COMS W4111-002, Fall 23:
Take Home Midterm

Overview

Instructions

Due Date: Sunday, October 22, 2023 at 11:59pm

You have one week to complete the take home 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. A citation would be of the form

```
I used this source for this question: https://stackoverflow.com/questions/298739/what-is-the-difference-between-a-schema-and-a-table-and-a-database.
```

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

There is a pinned thread on Edstem for corrections and calrifications. **Students are responsible for monitoring the thread. Do not waste our time by asking a question that has an answer.**

You may post **privately** on Edstem or attend OH for clarification questions. TAs will not be providing hints or help, except to clarify questions.

Students have a bad habit of posting questions, requests for help, etc., on forums and Edstem. We often see questions like,

- "This is my SQL for question 2, does anyone know why it does not work?"
- "I almost get the answer but it is off a little. Any suggestions?"

Do not do things like the above. That is a serious academic violation.

Submission Instructions

The TAs will post submission instructions on EdSTEM.

Environment Setup

You may need to change the MySQL userID and password in some of the cells below to match your configuration.

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

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

In [4]:
```
%sql SELECT 1
```

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

Out[4]:

1
1

In [5]:
```
import pandas as pd
from sqlalchemy import create_engine
```

In [6]:
```
sql_engine = create_engine("mysql+pymysql://root:dbuserdbuser@localhost")
```

In [15]:
```
from IPython.display import Image
```

Written Questions

Note: You may have to look in lecture notes, slides, slides associated with the recommended textbook, or online research to answer these questions. You will have to do online search and show initiatives in your careers to answer questions more complex then the ones in this exam.

You should cite online references/links. You do not need to cite the lectures, slides from lectures, or textbook material.

W1

Provide a short (two or three sentence) definition/description of the following terms. Some of these concepts do not have a single, precise, agreed definition. You may find slight differences in your research. Focus on the concept and grading will be flexible.

1. Super Key
2. Candidate Key
3. Primary Key
4. Alternate Key
5. Unique Key
6. Natural Key
7. Surrogate Key

8. Substitute Key
9. Foreign Key
10. External Key

Answer

- Super Key: A set of one or more attributes that can be used to uniquely identify a record in a table.
- Candidate Key: A minimal super key that can be used to uniquely identify a record.
- Primary Key: A arbitrarily chosen candidate key, usually immutable and contains no null values.
- Alternate Key: A candidate key that is not chosen as the primary key.
- Unique Key: An attribute or set of attributes that can uniquely identify a record within a table, similar to a primary key, but it allows for one null value.
- Natural Key: A type of unique key that has intrinsic meaning in the real world.
- Surrogate Key: An artificial, system-generated unique value used to uniquely identify a record in a table, having no semantic meaning.
- Substitute Key: Another term for a surrogate key.
- Foreign Key: An attribute or set of attributes in one table that refers to the primary key in another table, establishing a relationship between the two tables.
- External Key: A key linking a table to external system or dataset.

  I used this source for this question: https://blog.devart.com/surrogate-key-in-sql.html#:~:text=Conclusion-,What%20is%20a%20surrogate%20key%20in%20SQL,object%20generates%20this%20key%20itself.

W2

1. Define the concept of *immutable* column.

2. Why do some sources recommend that a primary key should be composed of immutable columns?

Answer

- An "immutable column" in a table means that once a value is inserted or initially set for that column, it should not be modified thereafter.
- Reasons: Changing primary keys can lead to confusion or errors in record referencing. It can disrupt foreign key relationships in other tables. Changing primary keys can disrupt consistent record identification, therefore disrupt data accuracy and enabling efficient data retrieval.

W3

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

Answer

- To hide original table information such as sensitive columns
- Since not everyone good at complex SQL, view make it easy to query
- When changes are made to physical level, views won't change and make sure upper level won't break

W4

Briefly explain the concepts of *procedural* language and *declarative* language. SQL is a declarative language. What are some advantages of a declarative language over a procedural language?

Answer

Procedural Language: A programming approach where the user specifies a sequence of steps to achieve a desired result. Declarative Language: A programming approach where the user defines what they want to achieve without explicitly outlining the steps to get there.

Pros:

- Simple, easier to read and understand
- Shorter and more concise, reducing the chances of errors and making code maintenance easier

Cons:

- Limited control over the execution process, hard to fine-tune performance or specify the order of operations in detail
- Debugging can be more challenging because we can't inspect each step of execution directly

W5

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

The diagram represents a three-tier architecture.

Presentation Tier: This is represented by the "User" and "Browser". It's where the user interacts with the system.

Logic Tier: Comprising the "Notebook server" and "Kernel", this is where processing occurs and business logic resides.

Database Tier: Represented by "MySQL", this tier handles data storage and retrieval.

W6

What is the difference between the database schema and the database instance? Do you use DDL for schema or instances? Do you use DML for schemas or instances?

Answer

Schema: A blueprint that defines the structure, organization, and constraints of a database.

Instance: An instance represents the actual data stored in a database at a specific point in time.

DDL is used for schema, DML is used for instances.

W7

The lecture slides and a previous homework defined a convention/notation for documenting the schema for a relation in the relational model. Use the notation to define the schema for a relation with the following columns:

- `product_category`
- `product_code`
- `product_name`
- `description`

The primary key is composed of `product_category` and `product_code`.

The following cell shows how to format text. Double click on the cell to see the source.

$$Something\ \underline{how\_to\_underline},\ some\ other\ stuff \tag{1}$$

Answer

$$Product(\underline{product\_category}, \underline{product\_code}, product\_name, description) \tag{2}$$

W8

Briefly define and explain:

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

Answer

Natural Join: A join that returns rows when there is a match in both tables based on all columns with the same name and data types. It automatically matches columns and discards duplicate columns.

Equi-join: A join on the equality between two columns, typically from two different tables. It uses the '=' operator to match rows based on the specified columns.

Theta Join: A more general form of join that uses any kind of binary comparison operator, to match rows based on a condition between two columns, like =, <, >, etc.

Self-join: A join operation in which a table is joined with itself. It's useful when comparing rows within the same table or finding relationships within the same dataset.

W9

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

Answer

- A table can have only one primary key constraint, but multiple unique constraints.
- The columns defined as the primary key cannot contain NULL values, while unique constraint columns can have NULLs.

W10

Briefly explan the difference between a *column (data) type* and the *column's domain*.

Answer

Domain: Set of all possible values for an attribute or column, defines the permissible range of values.

Data Type: Specifies the data format and associated constraints.

Domain is a constraint on value of data, while data type is a constraint on the format of data.

Entity Relationship Model

- This question tests transforming a high-level description of a data model into a more concrete *logical ER* diagram. You will produce a logical ER diagram using Lucidchart. You should use Crow's Foot notation and conventions we have used in lectures and examples.

- The data model is a simple representation of a university.

- The model has the following entity types:
  - School:
    - School code, e.g., "SEAS," "GSAS," "LAW," ... ...
    - School name, e.g., "School of Engineering and Applied Science."
  - Department:
    - Department code, e.g., "COMS," "MATH," "ECON," ... ...
    - Department name, e.g., "Department of Computer Science."
  - Faculty:
    - UNI
    - last name
    - first name
    - email
    - title, e.g., "Professor," "Adjunct Professor," ... ...
  - Student:
    - UNI
    - last name
    - first name
    - email
  - Course:
    - Course number is a composite key, e.g., "COMSW4111" is
      - Dept. code "COMS"

        - Faculty code "W"
        - Course number "4111"
      - Course title
      - Course description
    - Section:
      - Call number
      - Course number
      - Year
      - Semester
      - Section

- A Faculty has complex states and relationships.
  - A Faculty can have a role relative to a Department. The possible roles are `chair`, `professor`, `adjunct` and `emeritus`.
  - Roles change over time. The data model must support the ability to handle current roles and previous roles, and the dates for the roles.

- A Student has a relationship to Section.
  - The possible roles are `(Enrolled, Waitlist, Dropped, TA)`
  - The student has a current role, and there can be only one current role.
  - The data model must support the ability to handle roles changing over time and retain information about prior roles.

- A Faculty *may* teach a Section. All sections have exactly one Faculty.

- The relationship between Department and School is many-to-many. Each Department is in at least one school and each school has at least one department.

Notes:

1. There is no single correct answer to this question. You will have to make some design decisions and assumptions. You should document your decisions and assumptions. You can do this by putting a note/comment in your diagram.

2. You do not have to worry about **isA** relationships.
3. You do not have to document or worry about attribute types.
4. The ER diagram must be implementable in the relational/SQL model.

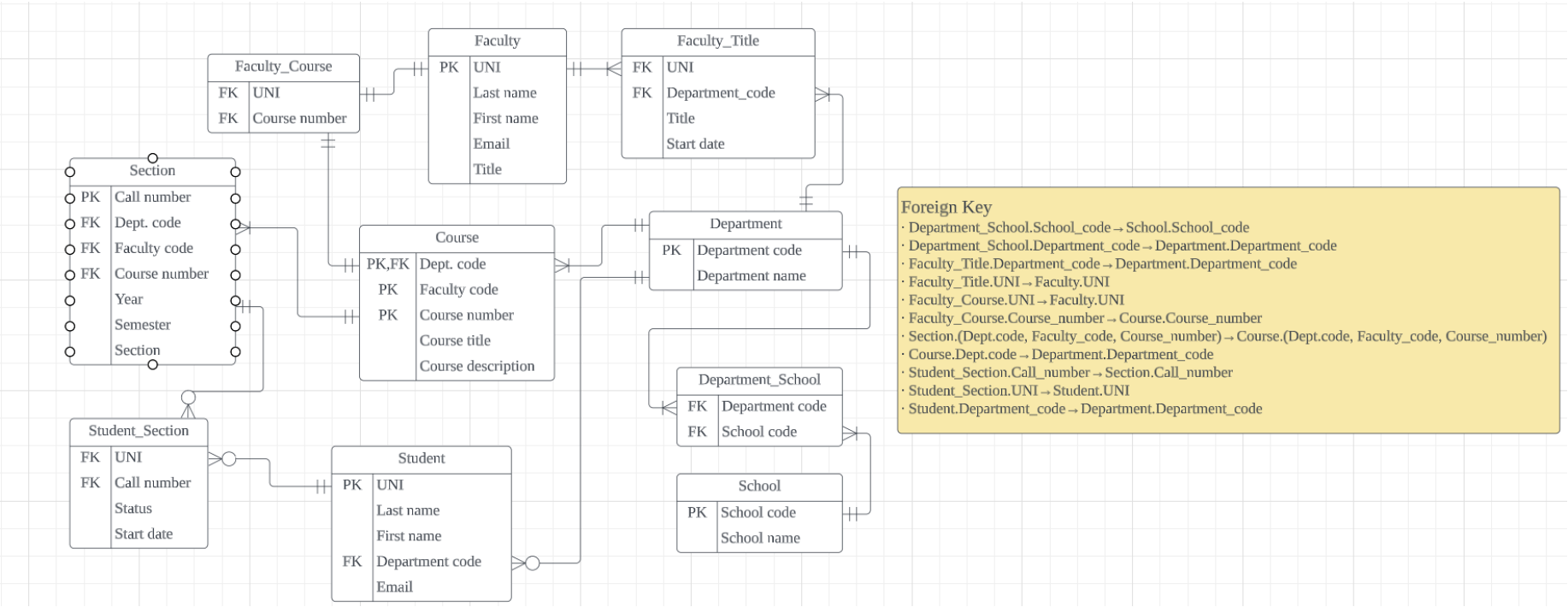
Answer

Assumptions:

1. Each student only belongs to one department
2. Each course is only under one department
3. A department can have zero to many students
4. A student can register zero to many sections(courses)

In [17]:
```
Image('ER.png')
```

Out[17]:



Relational Algebra

R1

Use the RelaX Calculator and the Silberschatz - UniversityDB for this question.

Two time slots *X* and *Y* obviously overlap if:

1. They are not the same time slot, i.e. do not have the same $time\_slot\_id$.
2. They have at least one lecture on *the same day,* the start hour for *X* is before the start hour for *Y*, and the end hour for *X* is after the start hour for *Y.*
3. To make the question easier, you do not need to consider minutes in computing overlap but must show minutes in the result.

Write the relational algebra expression that identifies obviously overlapping time slots, and only lists overlapping pairs of time slots once.

Your output must match the answer below.

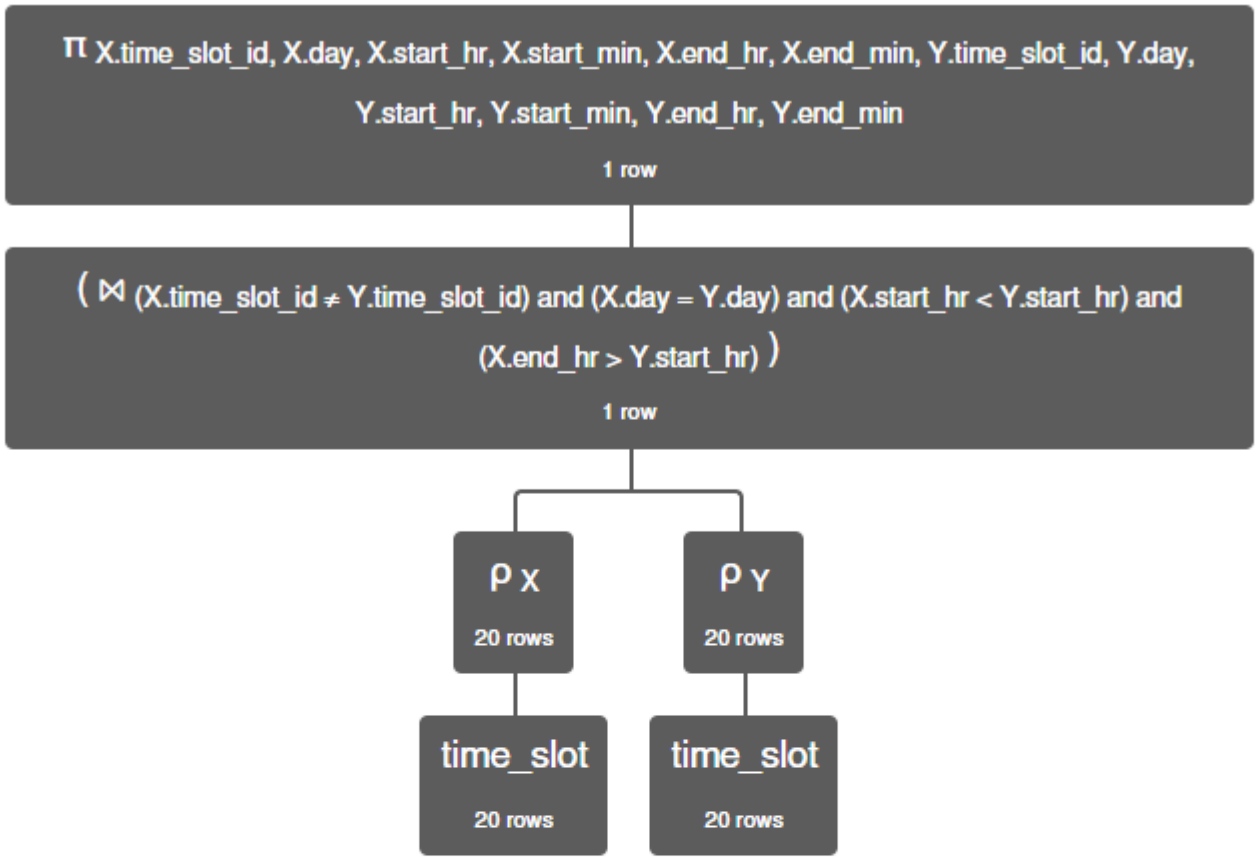
Answer

```
    π X.time_slot_id,
      X.day,
      X.start_hr,
      X.start_min,
      X.end_hr,
      X.end_min,
      Y.time_slot_id,
      Y.day,
      Y.start_hr,
      Y.start_min,
      Y.end_hr,
      Y.end_min(
    ρ X(time_slot) ⋈
          (X.time_slot_id ≠ Y.time_slot_id)∧
          (X.day = Y.day)∧
          (X.start_hr < Y.start_hr)∧
```

```
          (X.end_hr > Y.start_hr)
    ρ Y(time_slot))
```

In [18]:
```
Image('R1 result.png')
```

Out[18]:



π X.time_slot_id, X.day, X.start_hr, X.start_min, X.end_hr, X.end_min, Y.time_slot_id, Y.day, Y.start_hr, Y.start_min, Y.end_hr, Y.end_min (ρ X (time_slot) ⋈ (X.time_slot_id ≠ Y.time_slot_id) and (X.day = Y.day) and (X.start_hr < Y.start_hr) and (X.end_hr > Y.start_hr) ρ Y (time_slot))

Execution time: 0 ms

X.time_slot_id	X.day	X.start_hr	X.start_min	X.end_hr	X.end_min	Y.time_slot_id	Y.day	Y.sta

'H'	'W'	10	0	12	30	'C'	'W'	1

R2

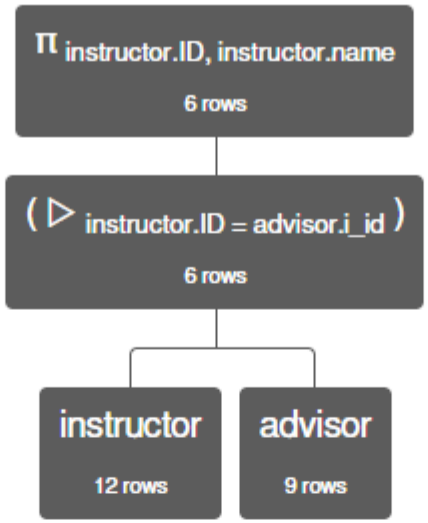
1. You **may not** use the subtraction operator `-` to write this query.
2. Produce a relation that:
   - Has column names `instructor_ID, instructor_name.`
   - Contains the `ID` and `name` of instructors who do not advise any students.

Answer

π instructor.ID, instructor.name (instructor $\triangleright$ instructor.ID = advisor.i_id advisor)

In [19]:
```
Image('R2_result.png')
```

Out[19]:



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

Execution time: 10 ms

instructor.ID	instructor.name
12121	'Wu'
15151	'Mozart'
32343	'El Said'
33456	'Gold'
58583	'Califieri'

83821 'Brandt'

SQL Schema and DDL

Objective

- You have a logical datamodel ER diagram (see below).

- You need to use DDL to define a schema that realizes the model. Name your schema `f23_w4111_midterm_medical`.

- 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 may need to look do some self-study to find information about concepts, e.g. `GENERATED`. Cite your sources.

ER Diagram

ER Diagram

Answer

Design Decisions, Notes, etc.

I used this source for this question: https://www.simplilearn.com/tutorials/sql-tutorial/auto-increment-in-sql#:~:text=The%20auto%20increment%20in%20SQL,for%20every%20record%20you%20add.

Here is my logic in building this schema:

- All the common columns between table are not specified in the ER diagram, therefore should be added specifically. If A to B is one to many relationship and B to A is one to one relationship, refer table B to table A when adding foerign keys by including PK of table A to table B and establish the FK between two columns.
- Since one insurance company can insure many patients, insured_person column should not appear in Insurance_Company table, but should add insurance company id as a column in Patient table.
- Auto increment are used to generate receipt_no in Payment since no column in other table is refering this column.

DDL

- Execute your DDL in the cell below. You may use DataGrip or other tools to help build the DDL statements.

- You can copy and paste the SQL `CREATE/ALTER TABLE` statements below, but you MUST execute the statements in the notebook.

In [126…
```
%%sql

# DDL in cells below.

DROP SCHEMA IF EXISTS f23_w4111_midterm_medical;
CREATE SCHEMA f23_w4111_midterm_medical;
USE f23_w4111_midterm_medical;
```

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

Out[126]:
```
[]
```

In [127…
```
%%sql

create table if not exists f23_w4111_midterm_medical.insurance_company
(
    company_id  varchar(50)  not null
```

```
        primary key,
    phone_no    varchar(12)  null,
    street1     varchar(255) null,
    city        varchar(255) null,
    state       varchar(50)  null,
    postal_code varchar(50)  null
);

create table if not exists f23_w4111_midterm_medical.patient
(
    patient_ID       varchar(50)  not null
        primary key,
    last_name        varchar(9)   not null,
    first_name       varchar(9)   not null,
    street1          varchar(255) null,
    city             varchar(255) null,
    state            varchar(50)  null,
    postal_code      varchar(10)  null,
    insur_company_id varchar(50)  null,
    constraint patient_insurance_company_company_id_fk
        foreign key (insur_company_id) references f23_w4111_midterm_medical.insurance_company (company_id)
);

create table if not exists f23_w4111_midterm_medical.physician
(
    physician_ID   varchar(50)             not null
        primary key,
    last_name      varchar(9)              not null,
    first_name     varchar(9)              not null,
    physician_type enum ('NP', 'MD', 'DO') not null
);

create table if not exists f23_w4111_midterm_medical.appointment
(
    appt_ID       varchar(50)    not null
        primary key,
    appt_date     date           not null,
    appt_time     time           not null,
    appt_duration decimal(10, 2) not null,
    appt_reason   text           null,
    physician_ID  varchar(50)    not null,
    patient_ID    varchar(50)    not null,
    constraint appointment_patient_patient_ID_fk
        foreign key (patient_ID) references f23_w4111_midterm_medical.patient (patient_ID),
```

```
    constraint appointment_physician_physician_ID_fk
        foreign key (physician_ID) references f23_w4111_midterm_medical.physician (physician_ID)
);

create table if not exists f23_w4111_midterm_medical.bill
(
    bill_no            varchar(50)                    not null
        primary key,
    amount_insured     decimal(10, 2)                 not null,
    amount_not_insured decimal(10, 2)                 not null,
    bill_total         decimal(10, 2)                 null,
    bill_date          date                           null,
    bill_status        varchar(9) default 'Pending'   null,
    appt_ID            varchar(50)                    null,
    constraint bill_appointment_appt_ID_fk
        foreign key (appt_ID) references f23_w4111_midterm_medical.appointment (appt_ID)
);

create table if not exists f23_w4111_midterm_medical.payment
(
    receipt_no       int auto_increment
        primary key,
    paid_amount      decimal(10, 2)                    not null,
    paid_date        date                              null,
    paid_type        enum ('CC', 'Check', 'Transfer')  not null,
    bill_no          varchar(9)                        null,
    insur_company_id varchar(50)                       null,
    patient_ID       varchar(50)                       null,
    constraint payment_bill_bill_no_fk
        foreign key (bill_no) references f23_w4111_midterm_medical.bill (bill_no),
    constraint payment_insurance_company_company_id_fk
        foreign key (insur_company_id) references f23_w4111_midterm_medical.insurance_company (company_id),
    constraint payment_patient_patient_ID_fk
        foreign key (patient_ID) references f23_w4111_midterm_medical.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.
```

Out[127]: []

In [ ]:

Complex SQL

Birth Countries and Death Countries

- **Note:** Use the instructions from HW1 Part 2 Nonprogramming to load the Lahman's Baseball Database data into MySQL if you have not already done so.

- In Lahman's Baseball Database table `people`, there is information about people's `birthCountry` and `deathCountry`.

- There are countries in which at least one person was born but in which no person has died.

- Write a query that produces a table of the form:
  - `birthCountry`
  - `no_of_births`, which is the total number of births in the country.

- The table contains all rows for countries in which there with births but no deaths.

- You may create indexes to improve performance.

Answer

In [96]:
```
%%sql

USE lahmans_hw1;
```

```
WITH death AS (
    SELECT deathCountry, COUNT(*) AS no of deaths
    FROM people
    GROUP BY deathCountry
),

birth AS (
    SELECT birthCountry, COUNT(*) AS no of births
    FROM people
    GROUP BY birthCountry
)

SELECT birthCountry, no of births
FROM birth
LEFT JOIN death ON birth.birthCountry = death.deathCountry
WHERE death.deathCountry IS NULL AND birthCountry != 'None';
```

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

Out[96]:

birthCountry	no_of_births
Russia	9
Curacao	16
Colombia	28
Nicaragua	15
Germany	45
Norway	3
Italy	7
South Korea	27
Czech Republic	7
Aruba	6
Sweden	4
Hong Kong	1
Afghanistan	1
Spain	4
Greece	1
Jamaica	4
Poland	6
Honduras	2
Brazil	5
Viet Nam	1
Guam	2
Denmark	1
Switzerland	1
Singapore	1

birthCountry	no_of_births
Belgium	1
Peru	1
Belize	1
Indonesia	1
Finland	1
Lithuania	1
South Africa	2
Slovakia	1
Saudi Arabia	2
Portugal	1
Latvia	1

Best Baseball Players

- This question uses `lahmansbaseballdb.batting`, `lahmansbaseballdb.pitching` and `lahmansbaseballdb.people`.

- These query computes performance metrics:

  - Batting:
    - On-base percentage: OBP is `(sum(h) + sum(BB))/(sum(ab) + sum(BB))`
    - Slugging percentage: SLG is

$$\frac{(sum(h) - sum(`2b`) - sum(`3b`) - sum(hr)) + 2 * sum(`2b`) + 3 * sum(`3b`) + 4 * sum(hr))}{sum(ab)} \tag{3}$$

    - On-base percentage plus slugging: OPS is is `(obp + slg)`.

    - Pitching:
        - total_wins is `sum(w)`.
        - total_loses is `sum(l)`.
        - win_percentage is `sum(w)/(sum(w) + sum(l))`.

- Professor Ferguson has two criteria for someone being a great baseball player.

    - Batting:
        - Total number of `ab >= 3000`.
        - OPS: Career `OPS >= 1.000`
    - Pitching:
        - `(sum(w) + sum(l)) >= 200`.
        - `win_percentage >= 0.70)` or `sum(w) >= 300.`

- This is one of the rare cases where Prof. Ferguson will provide the answer. So, please produce the table below. Some notes:

    - `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.

- There is a CSV file in the directory with my sample answer. The columns are:
    - `playerid`
    - `nameLast` from `People`
    - `nameFirst` from `People.`
    - `great_because`
    - `slg`
    - `obp`
    - `op_slg` is on-base percentage plus slugging.
    - `total_abs` is total at bats.
    - `total_w` is the total wins.

- `total_l` is the total loses.
- `total_d` is some of `total_w` and `total_l`.
- `win_percentage` is the winning percentage.

- **Note:** Since I saved the query to Pandas and then read it back it,
  - All number become floating point. Your answer uses SQL and the values should be integers except for `win_percentage.`
  - Pandas converts `NULL` to `NaN`. In your SQL result, you will have `None` instead of `NaN.`

In [45]:
```
bb_greats = pd.read_csv('./baseball_greats.csv')
bb_greats
```

Out[45]:

	playerid	nameLast	nameFirst	great_because	slg	obp	obp_slg	total_abs	total_w	total_l	total_d	win_percentage
0	alexape01	Alexander	Pete	Pitcher	NaN	NaN	NaN	NaN	373.0	208.0	581.0	0.6420
1	bondsba01	Bonds	Barry	Batter	0.6069	0.4428	1.0497	9847.0	NaN	NaN	NaN	NaN
2	carltst01	Carlton	Steve	Pitcher	NaN	NaN	NaN	NaN	329.0	244.0	573.0	0.5742
3	carutbo01	Caruthers	Bob	Pitcher	NaN	NaN	NaN	NaN	218.0	99.0	317.0	0.6877
4	clarkjo01	Clarkson	John	Pitcher	NaN	NaN	NaN	NaN	328.0	178.0	506.0	0.6482
5	clemero02	Clemens	Roger	Pitcher	NaN	NaN	NaN	NaN	354.0	184.0	538.0	0.6580
6	fordwh01	Ford	Whitey	Pitcher	NaN	NaN	NaN	NaN	236.0	106.0	342.0	0.6901
7	foutzda01	Foutz	Dave	Pitcher	NaN	NaN	NaN	NaN	147.0	66.0	213.0	0.6901
8	foxxji01	Foxx	Jimmie	Batter	0.6093	0.4275	1.0368	8134.0	NaN	NaN	NaN	NaN
9	galvipu01	Galvin	Pud	Pitcher	NaN	NaN	NaN	NaN	365.0	310.0	675.0	0.5407
10	gehrilo01	Gehrig	Lou	Batter	0.6324	0.4447	1.0772	8001.0	NaN	NaN	NaN	NaN
11	glavito02	Glavine	Tom	Pitcher	NaN	NaN	NaN	NaN	305.0	203.0	508.0	0.6004
12	greenha01	Greenberg	Hank	Batter	0.6050	0.4103	1.0153	5193.0	NaN	NaN	NaN	NaN
13	grovele01	Grove	Lefty	Pitcher	NaN	NaN	NaN	NaN	300.0	141.0	441.0	0.6803
14	hornsro01	Hornsby	Rogers	Batter	0.5765	0.4308	1.0073	8173.0	NaN	NaN	NaN	NaN
15	johnsra05	Johnson	Randy	Pitcher	NaN	NaN	NaN	NaN	303.0	166.0	469.0	0.6461
16	johnswa01	Johnson	Walter	Pitcher	NaN	NaN	NaN	NaN	417.0	279.0	696.0	0.5991
17	keefeti01	Keefe	Tim	Pitcher	NaN	NaN	NaN	NaN	342.0	225.0	567.0	0.6032
18	kershcl01	Kershaw	Clayton	Pitcher	NaN	NaN	NaN	NaN	185.0	84.0	269.0	0.6877
19	maddugr01	Maddux	Greg	Pitcher	NaN	NaN	NaN	NaN	355.0	227.0	582.0	0.6100
20	martipe02	Martinez	Pedro	Pitcher	NaN	NaN	NaN	NaN	219.0	100.0	319.0	0.6865
21	mathech01	Mathewson	Christy	Pitcher	NaN	NaN	NaN	NaN	373.0	188.0	561.0	0.6649
22	nichoki01	Nichols	Kid	Pitcher	NaN	NaN	NaN	NaN	362.0	208.0	570.0	0.6351
23	niekrph01	Niekro	Phil	Pitcher	NaN	NaN	NaN	NaN	318.0	274.0	592.0	0.5372

