COMS W4111-002, Fall 21: Take Home Midterm

Aditya Kelvianto Sidharta, aks2266

Overview

Instructions

Due Date: Sunday, October 31, 2021 at 11:59pm

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.

Environment Setup

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

In [42]:
```
%load_ext sql
```

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

In [43]:
```
%sql mysql+pymysql://root:password@127.0.0.1/lahmansbaseballdb
```

```
(pymysql.err.OperationalError) (1045, "Access denied for user 'root
'@'localhost' (using password: YES)")
(Background on this error at: http://sqlalche.me/e/14/e3q8) (htt
p://sqlalche.me/e/14/e3q8))
Connection info needed in SQLAlchemy format, example:
               postgresql://username:password@hostname/dbname
               or an existing connection: dict_keys(['mysql+pymysq
l://root:***@127.0.0.1/lahmansbaseballdb', 'mysql+pymysql://root:**
```

In [44]:
```python
from sqlalchemy import create_engine
```

In [45]:
```python
sql_engine = create_engine("mysql+pymysql://root:password@127.0.0.1/
```

In [46]:
```python
import pandas as pd
```

Written Questions

W1

Provide a short (two or three sentence) definition/description of the following terms. Provide an example from the Lahman's Baseball DB for each.

Notes:

- Lahman's Baseball DB uses auto-increment ID columns for primary keys. If ignoring the ID column makes answering the question easier, you can assume that the column and current primary keys are not defined.
- 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

Ref :

- https://beginnersbook.com/2015/04/super-key-in-dbms/ (https://beginnersbook.com/2015/04/super-key-in-dbms/)
- https://blog.sqlauthority.com/2009/10/22/sql-server-difference-candidate-keys-primary-key-simple-words/ (https://blog.sqlauthority.com/2009/10/22/sql-server-difference-candidate-keys-primary-key-simple-words/)
- https://www.tutorialspoint.com/Alternate-Key-in-RDBMS (https://www.tutorialspoint.com/Alternate-Key-in-RDBMS)
- https://www.javatpoint.com/unique-key-in-sql (https://www.javatpoint.com/unique-key-in-sql)
- https://www.guru99.com/difference-between-primary-key-and-unique-key.html (https://www.guru99.com/difference-between-primary-key-and-unique-key.html)
- http://www.agiledata.org/essays/keys.html (http://www.agiledata.org/essays/keys.html)
- https://docs.microsoft.com/en-us/dynamicsax-2012/developer/table-keys-surrogate-alternate-replacement-primary-and-foreign (https://docs.microsoft.com/en-us/dynamicsax-2012/developer/table-keys-surrogate-alternate-replacement-primary-and-foreign)
- https://web.csulb.edu/colleges/coe/cecs/dbdesign/dbdesign.php?page=keys.php (https://web.csulb.edu/colleges/coe/cecs/dbdesign/dbdesign.php?page=keys.php)

1. Super Key: Sets of Attributes that can be used to identify rows in a table (batting - playerID, yearId, stint, teamID, G)
2. Candidate Key: Sets of Minimal Attributes that can be used to identify rows in a table (batting - playerID, yearID, stint, teamID)
3. Primary Key: Set of Minimal Attributes that can be used to identify rows in a table. One of the Candidate Keys is chosen as Primary Key (batting - playerID, yearID, stint, teamID
4. Alternate Key: Candidate Keys that are not chosen as the Primary Key (batting - playerID, yearID, stint, team_ID)
5. Unique Key: Column / Set of Columns that uniquely identifies every row in a table, able to have at most ONE null values in its row content. (batting - playerID, yearID, stint, team_ID)
6. Natural Key: Primary key that is part of the real world (batting - playerID, yearID, stint, teamID)
7. Surrogate Key: Primary key that is does not have any real meaning (batting - ID)
8. Substitute Key: Alternate key that system can display on forms. (batting - playerID, yearID, stint, team_ID)
9. Foreign Key: Column in a table that references data from another table (batting - playerID)
10. External Key: Surrogate or Substitute Keys that has been defined by external parties

W2

- 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

- https://softwareengineering.stackexchange.com/questions/8187/should-a-primary-key-be-immutable (https://softwareengineering.stackexchange.com/questions/8187/should-a-primary-key-be-immutable)
- https://stackoverflow.com/questions/12166523/immutable-sql-columns (https://stackoverflow.com/questions/12166523/immutable-sql-columns)

1. Immutable column refers to column or key refers to column values that are not allowed to be updated once it has been created
2. The reason is because if the primary key is used in external table / other applications, then modifying the value of the primary key will break the other applications
3. Assuming a table with primary_key, col1, col2, col3

```
grant insert, select, delete on the_table to the_user;
grant update (col1, col2, col3) on the_table to the_user;
```

W3

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

Answer

- https://www.c-sharpcorner.com/blogs/advantages-and-disadvantages-of-views-in-sql-server1 (https://www.c-sharpcorner.com/blogs/advantages-and-disadvantages-of-views-in-sql-server1)

1. View can be used for security reasons : i.e only allow specific data to be accessed by a user
2. View can abstract out complicated subqueries, to be used in various data manipulation processes
3. View can provide a consistent, unchanged image of the structure of the database, even though the underlying source tables has been modified

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?

