COMS W4111: Introduction to Databases
Spring 2023, Sections 002

Take Home Midterm

Overview

Instructions

Due Date: Sunday, 2023-MAR-12 at 11:59pm

You may not use late days.

You have one week to complete the take home portion of the midterm. All of the work must be your own, you may not work in groups or teams. You may use outside sources so long as you cite them and provide links.

Points will be taken off for any answers that are extremely verbose. Try to stay between 2-3 sentences for definitions and 5 sentences for longer questions.

You may post **privately** on Ed or attend OH for clairification questions. TAs will not be providing hints.

There is a [post on Ed (https://edstem.org/us/courses/32981/discussion/2716284)](https://edstem.org/us/courses/32981/discussion/2716284) that:

- Provides submission instructions.
- Clarifications and corrections on questions.

Students are responsible for reading and monitoring the post.

Environment Setup

- **Note:** You will need to change the MySQL userID and password in some of the cells below to match your configuration.

Environments

- Different applications and different personas use different ways to interaction with databases

- We use three different connection/interaction models to give students experience with the various options.

ipython-SQL

In [1]:

```
%load_ext sql
```

In [2]:

```
#
# Set the userid and password for connecting to your instance of SQL.
#
mysql_user = "root"
mysql_password = "dbuserdbuser"

mysql_url = f"mysql+pymysql://{mysql_user}:{mysql_password}@localhost"

print("Your connection URL is", mysql_url)
```

```
Your connection URL is mysql+pymysql://root:dbuserdbuser@localhost
```

In [3]:

```
#
# Connect. See the ipython-sql documentation for the $variable syntax.
#
%sql $mysql_url
```

SQL Alchemy and Pandas

In [4]:

```
#
# Yes, I know the cool kids import as pd. I am not cool.
#
import pandas
```

In [5]:

```
#
# Pandas SQL operations require a SQL Alchemy engine.
#
from sqlalchemy import create_engine
```

In [6]:

```
sql_engine = create_engine(mysql_url)
```

pymysql

In [7]:

```
import pymysql
```

In [8]:

```
pymysql_con = pymysql.connect(
    user= mysql_user,
    password= mysql_password,
    host= "localhost",
    port= 3306,
    autocommit= True,
    cursorclass= pymysql.cursors.DictCursor)
```

Data Loading

Classic Models

- We will use the [Classic Models (https://www.mysqltutorial.org/mysql-sample-database.aspx)](https://www.mysqltutorial.org/mysql-sample-database.aspx) sample database for many of the questions on this exam.
- The directory containing this notebook contains a file `classic-models-sample.sql` .
- Load the data:
  - Open the file in DataGrip using `File -> Open` dialog.
  - Select all of the text/SQL in the file.
  - Click the green arrowhead to run the files contents.
- Running the following queries will test if the load worked.

In [9]:

```
%sql use classicmodels;
```

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

Out[9]:

```
[]
```

In [10]:

```
%sql show tables;
```

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

Out[10]:

Tables_in_classicmodels
customers
employees
offices
orderdetails
orders
payments
productlines
products

In [11]:
```
%sql select count(*) as count from orders join orderdetails using(orderNumb
```

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

Out[11]:

count
2996

Lahman's Baseball Database

- You previously loaded information from [Lahman's Baseball Database. (https://www.seanlahman.com/)](https://www.seanlahman.com/)
- If you have not done so, the following code will load the data into a new schema `lahmansdb_midterm`.

In [12]:
```
%sql create schema lahmansdb_midterm
```

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

Out[12]:
```
[]
```

In [13]:
```
people_df = pandas.read_csv("./People.csv")
people_df.to_sql("people", schema="lahmansdb_midterm", con=sql_engine,index
```

Out[13]:
```
20370
```

In [14]:
```
batting_df = pandas.read_csv("./Batting.csv")
batting_df.to_sql("batting", schema="lahmansdb_midterm", con=sql_engine,ind
```

Out[14]:
```
110495
```

In [15]:
```
pitching_df = pandas.read_csv("./Pitching.csv")
pitching_df.to_sql("pitching", schema="lahmansdb_midterm", con=sql_engine,i
```

Out[15]:
```
49430
```

- This will test the data loading.

In [16]:
```
%sql select count(*) as people_count from lahmansdb_midterm.people;
```

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

Out[16]:

people_count
20370

```
In [17]: %sql select count(*) as batting_count from lahmansdb_midterm.batting;
```

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

Out[17]:

batting_count
110495

```
In [18]: %sql select count(*) as pitching_count from lahmansdb_midterm.pitching;
```

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

Out[18]:

pitching_count
49430

Written Questions

W1

Question

- Define the concept of *immutable* column and key.
- Why do some sources recommend that a primary key should be immutable?
- How would to implement immutability for a primary key in a table?

Answer

- An `immutable column` is an column that cannot be modified after being created, meaning that its data type and constrains cannot be changed once it is created.
- An `immutable key` is a key or unique constraint given to a set of columns, once created, the values in those columns cannot be changed.
- Primary key is used to uniquely identify a tuple, it is recommended to be immutable because if one changes the primary key, one needs to either delete the row or reinsert a new primary key value. This may lead to problems in a large schema if the orginal primary key value was referenced in other tables. By making the primary key immutable, these problems can be avoided.
- A `update trigger` could be created so that an error message will show when the user tries to change the value of a primary key.

W2

Question

Views are a powerful concept in relational database management systems. List and briefly explain 3 benefits of/reasons for creating a view.

Answer

1. Security: a table may contain sensitive or private information that may not be displayed publically (such as salaries). In this case, a view can be created by selecting only the information that is able to be publically displayed. This can also be used together with `grants` to restrict user access to tables containing sensitive information.
2. Simplification for "naive" users: users may only be interested in selected attributes from different tables, it can be hard for naive users to write complex queries to view desired information. In this case, a view can be created beforehand to combine attributes of high interest and the naive users can directly call the view and perform simple queries.
3. Protect applications from schema changes: application can be written against the view while actual datas in the table are evolving over time. The view can be updated later with evolved data and the application will still be intact.

W3

Question

Briefly explain the concepts of *procedural* language and *declarative* language. SQL is primarily a declarative language. SQL added procedure language capabilities in functions, procedures and triggers. What is a reason for this addition?