	playerid	nameLast	nameFirst	great_because	slg	obp	obp_slg	total_abs	total_w	total_l	total_d	win_percentage
24	perryga01	Perry	Gaylord	Pitcher	NaN	NaN	NaN	NaN	314.0	265.0	579.0	0.5423
25	planked01	Plank	Eddie	Pitcher	NaN	NaN	NaN	NaN	326.0	194.0	520.0	0.6269
26	radboch01	Radbourn	Old Hoss	Pitcher	NaN	NaN	NaN	NaN	310.0	194.0	504.0	0.6151
27	ruthba01	Ruth	Babe	Batter	0.6898	0.4718	1.1616	8398.0	NaN	NaN	NaN	NaN
28	ryanno01	Ryan	Nolan	Pitcher	NaN	NaN	NaN	NaN	324.0	292.0	616.0	0.5260
29	seaveto01	Seaver	Tom	Pitcher	NaN	NaN	NaN	NaN	311.0	205.0	516.0	0.6027
30	spahnwa01	Spahn	Warren	Pitcher	NaN	NaN	NaN	NaN	363.0	245.0	608.0	0.5970
31	spaldal01	Spalding	Al	Pitcher	NaN	NaN	NaN	NaN	252.0	65.0	317.0	0.7950
32	suttodo01	Sutton	Don	Pitcher	NaN	NaN	NaN	NaN	324.0	256.0	580.0	0.5586
33	welchmi01	Welch	Mickey	Pitcher	NaN	NaN	NaN	NaN	307.0	210.0	517.0	0.5938
34	willite01	Williams	Ted	Batter	0.6338	0.4806	1.1144	7706.0	NaN	NaN	NaN	NaN
35	wynnea01	Wynn	Early	Pitcher	NaN	NaN	NaN	NaN	300.0	244.0	544.0	0.5515
36	youngcy01	Young	Cy	Pitcher	NaN	NaN	NaN	NaN	511.0	315.0	826.0	0.6186

Answer

In [55]:
```
%%sql

USE lahmans_hw1;

WITH BattingStats AS (SELECT playerid,
                             (sum(H) + sum(BB)) / (sum(AB) + sum(BB)) AS obp,
                             (sum(H) - sum(2B) - sum(3B) - sum(HR) + 2 * sum(2B) + 3 * sum(3B) + 4 * sum(HR)) /
                             sum(AB)                                  AS slg,
                             ((sum(H) + sum(BB)) / (sum(AB) + sum(BB))) +
                             ((sum(H) - sum(2B) - sum(3B) - sum(HR) + 2 * sum(2B) + 3 * sum(3B) + 4 * sum(HR)) /
                              sum(AB))                                AS obp_slg,
                             sum(AB)                                  AS total_abs
                      FROM Batting
                      GROUP BY playerid
                      HAVING total_abs >= 3000
                         AND obp_slg >= 1.000),
```

```
    PitchingStat AS (SELECT playerid,
                            sum(W)                      as total_w,
                            sum(L)                      as total_l,
                            sum(W) / (sum(W) + sum(L)) AS win_percentage,
                            sum(W) + sum(L)             AS total_d
                     FROM Pitching
                     GROUP BY playerid
                     HAVING total_d >= 200
                        AND (win_percentage >= 0.70 OR total_w >= 300))

SELECT People.playerid,
       People.nameLast,
       People.nameFirst,
       CASE
           WHEN BattingStats.playerid IS NOT NULL THEN 'Batter'
           WHEN PitchingStat.playerid IS NOT NULL THEN 'Pitcher'
           END                      AS great_because,
       BattingStats.slg             AS slg,
       BattingStats.obp             AS obp,
       BattingStats.obp_slg         AS obp_slg,
       BattingStats.total_abs       AS total_abs,
       PitchingStat.total_w         AS total_w,
       PitchingStat.total_l         AS total_l,
       PitchingStat.total_d         AS total_d,
       PitchingStat.win_percentage AS win_percentage
FROM People
         LEFT JOIN BattingStats
                   ON People.playerid = BattingStats.playerid
         LEFT JOIN PitchingStat
                   ON People.playerid = PitchingStat.playerid
WHERE BattingStats.playerid IS NOT NULL
   OR PitchingStat.playerid IS NOT NULL;
```

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

Out[55]:

playerid	nameLast	nameFirst	great_because	slg	obp	obp_slg	total_abs	total_w	total_l	total_d	win_percentage
alexape01	Alexander	Pete	Pitcher	None	None	None	None	373	208	581	0.6420
bondsba01	Bonds	Barry	Batter	0.6069	0.4428	1.0497	9847	None	None	None	None
carltst01	Carlton	Steve	Pitcher	None	None	None	None	329	244	573	0.5742
clarkjo01	Clarkson	John	Pitcher	None	None	None	None	328	178	506	0.6482
clemero02	Clemens	Roger	Pitcher	None	None	None	None	354	184	538	0.6580
foxxji01	Foxx	Jimmie	Batter	0.6093	0.4275	1.0368	8134	None	None	None	None
galvipu01	Galvin	Pud	Pitcher	None	None	None	None	365	310	675	0.5407
gehrilo01	Gehrig	Lou	Batter	0.6324	0.4447	1.0772	8001	None	None	None	None
glavito02	Glavine	Tom	Pitcher	None	None	None	None	305	203	508	0.6004
greenha01	Greenberg	Hank	Batter	0.6050	0.4103	1.0153	5193	None	None	None	None
grovele01	Grove	Lefty	Pitcher	None	None	None	None	300	141	441	0.6803
hornsro01	Hornsby	Rogers	Batter	0.5765	0.4308	1.0073	8173	None	None	None	None
johnsra05	Johnson	Randy	Pitcher	None	None	None	None	303	166	469	0.6461
johnswa01	Johnson	Walter	Pitcher	None	None	None	None	417	279	696	0.5991
keefeti01	Keefe	Tim	Pitcher	None	None	None	None	342	225	567	0.6032
maddugr01	Maddux	Greg	Pitcher	None	None	None	None	355	227	582	0.6100
mathech01	Mathewson	Christy	Pitcher	None	None	None	None	373	188	561	0.6649
nichoki01	Nichols	Kid	Pitcher	None	None	None	None	362	208	570	0.6351
niekrph01	Niekro	Phil	Pitcher	None	None	None	None	318	274	592	0.5372
perryga01	Perry	Gaylord	Pitcher	None	None	None	None	314	265	579	0.5423
planked01	Plank	Eddie	Pitcher	None	None	None	None	326	194	520	0.6269
radboch01	Radbourn	Old Hoss	Pitcher	None	None	None	None	310	194	504	0.6151
ruthba01	Ruth	Babe	Batter	0.6898	0.4718	1.1616	8398	None	None	None	None
ryanno01	Ryan	Nolan	Pitcher	None	None	None	None	324	292	616	0.5260

playerid	nameLast	nameFirst	great_because	slg	obp	obp_slg	total_abs	total_w	total_l	total_d	win_percentage
seaveto01	Seaver	Tom	Pitcher	None	None	None	None	311	205	516	0.6027
spahnwa01	Spahn	Warren	Pitcher	None	None	None	None	363	245	608	0.5970
spaldal01	Spalding	Al	Pitcher	None	None	None	None	252	65	317	0.7950
suttodo01	Sutton	Don	Pitcher	None	None	None	None	324	256	580	0.5586
welchmi01	Welch	Mickey	Pitcher	None	None	None	None	307	210	517	0.5938
willite01	Williams	Ted	Batter	0.6338	0.4806	1.1144	7706	None	None	None	None
wynnea01	Wynn	Early	Pitcher	None	None	None	None	300	244	544	0.5515
youngcy01	Young	Cy	Pitcher	None	None	None	None	511	315	826	0.6186

In [ ]:

Data and Schema Cleanup

Part 1: Countries and Cities

- There is a file `worldcities.csv` in the same folder as this notebook.

- In the following code cell, use Pandas to:
  - Read the CSV file into a Data Frame.
  - Convert the Data Frame to contain only the following columns:
    - `city`
    - `city_ascii`
    - `lat`
    - `lng`
    - `country`
    - `iso2`

    - `iso3`
    - `id`
  - Write the data to the table `worldcities` in the schema `F23W4111Midterm`.

- Use the SQL after the code cell to display part of your new table.

Note: In lecture examples and HW, you have seen how to use Pandas to read CSV files and create tables.

Answer

In [7]:
```
%%sql

DROP SCHEMA IF EXISTS F23W4111Midterm;
CREATE SCHEMA F23W4111Midterm;
USE F23W4111Midterm;
```

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

Out[7]:
```
[]
```

In [8]:
```
worldcities = pd.read_csv('./worldcities.csv')
worldcities = worldcities[['city', 'city_ascii', 'lat', 'lng', 'country', 'iso2', 'iso3', 'id']]
worldcities.to_sql(
    'worldcities', schema="F23W4111Midterm",
    con=sql_engine,
    if_exists="replace",
    index=False
)
```

Out[8]:
```
41001
```

- Display data.

In [9]:
```
%sql select * from F23W4111Midterm.worldcities order by city limit 30;
```

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

Out[9]:

city	city_ascii	lat	lng	country	iso2	iso3	id
ʻAdrā	`Adra	33.6	36.515	Syria	SY	SYR	1760640037
ʻAjlūn	`Ajlun	32.3325	35.7517	Jordan	JO	JOR	1400775371
ʻAjmān	`Ajman	25.3994	55.4797	United Arab Emirates	AE	ARE	1784337875
ʻAkko	`Akko	32.9261	35.0839	Israel	IL	ISR	1376781950
ʻAlavīcheh	`Alavicheh	33.0528	51.0825	Iran	IR	IRN	1364605877
ʻAmrān	`Amran	15.6594	43.9439	Yemen	YE	YEM	1887433410
ʻĀmūdā	`Amuda	37.1042	40.93	Syria	SY	SYR	1760247135
ʻAnadān	`Anadan	36.2936	37.0444	Syria	SY	SYR	1760993442
ʻAssāl al Ward	`Assal al Ward	33.8658	36.4133	Syria	SY	SYR	1760181042
ʻAtaq	`Ataq	14.55	46.8	Yemen	YE	YEM	1887172893
ʻAyn ʻĪsá	`Ayn `Isa	36.3858	38.8472	Syria	SY	SYR	1760078370
ʻIbrī	`Ibri	23.2254	56.517	Oman	OM	OMN	1512077267
ʻAïn Abessa	'Ain Abessa	36.3	5.295	Algeria	DZ	DZA	1012074116
ʻAïn Arnat	'Ain Arnat	36.1833	5.3167	Algeria	DZ	DZA	1012453452
ʻAïn Azel	'Ain Azel	35.8433	5.5219	Algeria	DZ	DZA	1012746080
ʻAïn el Hammam	'Ain el Hammam	36.5647	4.3061	Algeria	DZ	DZA	1012595495
ʻAïn Leuh	'Ain Leuh	33.2833	-5.3833	Morocco	MA	MAR	1504668626
ʻAïn Roua	'Ain Roua	36.3344	5.1806	Algeria	DZ	DZA	1012529757
ʻAli Ben Sliman	'Ali Ben Sliman	31.9053	-7.2144	Morocco	MA	MAR	1504127885
ʻAyn Bni Mathar	'Ayn Bni Mathar	34.0889	-2.0247	Morocco	MA	MAR	1504845272
ʻs-Hertogenbosch	's-Hertogenbosch	51.6833	5.3167	Netherlands	NL	NLD	1528012333
A Coruña	A Coruna	43.3713	-8.4188	Spain	ES	ESP	1724417375
Aachen	Aachen	50.7762	6.0838	Germany	DE	DEU	1276805572
Aadorf	Aadorf	47.4939	8.8975	Switzerland	CH	CHE	1756022542

city	city_ascii	lat	lng	country	iso2	iso3	id
Aalborg	Aalborg	57.0337	9.9166	Denmark	DK	DNK	1208789278
Aalen	Aalen	48.8372	10.0936	Germany	DE	DEU	1276757787
Aalsmeer	Aalsmeer	52.2639	4.7625	Netherlands	NL	NLD	1528899853
Aalst	Aalst	50.9333	4.0333	Belgium	BE	BEL	1056695813
Aalten	Aalten	51.925	6.5808	Netherlands	NL	NLD	1528326020
Äänekoski	Aanekoski	62.6042	25.7264	Finland	FI	FIN	1246710490

Part 2: Modify World City Data

- Having multiple rows that repeat `country`, `iso2`, and `iso3` is poor design.

- Create two new tables:
  - `countries` that contains `country`, `iso2`, and `iso3`.
  - `cities` that contains only the remaining fields.
  - Pick either `iso2` or `iso3` to define a foreign key between the tables.

- Add primary keys, unique keys, select column data types, etc., to define a better schema for the two tables.

- Show your SQL statements for creating and modifying the tables below.

- **Note:** A small number of the ISO2 and ISO3 codes are incorrect and will prevent you from creating keys. You must correct this data and document your changes.

- Show your DDL below.

Answer

In [10]:
```
%%sql

USE F23W4111Midterm;

update worldcities
set iso3 = 'ARM'
where iso2 = 'AM';

update worldcities
set iso2 = 'NA'
where iso3 = 'NAM';

create table countries as select distinct country, iso2, iso3 from worldcities;
create table cities as select city, city ascii, lat, lng, iso3, id from worldcities;

alter table countries
    modify country varchar(50) not null;

alter table countries
    modify iso2 varchar(2) not null;

alter table countries
    modify iso3 varchar(3) not null;

alter table countries
    add constraint countries pk
        primary key (country);

alter table countries
    add constraint countries pk2
        unique (iso2, iso3);
```

```
 * mysql+pymysql://root:***@localhost
0 rows affected.
67 rows affected.
31 rows affected.
237 rows affected.
41001 rows affected.
237 rows affected.
237 rows affected.
237 rows affected.
0 rows affected.
0 rows affected.
```

Out[10]: []

In [11]:
```
%%sql

alter table cities
    modify city varchar(50) not null;

alter table cities
    modify city_ascii varchar(50) not null;

alter table cities
    modify lat decimal(9, 4) not null;

alter table cities
    modify lng decimal(9, 4) not null;

alter table cities
    modify iso3 varchar(3) not null;

alter table cities
    add constraint cities_pk
        primary key (id);

create index countries_iso3_index
    on countries (iso3);

alter table cities
    add constraint cities_countries_iso3_fk
        foreign key (iso3) references countries (iso3);
```

```
 * mysql+pymysql://root:***@localhost
41001 rows affected.
41001 rows affected.
41001 rows affected.
41001 rows affected.
41001 rows affected.
0 rows affected.
0 rows affected.
41001 rows affected.
[]
```

Out[11]:

Part 3: An Easy Question

- An interesting question. Is there a better SQL type for latitude and longitude than `DOUBLE`? If you think there is a better type, what would it be? (You do not need to perform any conversions)

We can combine them as `POINT` data type.

Part 4: Final Create Table Statements

- Use the DataGrip tool to generate final `CREATE TABLE` statements below. You do not need to execute the statements.

Answer

```
create table if not exists f23w4111midterm.countries
(
    country varchar(50) not null
        primary key,
    iso2    varchar(2)  not null,
    iso3    varchar(3)  not null,
    constraint countries_pk2
        unique (iso2, iso3)
);

create table if not exists f23w4111midterm.cities
(
```

```
    city        varchar(50)    not null,
    city_ascii  varchar(50)    not null,
    lat         decimal(9, 4)  not null,
    lng         decimal(9, 4)  not null,
    iso3        varchar(3)     not null,
    id          bigint         not null
            primary key,
    constraint cities_countries_iso3_fk
            foreign key (iso3) references f23w4111midterm.countries (iso3)
);

create index countries_iso3_index
    on f23w4111midterm.countries (iso3);
```

Part 5: Fixing People Table

Create a Copy People

- Create a table `F23W4111Midterm.people_modified` that has the same schema and data as `People`.

In [12]:
```
%%sql

USE F23W4111Midterm;

CREATE TABLE IF NOT EXISTS people_modified AS
    SELECT * FROM lahmans_hw1.People;
```

```
 * mysql+pymysql://root:***@localhost
0 rows affected.
20370 rows affected.
```

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

Fixing ```birthCountry```

- The query below indicates that some `birthCountry` entries in `people` do not map to a known country.

In [13]:
```
%%sql
```

```
select distinct birthCountry, count(*) as count
from people modified
where
    birthCountry not in (select country from countries)
group by birthCountry;
```

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

Out[13]:

birthCountry	count
USA	17601
D.R.	826
CAN	258
P.R.	276
Bahamas	7
South Korea	27
Czech Republic	7
V.I.	14
Viet Nam	1
At Sea	1

- My proposed corrections for `birthCountry` are:

birthCountry	ISO3	ISO2	Correct Country Name
Bahamas	BHS	BS	Bahamas, The
CAN	CAN	CA	Canada
Czech Republic	CZE	CZ	Czechia
South Korea	KOR	KR	Korea, South
USA	USA	US	United States
Viet Nam	VNM	VN	Vietnam

birthCountry	ISO3	ISO2	Correct Country Name
D.R.	DOM	DO	Dominican Republic
P.R.	PRI	PR	Puerto Rico
V.I.	USA	US	United States
At Sea	NULL	NULL	NULL

- Correct `people_modified`, making the following changes:
  1. Add a column `birthCountryISO3`
  2. Correct the entries for `birthCountry`
  3. Populate the values for `birthCountryISO3`
  4. Set up a foreign key relationship from `people_modified` to `countries`
  5. Drop the column `birthCountry`

Answer

- Show your SQL statements for altering the table below.
- Run your queries to show correctly modified table.

In [14]:
```
%%sql

alter table people_modified
    add birthCountryISO3 VARCHAR(3) null;

update people_modified
set birthCountry = 'Bahamas, The',
    birthCountryISO3 = 'BHS'
where birthCountry = 'Bahamas';

update people_modified
set birthCountry = 'Canada',
    birthCountryISO3 = 'CAN'
where birthCountry = 'CAN';

update people_modified
set birthCountry = 'Czechia',
```

```
    birthCountryISO3 = 'CZE'
where birthCountry = 'Czech Republic';

update people_modified
set birthCountry = 'Korea, South',
    birthCountryISO3 = 'KOR'
where birthCountry = 'South Korea';

update people_modified
set birthCountry = 'United States',
    birthCountryISO3 = 'USA'
where birthCountry = 'USA';

update people_modified
set birthCountry = 'Vietnam',
    birthCountryISO3 = 'VNM'
where birthCountry = 'Viet Nam';

update people_modified
set birthCountry = 'Dominican Republic',
    birthCountryISO3 = 'DOM'
where birthCountry = 'D.R.';

update people_modified
set birthCountry = 'Puerto Rico',
    birthCountryISO3 = 'PRI'
where birthCountry = 'P.R.';

update people_modified
set birthCountry = 'United States',
    birthCountryISO3 = 'USA'
where birthCountry = 'V.I.';

update people_modified
set birthCountry = NULL,
    birthCountryISO3 = NULL
where birthCountry = 'At Sea';

alter table people_modified
    add constraint people_modified_countries_iso3_fk
        foreign key (birthCountryISO3) references countries (iso3);

alter table people_modified
    drop column birthCountry;
```

```
 * mysql+pymysql://root:***@localhost
0 rows affected.
7 rows affected.
258 rows affected.
7 rows affected.
27 rows affected.
17601 rows affected.
1 rows affected.
826 rows affected.
276 rows affected.
14 rows affected.
1 rows affected.
20370 rows affected.
0 rows affected.
```

Out[14]: []