Question

Consider the following relation definitions.

$$Customers(ID, last\_name, first\_name) \tag{2}$$

$$Accounts(ID, balance, customer\_ID) \tag{3}$$

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

Answer

Accounts.ID are view as same column with Customers.ID to pair up in natural join, which is actually not correct.

Accounts.customers_ID will be the correct column to join. We can clarify it by add ON statement

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

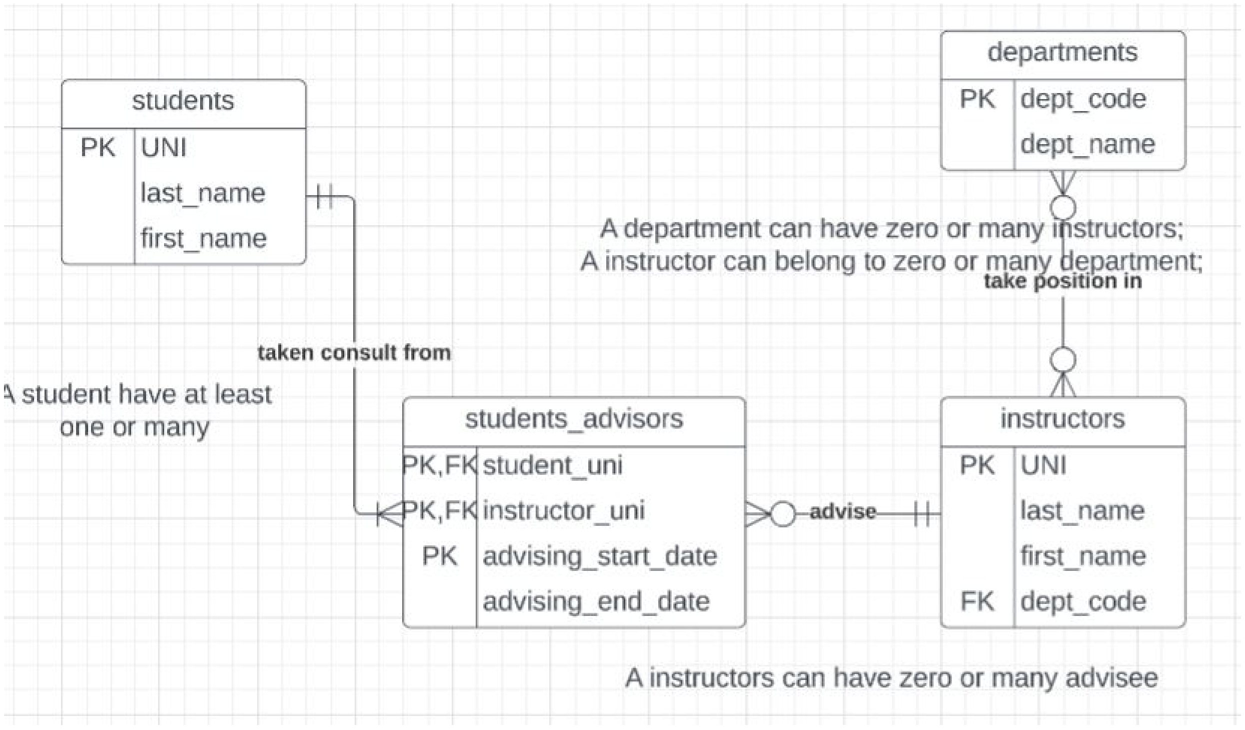
- *Put your screen capture in the same directory as the midterm.*
- *Add `<img src='./myfile.name'>` in the Markdown cell, using the actual file name.*

In [20]:

```
#above <img src="./jupyter-notebook.png"> can run, but cannot show after export html,
from IPython.display import Image

Image("ER1.jpg")
```

Out[20]:



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 [21]:
```
%sql select * from classicmodels.orders limit 10;
```

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

Out[21]:

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

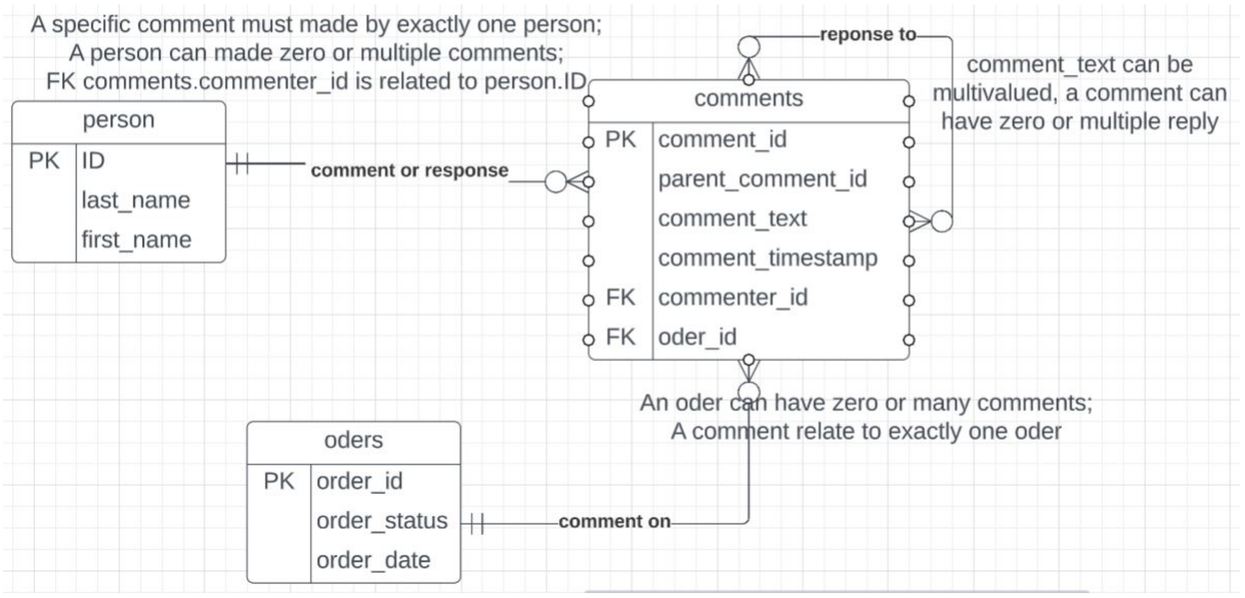
- *Put your screen capture in the same directory as the midterm.*

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

In [22]:
```
#above <img src="./jupyter-notebook.png"> can run, but cannot show after export html,
from IPython.display import Image

Image("ER3.jpg")
```

Out[22]:


A specific comment must made by exactly one person;
A person can made zero or multiple comments;
FK comments.commenter_id is related to person.ID
person
PK ID
last_name
first_name
comment or response
reponse to
comments
PK comment_id
parent_comment_id
comment_text
comment_timestamp
FK commenter_id
FK oder_id
comment_text can be multivalued, a comment can have zero or multiple reply
An oder can have zero or many comments;
A comment relate to exactly one oder
oders
PK order_id
order_status
order_date
comment on

Relational Algebra

- *Use the RelaX Calculator and the Silberschatz calculator 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:*

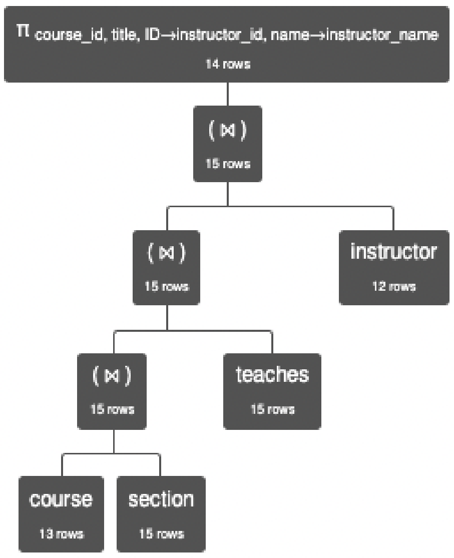
```
π course_id, title,
    instructor_id←ID,
    instructor_name←name
(
    (
        (course ⋈ section)
        ⋈
        teaches
    )
    ⋈
    instructor
)
```

In [23]:

```python
#above <img src="./jupyter-notebook.png"> can run, but cannot show after export html w
from IPython.display import Image

Image("relational-example.png")
```

Out[23]:



$\pi_{\text{course\_id, title, ID→instructor\_id, name→instructor\_name}}$ (((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*

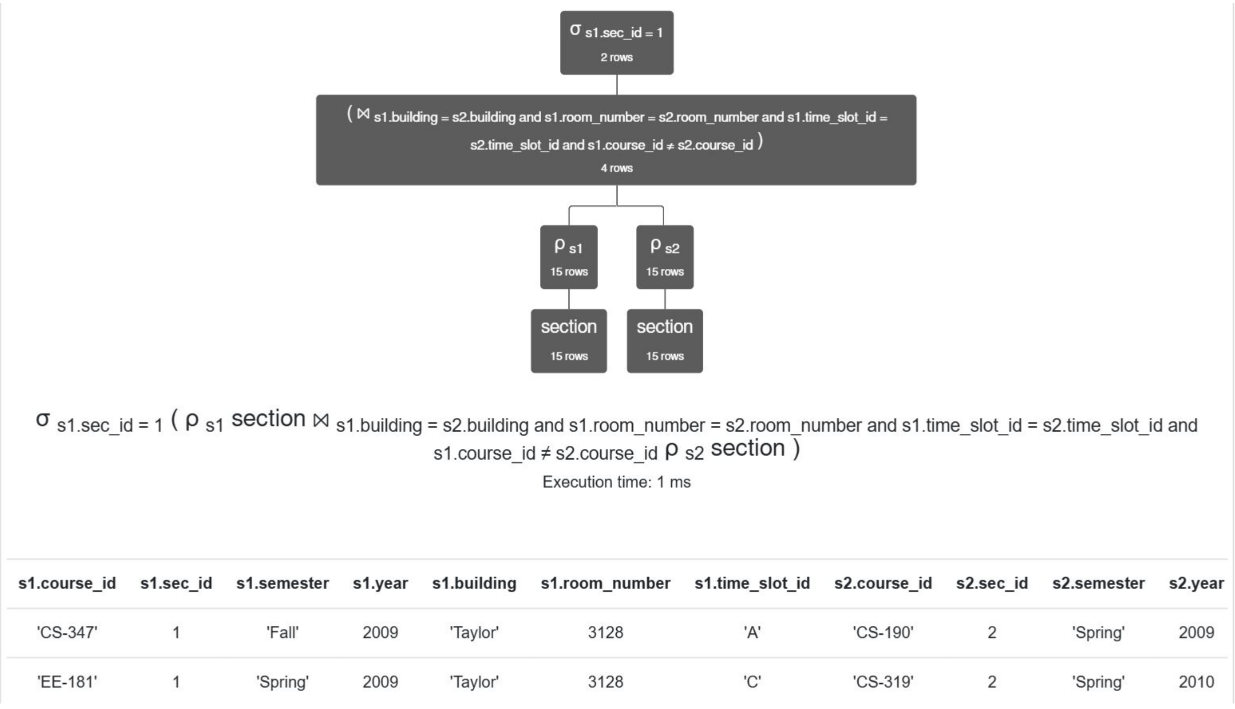
- *Your answer cannot include courses and sections that conflict with themselves, or have two rows that show the same conflict.*

```
σ s1.sec_id = 1 ( ρ s1 section ⋈ s1.building = s2.building and
s1.room_number = s2.room_number and s1.time_slot_id = s2.time_slot_id
and s1.course_id ≠ s2.course_id ρ s2 section )
```

In [24]:
```
#above <img src="./R1.png"> can run, but cannot show after export html, thus, I use in
from IPython.display import Image

Image("R1.jpg")
```

Out[24]:


σ s1.sec_id = 1
2 rows
(⋈ s1.building = s2.building and s1.room_number = s2.room_number and s1.time_slot_id = s2.time_slot_id and s1.course_id ≠ s2.course_id)
4 rows
ρ s1
15 rows
ρ s2
15 rows
section
15 rows
section
15 rows

σ s1.sec_id = 1 (ρ s1 section ⋈ s1.building = s2.building and s1.room_number = s2.room_number and s1.time_slot_id = s2.time_slot_id and s1.course_id ≠ s2.course_id ρ s2 section)
Execution time: 1 ms

s1.course_id	s1.sec_id	s1.semester	s1.year	s1.building	s1.room_number	s1.time_slot_id	s2.course_id	s2.sec_id	s2.semester	s2.year
'CS-347'	1	'Fall'	2009	'Taylor'	3128	'A'	'CS-190'	2	'Spring'	2009
'EE-181'	1	'Spring'	2009	'Taylor'	3128	'C'	'CS-319'	2	'Spring'	2010

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)`

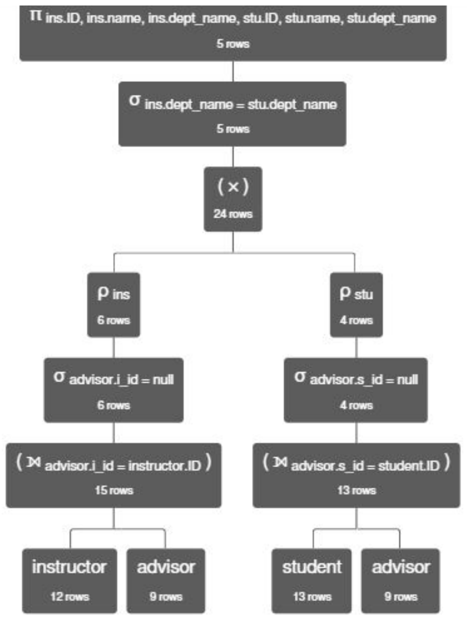
- *That matches instructors that do not advise students and students that do not have advisors and are in the same department.*

π ins.ID, ins.name, ins.dept_name, stu.ID, stu.name, stu.dept_name σ ins.dept_name = stu.dept_name (ρ ins (instructor ▷ advisor.i_id=instructor.ID advisor) × ρ stu (student ▷ advisor.s_id=student.ID advisor))

In [25]:
```
#above <img src="./R2.png"> can run, but cannot show after export html working, thus,
from IPython.display import Image

Image("R2.jpg")
```

Out[25]:



$\pi_{\text{ins.ID, ins.name, ins.dept\_name, stu.ID, stu.name, stu.dept\_name}}\ \sigma_{\text{ins.dept\_name = stu.dept\_name}} (\rho_{\text{ins}} (\sigma_{\text{advisor.i\_id = null}} (\text{instructor} ⟕_{\text{advisor.i\_id = instructor.ID}} \text{advisor})) \times \rho_{\text{stu}} (\sigma_{\text{advisor.s\_id = null}} (\text{student} ⟕_{\text{advisor.s\_id = student.ID}} \text{advisor})))$

Execution time: 2 ms

ins.ID	ins.name	ins.dept_name	stu.ID	stu.name	stu.dept_name
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.***

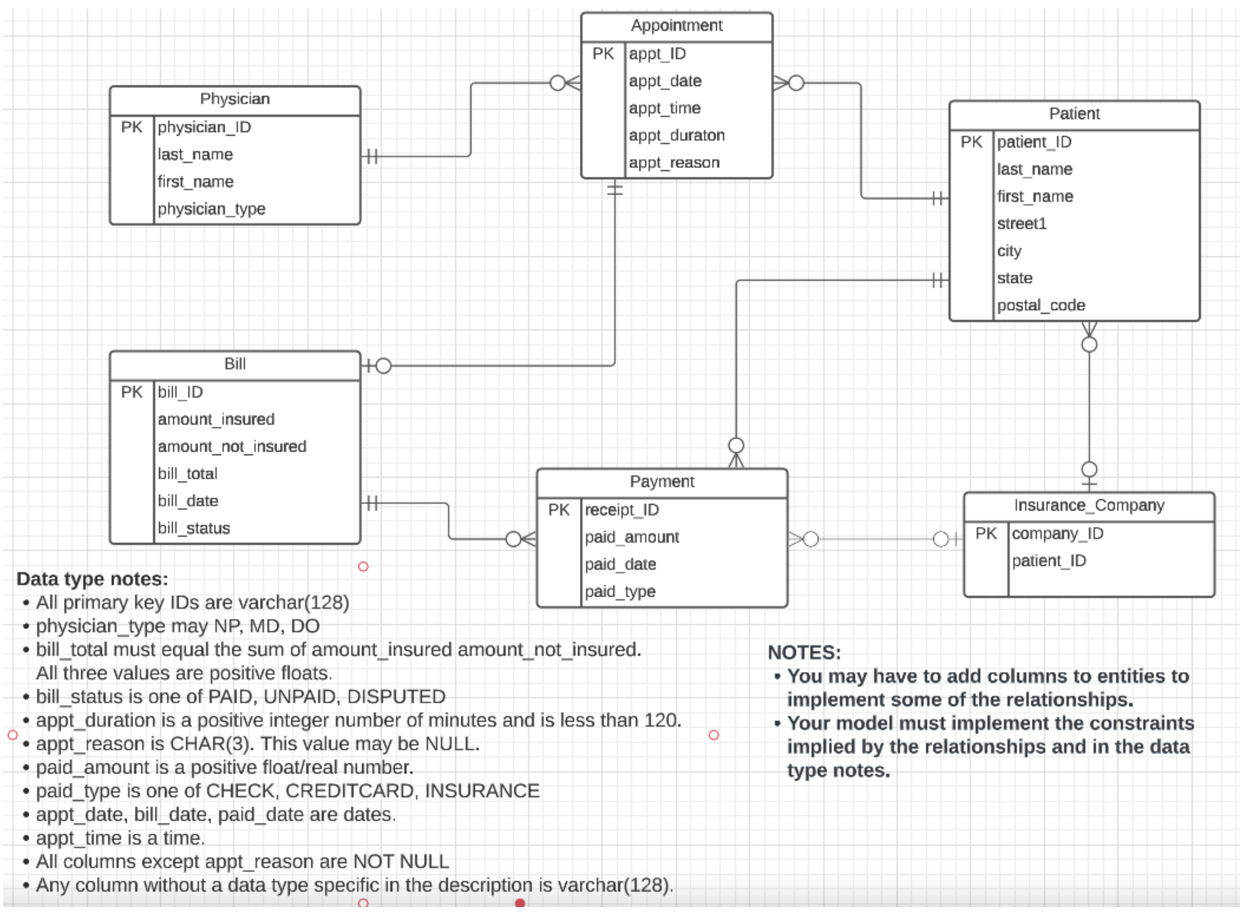
- *You should document any reasonable assumptions you make.*

ER Diagram

In [26]:
```
#above <img src="./jupyter-notebook.png"> can run, but cannot show after export html w
from IPython.display import Image

Image("s23-midterm-er-to-sql-v2.png")
```

Out[26]:



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.**

Answer

Design Decisions, Notes, etc:

1. *Add primiary key, foregin key by ALTER table. Impletement bill_total = amount_insured +amoutnt_not insured by* ***ADD CONSTRAINT check_total_amount****.*

2. *Because there are* ***amount_insured and amount_not_insured*** *in Bill table, it means bill can pay by two different paid_type. However,* ***paid_type*** *in Payment table can only have one value at a time. Thus, I will add one table* ***Pay_Method****, with extracted columns from Payment as weak entity of Payment table.*

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 [27]:
```
%sql drop schema if exists s23_midterm_medical


%sql create schema s23_midterm_medical
```

```
 * mysql+pymysql://root:***@localhost
7 rows affected.
 * mysql+pymysql://root:***@localhost
1 rows affected.
```
Out[27]:
```
[]
```

In [28]:
```
%%sql
use s23_midterm_medical;

drop table if exists Appointments;
create table if not exists Appointments
(
    appt_ID varchar(128)   not null
        primary key,
    appt_date date not null,
    appt_time time not null,
    appt_duration int CHECK(appt_duration > 0 AND appt_duration < 120) not null,
    appt_reason char(3),
    patient_ID varchar(128)   not null,
    physician_ID varchar(128)   not null
);
```

```
 * mysql+pymysql://root:***@localhost
0 rows affected.
0 rows affected.
0 rows affected.
```
Out[28]:
```
[]
```

In [29]:
```
%%sql
use s23_midterm_medical;

drop table if exists Patient;
create table if not exists Patient
(
    patient_ID varchar(128)   not null
        primary key,
    first_name varchar(128)   not null,
    last_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,
    company_ID varchar(128)   not null
);
```

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

Out[29]:
```
[]
```

In [30]:
```
%%sql
drop table if exists Insurance_Company;
create table if not exists Insurance_Company
(
    company_ID varchar(128)   not null
        primary key,
    patient_ID varchar(128)   not null
);

drop table if exists Payment;
create table if not exists Payment
(
    receipt_ID varchar(128)   not null
        primary key,
    paid_amount FLOAT CHECK(paid_amount > 0) not null,
    patient_ID varchar(128)   not null,
    company_ID varchar(128)   not null,
    bill_ID varchar(128)   not null
);

#This table is weak entity of Payment table. Reason is desribed in design note in beg
drop table if exists Payment_Method;
create table if not exists Payment_Method
(
    paid_date date not null,
    paid_type enum('CHECK','CREDITCARD','INSURANCE') not null
);

drop table if exists Bill;
create table if not exists Bill
(
    bill_ID varchar(128)   not null
        primary key,
    amount_insured FLOAT CHECK(amount_insured > 0) not null,
    amount_not_insured FLOAT CHECK(amount_not_insured > 0) not null,
    bill_total FLOAT CHECK(bill_total > 0) not null,
    bill_date date not null,
    bill_status enum ('PAID','UNPAID','DISPUTED') not null,
    appt_ID varchar(128)   not null
);

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

```
 * 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.
0 rows affected.
0 rows affected.
0 rows affected.
```

Out[30]:

```
[]
```

In [31]:

```
%%sql

ALTER TABLE Bill ADD FOREIGN KEY(appt_ID) references Appointments(appt_ID);
ALTER TABLE Payment ADD FOREIGN KEY(patient_ID) references Patient (patient_ID);
ALTER TABLE Payment ADD FOREIGN KEY(company_ID) references Insurance_Company (company_
ALTER TABLE Payment ADD FOREIGN KEY(bill_ID) references Bill (bill_ID);
ALTER TABLE Insurance_Company ADD FOREIGN KEY(patient_ID) references Patient (patient_
ALTER TABLE Patient ADD FOREIGN KEY(company_ID) references Insurance_Company (company_
ALTER TABLE Appointments ADD FOREIGN KEY(physician_ID) references Physician (physiciar
ALTER TABLE Appointments ADD FOREIGN KEY(patient_ID) references Patient (patient_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.
0 rows affected.
```

Out[31]:

```
[]
```

In [32]:

```
#add constraint to make sure total=insured+not insured
%sql ALTER TABLE Bill ADD CONSTRAINT check_total_amount CHECK (bill_total =amount_insu
```

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

Out[32]:

```
[]
```

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.*