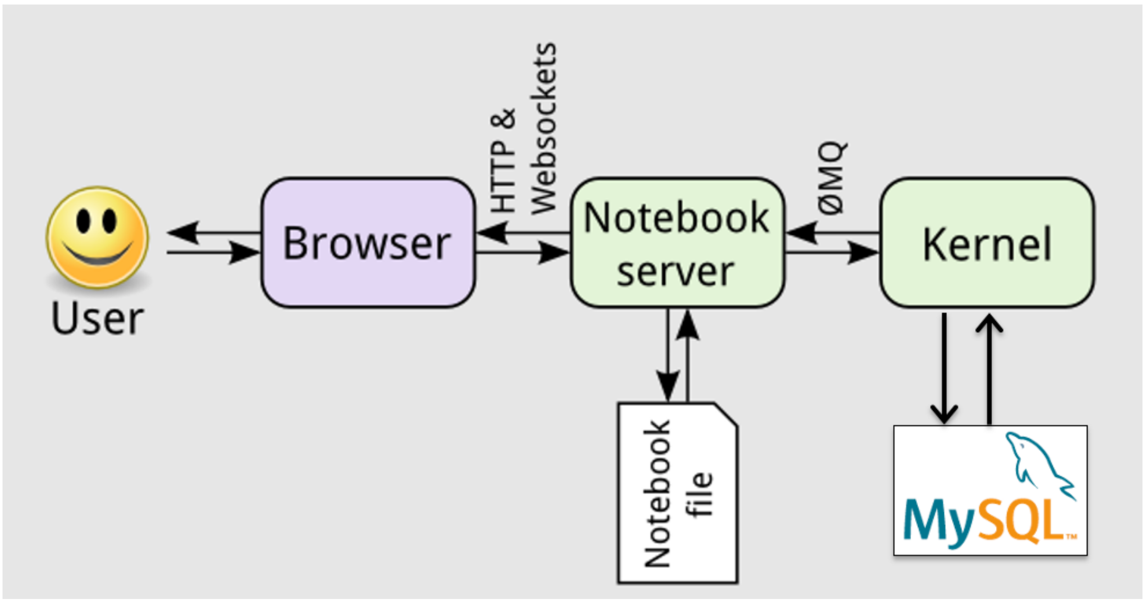
Answer

- https://stackoverflow.com/questions/1619834/what-is-the-difference-between-declarative-and-procedural-programming-paradigms (https://stackoverflow.com/questions/1619834/what-is-the-difference-between-declarative-and-procedural-programming-paradigms)

Procedural Language attempts to perform certain action by explicitly describing each steps that need to be taken by the algorithm. However, declarative language achieve the same action by describing the end state that wants to be achieved. The advantage of Declarative programming is that the underlying mechanism of step-by-step actions is abstracted away

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

It is a three tier architecture, as the client (Browser) does not interact directly with the database (MySQL), but instead communicate through an application server, which in this case is a notebook server

W6

What is the difference between the database schema and the database instance? Use the following conventions of the relational model for documenting a relation schema to answer the question if:

1. $country\_code$ is the country code for the phone number, e.g. +1
2. $main\_number$ is the main number, e.g. 212-555-1212.
3. $extension$ is the extension, e.g. x1002
4. $phone\_type$ is an enum, e.g. *work, home, mobile, others.*

$$PhoneNumber(\underline{country\_code}), \underline{main\_number}, \underline{extension}, phone\_type)$$

Answer

- https://pediaa.com/what-is-the-difference-between-schema-and-instance/ (https://pediaa.com/what-is-the-difference-between-schema-and-instance/)

Database schema referes to the structural view in the database. In other words, it gives information about the set of columns it has, the type for each of the columns, as well as the metadata surrounding it. In this case, the schema refers to the types of the country code as well as which columns are the primary keys. In contrast, databse instance refers to the actual data that is stored in the database.

W7

Briefly explain the differences between:

- Database stored procedure
- Database function
- Database trigger

Answer

- https://stackoverflow.com/questions/1179758/function-vs-stored-procedure-in-sql-server (https://stackoverflow.com/questions/1179758/function-vs-stored-procedure-in-sql-server)
- https://www.geeksforgeeks.org/sql-trigger-student-database/ (https://www.geeksforgeeks.org/sql-trigger-student-database/)
- Database stored procedure: SQL code that is saved so that it can be used multiple times
- Database function: Function that can be used to perform mapping between input and output. It differs from the database stored procedure as it cannot perform changes to the SQL server
- Database trigger is a stored procedure that are automatically called whenever there is a specified special event occuring in the databsae. For example, a specific database trigger might be called whenever certain columns are inserted / updated.

W8

Briefly explain:

- Natural join
- Equi-join
- Theta join

- Self-join

Answer

- https://stackoverflow.com/questions/7870155/difference-between-a-theta-join-equijoin-and-natural-join/7870216 (https://stackoverflow.com/questions/7870155/difference-between-a-theta-join-equijoin-and-natural-join/7870216)
- Theta Join is a join between two tables that allows for various comparison relationship (EQ, GT, IT, etc)
- Equi-Join is a Theta-join that uses equality operator
- Natural Join is an equi-join on columns that have the same name in the relationship. This will also remove duplicates on the resulting column
- Self-join is a join with a replica of the same table, conditioned on certain clause.

W9

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

Answer

Unique key constraint allows the column to have at most one NULL values, as the only constraint is for each individual values in the column to be unique. Primary key constraint is equal to having NOT NULL and UNIQUE constraint, and thus it could not have any null values at all. In addition, you could not have multiple primary key constraints in a single table

W10

Briefly explan *domain constraint* and give an example.

Answer

- https://www.geeksforgeeks.org/domain-constraints-in-dbms/ (https://www.geeksforgeeks.org/domain-constraints-in-dbms/)

Domain constraints are user-defined data-type to make sure that the input data is appropriate. It consists of the data-type of the column and the constraints that we choose

Example:

```
create domain age_value int
constraint age_test
```

Entity Relationship Model

<u>Question</u>

- 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, or a similar tool. You should use Crow's Foot notation and conventions we have used in lectures.

