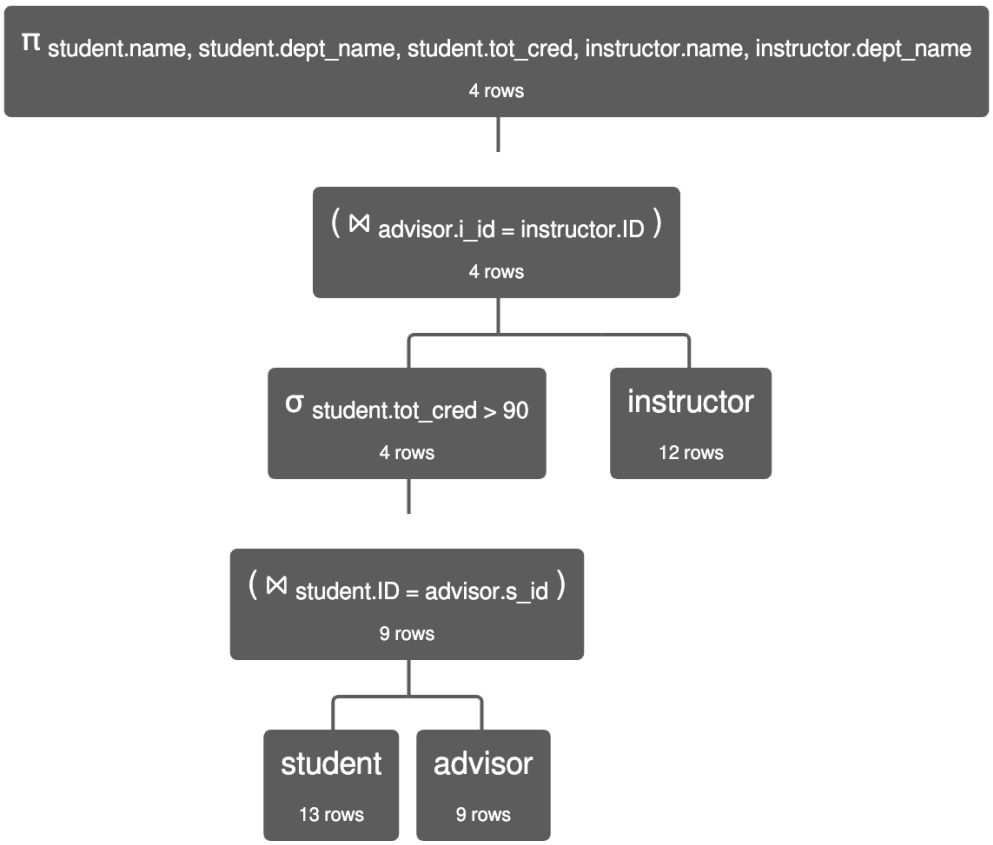
1. You will have to use the `student`, `instructor`, and `advisor` relations.
2. You should only include students that have an advisor, i.e., `instructor.name` and `instructor.dept_name` should be non-null for all rows.

Algebra statement:

```
π student.name, student.dept_name, student.tot_cred,
instructor.name, instructor.dept_name (
```

```
σ student.tot_cred > 90 (
    student ⋈ student.ID = advisor.s_id advisor)
    ⋈ advisor.i_id = instructor.ID instructor)
```

Execution:



R1 Execution Result

R2

Write a relational algebra statement that produces a relation with the columns:

- `course_id`
- `title`
- `prereq_course_id`
- `prereq_course_title`

This relation represents courses and their prereqs.

Note:

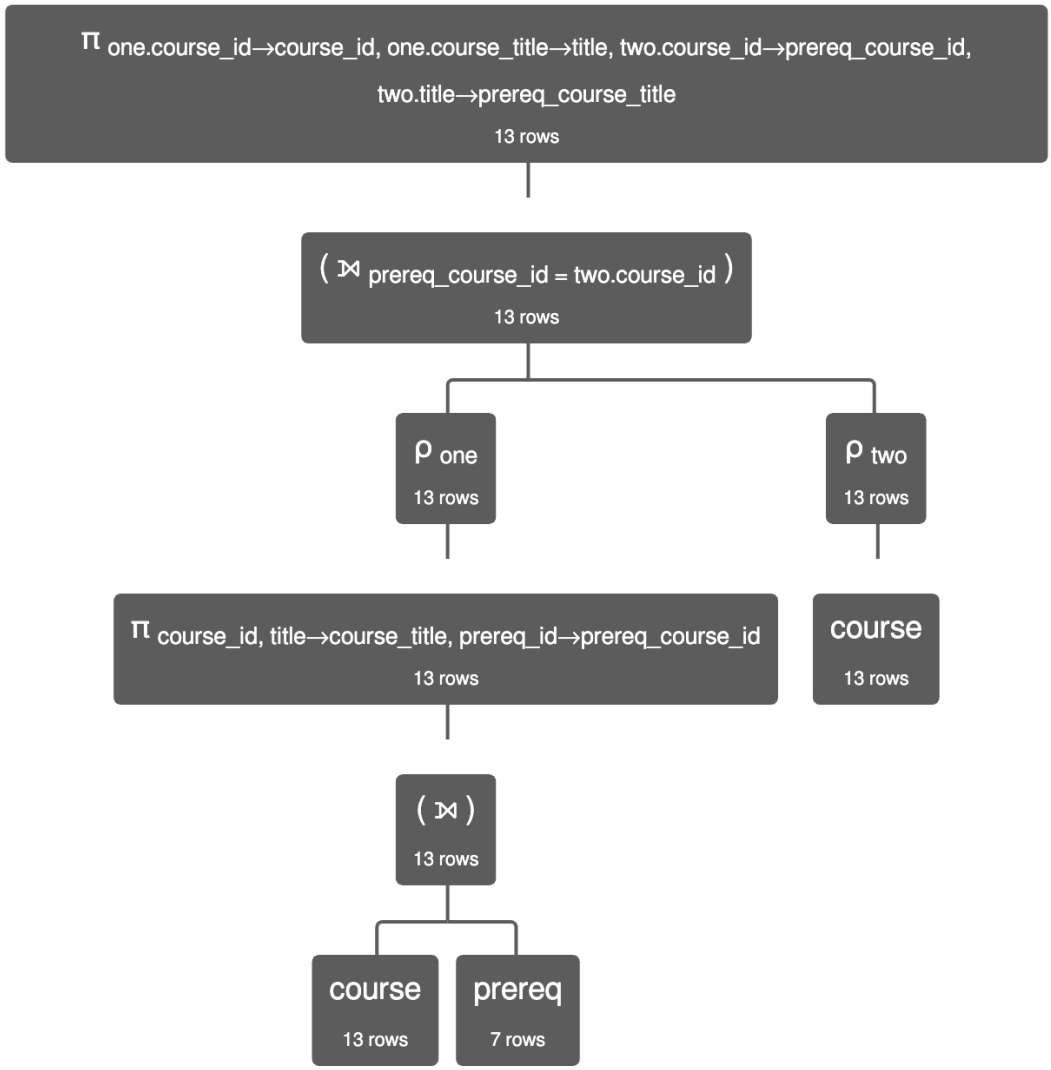
1. This query requires the `course` and `prereq` tables.
2. Your answer should only include courses in the `Comp. Sci.` department.
3. If a course has no prereqs, `prereq_course_id` and `prereq_course_title` should both be *null*.

4. You *may* have to use table and column renaming.

Algebra statement:

```
π course_id <- one.course_id, title <- one.course_title,
prereq_course_id <- two.course_id, prereq_course_title <-
two.title (
    (ρ one (π course_id, course_title <- title,
prereq_course_id <- prereq_id (course ⋈ prereq)))
⋈ prereq_course_id = two.course_id
(ρ two (course))
)
```

Execution:



R2 Execution Result

SQL

New Database

MySQL Tutorial is a good site with information that complements and extends the core material in our course. Much of the material the site covers is applicable to other SQL products. MySQL Tutorial uses an interesting dataset that is more complex than the simple "db_book" database. This is the Classic Models Dataset. The complexity allows us to better appreciate more complex SQL concepts.

You learned how to run a SQL script/file as part of HW0. **Use the same approach to load and create the `Classic Models Database`.** The file is `classic-models-database.sql` and is in the HW1 folder.

To test loading the data, you can use the cell below.

In [6]:
```
%sql USE classicmodels;
```

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

Out[6]:
```
[]
```

In [7]:
```
%sql show tables;
```

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

Out[7]:

Tables_in_classicmodels
customers
employees
offices
orderdetails
orders
payments
productlines
products

SQL 1

This query uses `customers` and `employees`.