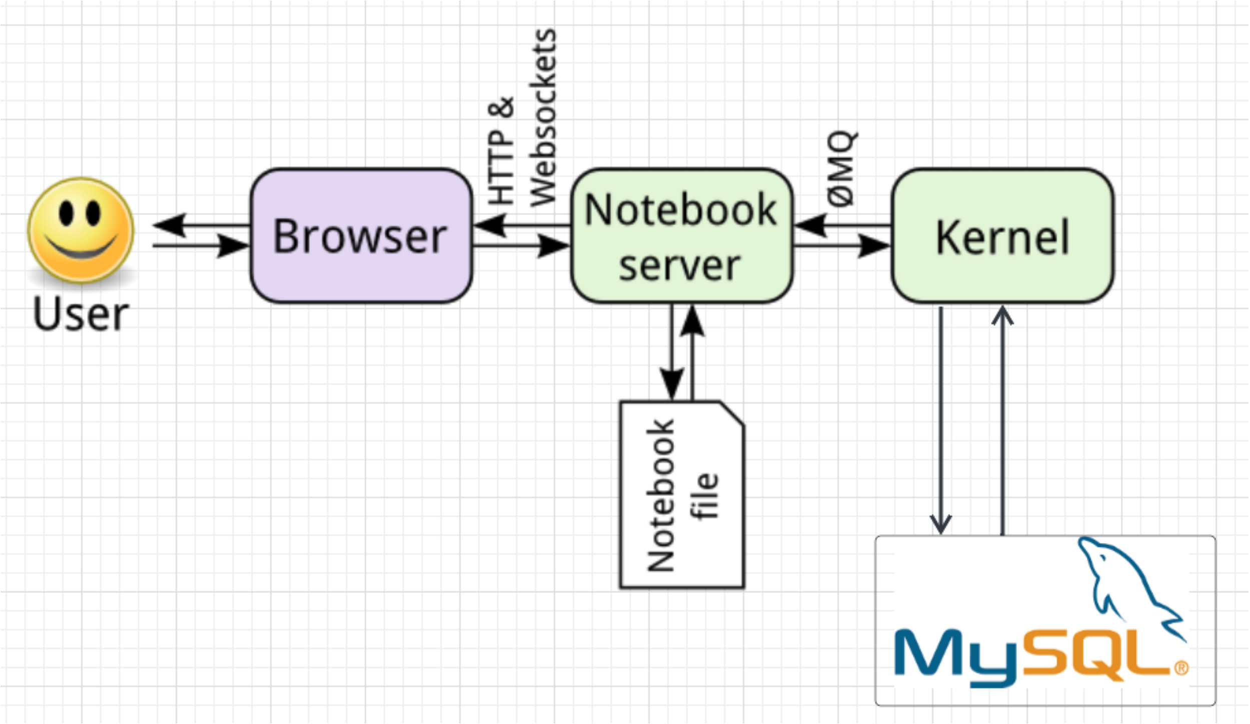
Answer

- `Procedural language` requires the user to specify what data are needed and steps of how to get the data. This is usually harder to learn and use than declarative language
- `Declarative language` only requires the user to specify what data are needed and the program is able to get the data without specific instructions from the user.
- Since SQL is a declarative language, it is not Turing complete and has limited computational capabilities. SQL added procedure language capabilities in functions, procedures and triggers to increase flexibility and power of the language, allowing for more complex data processing operations and logics such as immutability of attributes or data aggregations.

W4

Question

The following diagram is a simple representation of the architecture of a Jupyter notebook using MySQL. Is this a two-tier architecture or a three-tier architecture? Explain your answer briefly.



Answer

- Jupyter Notebooks are a `3-tier application` because the user is accessing the Jupyter Notebook in a browser (application client), which is connected to a notebook server (application server), which in turn interacts with the database through pyMysql if a code cell is added to the notebook.

W5

Question

- Consider a US Social Security Number. An example is "012-34-6789".
- The data *type* is character string.
- The relational model requires that columns (attributes) are from a *domain.*
- Use the Social Security Number example to explain the difference between a type and a domain.

Answer

- A `type` is the kind of value (date, string, integer) that can be entered into each column. In this case, it is the data type string including but not limited to alphabeticall letters (a, b, c, d...) and numebrs (1, 2, 3...) and symbols (,!?- and so on).
- A `domain` is the set of allowed values for each attribute. In this case it is only numbers and the symbol (-) that divides the groups of numbers.

W6

Question

Briefly explain the differences between:

- Database stored procedure
- Database function
- Database trigger

Answer

- `Database stored procedure` is a set of SQL statements with typed input and output parameters, but function doesn't have output parameter (return type instead).
- `Database function` is pre-defined and needs to be called explicitly when needed. It is able to return values.
- `Database trigger` is executed automatically when an event, such as delete/insert/update, happens to a database object. It doesn't need to be called explicitly.

W7

Question

Briefly explain:

- Natural join
- Equi-join
- Theta join
- Self-join

Answer

- `Natural join` combines rows in tables based on the column with same name and data type in both tables.
- `Equi-join` combines rows in tables based on specified equality condition between two columns. It requires the "ON" keyword to specify which columns need to be matched.

- `Theta join` combines rows in tables based on any comparison operator (including >, <, >=, and <=). The condition also needs to be specified.
- `Self join` combines a table with itself. It requires an alias to be assigned to the tables in order to differentiate the two tables being joined.

W8

Question

Briefly explain the difference between a *unique (key) constraint* and a *primary key constraint?*

Answer

Both `unique constraint` and `primary key constraint` are able to ensure the values in a column or a set of columns to be unqiue across all rows. However, a unique constraint can have a *NULL* value while a primary key constraint doesn't allow a *NULL* values. This is because the primary key constraint is used to uniquely identify tuples.

W9

Question

Give two reasons for using an *associative entity* to implement a relationship instead of using a foreign key.

Answer

- Associative entity can store more information about the relationship between two entities. For example, the relationship between a user and a comment can include information such as comment time and the user's ip address when making the comment. On the other hand, foreign key only indicates that the two tables are related by certain attribute.
- Associative entity also simplifies the query process as it avoids complex join operations that may be required when a foreign key is used to represent the relationship.

W10

Question

Briefly explain the concepts of:

- Conceptual model
- Logical model
- Physical model

For data modeling.

Answer

- `Conceptual model` only include the name of entities and relationship between the entities. It describes the overall structure of the model without specifying how it will be implemented.
- `Logical model` include entity names, entity relationships, attributes, primary keys, and foreign keys. It is more detailed than a conceptual model.
- `Physical model` is the most detailed model as it includes constraints (primary and foreign keys), table names, column names and data types. It is used to describe implementation details of how data will be physically stored in the database.

W11

Question

Briefly explain the concepts of:

- Data manipulation language
- Data definition language

Given an example statement in SQL for DML and for DDL.

Answer

- Data manipulation language (DML) is used to query information from database and to insert, delete, update data stored in tables. An example statement would be `select * from students` , which retrieve data from a table called `students` .
- Data definition language (DDL) is used to define and modify the structure database objects and to create, modify, drop tables, views, or other database objects. An example statement to create a table called `students` would be:

  ```
  create table students (
    ID int primary key,
    last_name varchar(32) not null,
    first_name varchar(32) not null;
  );
  ```

W12

Question

Codd's 4th rule is:

Rule 4 - Dynamic online catalog based on the relational model:

The data base description is represented at the logical level in the same way as ordinary data, so that authorized users can apply the same relational language to its interrogation as they apply to the regular data.

Explain what this means, and use SQL to provide examples.

Answer

This means that in relational database, the information about the schema itself is data (called metadata) and users can query the information about the data.

```
In [34]: %sql use information_schema;
```

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

```
Out[34]: []
```

```
In [35]: %sql select * from tables where table_schema = 'db_book';
         #gives all the information about the tables in the db_book schema
```

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

Out[35]:

TABLE_CATALOG	TABLE_SCHEMA	TABLE_NAME	TABLE_TYPE	ENGINE	VERSION	ROW_FORMAT
def	db_book	advisor	BASE TABLE	InnoDB	10	Dynamic
def	db_book	classroom	BASE TABLE	InnoDB	10	Dynamic
def	db_book	course	BASE TABLE	InnoDB	10	Dynamic
def	db_book	department	BASE TABLE	InnoDB	10	Dynamic
def	db_book	instructor	BASE TABLE	InnoDB	10	Dynamic
def	db_book	prereq	BASE TABLE	InnoDB	10	Dynamic
def	db_book	section	BASE TABLE	InnoDB	10	Dynamic
def	db_book	student	BASE TABLE	InnoDB	10	Dynamic
def	db_book	takes	BASE TABLE	InnoDB	10	Dynamic
def	db_book	teaches	BASE TABLE	InnoDB	10	Dynamic
def	db_book	time_slot	BASE TABLE	InnoDB	10	Dynamic

W13

Question

The formal definition of a theta join is

$$r \bowtie_\theta s = \sigma_\theta(r \times s)$$

Briefly explain the definition and give an example.

Why is the fact that the relational algebra is closed is important to this definition?

Answer

- r and s are relations being joined. `r × s` is the cartesian product of the two relations, producing every possible combination of tuples from R and S. `σ` is the selection operator and `σ`θ selects the only tuples from the cartesian product that satisfy the specified condition θ.
- An example would be

  ```
  σ instructor.id = teaches.id (instructor × teaches)
  ```

  which is equivalent to

  ```
  instructor ⋈ instructor.id = teaches.id teaches
  ```

- The fact that the relational algebra is closed is important to this definition because the table produced by the cartesian product can still be used as an inout to the selection operator.

W14

Question

Consider two different statements in the relational algebra or SQL.

Despite being different statements, the statements may be <u>equivalent.</u> Briefly explain what this means.

Answer

An example would be:

```
select * from instructor where salary > 70000;
```

is equivalent to

```
select * from instructor where not (salary <= 70000);
```

This means that the two statements perform the same logical operation and retrieve the same set of data from the instructor table.