- 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, e.g. Chair.
- Roles are a small set of possible values.
- Roles change over time. The data model must support the ability to handle current roles and previous roles.
- A Faculty can have a role in a department at most once.

- 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 row.
  - The data model must support the ability to handle roles changing over time and retaining 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.

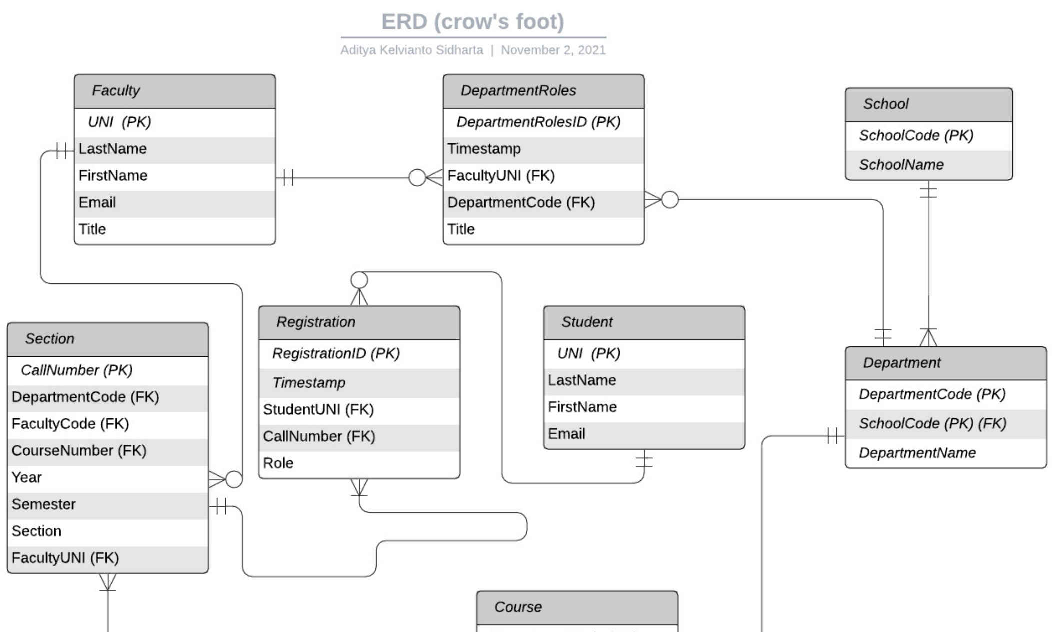
Notes:

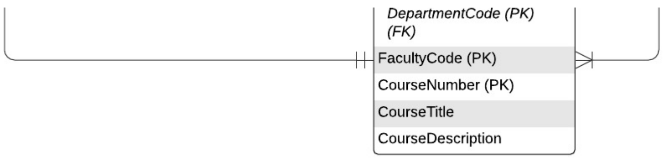
1. There is no single correct answer to this question. You will have to make some design decisions and assumptions. You should document your decisions and assumptions.
2. You do not have to worry about **isA** relationships.
3. You do not have to document or worry about attribute types.
4. The ER diagram must be implementable in the relational/SQL model.

Answer

Assumptions, Decsions and Notes:

1. Each Student may or may not take classes, it can take more than 1 classes
2. Each Faculty member might take multiple roles. Its possible that a certain role in a department is empty





Relational Algebra

R1

Use the [RelaX Calculator and the Silberschatz - UniversityDB (https://dbis-uibk.github.io/relax/calc/gist/4f7866c17624ca9dfa85ed2482078be8/relax-silberschatz-english.txt/0)](https://dbis-uibk.github.io/relax/calc/gist/4f7866c17624ca9dfa85ed2482078be8/relax-silberschatz-english.txt/0) for this question.

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

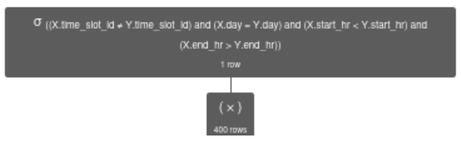
1. They are not the same time slot, i.e. do not have the same $time\_slot\_id$.
2. They have at least one lecture on *the same day,* the start hour for *X* is before the start hour for *Y*, and the end hour for *X* is after the start hour for *Y.*
3. To make the question easier, you do not need to consider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

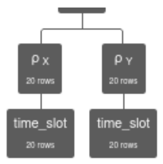
Write the relational algebra expression that identifies obviously overlapping time slots, and only lists overlapping pairs of time slots once.

Your output must match the answer below.

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
'H'	'W'	10	0	12	30	'C'	'W'	11	0	11	50

Answer


σ ((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.end_hr))
1 row
(×)
400 rows



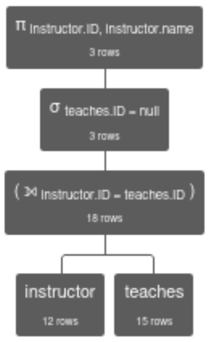
$\sigma_{((X.time\_slot\_id \neq Y.time\_slot\_id) \text{ and } (X.day = Y.day) \text{ and } (X.start\_hr < Y.start\_hr) \text{ and } (X.end\_hr > Y.end\_hr))} ((\rho_X (time\_slot)) \times (\rho_Y (time\_slot)))$

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
'H'	'W'	10	0	12	30	'C'	'W'	11	0	11	50

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



$\pi_{instructor.ID,\ instructor.name}\ \sigma_{teaches.ID = null} (instructor ⟕_{instructor.ID = teaches.ID} teaches)$

instructor.ID	instructor.name
33456	'Gold'
58583	'Califieri'
76543	'Singh'

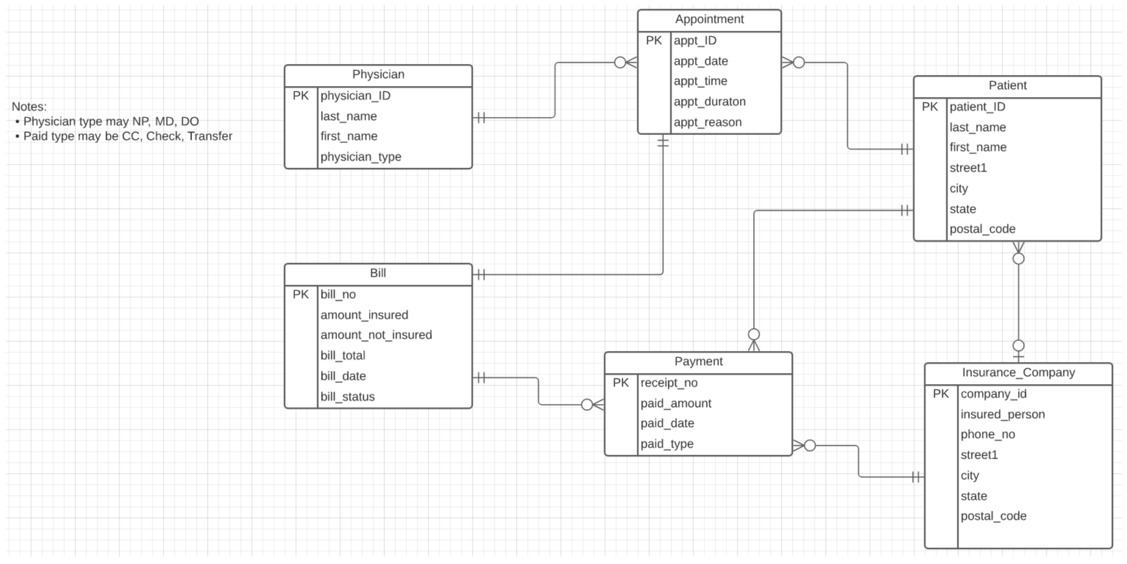
SQL Schema and DDL

Objective

- You have a logical datamodel ER-diagram (see below).
- You need to use DDL to define a schema that realizes the model.
- 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.**

ER Diagram



ER Diagram

Answer

Design Decisions, Notes, etc.

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.

```
CREATE DATABASE `midterm`;

USE midterm;

CREATE TABLE `Physician` (
  `physician_ID` varchar(256) NOT NULL,
  `last_name` varchar(256),
  `first_name` varchar(256),
  `physician_type` enum('NP','MD','DO'),
  PRIMARY KEY (`physician_ID`)
);

CREATE TABLE `Patient` (
    `patient_ID` varchar(256) NOT NULL,
    `last_name` varchar(256),
    `first_name` varchar(256),
    `street1` varchar(256),
    `city` varchar(256),
    `state` varchar(2),
    `postal_code` int,
    PRIMARY KEY (`patient_ID`)
);

CREATE TABLE `Insurance_Company` (
    `company_id` varchar(256) NOT NULL,
    `insured_person` varchar(256) NOT NULL,
    `phone_no` varchar(256),
    `street1` varchar(256),
    `city` varchar(256),
    `state` varchar(2),
    `postal_code` int,
    PRIMARY KEY (`company_id`),
    FOREIGN KEY (`insured_person`) REFERENCES `Patient` (`pa
tient_ID`)
);

CREATE TABLE `Payment` (
    `receipt_no` varchar(256) NOT NULL,
    `paid_amount` double,
    `paid_date` date,
    `paid_type` ENUM('CC', 'Check', 'Transfer'),
    `bill_no` varchar(256) NOT NULL,
    PRIMARY KEY (`receipt_no`),
    FOREIGN KEY (`bill_no`) REFERENCES `Bill` (`bill_no`)
);

CREATE TABLE `Bill` (
    `bill_no` varchar(256) NOT NULL,
    `amount_insured` float,
    `amount_not_insured` float,
    `bill_total` float,
    `bill_date` date,
    `bill_status` varchar(256),
    PRIMARY KEY (`bill_no`)
);

CREATE TABLE `Appointment` (
  `appt_ID` varchar(256) NOT NULL,
  `appt_date` date,
  `appt_time` timestamp,
  `appt_duration` int,
  `appt_reason` varchar(256),
  `physician_ID` varchar(256) NOT NULL,
  `patient ID` varchar(256) NOT NULL.
```

Complex SQL

Birth Countries and Death Countries

Question

- In `lahmansbaseballdb.people` there is information about people's `birthCountry` and `deathCountry`.
- There are countries in which at least 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 in which there with births but no deaths.

Answer

In [50]:
```
%%sql

SELECT birthCountry, no_of_births
FROM (
    SELECT birthCountry, COUNT(DISTINCT playerID) as no_of_births
    FROM people
    GROUP BY birthCountry) as lefttable
LEFT JOIN (SELECT DISTINCT deathCountry
    FROM people
    WHERE deathYear IS NOT NULL) as righttable
ON birthCountry = deathCountry
WHERE deathCountry IS null AND birthCountry NOT LIKE "%None%"
ORDER BY no_of_births DESC
```

```
   mysql+pymysql://root:***@127.0.0.1/F21W4111Midterm
 * mysql+pymysql://root:***@127.0.0.1/lahmansbaseballdb
35 rows affected.
```

Out[50]:

birthCountry	no_of_births
Germany	45
Colombia	24
South Korea	23
Curacao	15
Nicaragua	15
Russia	9
Italy	7
Czech Republic	6
Aruba	5
Brazil	5
Poland	5
Jamaica	4
Sweden	4
Spain	4
Norway	3
Honduras	2
Guam	2
South Africa	2
Saudi Arabia	2
Portugal	1
Viet Nam	1
Switzerland	1
Belgium	1
Belize	1
Denmark	1
Slovakia	1
Singapore	1
Finland	1
Hong Kong	1
Greece	1
Peru	1
Lithuania	1
Latvia	1
Afghanistan	1
Indonesia	1

Best Baseball Players

Question

- This question uses `lahmansbaseballdb.batting`, `lahmansbaseballdb.pitching` and `lahmansbaseballdb.people`.
- There query computes performance metrics:
  - Batting:
    - On-base percentage: OBP is (sum(h) + sum(BB))/(sum(ab) + sum(BB))
    - Slugging percentage: SLG is

      ```
      (
      (sum(h) - sum(`1b`) - sum(`2b`) - sum(`3b`) - sum
      (hr)) +
      2*sum(`2b`) + 3*sum(`3b`) + 4*hr
      )/sum(ab)
      ```

    - 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 >= 1000`.
    - OPS: Career `OPS >= .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.

Answer

In [4]:
```
%%sql


WITH raw AS (
SELECT playerID,
        nameLast,
       nameFirst,
        debut,
       finalGame,
       sum(ab) AS ab_sum,
```

```
       sum(w) + sum(l) AS game_sum,
       (sum(batting.h) + sum(batting.BB))/(sum(ab) + sum(batting.BB)
       ((sum(batting.h) - sum(`2b`) - sum(`3b`) - sum(batting.hr)) +
       ((sum(batting.h) + sum(batting.BB))/(sum(ab) + sum(batting.BB
       sum(w) AS total_wins,
       sum(l) AS total_losses,
       sum(w)/(sum(w) + sum(l)) AS win_percent
FROM people
lEFT JOIN batting USING (playerID)
LEFT JOIN pitching USING (playerID)
GROUP BY 1,2,3,4,5
HAVING (ab_sum > 1000 AND ops > 1.0) OR (game_sum >= 200 AND ((win_pe
)

SELECT playerID                                                AS play
       nameLast                                                AS name
       nameFirst                                               AS name
       IF((ab_sum > 1000 AND ops > 1.0), 'Batter', 'Pitcher') AS grea
       debut                                                   AS debu
       finalGame                                               AS fina
       playerID                                                AS play
       IF((ab_sum > 1000 AND ops > 1.0), obp, NULL)            AS obp,
       IF((ab_sum > 1000 AND ops > 1.0), slg, NULL)            AS slg,
       IF((ab_sum > 1000 AND ops > 1.0), ops, NULL)            AS ops,
       IF((ab_sum > 1000 AND ops > 1.0), NULL, total_wins)     AS tota
       IF((ab_sum > 1000 AND ops > 1.0), NULL, total_losses)   AS tota
       IF((ab_sum > 1000 AND ops > 1.0), NULL, win_percent)    AS win_
FROM raw
LIMIT 20
```

```
 * mysql+pymysql://root:***@127.0.0.1/lahmansbaseballdb
20 rows affected.
```

Out[4]:

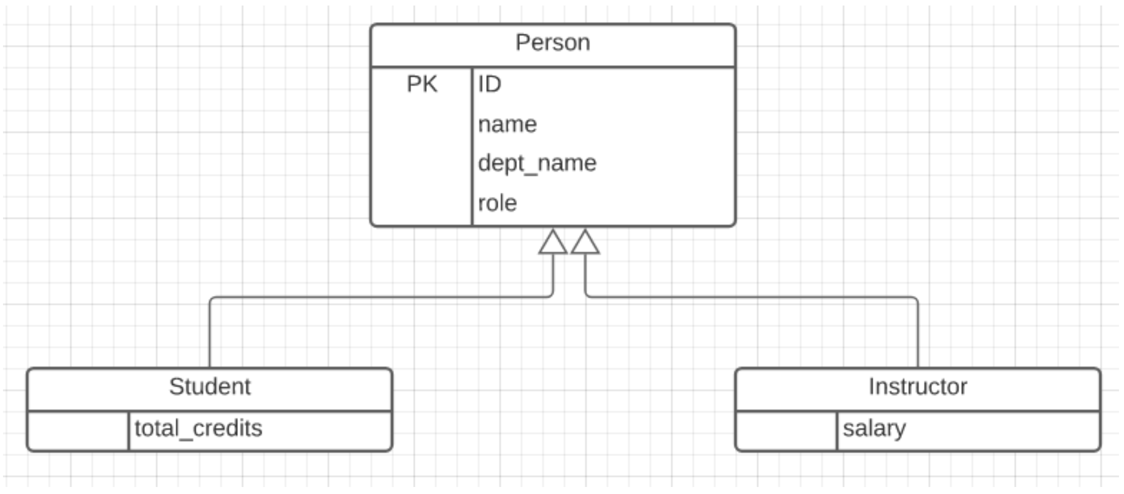
playerid	nameLast	nameFirst	great_because	debut_date	finalgame_date	playerid_1	o
aasedo01	Aase	Don	Pitcher	1977-07-26	1990-10-03	aasedo01	No
abbotgl01	Abbott	Glenn	Pitcher	1973-07-29	1984-08-08	abbotgl01	No
abbotji01	Abbott	Jim	Pitcher	1989-04-08	1999-07-21	abbotji01	No
abbotpa01	Abbott	Paul	Pitcher	1990-08-21	2004-08-07	abbotpa01	No
abernte02	Abernathy	Ted	Pitcher	1955-04-13	1972-09-30	abernte02	No
ackerji01	Acker	Jim	Pitcher	1983-04-07	1992-06-14	ackerji01	No
adamsba01	Adams	Babe	Pitcher	1906-04-18	1926-08-11	adamsba01	No
adamste01	Adams	Terry	Pitcher	1995-08-10	2005-05-23	adamste01	No
affelje01	Affeldt	Jeremy	Pitcher	2002-04-06	2015-10-04	affelje01	No
agostju01	Agosto	Juan	Pitcher	1981-09-07	1993-06-19	agostju01	No
aguilri01	Aguilera	Rick	Pitcher	1985-06-12	2000-09-06	aguilri01	No
aguirha01	Aguirre	Hank	Pitcher	1955-09-10	1970-06-24	aguirha01	No
akerja01	Aker	Jack	Pitcher	1964-05-03	1974-09-27	akerja01	No
alberma01	Albers	Matt	Pitcher	2006-07-25	2019-09-28	alberma01	No
aldrivi01	Aldridge	Vic	Pitcher	1917-04-15	1928-08-29	aldrivi01	No