Write and execute a SQL query that produces a table with the following columns:

- `customerContactName`
- `customerPhone`
- `salesRepName`

Only keep customers from France. Order your output by `customerContactName`.

Notes:

- The names of your columns must match exactly with what is specified.
- `customerContactName` can be formed by combining `customers.contactFirstName` and `customers.contactLastName`.
- `salesRepName` can be formed by combining `employees.firstName` and `employees.lastName`.

In [8]:
```
%%sql
SELECT CONCAT(customers.contactFirstName, ' ',customers.contactLastName) as cus
    customers.phone as customerPhone,
    CONCAT(employees.firstName, ' ', employees.lastName) as salesRepName
FROM customers join employees on customers.salesRepEmployeeNumber = employees.
where country = 'France'
order by customerContactName
```

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

Out[8]:

customerContactName	customerPhone	salesRepName
Annette Roulet	61.77.6555	Gerard Hernandez
Carine Schmitt	40.32.2555	Gerard Hernandez
Daniel Tonini	30.59.8555	Gerard Hernandez
Daniel Da Silva	+33 1 46 62 7555	Loui Bondur
Dominique Perrier	(1) 47.55.6555	Loui Bondur
Frédérique Citeaux	88.60.1555	Gerard Hernandez
Janine Labrune	40.67.8555	Gerard Hernandez
Laurence Lebihan	91.24.4555	Loui Bondur
Marie Bertrand	(1) 42.34.2555	Loui Bondur
Martine Rancé	20.16.1555	Gerard Hernandez
Mary Saveley	78.32.5555	Loui Bondur
Paul Henriot	26.47.1555	Loui Bondur

SQL 2

This query uses `employees`, `customers`, `orders`, `orderdetails`.

Write and execute a SQL query that produces a table showing the amount of money each sales rep has generated.

Your table should have the following columns:

- `salesRepName`
- `moneyGenerated`

Order your output from greatest to least `moneyGenerated`.

Notes:

- The names of your columns must match exactly with what is specified.
- `salesRepName` can be formed by combining `employees.firstName` and `employees.lastName`.
- To calculate `moneyGenerated`:
  - Every order in `orders` is associated with multiple rows in `orderdetails`. The total amount of money spent on an order is the sum of `quantityOrdered * priceEach` for all the associated rows in `orderdetails`. **Only consider orders that are `Shipped`.**
  - A customer can have multiple orders. The total amount of money a customer has spent is the sum of the money spent on all that customer's orders.
  - A sales rep can have multiple customers. `moneyGenerated` is the sum of the money spent by all that sales rep's customers.
- You may find the WITH keyword to be useful for cleaner code.

In [9]:
```
%%sql
with
    one as (
        select * from customers left join orders using(customerNumber)
    ),

    two as (
        select orderNumber, customerNumber, sum(quantityOrdered * priceEach) as
        from orders natural join orderdetails
        where status = 'Shipped'
        group by orderNumber, customerNumber
    ),

    three as (
        select customerNumber, customerName, country, orderRevenue
        from one join two using (customerNumber, orderNumber)
    ),

    four as (select customerName, sum(orderRevenue) as totalRevenue
             from three
             group by customerNumber, customerName
             order by totalRevenue
    ),

    five as (
         select employeeNumber, CONCAT(firstName, ' ', lastName) as salesRepNa
         from employees left join customers on (employeeNumber = salesRepEmplo
         where jobTitle = 'Sales Rep'
    )
select salesRepName, ifnull(sum(totalRevenue),0) as moneyGenerated
from five left join four using(customerName)
group by salesRepName
order by moneyGenerated
```

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

Out[9]:

salesRepName	moneyGenerated
Tom King	0.00
Yoshimi Kato	0.00
Leslie Thompson	307952.43
Julie Firrelli	386663.20
Martin Gerard	387477.47
Steve Patterson	449219.13
Mami Nishi	457110.07
Foon Yue Tseng	488212.67
Andy Fixter	509385.82
Peter Marsh	523860.78
Loui Bondur	569485.75
George Vanauf	584406.80
Barry Jones	637672.65
Larry Bott	686653.25
Pamela Castillo	790297.44
Leslie Jennings	1021661.89
Gerard Hernandez	1065035.29

In [ ]: