# 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 [10]:
```
# Print your name, uni, and track below

name = "Arman Ozcan"
uni = "ao2794"
track = "Nonprogramming Track"

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

```
Arman Ozcan
ao2794
Nonprogramming Track
```

# Setup

## SQL Magic

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

In [11]:
```
%load_ext sql
```

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

You may need to change the password below.

In [12]:
```
%sql mysql+pymysql://root:dbuserdbuser@localhost
```

In [13]:
```
%sql SELECT * FROM db_book.student WHERE ID = 12345
```

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

Out[13]:

| ID | name | dept_name | tot_cred |
|---|---|---|---|
| 12345 | Shankar | Comp. Sci. | 32 |

## Python Libraries

In [14]:
```
from IPython.display import Image
import pandas
```

# Written Questions

Chapter 1 from the recommended textbook Database System Concepts, Seventh Edition (https://codex.cs.yale.edu/avi/db-book/) 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 (https://codex.cs.yale.edu/avi/db-book/slides-dir/index.html) 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?

A database-management system (DBMS) is a collection of interrelated data and a set of programs to access those data. The primary goal of a DBMS is to provide a way to store and retrieve database information that is both convenient and efficient. Relational databases organize data as a collection of tables, representing both data and the relationships among those data. Each table has multiple columns, and each column has a unique name. Tables are also known as relations.

# 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?

First, when multiple users edit the document at the same time in Google Sheets, there can be conflicts that result in overwrites or lost changes, while DMBS handle that well with concurrency control. DBMS provide comprehensive data integrity features, such as primary and foreign key constraints and unique constraints, to ensure that the data stored in the database remains accurate and consistent, which is much harder to maintain in Google Sheets where there is no such constraints. Unlike Google Sheets, DBMS also support complex querying capabilities, with SQL (Structured Query Language), which allows for complex data retrieval, manipulation, and analysis operations across multiple tables and datasets. Lastly, DBMS are designed to efficiently handle large volumes of data and high transaction volumes and optimize performance, whereas Google Sheets are not as scalable.

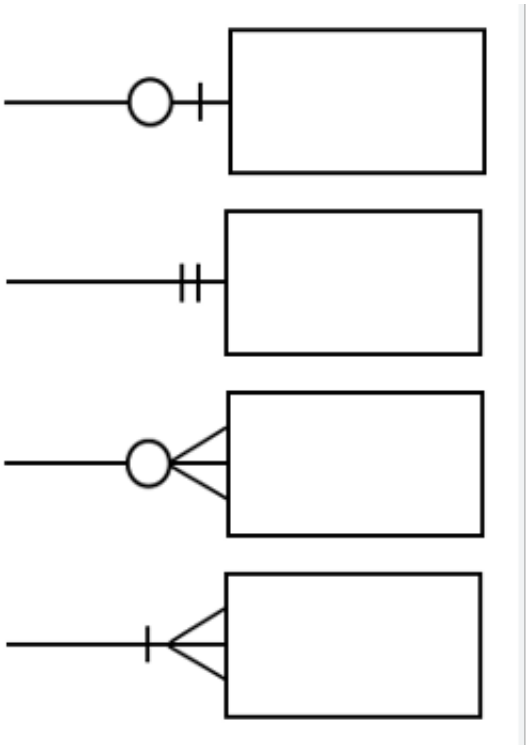
# W3

Explain the differences between SQL, MySQL Server and DataGrip.

SQL is a programming language specifically designed for managing and manipulating relational databases. It provides a set of commands for tasks such as creating and modifying database structures and inserting, updating, and deleting, querying data. MySQL is a relational database management system (RDBMS) that uses SQL as its query language. MySQL Server provides a platform for creating and managing relational databases. DataGrip is not a database system but rather an integrated development environment (IDE) for SQL that supports database systems such as MySQL.

# W4

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



This notation is used in the design of relational databases to illustrate how entities relate to each other. Here are the meanings of each ending:

1. The instance of the entity on the left is associated with zero or one instances of the entity on the right.
2. The instance of the entity on the left is associated with exactly one instance of the entity on the right.
3. The instance of the entity on the left is associated with zero, one, or many instances of the entity on the right.
4. The instance of the entity on the left is associated with one or many instances of the entity on the right.

# W5

What is a primary key and why is it important?

Primary key is a candidate key that is chosen by the database designer as the principal means of identifying tuples within a relation, where candidate keys are a minimal set of attributes that allow us to identify uniquely a tuple in the relation. That is, any two individual tuples in the relation are prohibited from having the same value on their primary key at the same time, which is called the primary key constraint. For example, a primary key for a "course" table may be "courseID", which uniquely identifies each course.

# W6

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

That means that every relational algebra operators applied on some relations results in a new relation. In other words, it ensures that we can combine any number of operations and will always end up with a valid relation. An example would be as follows:

1. Say "course" is a relation with an attribute "dept_name". Then "σ course.dept_name = 'Comp. Sci.' course" will also be a relation as a result of select operation.
2. Say "prereq" is another relation. Then "course × prereq" will be another relation as a result of cross join operation.
3. σ course.dept_name = 'Comp. Sci.' (course × prereq) is yet another relation as a combination of two operations.

# 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?

For each attribute of a relation, there is a set of permitted values, called the domain of that attribute, which is the set of all possible couples of years and one of three semesters for the year/semester attribute of the section relation in this example. A domain is atomic if elements of the domain are considered to be indivisible units, which is the case for the domain of year/semester because it always contains exactly a single entity that represents both the year and the semester of a section together instead of separating the year and the semester.

Type refers to the data type of an attribute, which dictates the storage method and format of data it can store, such as integer or character. In the example, CHAR(6) is the data type, indicating that the attribute stores characters and has a fixed length of 6 characters, which is different than the domain for the year/semester attribute that instead defines the logical rules and constraints (e.g., first four must be a valid year, and the last character must be 1, 2, or 3) that dictate what constitutes a valid value within that type, such as "2023_3".

# W8

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

The database schema is the logical design and structure of the database, whereas the database instance is a snapshot of the data in the database at a given instant in time. One can think of a database schema as the static blueprint that formally defines how data is organized in a database, specifying tables, data types, constraints, etc. A database instance, on the other hand, is a state of the database at some point in time with the actual data stored in it, which dynamically changes over time through data insertions, deletions and modifications.

## W9

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

Data definition language (DDL) deals with the schema or structure of the database, whereas data manipulation language (DML) is concerned with the manipulation of the data itself within the database. DDL allows databases to be created, altered, and deleted and provides facilities to specify certain consistency constraints but do not deal with the data within the tables. DML, on the other hand, allows for the insertion, updating, deleting, querying of data within tables in a database. In practice, DDL and DML are not two separate languages, but they simply form parts of a single database language, such as the SQL language.

## W10

What is physical data independence?

There are three levels of data abstraction in a database: physical, logical, and view levels. Physical Data Independence refers to the capacity of databases to make changes in the structure of the physical level, the lowest level of the Database Management System, without affecting the higher-level schemas, logcal and view levels. Although implementation of the simple structures at the logical level may involve complex physical-level structures, the user of the logical level does not need to be aware of this complexity as physical data independence makes sure that the physical storage structures and access methods or any modifications to them do not affect the conceptual view of the data or the application logic that accesses it.

---

# 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.

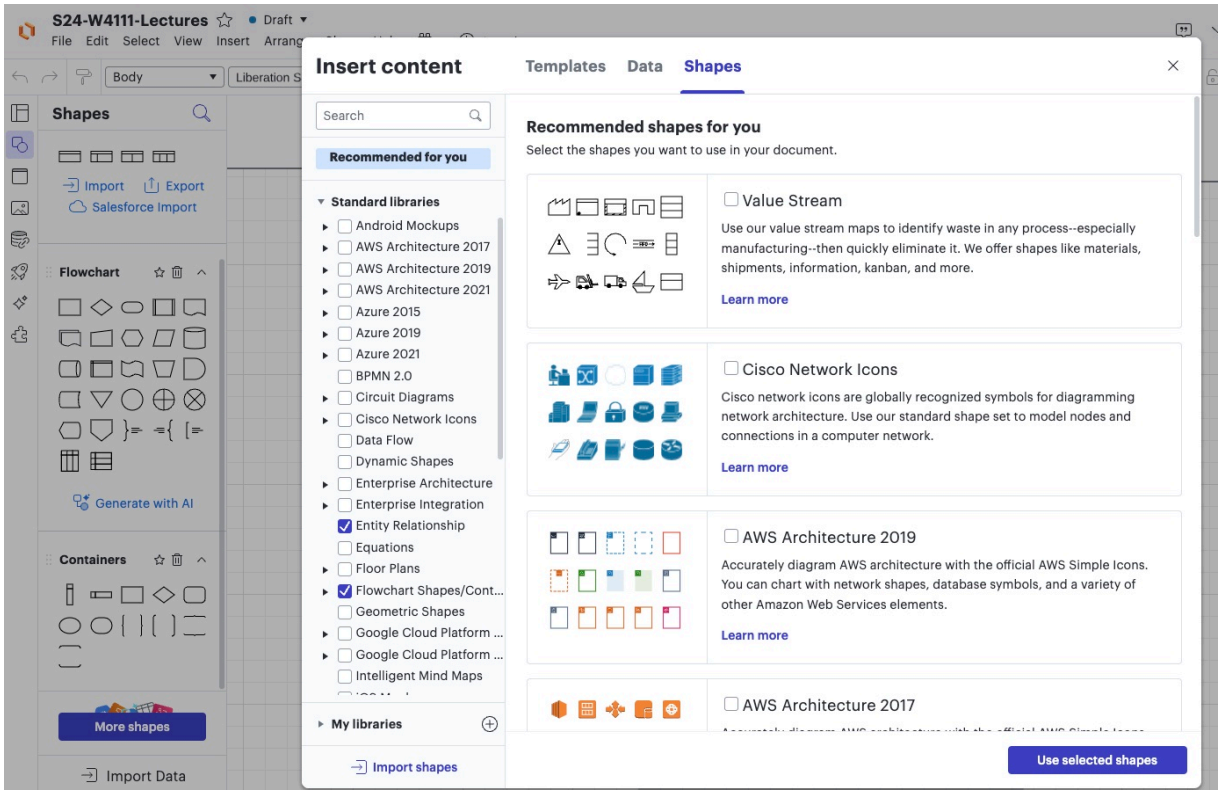
The users and business stakeholder often can understand and interact using a *conceptual model* but details like keys, foreign keys, ... are outside their scope.

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. (https://www.lucidchart.com/pages/landing)](https://www.lucidchart.com/pages/landing) 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 (https://lucid.app/lucidchart/828777b1-7b2d-4828-bedb-37b6d456c33e/edit?invitationId=inv_a142899a-7e60-44e9-b18e-335d7c9767fc) 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

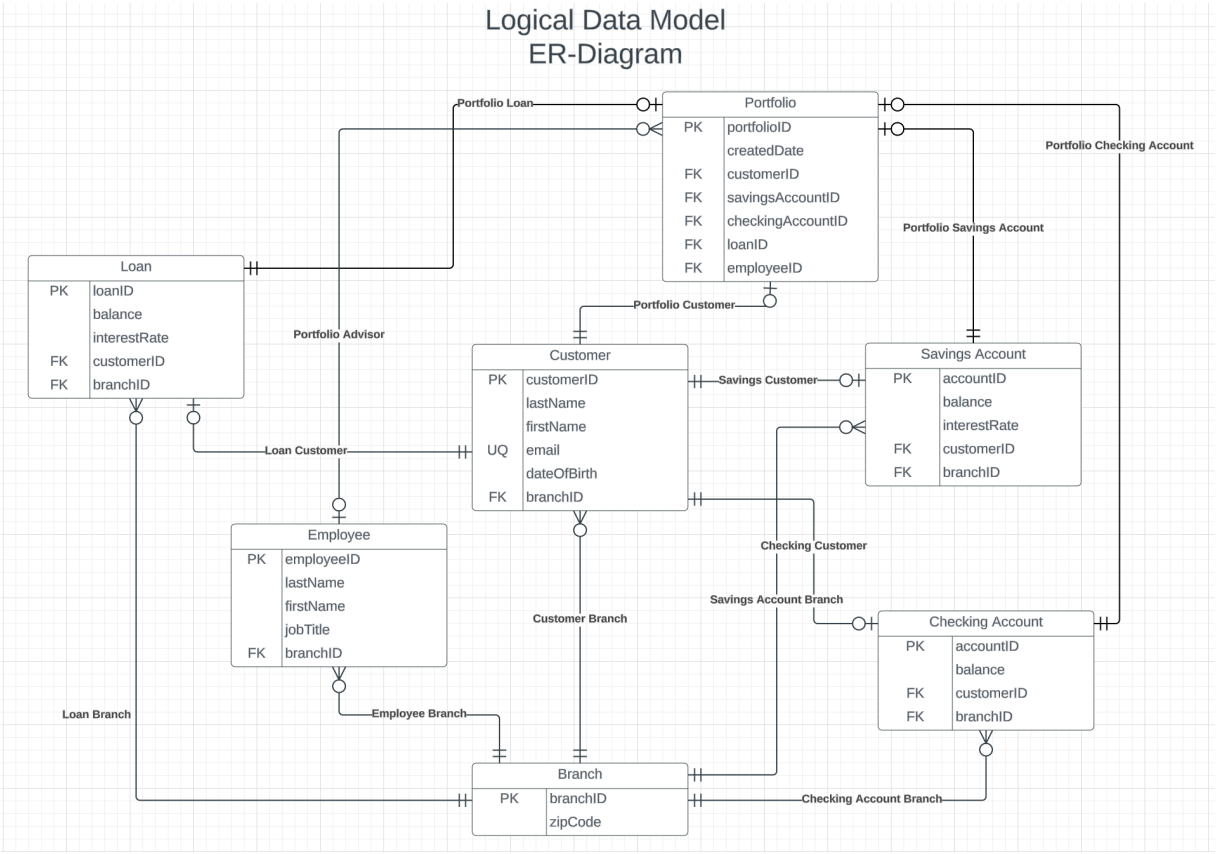
1. Place your Logical Model diagram below.
2. You *may* have to add attributes to entities to implement the model.
3. You *may* make reasonable assumptions. Please document your assumptions below. You may add comments/notes to your diagram for clarity.

Assumptions:

1. Each customer must have unique email.

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.

classroom(building, room_number, capacity)

# Answer Format

For the answers to the relational algebra questions, you will use the RelaX calculator (https://dbis-uibk.github.io/relax/calc/gist/4f7866c17624ca9dfa85ed2482078be8/relax-silberschatz-english.txt/0) 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

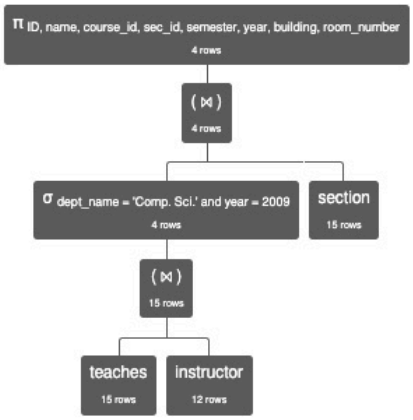
**Note:**

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.

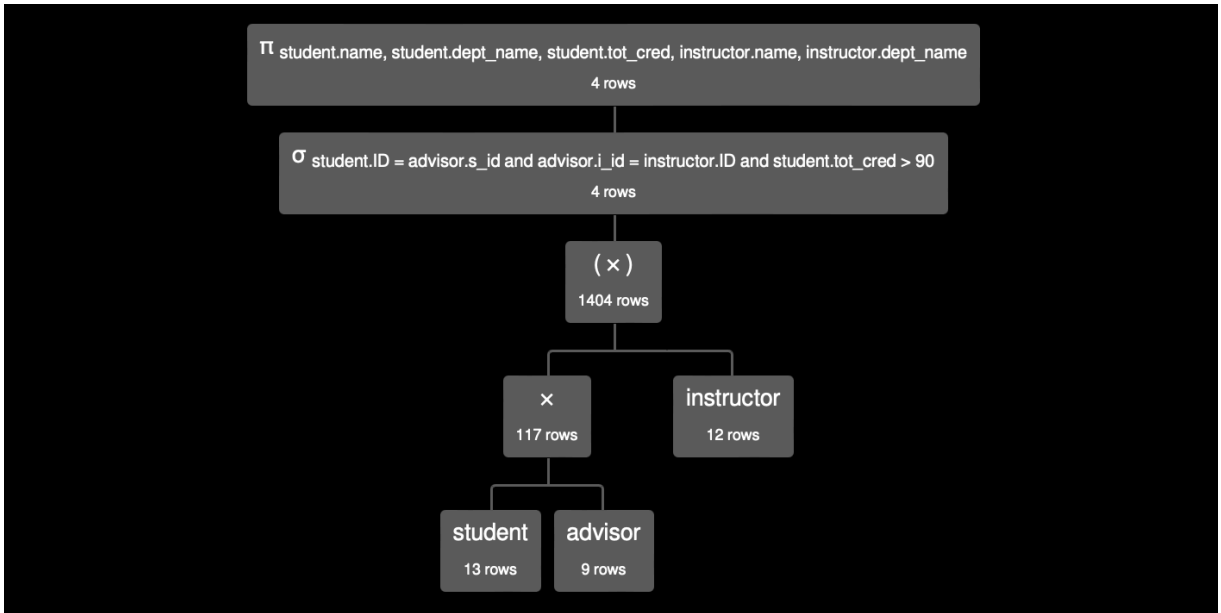
**Note:**

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, instructo
r.name, instructor.dept_name
    (σ student.ID = advisor.s_id ∧ advisor.i_id = instructor.I
D ∧ student.tot_cred > 90
        (student × advisor × instructor)
        )
```

Execution:




π student.name, student.dept_name, student.tot_cred, instructor.name, instructor.dept_name ( σ student.ID = advisor.s_id and advisor.i_id = instructor.ID and student.tot_cred > 90 ( ( student × advisor ) × instructor ) )

Execution time: 4 ms

| student.name | student.dept_name | student.tot_cred | instructor.name | instructor.dept_name |
|---|---|---|---|---|
| 'Zhang' | 'Comp. Sci.' | 102 | 'Katz' | 'Comp. Sci.' |
| 'Chavez' | 'Finance' | 110 | 'Singh' | 'Finance' |
| 'Bourikas' | 'Elec. Eng.' | 98 | 'Kim' | 'Elec. Eng.' |
| 'Tanaka' | 'Biology' | 120 | 'Crick' | 'Biology' |

**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:

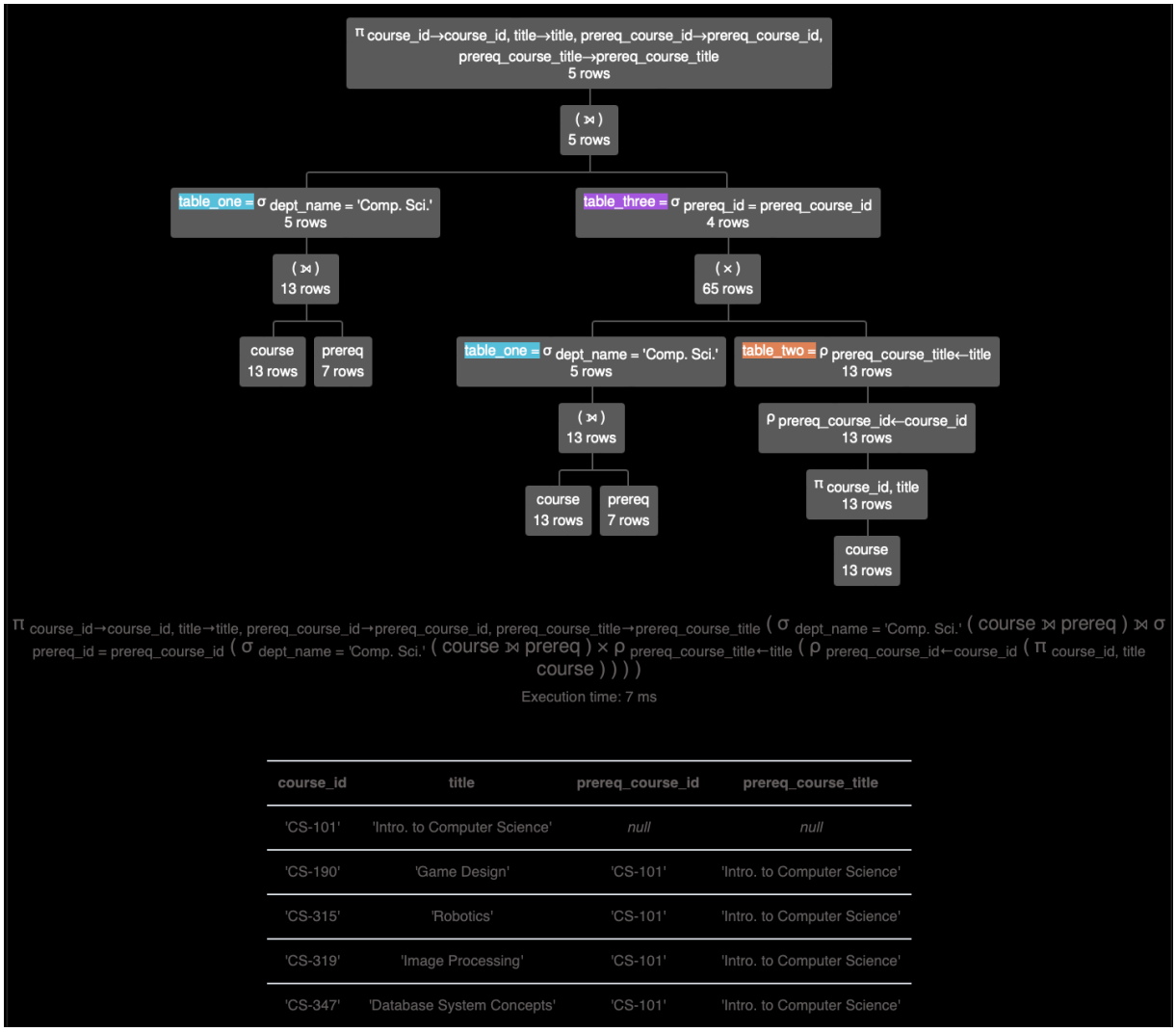
```
table_one = σ dept_name = 'Comp. Sci.' (course ⋈ prereq)

table_two = ρ prereq_course_title ← title (ρ prereq_course_id
← course_id (π course_id, title course))

table_three = σ prereq_id = prereq_course_id (table_one × tabl
e_two)

π course_id←course_id, title←title, prereq_course_id←prereq_co
urse_id, prereq_course_title←prereq_course_title (table_one ⋈
table_three)
```

Execution:




| course_id | title | prereq_course_id | prereq_course_title |
|---|---|---|---|
| 'CS-101' | 'Intro. to Computer Science' | *null* | *null* |
| 'CS-190' | 'Game Design' | 'CS-101' | 'Intro. to Computer Science' |
| 'CS-315' | 'Robotics' | 'CS-101' | 'Intro. to Computer Science' |
| 'CS-319' | 'Image Processing' | 'CS-101' | 'Intro. to Computer Science' |
| 'CS-347' | 'Database System Concepts' | 'CS-101' | 'Intro. to Computer Science' |

**R2 Execution Result**

# SQL

## New Database

MySQL Tutorial (https://www.mysqltutorial.org/) 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 (https://www.mysqltutorial.org/getting-started-with-mysql/mysql-sample-database/). 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 [15]:
```
%sql show tables;
```

```
 * mysql+pymysql://root:***@localhost
(pymysql.err.OperationalError) (1046, 'No database selected')
[SQL: show tables;]
(Background on this error at: https://sqlalche.me/e/20/e3q8) (https://sqlalche.me/e/20/e3q8))
```

In [16]:
```
%sql USE classicmodels;
```

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

Out[16]:
```
[]
```

# 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 [17]:
```
%%sql

SELECT
    CONCAT(customers.contactFirstName, ' ', customers.contactLastName)
    customers.phone AS customerPhone,
    CONCAT(employees.firstName, ' ', employees.lastName) AS salesRepNa
FROM
    customers
INNER JOIN
    employees ON customers.salesRepEmployeeNumber = employees.employee
WHERE
    customers.country = 'France'
ORDER BY
    customerContactName;
```

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

Out[17]:

| 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 (https://www.tutorialspoint.com/mysql/mysql_with.htm) to be useful for cleaner code.

In [18]:
```
%%sql

WITH OrderTotalSpending AS (
    SELECT
        o.customerNumber,
        SUM(od.quantityOrdered * od.priceEach) AS orderTotal
    FROM
        orders o
    INNER JOIN
        orderdetails od ON o.orderNumber = od.orderNumber
    WHERE
        o.status = 'Shipped'
    GROUP BY
        o.orderNumber
),
CustomerTotalSpending AS (
```

```
    SELECT
        c.salesRepEmployeeNumber,
        SUM(oa.orderTotal) AS customerTotal
    FROM
        OrderTotalSpending oa
    INNER JOIN
        customers c ON oa.customerNumber = c.customerNumber
    GROUP BY
        c.salesRepEmployeeNumber
)
SELECT
    CONCAT(e.firstName, ' ', e.lastName) AS salesRepName,
    SUM(ct.customerTotal) AS moneyGenerated # this SUM is not really n
FROM
    CustomerTotalSpending ct
INNER JOIN
    employees e ON ct.salesRepEmployeeNumber = e.employeeNumber
GROUP BY
    e.employeeNumber
ORDER BY
    moneyGenerated DESC;
```

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

Out[18]:

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