alexado01	Alexander	Doyle	Pitcher	1971-06-26	1989-09-27	alexado01	Nc
alexape01	Alexander	Pete	Pitcher	1911-04-15	1930-05-28	alexape01	Nc
alfonan01	Alfonseca	Antonio	Pitcher	1997-06-17	2007-09-23	alfonan01	Nc
allenfr01	Allen	Frank	Pitcher	1912-04-24	1917-09-19	allenfr01	Nc

Putting Together DDL, DML, Functions, Triggers

Question

- Use the database that comes with the textbook for this question.
    - Create a new database `db_book_midterm.`
    - Copy the data and table definitions for `Student` and `Instructor`
    - You may have to remove some constraints from the copied data/definition to make it work.
- Base tables:
    - `Student` has the form `Student(ID, name, dept_name, total_cred)` .
- `Instructor` has the form `Instructor(ID, name, dept_name, salary)` .
- There is a *logical* base type `Person.` The logical `isA` model is:



Logical Model

- `role` is either `S` or `F` based on whether the `Person` is a `Student` or `Instructor` .
- Implement a *two table* solution to realize `Person.` This means define `Person` as a view.
- You do not need to worry about generating the primary key ID. Your implementation MUST, however, enforce the rule that the ID is immutable.
- You must also create a *stored procedure* `create_person` . The template for the implementation is:

```
CREATE DEFINER=`root`@`localhost` PROCEDURE `create_person`(
    in person_name varchar(32),
    in dept_name varchar(32),
    in total_cred decimal(3,0),
    in salary decimal(8,2),
    out ID varchar(5)
    )
BEGIN

    declare bad_person boolean;
    declare new_id varchar(12);

    set bad_person = false;
    set new_id = '00000';



    /*
        The logic of the stored procedure is the following:
            - The request is invalid and sets bad_person to
true if:
                - Any of person_name, dept_name is NULL.
                - Either total_cred is NULL and salary is NO
T NULL, or salary is NULL and total_cred is NULL.
            - The procedure must compute a new, unique ID. T
he approach is to find the manimum
                ID value over Student and Instructor. Add 1
to the value to produce the new, unique ID.
            - The procedure then adds the information to Stu
dent or Instructor based on whether
                total_cred is NULL or salary is NULL.
    */

    if person_name IS NULL or dept_name IS NULL OR total_cre
d IS NULL then
        set bad_person = true
     end if;

    if bad_person is true then
        SIGNAL SQLSTATE '50001'
            SET MESSAGE_TEXT = 'Invalid Person information i
nput';
    end if;

    /* NOTE: ID is the out parameter. You must set ID to the
new, unique ID */
```

Answer

Create view statement

```
CREATE VIEW person AS
(SELECT ID, name, dept_name, 'S' AS role
 FROM student
UNION ALL
 SELECT ID, name, dept_name, 'F' AS role
 FROM instructor
)
```

Create procedure statement

```
CREATE DEFINER=`root`@`localhost` PROCEDURE `create_person`(
    in person_name varchar(32),
    in dept_name varchar(32),
    in total_cred decimal(3,0),
    in salary decimal(8,2),
    out ID varchar(5)
    )
BEGIN
```

Tests

Run the SQL in the following cells to test your solution.

- Test 1: Show the view.

In [56]:
```
%sql select * from Person;
```

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

Out[56]:

ID	name	dept_name	I
10101	Srinivasan	Comp. Sci.	I
12121	Wu	Finance	I
15151	Mozart	Music	I
22222	Einstein	Physics	I
32343	El Said	History	I
33456	Gold	Physics	I
45565	Katz	Comp. Sci.	I
58583	Califieri	History	I
76543	Singh	Finance	I
76766	Crick	Biology	I
83821	Brandt	Comp. Sci.	I
98345	Kim	Elec. Eng.	I
00128	Zhang	Comp. Sci.	S
12345	Shankar	Comp. Sci.	S
19991	Brandt	History	S
23121	Chavez	Finance	S
44553	Peltier	Physics	S
45678	Levy	Physics	S
54321	Williams	Comp. Sci.	S
55739	Sanchez	Music	S
70557	Snow	Physics	S

76543	Brown	Comp. Sci.	S
76653	Aoi	Elec. Eng.	S
98765	Bourikas	Elec. Eng.	S

Create an Instructor and Student

In [57]:
```
%sql CALL create_person('Ferguson', 'Comp. Sci.',  NULL, 30000.00, @p
```

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

Out[57]:
```
[]
```

In [59]:
```
%sql SELECT @prof_id;
```

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

Out[59]:

@prof_id
98989

In [63]:
```
%sql CALL create_person('Ferguson', 'Comp. Sci.',  100.0, NULL, @stud
```

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

Out[63]:
```
[]
```

In [64]:
```
%sql SELECT @student_id;
```

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

Out[64]:

@student_id
98990

- Try an error

In [65]:
```
try:
    %sql CALL create_person('Ferguson', 'Comp. Sci.',  100.0, 30000,
except Exception as e:
    print(e)
res = %sql select @student_id;
```

```
 * mysql+pymysql://dbuser:***@localhost
(pymysql.err.OperationalError) (1644, 'Invalid Person information i
nput')
[SQL: CALL create_person('Ferguson', 'Comp. Sci.',  100.0, 30000, @
student_id);]
(Background on this error at: http://sqlalche.me/e/e3q8) (http://sq
lalche.me/e/e3q8))
 * mysql+pymysql://dbuser:***@localhost
1 rows affected.
```

Include DDL that Show Enforcing Immutable ID

Write a test that shows you implemented immutable IDs

Data and Schema Cleanup

Part 1 — Countries and Cities

- There is a file `worldcities` 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`
    - `iso2`
    - `id`
  - Write the data to the table `worldcities` in the schema `F21W4111Midterm.`
- Use the SQL after the code cell to display part of your new table.

Answer

In [10]:
```
df = pd.read_csv("/home/adityasidharta/columbia/database/midterm/worl
```

In [12]:
```
df = df[['city', 'city_ascii', 'lat', 'lng', 'country', 'iso2', 'iso3
```

In [16]:
```
df.to_sql('worldcities', con=sql_engine)
```

- Display data.

In [20]:
```
%sql mysql+pymysql://root:password@127.0.0.1/F21W4111Midterm
```

In [22]:
```
%sql select * from F21W4111Midterm.worldcities order by city limit 30
```

```
 * mysql+pymysql://root:***@127.0.0.1/F21W4111Midterm
   mysql+pymysql://root:***@127.0.0.1/lahmansbaseballdb
30 rows affected.
```

Out[22]:

index	city	city_ascii	lat	lng	country	iso2	iso3	id
19249	‘Adrā	`Adra	33.6	36.515	Syria	SY	SYR	1760640037

9815	‘Ajlūn	`Ajlun	32.3325	35.7517	Jordan	JO	JOR	1400775371
2469	‘Ajmān	`Ajman	25.3994	55.4797	United Arab Emirates	AE	ARE	1784337875
13032	‘Akko	`Akko	32.9261	35.0839	Israel	IL	ISR	1376781950
23726	‘Alavīcheh	`Alavicheh	33.0528	51.0825	Iran	IR	IRN	1364605877
9612	‘Amrān	`Amran	15.6594	43.9439	Yemen	YE	YEM	1887433410
13142	‘Āmūdā	`Amuda	37.1042	40.93	Syria	SY	SYR	1760247135
26026	‘Anadān	`Anadan	36.2936	37.0444	Syria	SY	SYR	1760993442
37808	‘Assāl al Ward	`Assal al Ward	33.8658	36.4133	Syria	SY	SYR	1760181042
6512	‘Ataq	`Ataq	14.55	46.8	Yemen	YE	YEM	1887172893
35329	‘Ayn ‘Īsá	`Ayn `Isa	36.3858	38.8472	Syria	SY	SYR	1760078370
4900	‘Ibrī	`Ibri	23.2254	56.517	Oman	OM	OMN	1512077267
22092	’Aïn Abessa	'Ain Abessa	36.3	5.295	Algeria	DZ	DZA	1012074116
13597	’Aïn Arnat	'Ain Arnat	36.1833	5.3167	Algeria	DZ	DZA	1012453452
13002	’Aïn Azel	'Ain Azel	35.8433	5.5219	Algeria	DZ	DZA	1012746080
19677	’Aïn el Hammam	'Ain el Hammam	36.5647	4.3061	Algeria	DZ	DZA	1012595495
28994	’Aïn Leuh	'Ain Leuh	33.2833	-5.3833	Morocco	MA	MAR	1504668626
26882	’Aïn Roua	'Ain Roua	36.3344	5.1806	Algeria	DZ	DZA	1012529757
20104	’Ali Ben Sliman	'Ali Ben Sliman	31.9053	-7.2144	Morocco	MA	MAR	1504127885
22485	’Ayn Bni Mathar	'Ayn Bni Mathar	34.0889	-2.0247	Morocco	MA	MAR	1504845272
3842	’s-Hertogenbosch	's-Hertogenbosch	51.6833	5.3167	Netherlands	NL	NLD	1528012333
1746	A Coruña	A Coruna	43.3713	-8.4188	Spain	ES	ESP	1724417375
2400	Aachen	Aachen	50.7762	6.0838	Germany	DE	DEU	1276805572
29829	Aadorf	Aadorf	47.4939	8.8975	Switzerland	CH	CHE	1756022542
4077	Aalborg	Aalborg	57.0337	9.9166	Denmark	DK	DNK	1208789278
11447	Aalen	Aalen	48.8372	10.0936	Germany	DE	DEU	1276757787
15609	Aalsmeer	Aalsmeer	52.2639	4.7625	Netherlands	NL	NLD	1528899853
10664	Aalst	Aalst	50.9333	4.0333	Belgium	BE	BEL	1056695813
17071	Aalten	Aalten	51.925	6.5808	Netherlands	NL	NLD	1528326020
20206	Äänekoski	Aanekoski	62.6042	25.7264	Finland	FI	FIN	1246710490

P2 – Modify World City Data

- Having multiple rows that repeat `country, iso2` and `iso3` is a 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 you SQL statements for creating and modifying the tables below.
- **Note:** A small number of the ISO2 and ISO3 codes are incorrect and will prevent creating keys. You must correct this data.
- Show you DDL below.

Answer

In [28]:
```
df[['country', 'iso2', 'iso3']].drop_duplicates().to_sql('countries',
```

In [31]:
```
df[['city','city_ascii','lat','lng','iso3','id']].drop_duplicates('ci
```

In [35]:
```
%%sql

ALTER TABLE cities
MODIFY COLUMN city varchar(256);

ALTER TABLE cities
MODIFY COLUMN iso3 varchar(256);

ALTER TABLE countries
MODIFY COLUMN country varchar(256);

ALTER TABLE countries
MODIFY COLUMN iso3 varchar(256);

ALTER TABLE countries
ADD PRIMARY KEY(`country`);

ALTER TABLE cities
ADD PRIMARY KEY(`city`, `iso3`);

ALTER TABLE cities
ADD FOREIGN KEY(`iso3`) REFERENCES countries (`iso3`);
```

```
 * mysql+pymysql://dbuser:***@localhost
20 rows affected.
```

Out[35]:

id	city	city_ascii	lat	lng	iso3	new_lat	new_lng
1004003059	Kandahār	Kandahar	31.6078	65.7053	AFG	31.6078	65.7053
1004016690	Qalāt	Qalat	32.1061	66.9069	AFG	32.1061	66.9069
1004047427	Sar-e Pul	Sar-e Pul	36.2214	65.9278	AFG	36.2214	65.9278
1004123527	Pul-e Khumrī	Pul-e Khumri	35.95	68.7	AFG	35.9500	68.7000
1004151943	Maḩmūd-e Rāqī	Mahmud-e Raqi	35.0167	69.3333	AFG	35.0167	69.3333
1004167490	Ghaznī	Ghazni	33.5492	68.4233	AFG	33.5492	68.4233
1004180853	Pul-e ‘Alam	Pul-e `Alam	33.9953	69.0227	AFG	33.9953	69.0227
1004227517	Kunduz	Kunduz	36.728	68.8725	AFG	36.7280	68.8725

1004237782	Herāt	Herat	34.3738	62.1792	AFG	34.3738	62.1792

P3 — 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)

Answer

BigInteger

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 F21W4111Midterm.cities
(
    `index` bigint null,
    city varchar(256) null,
    city_ascii text null,
    lat double null,
    lng double null,
    iso3 varchar(256) null,
    id bigint null
);

create index ix_cities_index
    on F21W4111Midterm.cities (`index`);
```

Fixing People Table

Create a Copy People

- Create a table `F21Midterm.people_modified` that has the same schema and data as `lahmansbaseballdb.people`.
- SQL:

Answer

In [38]:
```
%%sql

create table F21W4111Midterm.people_modified
(
    playerID       varchar(9)   not null
        primary key,
    birthYear      int          null,
    birthMonth     int          null,
    birthDay       int          null,
    birthCountry   varchar(255) null,
    birthState     varchar(255) null,
    birthCity      varchar(255) null,
    deathYear      int          null,
    deathMonth     int          null,
    deathDay       int          null,
    deathCountry   varchar(255) null,
    deathState     varchar(255) null,
    deathCity      varchar(255) null,
    nameFirst      varchar(255) null,
    nameLast       varchar(255) null,
    nameGiven      varchar(255) null,
    weight         int          null,
    height         int          null,
    bats           varchar(255) null,
    throws         varchar(255) null,
    debut          varchar(255) null,
    finalGame      varchar(255) null,
    retroID        varchar(255) null,
    bbrefID        varchar(255) null,
    birth_date     date         null,
    debut_date     date         null,
    finalgame_date date         null,
    death_date     date         null
);



INSERT INTO F21W4111Midterm.people_modified SELECT * from lahmansbase
```

```
 * mysql+pymysql://root:***@127.0.0.1/F21W4111Midterm
   mysql+pymysql://root:***@127.0.0.1/lahmansbaseballdb
0 rows affected.
19878 rows affected.
```

Out[38]:
```
[]
```

Fixing `birthCountry`

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

In [39]:
```
%%sql


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

```
 * mysql+pymysql://root:***@127.0.0.1/F21W4111Midterm
   mysql+pymysql://root:***@127.0.0.1/lahmansbaseballdb
10 rows affected.
```

Out[39]:

birthCountry	count
USA	17254
D.R.	761
CAN	255
P.R.	268
Bahamas	6
South Korea	23
Czech Republic	6
V.I.	14
Viet Nam	1
At Sea	1

- My proposed corrections for `birthCountry` are:

birthCountry	ISO3	ISO2	Correct Country Name

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
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. [illegible]

Answer

- Show your SQL statements for altering the table below.

Type *Markdown* and LaTeX: α^2

- Run a couple of queries to show correctly modified table.

```
ALTER TABLE people_modified
ADD COLUMN birthCountryISO3 varchar(256);

UPDATE people_modified
inner join countries on people_modified.birthCountry = count
ries.country
SET
    birthCountryISO3 = iso3;

ALTER TABLE people_modified
ADD FOREIGN KEY (`birthCountryISO3`) REFERENCES countries (`
iso3`);

ALTER TABLE people_modified
DROP COLUMN birthCountry;
```