COMS W4111-002, Fall 21: Take Home Midterm

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 [2]:

```
%load_ext sql
```

In [4]:

```
%sql mysql+pymysql://dbuser:dbuserdbuser@localhost
```

Out[4]:

```
'Connected: dbuser@None'
```

In [48]:

```
from sqlalchemy import create_engine
```

In [49]:

```
sql_engine = create_engine("mysql+pymysql://dbuser:dbuserdbuser@localhost")
```

In [50]:

```
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.

- Super Key
  It can be a single key or a group of keys than can help to uniqly identify tuples in a table.
- Candidate Key
  It is a subset of of super key which does not have any unnecessary information for uniqly identifying tuples in the data. eg. [playerID, yearID, stint] from the batting table
- Primary Key
  Out of the candidate keys, we can select one key to determine a unique row, which is called the Primary key. eg. playedrID in the people table. Values in the P.K are non-null and unique.
- Alternate Key
  Keys other than the Primary key which were not selected from the candidate keys can be alternate keys. eg: (yearID,teamID,lgID, playerID) in salaries
- Unique Key
  It's a set of columns that can uniqly identify a record, however different from the Primary key in terms of that it can take one NULL value. eg: [playerID, yearID, stint] from batting table
- Natural Key
  These are set of attributes that can identify a record and consist of a business meaning which can utilised outside of the database. eg. [playerID, yearID, stint]
- Surrogate Key
  It is simmilar to Natural key however, these set of attributes do not contain any business meaning, they only identify a unique record. eg: ID attributes in different tables
- Substitute Key
  It is a single attribute that has some descriptive value for the data such as an abbreviation.
- Foreign Key
  This key in a table (child) enables it to reference another table (parent) through the primary key. Values in a foreign key could only be one of values present in the Primary key or NULL. eg: teamID in batting table
- External Key:

Answer

W2

- Define the concept of *immutable* column and key.

An immutable column and key would not be modified once it is created. A new record or a column needs to be inserted if there is a modification to the data

- Why do some sources recommend that a primary key should be immutable?

Valeus in Primary key can be used across tables as foreign keys usually reference this data. Any change would need to be propogated throughout the database, hence it is advised to make the P.K. immutable. This prevents the chance of orphan records.

- How would to implement immutability for a primary key in a table? We can add the constrain ON UPDATE RESTRICT to the Primary key to make it immutable.

Answer

W3

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

- Views can help to control the degree of exposure of the underlying tables to the outer world. A user may have permission to query the view, while denied access to the rest of the base table.
- They can join multiple tables into a single vitual table and simplify working with them.
- Views do not take up much space to store since database contains only the definition of a view, not a copy of all the data

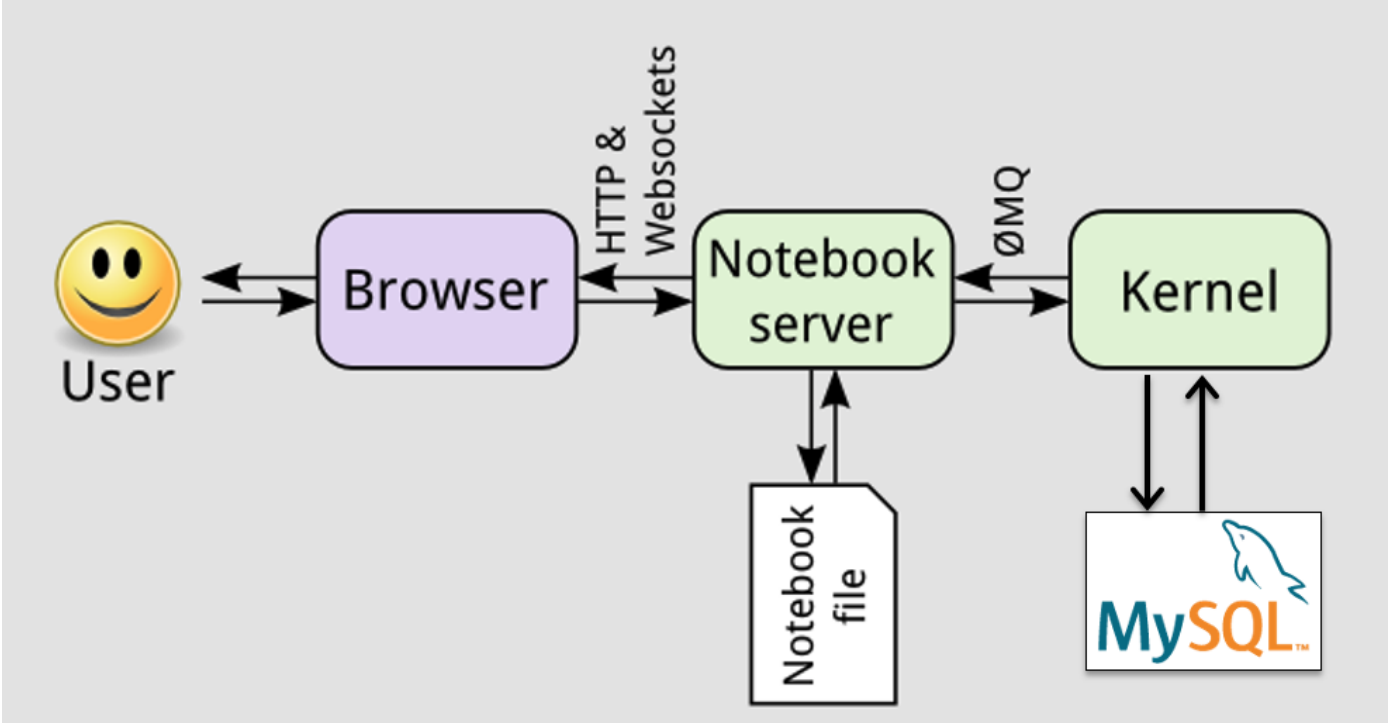
W4

`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?`

In a procedural language you define each step in detail, whereas in a declarative language you provide a command and get your result via a black-box method. In the declarative style you do not need to know about the underlying structure, while can still carry out the operations from a high level abstraction.

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.



The above diagram is an example of a 3 layer architecture. The browser can be termed as a client or the user interface, the Notebook is the data processing layer and the Kernel is the data storage layer.

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.*

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

The main difference between schema and instance is that schema is a structural view of the database, while the instance is the data stored in a database at a particular moment of time. For the example, schema is the structure like data type of country code, possiblle values in the enum phone_type or the length of the main_number. The instance is the record stored: +1, 212-555-1212, x1002, work

W7

Briefly explain the differences between:

- Database stored procedure A procedure is a combination of SQL statements written to perform a specified tasks. It helps in code re-usability and saves time and lines of code.

- Database function A database function is a specialised procedure that is pre-defined, it takes in a set of values to give an output.
- Database trigger A trigger is a special kind of procedure which executes only when some triggering event such as INSERT, UPDATE, DELETE operations occurs in a table.

The difference is procedures can be user-defined, database functions are pre-defined and triggers work on the execution of an event.

W8

Briefly explain:

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

- Equi-join: It type of join in which only can only use the equality operator and does so on the columns that are specified.
- Natural join: A natural join is an equi-join on attributes that have the same name. It only compares the same name columns.
- Theta join: A theta join allows for various comparison relationships like >, <, = etc.
- Self-join: When a table is joined with itself.

W9

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

The unique key constraint allows for non-duplicate values along with a single NULL value whereas primary key column cannot contain a null value. It only contains unique values.

W10

Briefly explan *domain constraint* and give an example.

Domain Constraints defines the domain of all the possible values that the attribute can take. For eg: We can specify the ranges of value a column take. Like a check can be placed to have salary > 20000. The constraint will make sure that all values entered satisfy this or will give an error.

Entity Relationship Model

Question

- 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

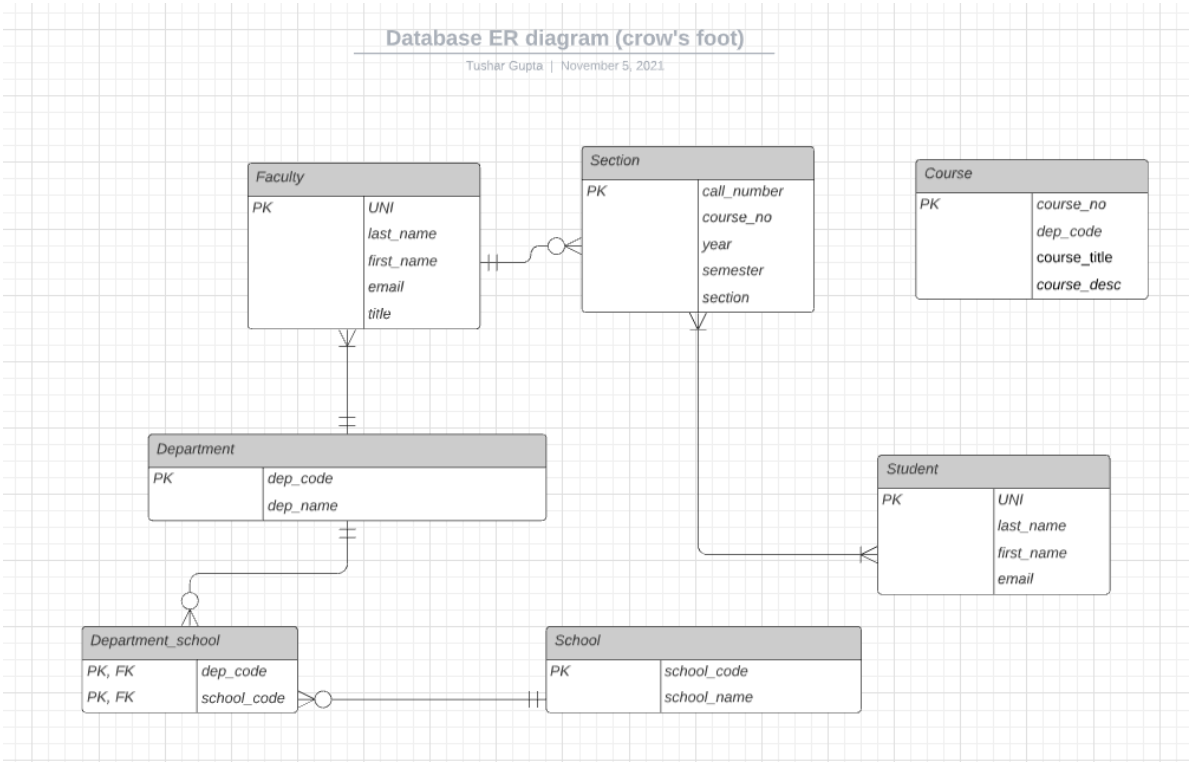
Assumptions, Decsions and Notes:

1. Faculty may teach 0, 1 or many sections.
2. Section can have 1 or many students

In [82]:

```
from IPython.display import Image
Image('./q2.png')
```

Out[82]:



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) for this question.

Two time slots *X_ and _Y* obviously overlap if:

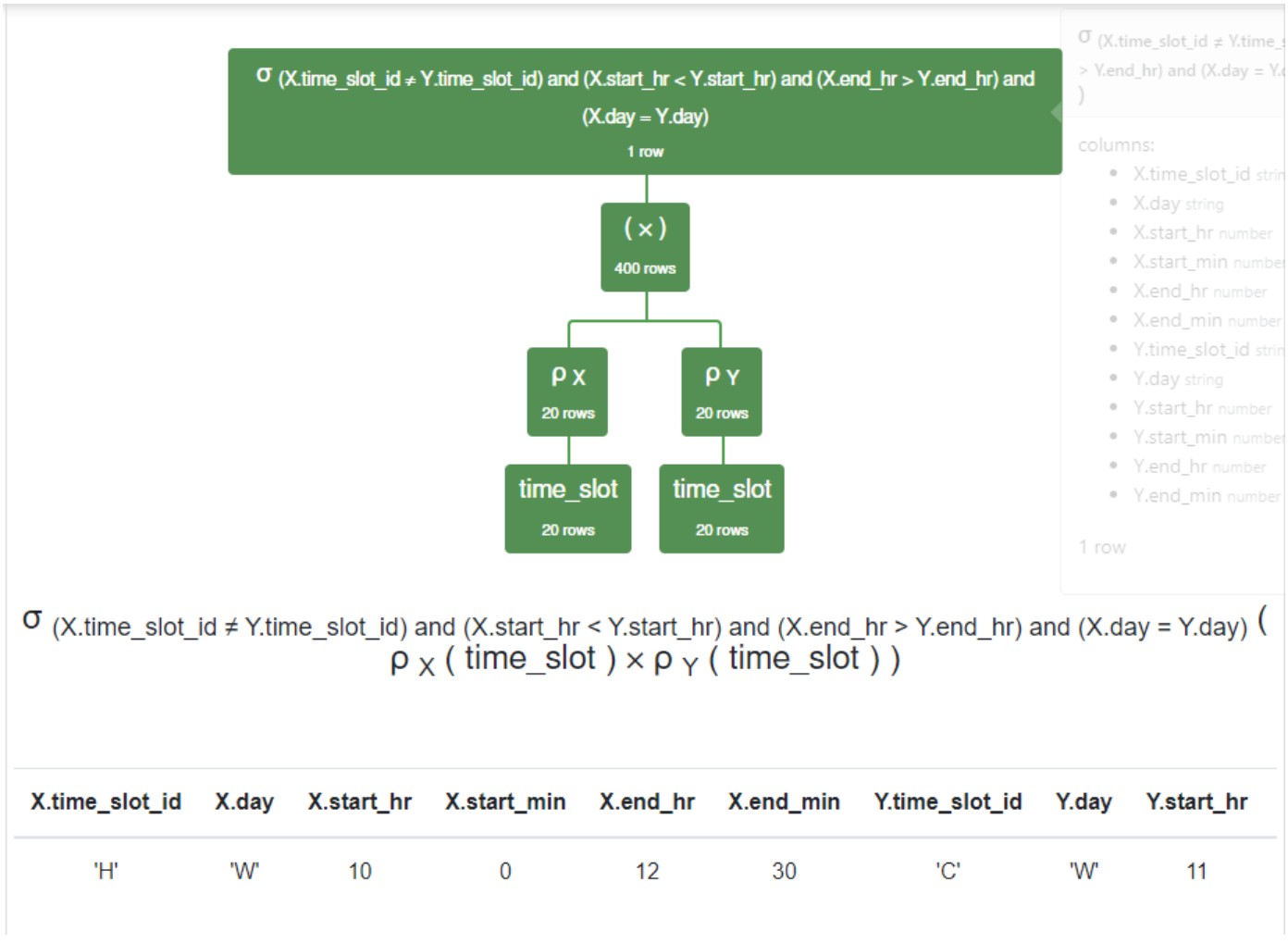
1. They are not the same time slot, i.e. do not have the same $time\_slot\_id$.
2. They have at least one lecture on *the same day,* the start hour for *X_ is before the start hour for _Y*, and the end hour for *X_ is after the start hour for _Y.*
3. To make the question easier, you do not need to consider 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