W15

Question

Consider the following relation definitions.

$$Customers(ID, last\_name, first\_name)$$

$$Accounts(ID, balance, customer\_ID)$$

What is problem with using natural join on the two tables?

Answer

Natural join combines rows in tables based on the column with same name and data type in both tables. Since both `Customers` and `Accounts` table have attributes named ID, a natural join operation would combine the two tables using matched ID, but in reality, the two tables should be combined with the condition `Customer.ID = Accounts.customer_ID` . In this case, an equi-join should be used.

Entity Relationship Modeling

ER-1

Question

This question tests your ability to "bottom up" model or "reverse engineering" a SQL schema to produce an explanatory ER-diagram.

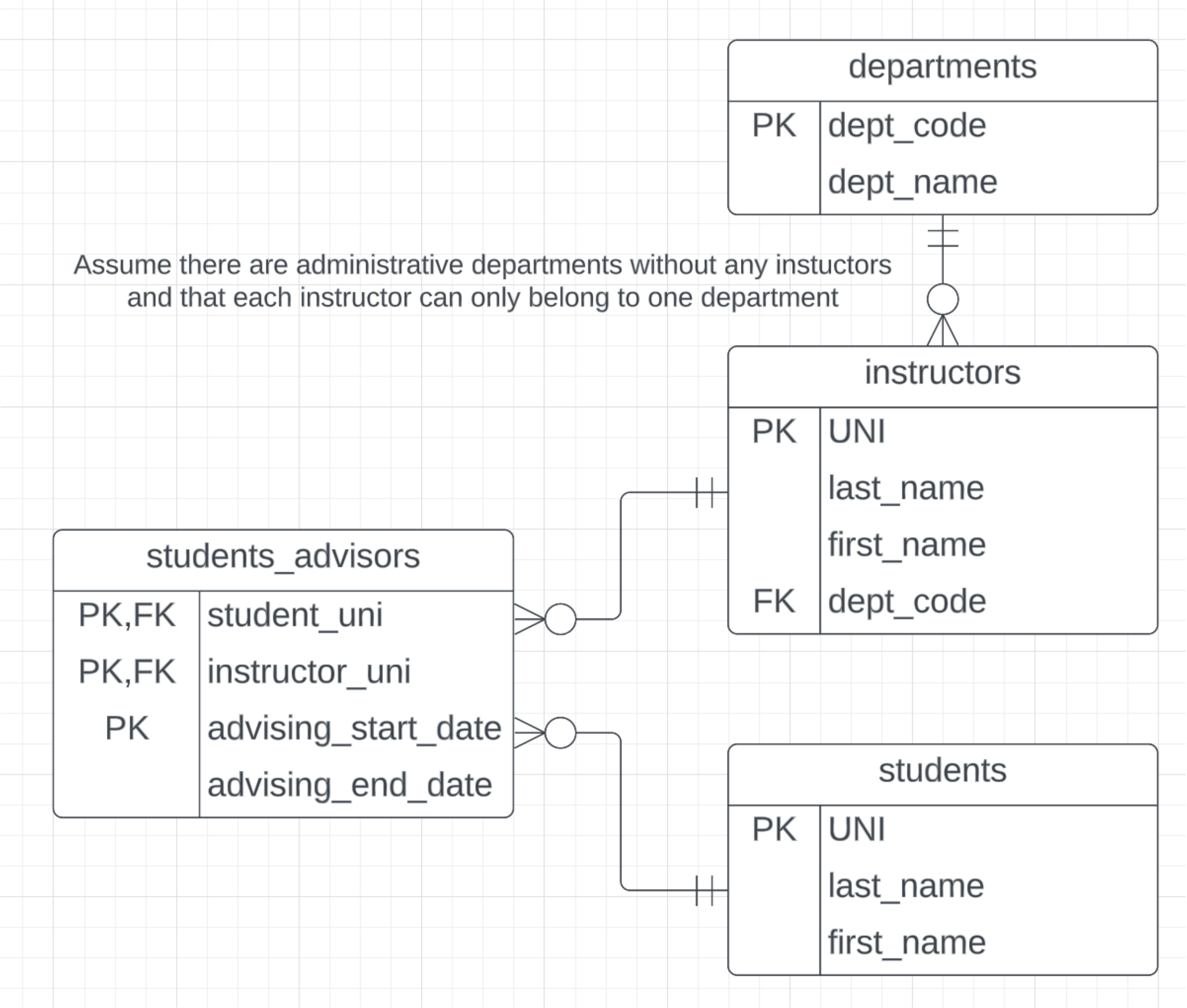
Use Lucidchart to draw a Crow's Foot notation diagram representing the following SQL.

You can use the simple table names, e.g. `students` instead of `s23_w4111_midterm.students` .

```
drop schema if exists s23_midterm;

create schema s23_midterm;

use s23_midterm;

drop table if exists departments;
create table if not exists departments
(
    dept_code varchar(4)   not null
        primary key,
    dept_name varchar(128) not null
);

drop table if exists instructors;
create table if not exists instructors
(
    UNI        varchar(12)  not null
        primary key,
    last_name  varchar(128) not null,
    first_name varchar(128) not null,
    dept_code  varchar(4)   null,
    constraint instructor_dept
        foreign key (dept_code) references departments (dept_code)
);

drop table if exists students;
create table if not exists students
(
    UNI        varchar(12)  not null
        primary key,
    last_name  varchar(128) null,
    first_name varchar(128) null
);

drop table if exists students_advisors;
create table if not exists students_advisors
(
    student_uni         varchar(12) not null,
    instructor_uni      varchar(12) not null,
    advising_start_date date        not null,
    advising_end_date   date        null,
    primary key (student_uni, instructor_uni, advising_start_date),
    constraint student_advisor_instructor
        foreign key (instructor_uni) references instructors (UNI),
    constraint student_advisors_student
        foreign key (student_uni) references students (UNI)
);
```

Answer

- Put your screen capture in the same directory as the midterm.
- Add `<img src='./myfile.name'>` in the Markdown cell, using the actual file name.



Assume there are administrative departments without any instuctors and that each instructor can only belong to one department

ER-2

Question

- This question tests your ability to convert a human language description of a data model into a Crow's Foot ER-Diagram.
- Consider the data model for Classic Models that you loaded.
- `orders` has a column `comments`.

```
In [36]: %sql select * from classicmodels.orders limit 10;
```

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

Out[36]:

orderNumber	orderDate	requiredDate	shippedDate	status	comments	customerNumber
10100	2003-01-06	2003-01-13	2003-01-10	Shipped	None	363
10101	2003-01-09	2003-01-18	2003-01-11	Shipped	Check on availability.	128
10102	2003-01-10	2003-01-18	2003-01-14	Shipped	None	181
10103	2003-01-29	2003-02-07	2003-02-02	Shipped	None	121
10104	2003-01-31	2003-02-09	2003-02-01	Shipped	None	141
10105	2003-02-11	2003-02-21	2003-02-12	Shipped	None	145
10106	2003-02-17	2003-02-24	2003-02-21	Shipped	None	278
10107	2003-02-24	2003-03-03	2003-02-26	Shipped	Difficult to negotiate with customer. We need more marketing materials	131
10108	2003-03-03	2003-03-12	2003-03-08	Shipped	None	385
10109	2003-03-10	2003-03-19	2003-03-11	Shipped	Customer requested that FedEx Ground is used for this shipping	486

- There are several issues with this design:
  - If there are multiple comments or responses to comments, the `comments` field becomes multi-valued.
  - The approach does not have information on when the comment was made, who made the comment and whether it is a response or elaboration.

- You will solve this problem in a simplified version of classic models. In the simplified model, there are three entity types:
  1. `person` has the following attributes:
     - `ID`
     - `last_name`
     - `first_name`
  2. `orders` has the following attributes:
     - `order_id`
     - `order_status`
     - `order_date`
  3. `comments` has the following attributes:
     - `comment_id` is a unique ID for all comments.
     - `parent_comment_id` is the `comment_id` of a comment for which this comment is a response or elaboration.
     - `comment_timestamp` , when the comment occured.
     - `commenter_id` is the ID of the person making the comment.
     - `order_id` is the ID of the order for to which this comment applies.

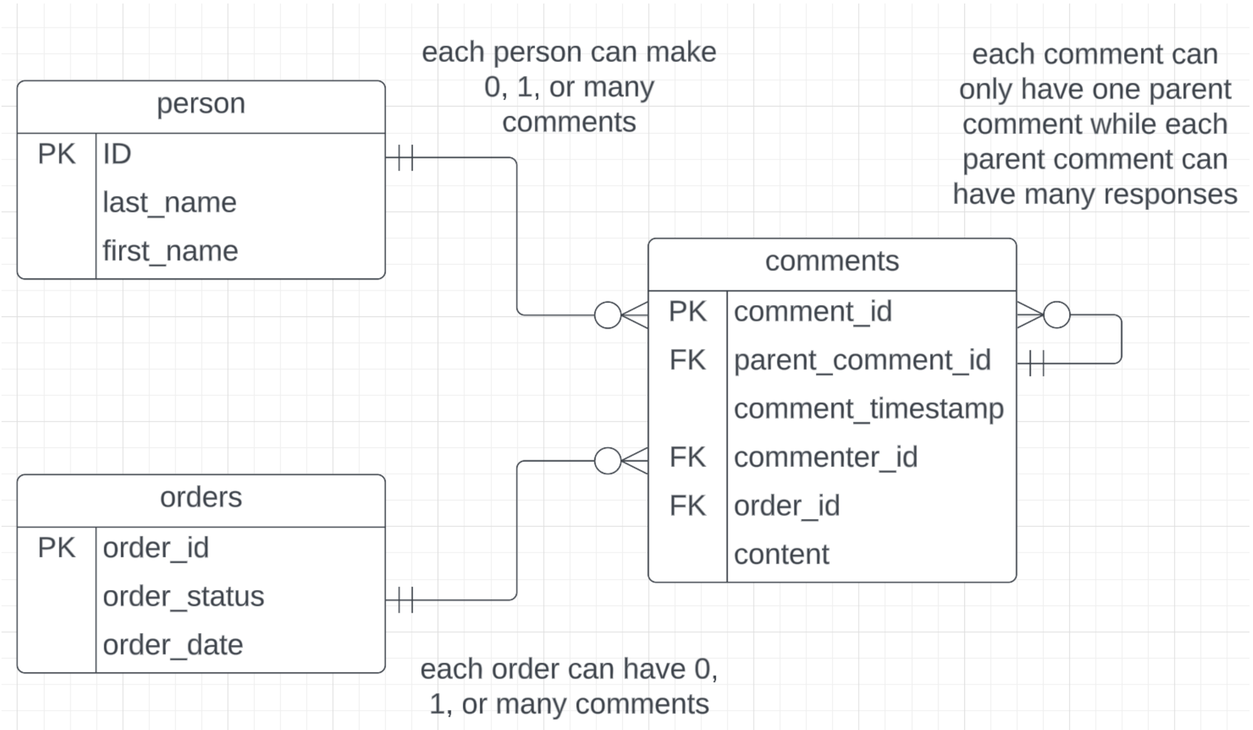
- Use Lucidchart to draw a *logical model* for the described datamodel.

- You may add notes to the diagram to document reasonable assumptions or desgn decisions.

Answer

- Put your screen capture in the same directory as the midterm.

- Add `<img src='./myfile.name'>` in the Markdown cell, using the actual file name.



each person can make 0, 1, or many comments

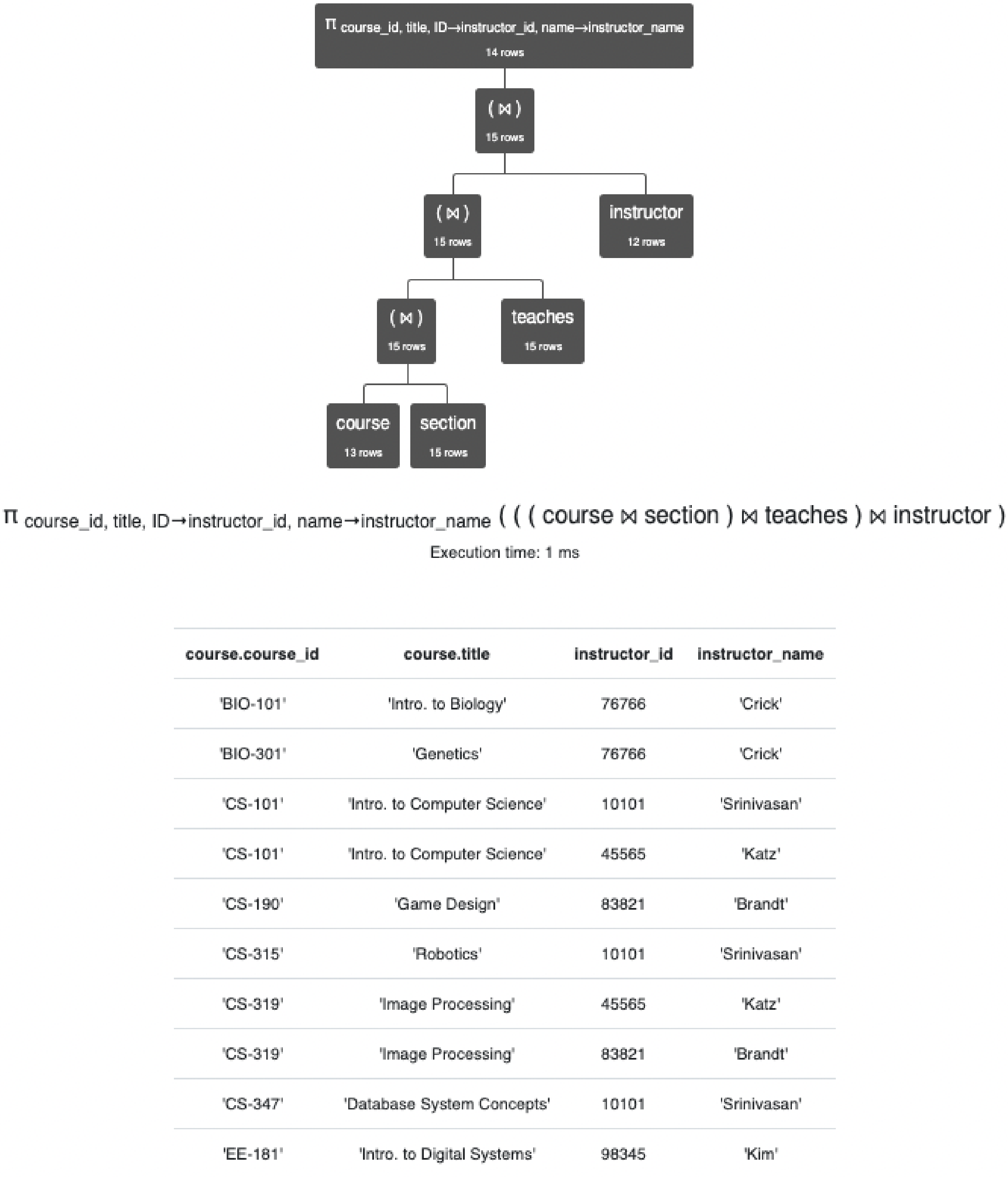
each comment can only have one parent comment while each parent comment can have many responses

each order can have 0, 1, or many comments

Relational Algebra

- Use the RelaX Calculator and the Silberschatz calculator (https://dbis-uibk.github.io/relax/calc/gist/4f7866c17624ca9dfa85ed2482078be8/relax-silberschatz-english.txt/0) with the Silberschatz database for these questions.
- Your answers will have two Markdown cells. The first is the relational statement you used to solve the problem. The second is a screen capture of the query execution and first page of result rows. And example is:

```
π course_id, title,
    instructor_id←ID,
    instructor_name←name
(
    (
        (course ⨝ section)
        ⨝
        teaches
    )
    ⨝
    instructor
)
```



$\pi_{\text{course\_id, title, ID}\rightarrow\text{instructor\_id, name}\rightarrow\text{instructor\_name}}$ (((course ⋈ section) ⋈ teaches) ⋈ instructor)

Execution time: 1 ms

course.course_id	course.title	instructor_id	instructor_name
'BIO-101'	'Intro. to Biology'	76766	'Crick'
'BIO-301'	'Genetics'	76766	'Crick'
'CS-101'	'Intro. to Computer Science'	10101	'Srinivasan'
'CS-101'	'Intro. to Computer Science'	45565	'Katz'
'CS-190'	'Game Design'	83821	'Brandt'
'CS-315'	'Robotics'	10101	'Srinivasan'
'CS-319'	'Image Processing'	45565	'Katz'
'CS-319'	'Image Processing'	83821	'Brandt'
'CS-347'	'Database System Concepts'	10101	'Srinivasan'
'EE-181'	'Intro. to Digital Systems'	98345	'Kim'

R1

Question

- Consider the relation produced by:

  ```
  π course_id, sec_id, building, room_number, time_slot_id (section)
  ```

- This contains sections, their time assignments and room assignments independent of the year and semester.

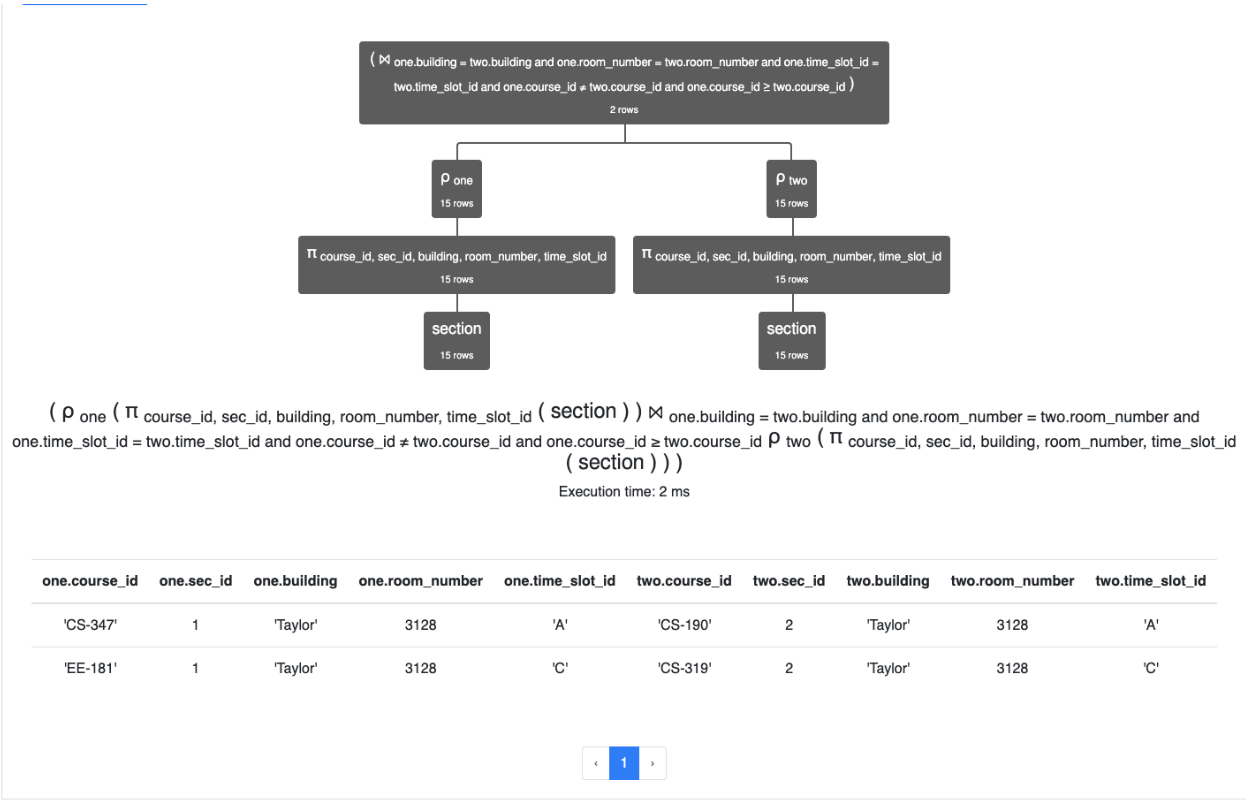
- Two sections in this derived table conflict if they have the same `building, room_number, time_slot_id`.
- My answer to this question is ... ...

one.course_id	one.sec_id	one.building	one.room_number	one.time_slot_id	two.course_id	two.sec_id
CS-347	1	Taylor	3128	A	CS-190	2
EE-181	1	Taylor	3128	C	CS-319	2

- Your answer cannot include courses and sections that conflict with themselves, or have two rows that show the same conflict.

Answer

```
ϱ one(π course_id, sec_id, building, room_number, time_slot_id (section))
⋈ one.building = two.building
    ∧ one.room_number = two.room_number
    ∧ one.time_slot_id = two.time_slot_id
    ∧ one.course_id ≠ two.course_id
    ∧ one.course_id ≥ two.course_id
ϱ two(π course_id, sec_id, building, room_number, time_slot_id (section))
```


(⋈ one.building = two.building and one.room_number = two.room_number and one.time_slot_id = two.time_slot_id and one.course_id ≠ two.course_id and one.course_id ≥ two.course_id)
2 rows
ρ one
15 rows
ρ two
15 rows
π course_id, sec_id, building, room_number, time_slot_id
15 rows
π course_id, sec_id, building, room_number, time_slot_id
15 rows
section
15 rows
section
15 rows

(ρ one (π course_id, sec_id, building, room_number, time_slot_id (section)) ⋈ one.building = two.building and one.room_number = two.room_number and one.time_slot_id = two.time_slot_id and one.course_id ≠ two.course_id and one.course_id ≥ two.course_id ρ two (π course_id, sec_id, building, room_number, time_slot_id (section)))

Execution time: 2 ms

one.course_id	one.sec_id	one.building	one.room_number	one.time_slot_id	two.course_id	two.sec_id	two.building	two.room_number	two.time_slot_id
'CS-347'	1	'Taylor'	3128	'A'	'CS-190'	2	'Taylor'	3128	'A'
'EE-181'	1	'Taylor'	3128	'C'	'CS-319'	2	'Taylor'	3128	'C'

‹ 1 ›

R2

Question

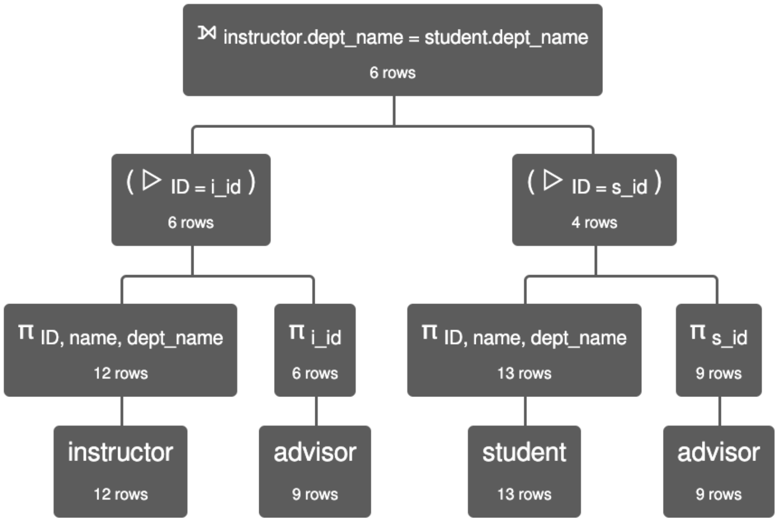
- You may use the following operators for this question: `π, σ, ϱ, ←`.
- Use the `instructor, student, advisor` tables for this question.
- There are some students that do not have advisors. That are some instructors that are not advisors.
- An `instructor` can be an advisor for a `student` if they are in the same department (`dept_name`).
- Produce a relation of the form

`(instructor_id, instructor_name, instructor_dept_name, student_id, student_name, student_dept_name)`

- That matches instructors that do not advise students and students that do not have advisors and are in the same department.

Answer

```
(π ID,name,dept_name(instructor)
▷ ID = i_id
π i_id(advisor))
⟕ instructor.dept_name = student.dept_name
(π ID,name,dept_name(student)
▷ ID = s_id
π s_id(advisor))
```



$$(\pi_{ID,\ name,\ dept\_name} (instructor) \triangleright_{ID = i\_id} \pi_{i\_id} (advisor)) ⟕_{instructor.dept\_name = student.dept\_name} (\pi_{ID,\ name,\ dept\_name} (student) \triangleright_{ID = s\_id} \pi_{s\_id} (advisor))$$

Execution time: 3 ms

instructor.ID	instructor.name	instructor.dept_name	student.ID	student.name	student.dept_name
12121	'Wu'	'Finance'	*null*	*null*	*null*
15151	'Mozart'	'Music'	55739	'Sanchez'	'Music'
32343	'El Said'	'History'	19991	'Brandt'	'History'
33456	'Gold'	'Physics'	70557	'Snow'	'Physics'
58583	'Califieri'	'History'	19991	'Brandt'	'History'
83821	'Brandt'	'Comp. Sci.'	54321	'Williams'	'Comp. Sci.'

SQL

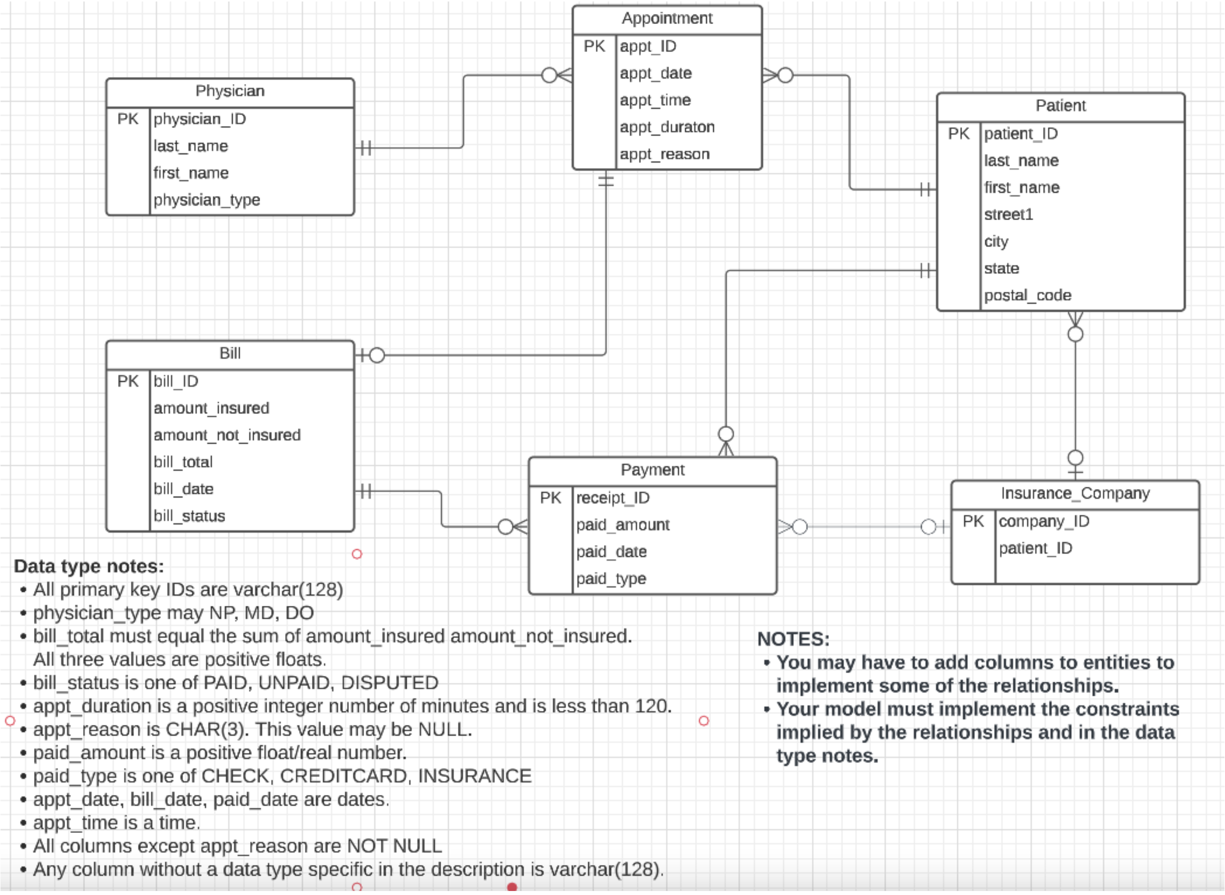
S1

Question

- You have a logical datamodel ER-diagram (see below).
- You need to use DDL to define a schema that realizes the model.
- Logical models are not specific enough for direct implementation. This means that:

- You will have to assign concrete types to columns, and choose things like `GENERATED, DEFAULT,` etc.
- You may have to decompose a table into two tables, or extract common attributes from multiple tables into a single, referenced table.
- Implementing the relationships may require adding columns and foreign keys, associative entities, etc.
- You may have to make other design and implementation choices. **This means that there is no single correct answer.**

- You should document any reasonable assumptions you make.



Data type notes:
- All primary key IDs are varchar(128)
- physician_type may NP, MD, DO
- bill_total must equal the sum of amount_insured amount_not_insured. All three values are positive floats.
- bill_status is one of PAID, UNPAID, DISPUTED
- appt_duration is a positive integer number of minutes and is less than 120.
- appt_reason is CHAR(3). This value may be NULL.
- paid_amount is a positive float/real number.
- paid_type is one of CHECK, CREDITCARD, INSURANCE
- appt_date, bill_date, paid_date are dates.
- appt_time is a time.
- All columns except appt_reason are NOT NULL
- Any column without a data type specific in the description is varchar(128).

NOTES:
- **You may have to add columns to entities to implement some of the relationships.**
- **Your model must implement the constraints implied by the relationships and in the data type notes.**

ER Diagram

Answer

Design Decisions, Notes, etc:

1. Foreign Keys physician_ID and patient_ID added to Appointment referencing to Physician and Patient respectively.
2. Foreign Key appt_ID Bill referencing to Appointment.
3. Foreign Keys bill_ID, patient_ID, and company_ID added to Payment referencing to Bill, Patient, and Insurance_Company respectively.

DDL

- Execute your DDL in the cell below. You may use DataGrip or other tools to help build the schema.
- You can copy and paste the `SQL CREATE TABLE` below, but you MUST execute the statements.

In [19]:

```
%sql drop schema if exists s23_midterm_medical


%sql create schema s23_midterm_medical
```

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

Out[19]:

```
[]
```

In [20]:

```
%%sql

use s23_midterm_medical;

create table if not exists Physician
(
    physician_ID   varchar(128)            not null,
    last_name      varchar(128)            not null,
    first_name     varchar(128)            not null,
    physician_type enum ('NP', 'MD', 'DO') not null,
    constraint Physician_pk
        primary key (physician_ID)
);

create table if not exists Patient
(
    patient_ID  varchar(128)  not null,
    last_name   varchar(128)  not null,
    first_name  varchar(128)  not null,
    street1     varchar(128)  not null,
    city        varchar(128)  not null,
    state       varchar(128)  not null,
    postal_code varchar(128)  not null,
    constraint Patient_pk
        primary key (patient_ID)
);

create table if not exists Appointment
(
    appt_ID       varchar(128) not null,
    appt_date     date         not null,
    appt_time     time         not null,
    appt_duration int          default 0
        check(0<=appt_duration<120),
    appt_reason   char(3)      null,
    physician_ID  varchar(128) not null,
    patient_ID    varchar(128) not null,
    constraint Appointment_pk
        primary key (appt_ID),
    constraint Appointment_Patient_patient_ID_fk
        foreign key (patient_ID) references Patient (patient_ID),
    constraint Appointment_Physician_physician_ID_fk
        foreign key (physician_ID) references Physician (physician_ID)
);

create table if not exists Bill
(
    bill_ID            varchar(128)                                        not null
        primary key,
    amount_insured     float                                               default
        check (amount_insured >= 0.0),
    amount_not_insured float                                               default
        check (amount_not_insured >= 0.0),
    bill_total         float as(amount_insured+amount_not_insured) not null
        check (bill_total >= 0.0),
    bill_date          date                                                not null
    bill_status        enum ('PAID', 'UNPAID', 'DISPUTED')                 not null
    appt_ID            varchar(128)                                not null
    constraint Bill_Appointment_appt_ID_fk
        foreign key (appt_ID) references Appointment (appt_ID)
);

create table if not exists Insurance_Company
(
    company_ID varchar(128) not null,
    patient_ID varchar(128) not null,
    constraint insurance_Company_pk
        primary key (company_ID),
    constraint insurance_Company_Patient_patient_ID_fk
        foreign key (patient_ID) references Patient (patient_ID)
);

create table if not exists Payment
(
    receipt_ID  varchar(128)                              not null,
    paid_amount float                                     default 0.0
        check (paid_amount >= 0.0),
    paid_date   date                                      not null,
    paid_type   enum ('CHECK', 'CREDITCARD', 'INSURANCE') not null,
    bill_ID     varchar(128)                              not null,
    patient_ID  varchar(128)                              not null,
    company_ID  varchar(128)                              not null,
    constraint Payment_pk
        primary key (receipt_ID),
    constraint Payment_Bill_bill_ID_fk
        foreign key (bill_ID) references Bill (bill_ID),
    constraint Payment_Patient_patient_ID_fk
        foreign key (patient_ID) references Patient (patient_ID),
    constraint Payment_insurance_Company_company_ID_fk
        foreign key (company_ID) references insurance_Company (company_ID)
);
```

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

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

S2

Question

- Use the classic models database that you loaded.
- Write a query that returns the following results:

```
(customerNumber, customerName, no_of_orders, total_revenue)
```

- where:
    - `customerNumber` and `customerName` are from `customers`.
    - `no_of_orders` is the number of orders the customer has placed.
    - `total_revenue` is the sum of `quantityOrdered*priceEach` for all `orderDetails` in `orders` associated with a customer.

- If a customer has not placed any orders, `no_of_orders` and `total_revenue` must be `0`.

Answer

In [22]:

```
%%sql

use classicmodels;

with b as (
    with a as (
        select orderNumber,
               sum(quantityOrdered*orderdetails.priceEach)as total
        from orderdetails
        group by orderNumber)
    select customerNumber, count(customerNumber) as no_of_orders,
           sum(total) as total_revenue
    from a join orders on a.orderNumber = orders.orderNumber
    group by customerNumber)
select customers.customerNumber, customerName,
       IF(no_of_orders is null, 0, no_of_orders) as no_of_orders,
       IF(total_revenue is null, 0, total_revenue) as total_revenue
from b right join customers
on b.customerNumber = customers.customerNumber;
```

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

Out[22]:

customerNumber	customerName	no_of_orders	total_revenue
103	Atelier graphique	3	22314.36
112	Signal Gift Stores	3	80180.98
114	Australian Collectors, Co.	5	180585.07
119	La Rochelle Gifts	4	158573.12
121	Baane Mini Imports	4	104224.79
124	Mini Gifts Distributors Ltd.	17	591827.34
125	Havel & Zbyszek Co	0	0
128	Blauer See Auto, Co.	4	75937.76
129	Mini Wheels Co.	3	66710.56

Best Baseball Players

Question

- This question uses `lahmansdb_midterm.batting` , `lahmansdb_midterm.pitching` and `lahmansdb_midterm.people` . You previously loaded this information.

- There query computes performance metrics:
  - Batting:
    - On-base percentage: OBP is (sum(h) + sum(BB))/(sum(ab) + sum(BB)). This value is `NULL` if `sum(ab) = 0` .
    - Slugging percentage: SLG is defined by the function below. The value is `NULL` if `sum(ab) = 0` .

      ```
      (
      (sum(h) - sum(`2b`) - sum(`3b`) - sum(hr)) +
      2*sum(`2b`) + 3*sum(`3b`) + 4*hr
      )/sum(ab)
      ```

  - Pitching:
    - total_wins is `sum(w)` .
    - total_loses is `sum(l)` .
    - win_percentage is `sum(w)/(sum(w) + sum(l))` . This value is NULL if `sum(w) + sum(l) = 0` .

- Professor Ferguson has two criteria for someone being a great baseball player. A play must meet at least one of the criteria to be a great baseball player.
  - Batting:
    - Total number of `ab >= 1500` .
    - SLG: Career `SLG >= .575`
  - Pitching:
    - `(sum(w) + sum(l)) >= 200` .
    - `win_percentage >= 0.70)` or `sum(w) >= 300.`

- In your result table there is some additional guidance.
  - `great_because` is either `Pitcher` or `Batter` based on whether the player matched the batting or pitching criteria.
  - The values from `batting` are `None` if the player did not qualify based on batting.
  - The values from `pitching` are `None` if the player did not qualify on pitching.

Note: For this query to run efficiently, you will need to create indexes on the tables.

Answer

- Execute your create index statements below.

In [23]:

```
%%sql

use lahmansdb_midterm;

create index batting_index on batting(playerID(9),AB);
create index pitching_index on pitching(playerID(9),W);
create index people_index on people(playerID(9));
```

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

Out[23]:

```
[]
```

- Execute your SQL statement producing the query result below.

In [24]:

```
%%sql

use lahmansdb_midterm;

with qualified_batters as (
    with career_basic_batting as (
        select
            playerid,
            (sum(h)-sum(hr)-sum(`2b`)-sum(`3b`)) as career_singles,
            sum(`2b`) as career_doubles,
            sum(`3b`) as career_triples,
            sum(hr) as  career_hrs,
            sum(ab) as career_abs,
            sum(h) as career_hits,
            sum(bb) as career_walks
        from
            batting
        group by playerid
    ),
        career_averages_batting as (
            select
                playerid, career_abs, career_hits,
                career_singles, career_doubles,
                career_triples, career_hrs,
                career_walks,
                if(career_abs = 0, null,
                 (career_hits+career_walks)/(career_abs+career_walks))
                    as OBP,
                if(career_abs = 0, null,
                    (career_singles+2*career_doubles+3*career_triples
                         +4*career_hrs)/career_abs
                    ) as SLG
            from career_basic_batting
    )
    select playerID, career_abs, SLG
    from career_averages_batting where career_abs>=1500 and SLG>=0.575
),
    qualified_pitchers as (
    with career_pitching as (
            select
                playerid,
                sum(w) as total_wins,
                sum(l) as total_loses,
                sum(w) + sum(l) as total_games,
                if(sum(w) + sum(l) = 0, null,sum(w)/(sum(w)+sum(l)))
                    as win_percentage
            from
                pitching
            group by playerid
    )
    select playerID, total_games, total_wins, win_percentage
    from career_pitching
    where total_games>=200 and (win_percentage>=0.70 or total_wins>=300)
)
select people.playerID, nameLast,nameFirst,
       'Batter' as great_because,
       career_abs, SLG,
       null as total_games, null as total_wins, null as win_percentage
from people join qualified_batters
on qualified_batters.playerID=people.playerID
union
select people.playerID, nameLast, nameFirst,
       'Pitcher' as great_because,
       null as career_abs, null as SLG,
       total_games, win_percentage, total_wins
from people join qualified_pitchers
on qualified_pitchers.playerID=people.playerID;
```