```
σ (X.time_slot_id ≠ Y.time_slot_id) ∧ (X.start_hr < Y.start_hr) ∧ (X.end_hr > Y.end_hr) ∧
(X.day = Y.day) (ρ X (time_slot) × ρ Y (time_slot))
```

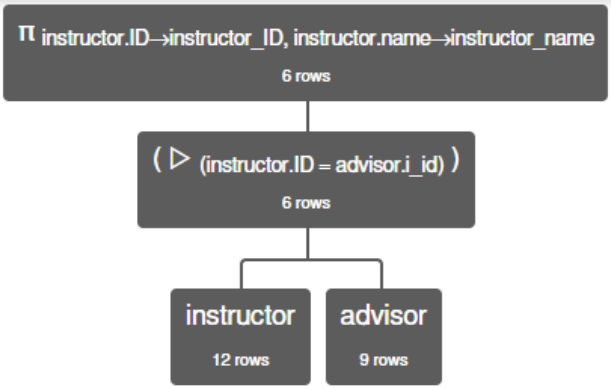


X.time_slot_id	X.day	X.start_hr	X.start_min	X.end_hr	X.end_min	Y.time_slot_id	Y.day	Y.start_hr
'H'	'W'	10	0	12	30	'C'	'W'	11

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.

π instructor_ID ← instructor.ID, instructor_name ← instructor.name (instructor ▷ (instructor.ID = advisor.i_id) advisor)



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

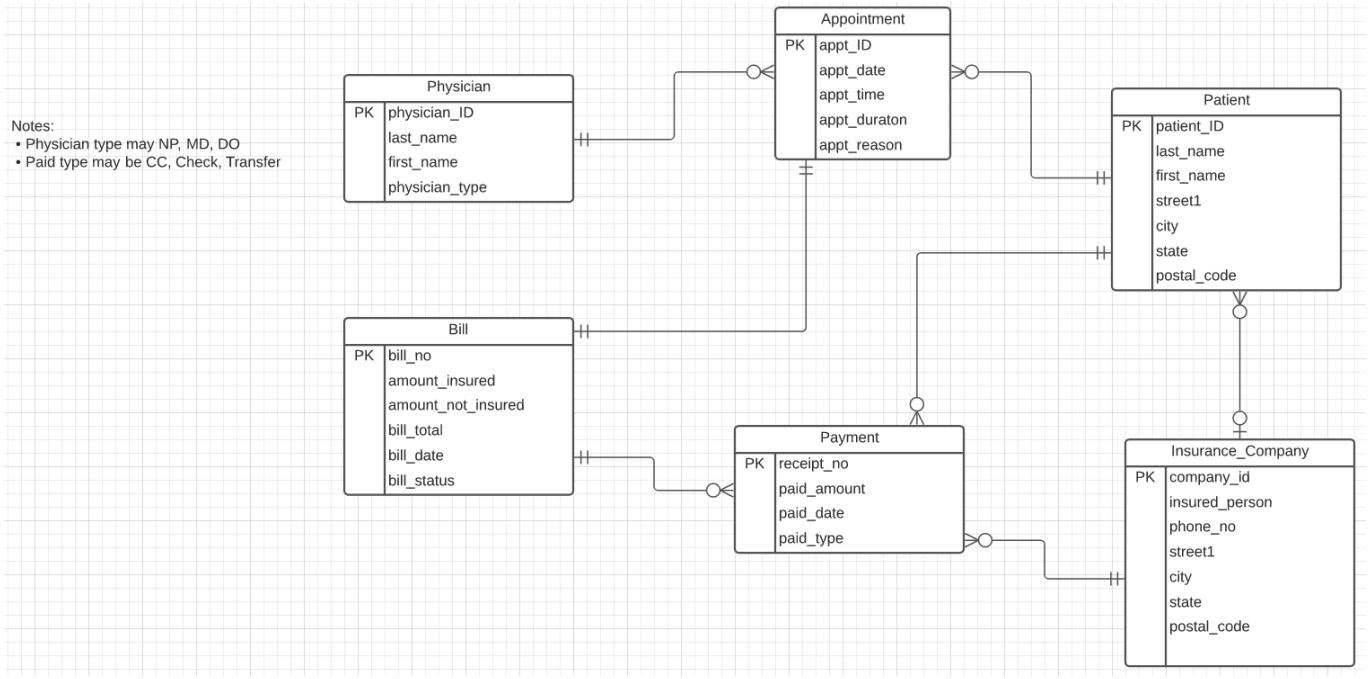
instructor_ID	instructor_name
12121	'Wu'
15151	'Mozart'
32343	'El Said'
33456	'Gold'
58583	'Califieri'
83821	'Brandt'

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.
- 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 table appointment
(
    appt_ID       int          not null
        primary key,
    appt_date     date         null,
    appt_time     timestamp    null,
    appt_duration int          null,
    appt_reason   varchar(255) null,
    physician_ID  int          null,
    patient_ID    int          null,
    bill_no       int          null,
    constraint appointment_bill_no_uindex
        unique (bill_no),
    constraint appointment_bill_bill_no_fk
        foreign key (bill_no) references bill (bill_no),
    constraint appointment_patient_patient_ID_fk
        foreign key (patient_ID) references patient (patient_ID),
    constraint appointment_physician_physician_ID_fk
        foreign key (physician_ID) references physician (physician_ID)
```

```
);

create table bill
(
    bill_no            int         not null
        primary key,
    amount_insured     float       null,
    amount_not_insured float       null,
    bill_total         float       null,
    bill_date          date        null,
    bill_status        varchar(20) null
);

create table insurance_company
(
    company_id     int          not null
        primary key,
    insured_person varchar(255) null,
    phone_no       int          null,
    street1        varchar(255) null,
    city           varchar(100) null,
    state          varchar(100) null,
    postal_code    int          null
);

create table patient
(
    patient_ID  int          not null
        primary key,
    last_name   varchar(100) null,
    first_name  varchar(100) null,
    street1     varchar(255) null,
    city        varchar(100) null,
    state       varchar(100) null,
    postal_code int          null,
    company_id  int          null,
    constraint patient_insurance_company_company_id_fk
        foreign key (company_id) references insurance_company (company_id)
);

create table payment
(
    receipt_no  int         not null
        primary key,
    paid_amount float       null,
    paid_date   date        null,
    paid_type   varchar(20) null,
    company_id  int         null,
    bill_no     int         null,
    patient_id  int         null,
    constraint payment_bill_bill_no_fk
        foreign key (bill_no) references bill (bill_no),
```

```
    constraint payment_insurance_company_company_id_fk
        foreign key (company_id) references insurance_company (company_id),
    constraint payment_patient_patient_ID_fk
        foreign key (patient_id) references patient (patient_ID)
);

create table physician
(
    physician_ID   int          not null
        primary key,
    last_name      varchar(100) null,
    first_name     varchar(100) null,
    physician_type varchar(5)   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 [2]:

```
import pymysql
import pandas as pd
conn=pymysql.connect(host='localhost',port=int(3306),user='root',passwd='admin123', db = 'l
pd.read_sql_query("select birthCountry, count(*) as no_of_births from \
                                                lahmansbaseballdb.people \
where birthCountry not in (select distinct(deathCountry) from lahmansbaseballdb.people wher
```

```
C:\Users\tushar\Anaconda3\lib\site-packages\pandas\compat\_optional.py:138:
UserWarning: Pandas requires version '2.7.0' or newer of 'numexpr' (version
'2.6.9' currently installed).
  warnings.warn(msg, UserWarning)
```

Out[2]:

	birthCountry	no_of_births
0	Germany	45
1	Colombia	24
2	South Korea	23
3	Curacao	15
4	Nicaragua	15
5	Russia	9
6	Italy	7
7	Czech Republic	6
8	Aruba	5
9	Poland	5
10	Brazil	5
11	Sweden	4
12	Jamaica	4
13	Spain	4
14	Norway	3
15	South Africa	2
16	Saudi Arabia	2
17	Honduras	2
18	Guam	2
19	Hong Kong	1
20	Afghanistan	1
21	Denmark	1
22	Switzerland	1
23	Singapore	1
24	Belgium	1
25	Peru	1
26	Belize	1

	birthCountry	no_of_births
27	Indonesia	1
28	Finland	1
29	Lithuania	1
30	Viet Nam	1
31	Slovakia	1
32	Greece	1
33	Portugal	1
34	Latvia	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 [3]:

```
import pymysql
import pandas as pd
conn=pymysql.connect(host='localhost',port=int(3306),user='root',passwd='admin123', db = 'l
pd.read_sql_query("with batting_cal as \
(select playerID, OBP, SLG, OBP + SLG as OPS, total_AB from (select playerID, (SUM(H) + SUM
       (((sum(H) - sum(2B) - sum(3b) - sum(HR)) + 2*sum(2B) + 3*sum(3B) + 4*sum(HR))/sum(AB
       from batting group by playerID) as t1) \
, pitching_cal as \
    (select playerID, sum(w) as total_wins, \
           sum(l) as total_losses, \
           ((sum(w)) / (sum(w) + sum(l))) as win_percentage from pitching group by playerID
, batting_criteria as \
    (select * from batting_cal where OPS >= 1.0 and total_AB >=1000) \
, pitching_criteria as \
    (select * from pitching_cal where total_losses + total_wins>= 200 and (total_wins >=300
, join_table as \
    ( select * from people \
        LEFT JOIN batting_criteria using(playerID) \
        LEFT JOIN pitching_criteria using(playerID)) \
, filter_table as \
    ( select * from join_table where total_AB is not NULL or total_wins is not NUll) \
select playerID, nameFirst, nameLast, debut_date, finalgame_date, OPS, OBP, SLG, total_wins
       (CASE \
           WHEN OPS is not NULL THEN 'Batter' \
           ELSE 'Pitcher' \
       END) as great_because from filter_table; \
", conn)
```

Out[3]:

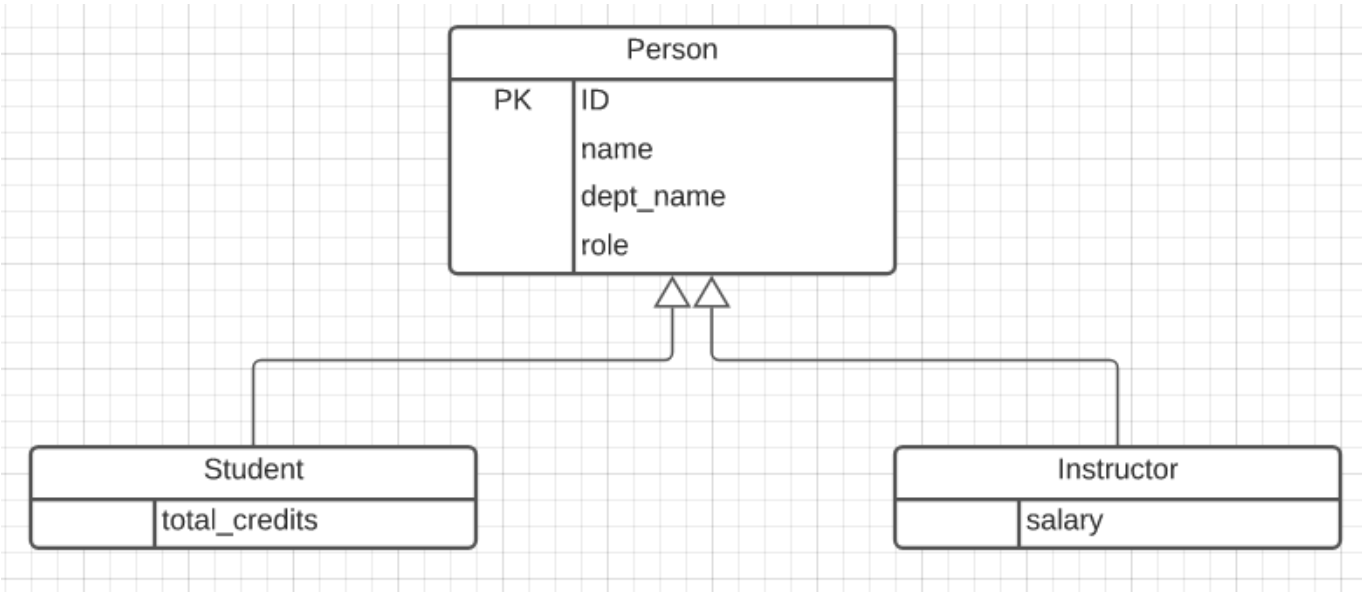
	playerID	nameFirst	nameLast	debut_date	finalgame_date	OPS	OBP	SLG	total
0	alexape01	Pete	Alexander	1911-04-15	1930-05-28	NaN	NaN	NaN	
1	bondsba01	Barry	Bonds	1986-05-30	2007-09-26	1.0497	0.4428	0.6069	
2	carltst01	Steve	Carlton	1965-04-12	1988-04-23	NaN	NaN	NaN	
3	clarkjo01	John	Clarkson	1882-05-02	1894-07-12	NaN	NaN	NaN	
4	clemero02	Roger	Clemens	1984-05-15	2007-09-16	NaN	NaN	NaN	
5	foxxji01	Jimmie	Foxx	1925-05-01	1945-09-23	1.0368	0.4275	0.6093	
6	galvipu01	Pud	Galvin	1875-05-22	1892-08-02	NaN	NaN	NaN	
7	gehrilo01	Lou	Gehrig	1923-06-15	1939-04-30	1.0771	0.4447	0.6324	
8	glavito02	Tom	Glavine	1987-08-17	2008-08-14	NaN	NaN	NaN	
9	greenha01	Hank	Greenberg	1930-09-14	1947-09-18	1.0153	0.4103	0.6050	
10	grovele01	Lefty	Grove	1925-04-14	1941-09-28	NaN	NaN	NaN	
11	hornsro01	Rogers	Hornsby	1915-09-10	1937-07-20	1.0073	0.4308	0.5765	
12	johnsra05	Randy	Johnson	1988-09-15	2009-10-04	NaN	NaN	NaN	
13	johnswa01	Walter	Johnson	1907-08-02	1927-09-30	NaN	NaN	NaN	
14	keefeti01	Tim	Keefe	1880-08-06	1893-08-15	NaN	NaN	NaN	
15	maddugr01	Greg	Maddux	1986-09-03	2008-09-27	NaN	NaN	NaN	
16	mathech01	Christy	Mathewson	1900-07-17	1916-09-04	NaN	NaN	NaN	

	playerID	nameFirst	nameLast	debut_date	finalgame_date	OPS	OBP	SLG	total
17	nichoki01	Kid	Nichols	1890-04-23	1906-05-18	NaN	NaN	NaN	
18	niekrph01	Phil	Niekro	1964-04-15	1987-09-27	NaN	NaN	NaN	
19	perryga01	Gaylord	Perry	1962-04-14	1983-09-21	NaN	NaN	NaN	
20	planked01	Eddie	Plank	1901-05-13	1917-08-06	NaN	NaN	NaN	
21	radboch01	Old Hoss	Radbourn	1880-05-05	1891-08-11	NaN	NaN	NaN	
22	ruthba01	Babe	Ruth	1914-07-11	1935-05-30	1.1616	0.4718	0.6898	
23	ryanno01	Nolan	Ryan	1966-09-11	1993-09-22	NaN	NaN	NaN	
24	seaveto01	Tom	Seaver	1967-04-13	1986-09-19	NaN	NaN	NaN	
25	spahnwa01	Warren	Spahn	1942-04-19	1965-10-01	NaN	NaN	NaN	
26	suttodo01	Don	Sutton	1966-04-14	1988-08-09	NaN	NaN	NaN	
27	welchmi01	Mickey	Welch	1880-05-01	1892-05-17	NaN	NaN	NaN	
28	willite01	Ted	Williams	1939-04-20	1960-09-28	1.1144	0.4806	0.6338	
29	wynnea01	Early	Wynn	1939-09-13	1963-09-13	NaN	NaN	NaN	
30	youngcy01	Cy	Young	1890-08-06	1911-10-06	NaN	NaN	NaN	

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 NOT NULL, or salary is N
ULL and total_cred is NULL.
            - The procedure must compute a new, unique ID. The approach is to find
the manimum
                ID value over Student and Instructor. Add 1 to the value to produc
e the new, unique ID.
            - The procedure then adds the information to Student or Instructor bas
ed on whether
                total_cred is NULL or salary is NULL.
    */

    /* <YOUR CODE GOES HERE> */

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

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

END
```

Answer

Create view statement

```
create view Person as
    select 'S' as role, ID, name, dept_name from student
    union
    select 'F' as role, ID, name, dept_name from instructor
```

Create procedure statement

```
create
    definer = root@localhost procedure create_person(IN person_name varchar(32), I
N dept_name varchar(32),
                                                     IN total_cred decimal(3), 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';


    /* <YOUR CODE GOES HERE> */
    IF person_name IS NULL OR dept_name IS NULL THEN
        SET bad_person = TRUE;
    end if;

    IF (salary IS NULL AND total_cred IS NULL) OR (salary IS NOT NULL AND total_cr
ed IS NOT NULL) THEN
        SET bad_person = TRUE;
    end if;

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

    SELECT MAX(person.ID) INTO new_id FROM person;
    SET new_id = new_id + 1;

    SELECT new_id;
    IF salary IS NOT NULL THEN
        INSERT INTO instructor values (new_id, person_name, dept_name, salary);
    end if;
```

```
        IF total_cred IS NOT NULL THEN
            INSERT INTO student values (new_id, person_name, dept_name, total_cred);
        end if;

        /* NOTE: ID is the out parameter. You must set ID to the new, unique ID */
        SET ID = new_id;

    END;
```

Tests

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

In [73]:

```
conn=pymysql.connect(host='localhost',port=int(3306),user='root',passwd='admin123', db = 'd
```

- Test 1: Show the view.

In [62]:

```
pd.read_sql_query("select * from Person;", conn)
```

Out[62]:

	role	ID	name	dept_name
0	S	12345	Shankar	Comp. Sci.
1	S	19991	Brandt	History
2	S	23121	Chavez	Finance
3	S	44553	Peltier	Physics
4	S	45678	Levy	Physics
5	S	54321	Williams	Comp. Sci.
6	S	55739	Sanchez	Music
7	S	70557	Snow	Physics
8	S	76543	Brown	Comp. Sci.
9	S	76653	Aoi	Elec. Eng.
10	S	98765	Bourikas	Elec. Eng.
11	S	98988	Tanaka	Biology
12	S	98989	Tushar	Comp
13	F	10101	Srinivasan	Comp. Sci.
14	F	12121	Wu	Finance
15	F	15151	Mozart	Music
16	F	22222	Einstein	Physics
17	F	32343	El Said	History
18	F	33456	Gold	Physics
19	F	45565	Katz	Comp. Sci.
20	F	58583	Califieri	History
21	F	76543	Singh	Finance
22	F	76766	Crick	Biology
23	F	83821	Brandt	Comp. Sci.

Create an Instructor and Student

In [69]:

```
pd.read_sql_query("CALL create_person('Ferguson', 'Comp. Sci.',  NULL, 31000.00, @prof_id)"
```

In [70]:

```
pd.read_sql_query("SELECT @prof_id;", conn)
```

Out[70]:

	@prof_id
0	98991

In [71]:

```
pd.read_sql_query("CALL create_person('Ferguson', 'Comp. Sci.',  100.0, NULL, @student_id);
```

Out[71]:

	new_id
0	98992

In [72]:

```
pd.read_sql_query("SELECT @student_id;", conn)
```

Out[72]:

	@student_id
0	98992

- Try an error

In [68]:

```
try:
    pd.read_sql("CALL create_person('Ferguson', 'Comp. Sci.',  100.0, 30000, @student_id);"
except Exception as e:
    print(e)
pd.read_sql("select @student_id;", conn)
```

```
Execution failed on sql 'CALL create_person('Ferguson', 'Comp. Sci.',  100.
0, 30000, @student_id);': (1644, 'Invalid Person information input')
```

Out[68]:

	@student_id
0	98991

Include DDL that Show Enforcing Immutable ID

```
create trigger student_immutable_id before update ON db_book_midterm.student
    for each row begin
        IF OLD.ID = NEW.ID THEN
            SIGNAL SQLSTATE '45000'
```

```
                SET MESSAGE_TEXT = 'CANNOT CHANGE RECORD, ALREADY EXISTS';
        end if;
    end;

    create trigger instructor_immutable_id before update ON db_book_midterm.instructor
        for each row begin
        IF OLD.ID = NEW.ID THEN
            SIGNAL SQLSTATE '45000'
                SET MESSAGE_TEXT = 'CANNOT CHANGE RECORD, ALREADY EXISTS';
        end if;
    end;
```

In [77]:

```
try:
    pd.read_sql_query("UPDATE db_book_midterm.student SET student.ID = 98765 WHERE student.
except Exception as e:
    print(e)
```

```
Execution failed on sql 'UPDATE db_book_midterm.student SET student.ID = 987
65 WHERE student.name = 'Bourikas';': (1644, 'CANNOT CHANGE RECORD, ALREADY
EXISTS')
```

In [78]:

```
try:
    pd.read_sql_query("UPDATE db_book_midterm.instructor SET instructor.ID = 76543 WHERE in
except Exception as e:
    print(e)
```

```
Execution failed on sql 'UPDATE db_book_midterm.instructor SET instructor.ID
= 76543 WHERE instructor.name = 'Ferguson';': (1062, "Duplicate entry '7654
3' for key 'instructor.PRIMARY'")
```

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 [9]:

```
import pandas as pd
world_cities = pd.read_csv("./worldcities.csv")
world_cities = world_cities[['city', 'city_ascii', 'lat', 'lng', 'country', 'iso2', 'iso3',
```

In [52]:

```
conn=pymysql.connect(host='localhost',port=int(3306),user='root',passwd='admin123', db = 'F
```

In [22]:

```
world_cities['iso2'].fillna()
```

Out[22]:

```
city          False
city_ascii    False
lat           False
lng           False
country       False
iso2           True
iso3          False
id            False
dtype: bool
```

In [34]:

```
from sqlalchemy import create_engine
#engine = create_engine('sqlite://', echo=False)
db_data = 'mysql+pymysql://' + 'root' + ':' + 'admin123' + '@' + 'localhost' + ':'+ str(330
engine = create_engine(db_data)

world_cities.to_sql("worldcities", engine, schema="F21W4111Midterm", if_exists='replace', i
```

- Display data.

In [37]:

```
pd.read_sql_query("select * from F21W4111Midterm.worldcities order by city limit 30;", conn
```

Out[37]:

	city	city_ascii	lat	lng	country	iso2	iso3	id
0	ʻAdrā	`Adra	33.6000	36.5150	Syria	SY	SYR	1760640037
1	ʻAjlūn	`Ajlun	32.3325	35.7517	Jordan	JO	JOR	1400775371
2	ʻAjmān	`Ajman	25.3994	55.4797	United Arab Emirates	AE	ARE	1784337875
3	ʻAkko	`Akko	32.9261	35.0839	Israel	IL	ISR	1376781950
4	ʻAlavīcheh	`Alavicheh	33.0528	51.0825	Iran	IR	IRN	1364605877
5	ʻAmrān	`Amran	15.6594	43.9439	Yemen	YE	YEM	1887433410
6	ʻĀmūdā	`Amuda	37.1042	40.9300	Syria	SY	SYR	1760247135
7	ʻAnadān	`Anadan	36.2936	37.0444	Syria	SY	SYR	1760993442
8	ʻAssāl al Ward	`Assal al Ward	33.8658	36.4133	Syria	SY	SYR	1760181042
9	ʻAtaq	`Ataq	14.5500	46.8000	Yemen	YE	YEM	1887172893
10	ʻAyn ʻĪsá	`Ayn `Isa	36.3858	38.8472	Syria	SY	SYR	1760078370
11	ʻIbrī	`Ibri	23.2254	56.5170	Oman	OM	OMN	1512077267
12	ʾAïn Abessa	'Ain Abessa	36.3000	5.2950	Algeria	DZ	DZA	1012074116
13	ʾAïn Arnat	'Ain Arnat	36.1833	5.3167	Algeria	DZ	DZA	1012453452
14	ʾAïn Azel	'Ain Azel	35.8433	5.5219	Algeria	DZ	DZA	1012746080
15	ʾAïn el Hammam	'Ain el Hammam	36.5647	4.3061	Algeria	DZ	DZA	1012595495
16	ʾAïn Leuh	'Ain Leuh	33.2833	-5.3833	Morocco	MA	MAR	1504668626
17	ʾAïn Roua	'Ain Roua	36.3344	5.1806	Algeria	DZ	DZA	1012529757
18	ʾAli Ben Sliman	'Ali Ben Sliman	31.9053	-7.2144	Morocco	MA	MAR	1504127885
19	ʾAyn Bni Mathar	'Ayn Bni Mathar	34.0889	-2.0247	Morocco	MA	MAR	1504845272
20	ʾs-Hertogenbosch	's-Hertogenbosch	51.6833	5.3167	Netherlands	NL	NLD	1528012333
21	A Coruña	A Coruna	43.3713	-8.4188	Spain	ES	ESP	1724417375
22	Aachen	Aachen	50.7762	6.0838	Germany	DE	DEU	1276805572
23	Aadorf	Aadorf	47.4939	8.8975	Switzerland	CH	CHE	1756022542
24	Aalborg	Aalborg	57.0337	9.9166	Denmark	DK	DNK	1208789278
25	Aalen	Aalen	48.8372	10.0936	Germany	DE	DEU	1276757787
26	Aalsmeer	Aalsmeer	52.2639	4.7625	Netherlands	NL	NLD	1528899853
27	Aalst	Aalst	50.9333	4.0333	Belgium	BE	BEL	1056695813
28	Aalten	Aalten	51.9250	6.5808	Netherlands	NL	NLD	1528326020
29	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 [44]:

```
world_cities['iso2'] = world_cities['iso2'].fillna("NA")
countries = world_cities[['country', 'iso2', 'iso3']].drop_duplicates()
countries.to_sql('countries', engine, schema="F21W4111Midterm", if_exists='replace', index=
```

In [45]:

```
cities = world_cities[['city', 'city_ascii', 'lat', 'lng', 'iso2', 'id']]
cities.to_sql('cities', engine, schema="F21W4111Midterm", if_exists='replace', index=False)
```

```
create table countries
(
    country text null,
    iso2    text null,
    iso3    text null
);

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 (iso2);


create table cities
(
    city       text   null,
```

```
    city_ascii text   null,
    lat        double null,
    lng        double null,
    iso2       text   null,
    id         bigint null
);

alter table cities modify city VARCHAR(255) null;

alter table cities modify city_ascii VARCHAR(255) null;

alter table cities modify iso2 VARCHAR(2) null;

alter table cities
    add constraint cities_countries_iso2_fk
        foreign key (iso2) references countries (iso2);
```

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)

DECIMAL can be used as a better data-type to represent the latitudes and longitudes. It is more optimal as we specify number of digits after the decimal as per the maximum precision present in the data. Whereas, a float type could be unnecessary if we do not have a large number of digits in our co-ordinates.

```
alter table cities modify new_lat decimal(7,4) null;

alter table cities modify new_lng decimal(8,4) null;

UPDATE cities SET new_lat = lat where lat is not NULL;
UPDATE cities SET new_lng = lng where lng is not NULL;
```

In [48]:

```
pd.read_sql_query("select * from cities ORDER BY id ASC limit 20;", conn)
```

Out[48]:

	city	city_ascii	lat	lng	iso2	id	new_lat	new_lng
0	Kandahār	Kandahar	31.6078	65.7053	AF	1004003059	31.6078	65.7053
1	Qalāt	Qalat	32.1061	66.9069	AF	1004016690	32.1061	66.9069
2	Sar-e Pul	Sar-e Pul	36.2214	65.9278	AF	1004047427	36.2214	65.9278
3	Pul-e Khumrī	Pul-e Khumri	35.9500	68.7000	AF	1004123527	35.9500	68.7000
4	Maḩmūd-e Rāqī	Mahmud-e Raqi	35.0167	69.3333	AF	1004151943	35.0167	69.3333
5	Ghaznī	Ghazni	33.5492	68.4233	AF	1004167490	33.5492	68.4233
6	Pul-e ‘Alam	Pul-e `Alam	33.9953	69.0227	AF	1004180853	33.9953	69.0227
7	Kunduz	Kunduz	36.7280	68.8725	AF	1004227517	36.7280	68.8725
8	Herāt	Herat	34.3738	62.1792	AF	1004237782	34.3738	62.1792
9	Asadābād	Asadabad	34.8742	71.1528	AF	1004251962	34.8742	71.1528
10	Sharan	Sharan	33.1757	68.7304	AF	1004273142	33.1757	68.7304
11	Bāmyān	Bamyan	34.8167	67.8167	AF	1004274041	34.8167	67.8167
12	Jalālābād	Jalalabad	34.4303	70.4528	AF	1004315012	34.4303	70.4528
13	Bāzārak	Bazarak	35.3129	69.5152	AF	1004374554	35.3129	69.5152
14	Mazār-e Sharīf	Mazar-e Sharif	36.7000	67.1167	AF	1004436363	36.7000	67.1167
15	Kōṯah-ye ‘Ashrō	Kotah-ye `Ashro	34.4500	68.8000	AF	1004450357	34.4500	68.8000
16	Faīẕābād	Faizabad	37.1298	70.5792	AF	1004452653	37.1298	70.5792
17	Gardēz	Gardez	33.5931	69.2297	AF	1004468894	33.5931	69.2297
18	Andkhōy	Andkhoy	36.9500	65.1167	AF	1004472345	36.9500	65.1167
19	Buynī Qarah	Buyni Qarah	36.3172	66.8747	AF	1004472874	36.3172	66.8747

FInal Create Table Statements

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

```
create table countries
(
    country varchar(50) not null,
    iso2    varchar(2)  not null
        primary key,
    iso3    varchar(3)  not null
);
```

```
create table cities
(
    city       varchar(255)  null,
    city_ascii varchar(255)  null,
    lat        double        null,
    lng        double        null,
    iso2       varchar(2)    null,
    id         bigint        null,
    new_lat    decimal(7, 4) null,
    new_lng    decimal(8, 4) null,
    constraint cities_countries_iso2_fk
        foreign key (iso2) references countries (iso2)
);
```

Fixing People Table

Create a Copy People

- Create a table `F21Midterm.people_modified` that has the same schema and data as `lahmansbaseballdb.people` .
- SQL: create table F21W4111Midterm.people_modified AS (SELECT * from people);

Answer

In [50]:

```
pd.read_sql_query("select * FROM people_modified limit 30;", conn)
```

Out[50]:

'ear	deathMonth	deathDay	...	throws	debut	finalGame	retroID	bbrefID	birth_date	c
Jan	NaN	NaN	...	R	2004-04-06	2015-08-23	aardd001	aardsda01	1981-12-27	
Jan	NaN	NaN	...	R	1954-04-13	1976-10-03	aaroh101	aaronha01	1934-02-05	
34.0	8.0	16.0	...	R	1962-04-10	1971-09-26	aarot101	aaronto01	1939-08-05	
Jan	NaN	NaN	...	R	1977-07-26	1990-10-03	aased001	aasedo01	1954-09-08	
Jan	NaN	NaN	...	L	2001-09-10	2006-04-13	abada001	abadan01	1972-08-25	
Jan	NaN	NaN	...	L	2010-07-28	2019-09-28	abadf001	abadfe01	1985-12-17	
05.0	5.0	17.0	...	R	1875-04-26	1875-06-10	abadj101	abadijo01	1850-11-04	
57.0	1.0	6.0	...	R	1897-09-04	1910-09-15	abbae101	abbated01	1877-04-15	
32.0	6.0	11.0	...	R	1892-06-14	1896-09-23	abbeb101	abbeybe01	1869-11-11	
26.0	4.0	27.0	...	L	1893-08-16	1897-08-19	abbec101	abbeych01	1866-10-14	
30.0	2.0	13.0	...	R	1890-04-19	1890-05-23	abbod101	abbotda01	1862-03-16	
35.0	6.0	11.0	...	R	1903-04-25	1905-09-20	abbof101	abbotfr01	1874-10-22	
Jan	NaN	NaN	...	R	1973-07-29	1984-08-08	abbog001	abbotgl01	1951-02-16	
Jan	NaN	NaN	...	L	1997-06-10	2001-09-29	abboj002	abbotje01	1972-08-17	
Jan	NaN	NaN	...	L	1989-04-08	1999-07-21	abboj001	abbotji01	1967-09-19	
Jan	NaN	NaN	...	R	1993-09-07	2001-04-13	abbok002	abbotku01	1969-06-02	
Jan	NaN	NaN	...	L	1991-09-10	1996-08-24	abbok001	abbotky01	1968-02-18	
33.0	4.0	13.0	...	R	1910-09-10	1910-10-15	abboo101	abbotod01	1886-09-05	
Jan	NaN	NaN	...	R	1990-08-21	2004-08-07	abbop001	abbotpa01	1967-09-15	
93.0	5.0	20.0	...	L	1950-09-15	1957-09-11	abera101	aberal01	1927-07-31	
39.0	11.0	11.0	...	None	1871-10-21	1871-10-21	aberd101	abercda01	1850-01-02	
Jan	NaN	NaN	...	R	2006-04-04	2008-09-28	aberr001	abercre01	1980-07-15	

'ear	deathMonth	deathDay	...	throws	debut	finalGame	retroID	bbrefID	birth_date	c
)6.0	2.0	19.0	...	R	1952-09-27	1952-09-27	aberb101	abernbi01	1929-01-30	
NaN	NaN	NaN	...	R	2001-06-25	2005-09-29	aberb001	abernbr01	1977-09-23	
)1.0	11.0	16.0	...	L	1942-09-19	1944-04-29	abert102	abernte01	1921-10-30	
)4.0	12.0	16.0	...	R	1955-04-13	1972-09-30	abert101	abernte02	1933-03-06	
)4.0	12.0	5.0	...	L	1946-07-28	1947-04-17	aberw101	abernwo01	1915-02-01	
'3.0	6.0	23.0	...	R	1947-07-18	1949-05-09	aberc101	aberscl01	1921-08-28	
51.0	2.0	8.0	...	L	1905-09-02	1911-05-05	ableh101	ablesha01	1883-10-04	
NaN	NaN	NaN	...	R	1987-09-08	1992-10-03	abnes001	abnersh01	1966-06-17	

Fixing `birthCountry`

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

In [41]:

```
%%sql


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

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

Out[41]:

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
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. Drop the column `birthCountry.`

```
UPDATE people_modified set birthCountry = (
    CASE
        WHEN birthCountry = 'Bahamas' THEN 'Bahamas, The'
        When birthCountry = 'CAN' THEN 'CANADA'
        WHEN birthCountry = 'Czech Republic' THEN 'Czechia'
        WHEN birthCountry = 'South Korea' THEN 'Korea, South'
        WHEN birthCountry = 'USA' THEN 'United States'
        WHEN birthCountry = 'Viet Nam' THEN 'Vietnam'
        WHEN birthCountry = 'D.R.' THEN 'Dominican Republic'
        WHEN birthCountry = 'P.R.' THEN 'Puerto Rico'
        WHEN birthCountry = 'P.R.' THEN 'Puerto Rico'
        WHEN birthCountry = 'V.I.' THEN 'United States'
        WHEN birthCountry = 'At Sea' THEN NULL
    END)
WHERE 1;

UPDATE people_modified
SET birthCountryISO2 = (select iso2 from countries where people_modified.birthCoun
try = countries.country);

alter table people_modified
    add constraint people_modified_countries_iso2_fk
        foreign key (birthCountryISO2) references countries (iso2);

alter table people_modified drop column birthCountry;
```

Type *Markdown* and LaTeX: α^2

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

In [56]:

```
pd.read_sql("select distinct birthCountryISO2, count(*) as count from people_modified \
            where birthCountryISO2 not in \
            (select iso2 from countries) group by birthCountryISO2;", conn)
```

Out[56]:

	birthCountryISO2	count