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

Out[24]:

playerID	nameLast	nameFirst	great_because	career_abs	SLG	total_games	total_wins	win_
ruthba01	Ruth	Babe	Batter	8398	0.6898	None	None	
hornsro01	Hornsby	Rogers	Batter	8173	0.5765	None	None	
gehrilo01	Gehrig	Lou	Batter	8001	0.6324	None	None	
foxxji01	Foxx	Jimmie	Batter	8134	0.6093	None	None	
greenha01	Greenberg	Hank	Batter	5193	0.6050	None	None	
dimagjo01	DiMaggio	Joe	Batter	6821	0.5788	None	None	
willite01	Williams	Ted	Batter	7706	0.6338	None	None	
bondsba01	Bonds	Barry	Batter	9847	0.6069	None	None	
mcgwima01	McGwire	Mark	Batter	6187	0.5882	None	None	
ramirma02	Ramirez	Manny	Batter	8244	0.5854	None	None	
troutmi01	Trout	Mike	Batter	4656	0.5831	None	None	
spaldal01	Spalding	Al	Pitcher	None	None	317	0.7950	
galvipu01	Galvin	Pud	Pitcher	None	None	675	0.5407	
keefeti01	Keefe	Tim	Pitcher	None	None	567	0.6032	
welchmi01	Welch	Mickey	Pitcher	None	None	517	0.5938	
radboch01	Radbourn	Old Hoss	Pitcher	None	None	504	0.6151	
clarkjo01	Clarkson	John	Pitcher	None	None	506	0.6482	
nichoki01	Nichols	Kid	Pitcher	None	None	570	0.6351	
youngcy01	Young	Cy	Pitcher	None	None	826	0.6186	
mathech01	Mathewson	Christy	Pitcher	None	None	561	0.6649	
planked01	Plank	Eddie	Pitcher	None	None	520	0.6269	
johnswa01	Johnson	Walter	Pitcher	None	None	696	0.5991	
alexape01	Alexander	Pete	Pitcher	None	None	581	0.6420	
grovele01	Grove	Lefty	Pitcher	None	None	441	0.6803	
wynnea01	Wynn	Early	Pitcher	None	None	544	0.5515	
spahnwa01	Spahn	Warren	Pitcher	None	None	608	0.5970	
perryga01	Perry	Gaylord	Pitcher	None	None	579	0.5423	
niekrph01	Niekro	Phil	Pitcher	None	None	592	0.5372	
carltst01	Carlton	Steve	Pitcher	None	None	573	0.5742	
ryanno01	Ryan	Nolan	Pitcher	None	None	616	0.5260	
suttodo01	Sutton	Don	Pitcher	None	None	580	0.5586	
seaveto01	Seaver	Tom	Pitcher	None	None	516	0.6027	
clemero02	Clemens	Roger	Pitcher	None	None	538	0.6580	
maddugr01	Maddux	Greg	Pitcher	None	None	582	0.6100	
glavito02	Glavine	Tom	Pitcher	None	None	508	0.6004	
johnsra05	Johnson	Randy	Pitcher	None	None	469	0.6461	

Data and Schema Cleanup

Explanation and Setup

- There are several issues with the schema for `clasicmodels.` Two of the issues are:
  - `customers.country:` Having programs or people enter country names is prone to errors.
  - `products.productCode` is clearly not an atomic value.
- The following SQL creates a schema with copies of the data. The SQL also loads a table of ISO country codes. (https://en.wikipedia.org/wiki/List_of_ISO_3166_country_codes)

In [25]:
```
%sql create schema classicmodels_midterm;
```

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

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

In [26]:
```
iso_df = pandas.read_csv('./wikipedia-iso-country-codes.csv')
iso_df.to_sql('country_codes', schema='classicmodels_midterm',
              con=sql_engine, index=False, if_exists="replace")
```

Out[26]:
```
246
```

In [27]:

```
%%sql

use classicmodels_midterm;

alter table classicmodels_midterm.country_codes
    change `English short name lower case` short_name text null;

alter table classicmodels_midterm.country_codes
    change `Alpha-2 code` alpha_2_code text null;

alter table classicmodels_midterm.country_codes
    change `Alpha-3 code` alpha_3_code text null;

alter table classicmodels_midterm.country_codes
    change `Numeric code` numberic_code bigint null;

alter table classicmodels_midterm.country_codes
    change `ISO 3166-2` iso_text text null;
```

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

Out[27]:

```
[]
```

In [28]:

```
%%sql

use classicmodels_midterm;


create table customers as select * from classicmodels.customers;


create table products as select * from classicmodels.products;
```

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

Out[28]:

```
[]
```

DE-1

Question

- There are four `country` values in `customers` that are not in `short_names` of `country_codes.`
- The four missing values are:

country
USA
Norway
UK
Russia

- Write an SQL query that returns the information about by querying `customers` and `country_codes`

Answer

In [29]:

```
%%sql

use classicmodels_midterm;

select country from customers
where country not in (select short_name from country_codes)
group by country;
```

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

Out[29]:

country
USA
Norway
UK
Russia

DE-2

Question

- Norway is on the list because there are spaces in the entry. The following query shows this fact.

In [30]:

```
%%sql

use classicmodels_midterm;

select customerNumber, customerName, country
from customers where length(country) != length(trim(country));
```

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

Out[30]:

customerNumber	customerName	country
167	Herkku Gifts	Norway
299	Norway Gifts By Mail, Co.	Norway

- The mapping of the other country names is:

customers.country	country_codes.short_name
USA	United States
UK	United Kingdom
Russia	Russian Federation

- Write a **single** `update` statement that corrects the values for `customers.country` .

Answer

In [31]:

```
%%sql

use classicmodels_midterm;

update customers
    set country = case
                    when country='USA' then 'United States'
                    when country='UK' then 'United Kingdom'
                    when country='Russia' then 'Russian Federation'
                    when country like '%Norway%' then 'Norway'
                    else country
                  end;
```

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

Out[31]: []

DE-3

Question

- The final tasks are:
    - Add a column `iso_code` to `customers` that is the `alpha_2_code` from `country_codes`.
    - Create a foreign key relationship `customers.iso_code -> country_codes.alpha_2_code`.
    - Drop `country` from `customers`.
    - Create a view `customers_country` of the form `(customerNumber, customerName, country, iso_code)`.

Answer

In [32]:

```
%%sql

use classicmodels_midterm;

/* Put the ALTER TABLE and CREATE VIEW statements here. */

update country_codes
    set alpha_2_code = 'NA' where short_name='Namibia';

alter table country_codes
    modify alpha_2_code char(2),
    add unique (alpha_2_code);

alter table customers
    add iso_code char(2) null;

update customers
    join country_codes on country = short_name
    set iso_code = alpha_2_code;

alter table customers
    add constraint customers_country_codes_alpha_2_code_fk
        foreign key (iso_code) references country_codes (alpha_2_code),
    drop country;

create or replace view customers_country as
    select customerNumber,customerName,short_name as country,iso_code
    from customers join country_codes
    on customers.iso_code = country_codes.alpha_2_code;
```

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

Out[32]:

```
[]
```

In [33]:

```
%%sql

use classicmodels_midterm;

/* Write a SELECT that displays 25 customers sorted by customerName */

select * from customers_country
order by customerName
limit 25;
```

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

Out[33]:

customerNumber	customerName	country	iso_code
242	Alpha Cognac	France	FR
168	American Souvenirs Inc	United States	US
249	Amica Models & Co.	Italy	IT
237	ANG Resellers	Spain	ES
276	Anna's Decorations, Ltd	Australia	AU
465	Anton Designs, Ltd.	Spain	ES
206	Asian Shopping Network, Co	Singapore	SG
348	Asian Treasures, Inc.	Ireland	IE
103	Atelier graphique	France	FR
471	Australian Collectables, Ltd	Australia	AU
114	Australian Collectors, Co.	Australia	AU
333	Australian Gift Network, Co	Australia	AU
256	Auto Associés & Cie.	France	FR
406	Auto Canal+ Petit	France	FR
198	Auto-Moto Classics Inc.	United States	US
187	AV Stores, Co.	United Kingdom	GB
121	Baane Mini Imports	Norway	NO
415	Bavarian Collectables Imports, Co.	Germany	DE
293	BG&E Collectables	Switzerland	CH
128	Blauer See Auto, Co.	Germany	DE
219	Boards & Toys Co.	United States	US
344	CAF Imports	Spain	ES
173	Cambridge Collectables Co.	United States	US
202	Canadian Gift Exchange Network	Canada	CA
339	Classic Gift Ideas, Inc	United States	US

DE-4

- Just kidding.
- My first intent was to have you fix `products`.
- Then, I thought I would make this an extra credit question.
- Finally, I decided that all students get 5 points added to there score for this exam. Since I never "curve up," you all get a bonus on final grade for putting up with the class.

In [ ]: