COMS W4111: Introduction to Databases
Fall 2023, Section 2

Homework 1, Part 2: Non-Programming

Introduction

This notebook contains HW1 Part 2 Nonprogramming. **Only those on the nonprogramming track should complete this part.** To ensure everything runs as expected, work on this notebook in Jupyter.

Submission instructions:

- You will submit **PDF and ZIP files** for this assignment. Gradescope will have two separate assignments for these.
- For the PDF:
  - The most reliable way to save as PDF is to go to your browser's menu bar and click `File -> Print`. **Switch the orientation to landscape mode**, and hit save.
  - **MAKE SURE ALL YOUR WORK (CODE AND SCREENSHOTS) IS VISIBLE ON THE PDF. YOU WILL NOT GET CREDIT IF ANYTHING IS CUT OFF.** Reach out for troubleshooting.
- For the ZIP:
  - Zip the folder that contains this notebook and any screenshots.

Add Student Information

In [120…

```
# Print your name, uni, and track below

name = "Yuanxi Li"
```

```
uni = "yl5127"
track = "Nonprogramming Track"

print(name)
print(uni)
print(track)
```

```
Yuanxi Li
yl5127
Nonprogramming Track
```

Setup

SQL Magic

In [121…
```
%load_ext sql
```

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

You may need to change the password below.

In [122…
```
%sql mysql+pymysql://root:dbuserdbuser@localhost
```

In [123…
```
%sql SELECT 1
```

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

Out[123]:

1
1

Python Libraries

In [124…
```
import os

from IPython.display import Image
import pandas
from sqlalchemy import create_engine
```

You may need to change the password below.

In [125…
```
engine = create_engine("mysql+pymysql://root:dbuserdbuser@localhost")
```

Load Data

We're going to load data into a new database called `lahmans_hw1`. The data is stored as CSV files in the `data/` directory.

In [126…
```
%sql DROP SCHEMA IF EXISTS lahmans_hw1
%sql CREATE SCHEMA lahmans_hw1
```

```
 * mysql+pymysql://root:***@localhost
6 rows affected.
 * mysql+pymysql://root:***@localhost
1 rows affected.
```
Out[126]:
```
[]
```

In [127…
```
def load_csv(data_dir, file_name, schema, table_name=None):
    """
    :param data_dir: The directory containing the file.
    :param file_name: The file name.
    :param schema: The database for the saved table.
    :param table_name: The name of the table to create. If the name is None, the function uses the name of
        the file before '.csv'. So, file_name 'cat.csv' becomes table 'cat'.
    :return: None
    """

    if table_name is None:
        table_name = file_name.split(".")
        table_name = table_name[0]
```

```
    full_file_name = os.path.join(data_dir, file_name)

    df = pandas.read_csv(full_file_name)
    df.to_sql(table_name, con=engine, schema=schema, if_exists="replace", index=False)
```

In [128…

```
data_dir = "data"
csv_files = [
    "People.csv",
    "Appearances.csv",
    "Batting.csv",
    "Pitching.csv",
    "Teams.csv",
    "Managers.csv",
]
schema = "lahmans_hw1"

for f in csv_files:
    load_csv(data_dir, f, schema)
    print("Loaded file:", f)
```

```
C:\Users\25631\AppData\Roaming\Python\Python310\site-packages\pandas\io\sql.py:1666: UserWarning: The provided table name 'People' is not found exactly as such in the database after writing the table, possibly due to case sensitivity issues. Consider using lower case table names.
  warnings.warn(msg, UserWarning)
Loaded file: People.csv
C:\Users\25631\AppData\Roaming\Python\Python310\site-packages\pandas\io\sql.py:1666: UserWarning: The provided table name 'Appearances' is not found exactly as such in the database after writing the table, possibly due to case sensitivity issues. Consider using lower case table names.
  warnings.warn(msg, UserWarning)
Loaded file: Appearances.csv
C:\Users\25631\AppData\Roaming\Python\Python310\site-packages\pandas\io\sql.py:1666: UserWarning: The provided table name 'Batting' is not found exactly as such in the database after writing the table, possibly due to case sensitivity issues. Consider using lower case table names.
  warnings.warn(msg, UserWarning)
Loaded file: Batting.csv
C:\Users\25631\AppData\Roaming\Python\Python310\site-packages\pandas\io\sql.py:1666: UserWarning: The provided table name 'Pitching' is not found exactly as such in the database after writing the table, possibly due to case sensitivity issues. Consider using lower case table names.
  warnings.warn(msg, UserWarning)
Loaded file: Pitching.csv
```

```
C:\Users\25631\AppData\Roaming\Python\Python310\site-packages\pandas\io\sql.py:1666: UserWarning: The provided table name 'Teams' is not found exactly as such in the database after writing the table, possibly due to case sensitivity issues. Consider using lower case table names.
  warnings.warn(msg, UserWarning)
Loaded file: Teams.csv
Loaded file: Managers.csv
C:\Users\25631\AppData\Roaming\Python\Python310\site-packages\pandas\io\sql.py:1666: UserWarning: The provided table name 'Managers' is not found exactly as such in the database after writing the table, possibly due to case sensitivity issues. Consider using lower case table names.
  warnings.warn(msg, UserWarning)
```

Data Cleanup

The `load_csv` function above created new tables and inserted data into them for us. Unfortunately, because it cannot guess our intentions, the tables have generic data types and are not related to each other. In this assignment, we'll fix these issues.

In [129…
```
%sql USE lahmans_hw1
```

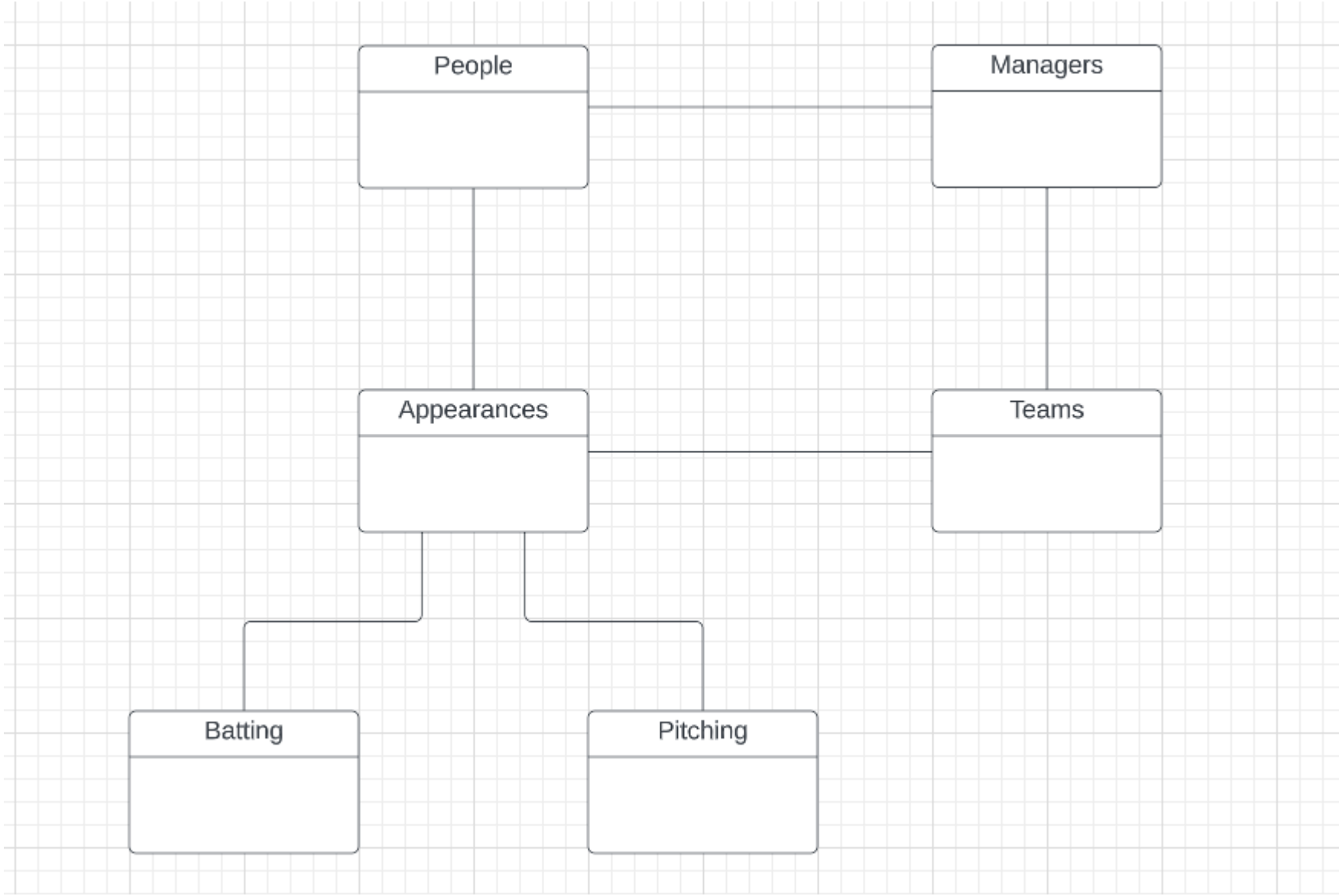
```
 * mysql+pymysql://root:***@localhost
0 rows affected.
```

Out[129]:
```
[]
```

Below is an overview of the six tables that we inserted and how they should be related.

In [130…
```
Image("./lahmans-conceptual.png")
```

Out[130]:



People

The `People` table is defined as

```
create table People
(
    playerID     text   null,
    birthYear    double null,
    birthMonth   double null,
    birthDay     double null,
    birthCountry text   null,
    birthState   text   null,
    birthCity    text   null,
    deathYear    double null,
    deathMonth   double null,
    deathDay     double null,
    deathCountry text   null,
    deathState   text   null,
    deathCity    text   null,
    nameFirst    text   null,
    nameLast     text   null,
    nameGiven    text   null,
    weight       double null,
    height       double null,
    bats         text   null,
    throws       text   null,
    debut        text   null,
    finalGame    text   null,
    retroID      text   null,
    bbrefID      text   null
);
```

You are to complete the following tasks:

1. Convert `playerID`, `retroID`, and `bbrefID` to **minimally sized** `CHAR`. Minimally sized means that the length passed into `CHAR` must be as small as possible while still being able to contain a `playerID` (i.e., don't simply choose a random large number).
2. Convert the `DOUBLE` columns to `INT`.
3. Convert `bats` and `throws` to `ENUM`.
4. Create two new columns, `dateOfBirth` and `dateOfDeath` of type `DATE`. Populate these columns based on `birthYear`, `birthMonth`, `birthDay`, `deathYear`, `deathMonth`, and `deathDay`. If any of these columns are null, you can set the corresponding new column to null (i.e., only keep full dates).

5. Convert `debut` and `finalGame` to `DATE`.

You should use `ALTER TABLE` to modify attributes (columns) and `UPDATE TABLE` to modify data (rows).

In [131…

```
%%sql

-- 1. Modify the structure:

-- Convert playerID, retroID, and bbrefID to CHAR
ALTER TABLE People
    MODIFY playerID CHAR(9),
    MODIFY retroID CHAR(8),
    MODIFY bbrefID CHAR(9);

-- Convert DOUBLE columns to INT
ALTER TABLE People
    MODIFY birthYear INT,
    MODIFY birthMonth INT,
    MODIFY birthDay INT,
    MODIFY deathYear INT,
    MODIFY deathMonth INT,
    MODIFY deathDay INT,
    MODIFY weight INT,
    MODIFY height INT;

-- Convert bats and throws to ENUM
ALTER TABLE People
    MODIFY bats ENUM ('L', 'R', 'B') DEFAULT NULL,
    MODIFY throws ENUM ('L', 'R', 'S') DEFAULT NULL;

-- Add dateOfBirth and dateOfDeath columns
ALTER TABLE People
    ADD dateOfBirth DATE,
    ADD dateOfDeath DATE;

-- Update dateOfBirth and dateOfDeath columns:
UPDATE People
SET dateOfBirth = CASE
    WHEN birthYear IS NOT NULL AND birthMonth IS NOT NULL AND birthDay IS NOT NULL
    THEN CONCAT(birthYear, '-', birthMonth, '-', birthDay)
    ELSE NULL
END,
dateOfDeath = CASE
```

```
    WHEN deathYear IS NOT NULL AND deathMonth IS NOT NULL AND deathDay IS NOT NULL
    THEN CONCAT(deathYear, '-', deathMonth, '-', deathDay)
    ELSE NULL
END;

-- Convert debut and finalGame to DATE type
ALTER TABLE People
    MODIFY debut DATE,
    MODIFY finalGame DATE;
```

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

Out[131]:
```
[]
```

Managers

The `Managers` table is defined as

```
create table Managers
(
    playerID text   null,
    yearID   bigint null,
    teamID   text   null,
    lgID     text   null,
    inseason bigint null,
    G        bigint null,
    W        bigint null,
    L        bigint null,
    `rank`   bigint null,
    plyrMgr  text   null
);
```

You are to complete the following tasks:

1. Convert `playerID`, `teamID`, and `lgID` to minimally sized `CHAR`.
2. Convert `yearID` to `CHAR(4)`.
3. Convert `plyrMgr` to `BOOLEAN`. This may require creating a temporary column.

You should use `ALTER TABLE` to modify attributes (columns) and `UPDATE TABLE` to modify data (rows).

In [132…

```
%%sql
-- Convert playerID, teamID, and lgID to minimally sized CHAR
ALTER TABLE Managers
    MODIFY playerID CHAR(9),
    MODIFY teamID CHAR(3),
    MODIFY lgID CHAR(2);

-- Convert yearID to CHAR(4)
ALTER TABLE Managers
    MODIFY yearID CHAR(4);

-- Convert plyrMgr to BOOLEAN. This may require creating a temporary column
ALTER TABLE Managers ADD COLUMN plyrMgr_temp TINYINT(1);
UPDATE Managers
SET plyrMgr_temp = CASE
                    WHEN plyrMgr = 'Y' THEN 1
                    WHEN plyrMgr = 'N' THEN 0
                    ELSE NULL
                END;
ALTER TABLE Managers DROP COLUMN plyrMgr;
ALTER TABLE Managers CHANGE plyrMgr_temp plyrMgr TINYINT(1);
```

```
 * mysql+pymysql://root:***@localhost
3684 rows affected.
3684 rows affected.
0 rows affected.
3684 rows affected.
0 rows affected.
0 rows affected.
```

Out[132]:

```
[]
```

Bonus point: MySQL has a `YEAR` type, but we choose to not use it for `yearID`. Can you figure out why?

The range for YEAR type is from 1901 to 2155, but earliest year in yearID is 1871, which makes it impossible for us to use YEAR type.

Appearances

The `Appearances` table is defined as

```
create table Appearances
(
    yearID    bigint null,
    teamID    text   null,
    lgID      text   null,
    playerID  text   null,
    G_all     bigint null,
    GS        double null,
    G_batting bigint null,
    G_defense double null,
    G_p       bigint null,
    G_c       bigint null,
    G_1b      bigint null,
    G_2b      bigint null,
    G_3b      bigint null,
    G_ss      bigint null,
    G_lf      bigint null,
    G_cf      bigint null,
    G_rf      bigint null,
    G_of      bigint null,
    G_dh      double null,
    G_ph      double null,
    G_pr      double null
);
```

You are to complete the following tasks:

1. Convert `yearID` to `CHAR(4)`.
2. Convert `teamID`, `lgID`, and `playerID` to minimally sized `CHAR`.

You should use `ALTER TABLE` to modify attributes (columns) and `UPDATE TABLE` to modify data (rows).

In [133…

```
%%sql
-- Convert yearID to CHAR(4)
ALTER TABLE Appearances
    MODIFY yearID CHAR(4);

-- Convert teamID, lgID, and playerID to minimally sized CHAR
ALTER TABLE Appearances
    MODIFY teamID CHAR(3),
    MODIFY lgID CHAR(2),
    MODIFY playerID CHAR(9);
```

```
 * mysql+pymysql://root:***@localhost
110422 rows affected.
110422 rows affected.
[]
```

Out[133]:

Batting

The `Batting` table is defined as

```
create table Batting
(
    playerID text   null,
    yearID   bigint null,
    stint    bigint null,
    teamID   text   null,
    lgID     text   null,
    G        bigint null,
    AB       bigint null,
    R        bigint null,
    H        bigint null,
    `2B`     bigint null,
    `3B`     bigint null,
    HR       bigint null,
    RBI      double null,
    SB       double null,
    CS       double null,
    BB       bigint null,
```

```
    SO      double null,
    IBB     double null,
    HBP     double null,
    SH      double null,
    SF      double null,
    GIDP    double null
);
```

You are to complete the following tasks:

1. Convert `playerID`, `teamID`, and `lgID` to minimally sized `CHAR`.
2. Convert `yearID` to `CHAR(4)`.

You should use `ALTER TABLE` to modify attributes (columns) and `UPDATE TABLE` to modify data (rows).

In [134…

```
%%sql
-- Convert playerID, teamID, and lgID to minimally sized CHAR
ALTER TABLE Batting
    MODIFY playerID CHAR(9),
    MODIFY teamID CHAR(3),
    MODIFY lgID CHAR(2);

-- Convert yearID to CHAR(4)
ALTER TABLE Batting
    MODIFY yearID CHAR(4);
```

```
 * mysql+pymysql://root:***@localhost
110493 rows affected.
110493 rows affected.
[]
```

Out[134]:

Pitching

The `Pitching` table is defined as

```
create table Pitching
(
    playerID text   null,
```

```
    yearID   bigint null,
    stint    bigint null,
    teamID   text   null,
    lgID     text   null,
    W        bigint null,
    L        bigint null,
    G        bigint null,
    GS       bigint null,
    CG       bigint null,
    SHO      bigint null,
    SV       bigint null,
    IPouts   bigint null,
    H        bigint null,
    ER       bigint null,
    HR       bigint null,
    BB       bigint null,
    SO       bigint null,
    BAOpp    double null,
    ERA      double null,
    IBB      double null,
    WP       bigint null,
    HBP      double null,
    BK       bigint null,
    BFP      double null,
    GF       bigint null,
    R        bigint null,
    SH       double null,
    SF       double null,
    GIDP     double null
);
```

You are to complete the following tasks:

1. Convert `playerID`, `teamID`, and `lgID` to minimally sized `CHAR`.
2. Convert `yearID` to `CHAR(4)`.

You should use `ALTER TABLE` to modify attributes (columns) and `UPDATE TABLE` to modify data (rows).

In [135…

```
%%sql
-- Convert playerID, teamID, and lgID to minimally sized CHAR
ALTER TABLE Pitching
    MODIFY playerID CHAR(9),
    MODIFY teamID CHAR(3),
    MODIFY lgID CHAR(2);

-- Convert yearID to CHAR(4)
ALTER TABLE Pitching
    MODIFY yearID CHAR(4);
```

```
 * mysql+pymysql://root:***@localhost
49430 rows affected.
49430 rows affected.
```

Out[135]:

```
[]
```

Teams

The `Teams` table is defined as

```
create table Teams
(
    yearID         bigint null,
    lgID           text   null,
    teamID         text   null,
    franchID       text   null,
    divID          text   null,
    `Rank`         bigint null,
    G              bigint null,
    Ghome          double null,
    W              bigint null,
    L              bigint null,
    DivWin         text   null,
    WCWin          text   null,
    LgWin          text   null,
    WSWin          text   null,
    R              bigint null,
    AB             bigint null,
```

```
    H              bigint null,
    `2B`           bigint null,
    `3B`           bigint null,
    HR             bigint null,
    BB             double null,
    SO             double null,
    SB             double null,
    CS             double null,
    HBP            double null,
    SF             double null,
    RA             bigint null,
    ER             bigint null,
    ERA            double null,
    CG             bigint null,
    SHO            bigint null,
    SV             bigint null,
    IPouts         bigint null,
    HA             bigint null,
    HRA            bigint null,
    BBA            bigint null,
    SOA            bigint null,
    E              bigint null,
    DP             bigint null,
    FP             double null,
    name           text   null,
    park           text   null,
    attendance     double null,
    BPF            bigint null,
    PPF            bigint null,
    teamIDBR       text   null,
    teamIDlahman45 text   null,
    teamIDretro    text   null
);
```

You are to complete the following tasks:

1. Convert `yearID` to `CHAR(4)`.
2. Convert `lgID`, `teamID`, `franchID`, and `divID` to minimally sized `CHAR`.

You should use `ALTER TABLE` to modify attributes (columns) and `UPDATE TABLE` to modify data (rows).

In [136…
```
%%sql
-- Convert yearID to CHAR(4)
ALTER TABLE Teams
    MODIFY yearID CHAR(4);

-- Convert playerID, teamID, and lgID to minimally sized CHAR
ALTER TABLE Teams
    MODIFY franchID CHAR(3),
    MODIFY teamID CHAR(3),
    MODIFY lgID CHAR(2),
    MODIFY divID CHAR(1);
```

```
 * mysql+pymysql://root:***@localhost
2985 rows affected.
2985 rows affected.
```

Out[136]:
```
[]
```

Primary Keys

Now we need to add primary keys to our tables. In the following cells, write and execute SQL statements that show the column/combination of columns that is a valid primary key for each of the 6 tables.

Recall the properties of primary keys and think about how you could represent them using queries. Note that you aren't simply selecting columns. You need to show **why** they can be a primary key.

answer

For an entity or a combination to be a PK, its values need to be unique and no duplicate. Meanwhile, we want it to be immutable. Below are the queries that checking the duplication for some entities. If no duplicate, it can be set as the PK for the table.

In [166…
```
%%sql -- People
select playerID, count(*) as count
from People group by 1
           having count>1
           order by 2 desc;
```

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

Out[166]:

playerID	count

In [167…

```
%%sql -- Managers
select playerID, yearID, inseason, count(*) as count
from Managers group by 1,2,3
           having count>1
           order by 4 desc;
```

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

Out[167]:

playerID	yearID	inseason	count

In [168…

```
%%sql -- Appearances
select playerID, teamID, yearID, count(*) as count
from Appearances group by 1,2,3
           having count>1
           order by 4 desc;
```

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

Out[168]:

playerID	teamID	yearID	count

In [169…

```
%%sql -- Batting
select playerID, yearID, stint, count(*) as count
from Batting group by 1,2,3
           having count>1
           order by 4 desc;
```

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

Out[169]:

playerID	yearID	stint	count

In [170…

```
%%sql -- Pitching
select playerID, yearID, stint, count(*) as count
from Pitching group by 1,2,3
           having count>1
           order by 4 desc;
```

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

Out[170]:

playerID	yearID	stint	count

In [171…

```
%%sql -- Teams
select yearID, teamID, count(*) as count
from Teams group by 1,2
           having count>1
           order by 3 desc;
```

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

Out[171]:

yearID	teamID	count

Write and execute `ALTER TABLE` statements to add your primary keys to the tables.

In [143…

```
%%sql

alter table people
    add constraint people_pk
        primary key (playerID);

alter table managers
    add constraint managers_pk
        primary key (playerID, yearID, inseason);

alter table appearances
    add constraint appearances_pk
        primary key (playerID, teamID, yearID);

alter table batting
    add constraint batting_pk
        primary key (playerID, yearID, stint);

alter table pitching
    add constraint pitching_pk
        primary key (playerID, yearID, stint);

alter table teams
    add constraint teams_pk
        primary key (yearID, teamID);
```

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

Out[143]:

Foreign Keys

Let's add foreign keys. **The conceptual ER diagram above should indicate to you which tables are related by foreign keys.** In the following cells, write and execute SQL statements that show the column/combination of columns that is a valid foreign key for each of the 6 relationships.

Recall the properties of foreign keys and think about how you could represent them using queries. Note that you aren't simply selecting columns. You need to show **why** they can be a foreign key.

answer

To add a FK, we need to reference table_1 to table_2 using their mutual entities. To do so, it is required that all the values in table_1's entity are in table_2's entity. Below queries are to check this relationship as the reason to set the FK.

In [144…

```sql
%%sql -- people-managers

select * from Managers where playerID not in (select playerID from People) limit 2;
select * from People where playerID not in (select playerID from Managers) limit 2; -- not able to use
```

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

Out[144]:

playerID	birthYear	birthMonth	birthDay	birthCountry	birthState	birthCity	deathYear	deathMonth	deathDay	deathCountry	deathState	deathCity
aardsda01	1981	12	27	USA	CO	Denver	None	None	None	None	None	None
aaronha01	1934	2	5	USA	AL	Mobile	2021	1	22	USA	GA	Atlanta

For entity playerID, all the values in Managers are in People as well. Therefore, we can set the FK between playerId referencing Managers to People.Appearances

In [145…

```
%%sql -- appearances-people

select * from Appearances where playerID not in (select playerID from People) limit 2;
select * from People where playerID not in (select playerID from Appearances) limit 2; -- not able to use
```

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

Out[145]:

playerID	birthYear	birthMonth	birthDay	birthCountry	birthState	birthCity	deathYear	deathMonth	deathDay	deathCountry	deathState	deathCity
actama99	1969	1	11	D.R.	San Pedro de Macoris	San Pedro de Macoris	None	None	None	None	None	None
adairbi99	1913	2	10	USA	AL	Mobile	2002	6	17	USA	AL	Bay Minette

For entity playerID, all the values in Managers are in People as well. Therefore, we can set the FK between playerId referencing Managers to People.Appearances

In [146…

```
%%sql -- appearances-teams

select * from Appearances where (teamID, yearID) not in (select teamID, yearID from Teams) limit 2;
select * from Teams where (teamID, yearID) not in (select teamID, yearID from Appearances) limit 2; -- both work
```

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

Out[146]: yearID lgID teamID franchID divID Rank G GHome W L DivWin WCWin LgWin WSWin R AB H 2B 3B HR BB SO SB CS HBP SF

For entity (teamID, yearID), all the values in Appearances are in Teams as well. Therefore, we can set the FK between (teamID, yearID) referencing Appearances to Teams.

In [147…
```
%%sql -- appearances-batting

select * from Appearances
    where (playerID,teamID, yearID) not in
    (select playerID,teamID, yearID from Batting)
limit 2; -- not able to use

select * from Batting
    where (playerID,teamID, yearID) not in
    (select playerID,teamID, yearID from Appearances)
limit 2;
```

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

Out[147]:

playerID	yearID	stint	teamID	lgID	G	AB	R	H	2B	3B	HR	RBI	SB	CS	BB	SO	IBB	HBP	SH	SF	GIDP

For the combination of (playerID, teamID, yearID), all the values in Batting are in Appearances as well. Therefore, we can set the FK between (playerID, teamID, yearID) referencing Batting to Appearances.

In [148…
```
%%sql -- appearances-pitching

select * from Pitching
    where (playerID,teamID, yearID) not in
    (select playerID,teamID, yearID from Appearances)
limit 2;

select * from Appearances
    where (playerID,teamID, yearID) not in
    (select playerID,teamID, yearID from Pitching)
limit 2; -- not able to use
```

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

Out[148]:

yearID	teamID	lgID	playerID	G_all	GS	G_batting	G_defense	G_p	G_c	G_1b	G_2b	G_3b	G_ss	G_lf	G_cf	G_rf	G_of	G_dh	G_ph	G_pr
1966	ATL	NL	aaronha01	158	157.0	158	158.0	0	0	0	2	0	0	0	5	158	158	0.0	1.0	0.0
1967	ATL	NL	aaronha01	155	152.0	155	152.0	0	0	0	1	0	0	0	11	142	152	0.0	3.0	0.0

For the combination of (playerID, teamID, yearID), all the values in Pitching are in Appearances as well. Therefore, we can set the FK between (playerID, teamID, yearID) referencing Pitching to Appearances.

In [149…

```
%%sql -- teams-managers

select * from Managers where teamID not in (select teamID from Teams) limit 2;
select * from Teams where teamID not in (select teamID from Managers) limit 2; -- both work
```

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

Out[149]:

yearID	lgID	teamID	franchID	divID	Rank	G	Ghome	W	L	DivWin	WCWin	LgWin	WSWin	R	AB	H	2B	3B	HR	BB	SO	SB	CS	HBP	SF

For entity teamID, all the values in Managers are in Teams as well. Therefore, we can set the FK between teamId referencing Managers to Teams.

Write and execute `ALTER TABLE` statements to add your foreign keys to the tables.

In [150…

```
%%sql

alter table Managers
    add constraint Managers_People_playerID_fk
        foreign key (playerID) references People (playerID);

alter table Appearances
    add constraint Appearances_People_playerID_fk
        foreign key (playerID) references People (playerID);

alter table Appearances
    add constraint Appearances_Teams_teamID_yearID_fk
        foreign key (yearID, teamID) references Teams (yearID, teamID);

alter table Managers
    add constraint Managers_Teams_teamID_yearID_fk
```

```
        foreign key (yearID, teamID) references Teams (yearID, teamID);

alter table Batting
    add constraint Batting_Appearances_playerID_yearID_teamID_fk
        foreign key (playerID, teamID, yearID) references Appearances (playerID, teamID, yearID);

alter table Pitching
    add constraint Pitching_Appearances_playerID_yearID_teamI_fk
        foreign key (playerID, teamID, yearID) references Appearances (playerID, teamID, yearID);
```

```
 * mysql+pymysql://root:***@localhost
3684 rows affected.
110422 rows affected.
110422 rows affected.
3684 rows affected.
110493 rows affected.
49430 rows affected.
```

Out[150]: []

SQL Queries

On-Base Percentage and Slugging

The formula for `onBasePercentage` is

\begin{equation} \frac{(H - 2B - 3B - HR) + 2 \times 2B + 3 \times 3B + 4 \times HR)}{AB} \end{equation}
Note that `2B`, `3B`, `HR`, and `AB` are their own columns, not multiplication.

Write a query that returns a table of form

```
(playerID, nameFirst, nameLast, yearID, stint, H, AB, G, onBasePercentage)
```

Your table should be sorted on `onBasePercentage` from highest to lowest, then on last name alphabetically (if there are any ties in `onBasePercentage`). **To avoid freezing your notebook, add a `LIMIT 10` to the end of your query to display only the first 10 rows.**

You may use the `Batting` and `People` tables.

In [157…

```
%%sql

select playerID,
       nameFirst,
       nameLast,
       yearID,
       stint,
       H,
       AB,
       G,
       (((H - 2B - 3B - HR) + 2 * 2B + 3 * 3B + 4 * HR) / AB) as onBasePercentage
from People
        join Batting using (playerID)
order by onBasePercentage desc, nameLast
limit 10
```

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

Out[157]:

playerID	nameFirst	nameLast	yearID	stint	H	AB	G	onBasePercentage
chacigu01	Gustavo	Chacin	2010	1	1	1	44	4.0000
hernafe02	Felix	Hernandez	2008	1	1	1	31	4.0000
lefebbi01	Bill	LeFebvre	1938	1	1	1	1	4.0000
motagu01	Guillermo	Mota	1999	1	1	1	51	4.0000
narumbu01	Buster	Narum	1963	1	1	1	7	4.0000
perrypa02	Pat	Perry	1988	2	1	1	35	4.0000
quirkja01	Jamie	Quirk	1984	2	1	1	1	4.0000
rogered01	Eddie	Rogers	2005	1	1	1	8	4.0000
sleatlo01	Lou	Sleater	1958	1	1	1	4	4.0000
yanes01	Esteban	Yan	2000	1	1	1	43	4.0000

Players and Managers

A person in `People` was a player if their `playerID` appears in `Appearances`. Similarly, a person in `People` was a manager if their `playerID` appears in `Managers`. Note that a person could have been both a player and manager.

Write a query that returns a table of form

```
(playerID, nameFirst, nameLast, careerPlayerGames, careerManagerGames)
```

`careerPlayerGames` is the sum of `Appearances.G_all` for a single player. It should be 0 if the person was never a player.

`careerManagerGames` is the sum of `Managers.G` for a single manager. It should be 0 if the person was never a manager.

Your table should be sorted on `careerPlayerGames + careerManagerGames` from highest to lowest. **To avoid freezing your notebook, add a `LIMIT 10` to the end of your query to display only the first 10 rows.**

You may use the `People`, `Appearances`, and `Managers` tables.

In [162…

```
%%sql

select playerID,
       nameFirst,
       nameLast,
       coalesce(sum(G_all), 0) as careerPlayerGames,
       coalesce(sum(G), 0)     as careerManagerGames
from People
        left join Appearances using (playerID)
        left join Managers using (playerID)
group by 1, 2, 3
order by (careerPlayerGames + careerManagerGames) desc
limit 10
```

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

Out[162]:

playerID	nameFirst	nameLast	careerPlayerGames	careerManagerGames
torrejo01	Joe	Torre	64061	77814
mcgrajo01	John	McGraw	39780	85842
mackco01	Connie	Mack	38372	85305
bakerdu01	Dusty	Baker	48936	70376
dykesji01	Jimmy	Dykes	50226	65164
ansonca01	Cap	Anson	53004	61776
durocle01	Leo	Durocher	42562	67302
pinielo01	Lou	Piniella	40181	63648
clarkfr01	Fred	Clarke	42674	59409
robinfr02	Frank	Robinson	44928	49324

Copy and paste your query from above. Modify it to only show people who were never managers. This should be a one-line change

In [163…

```
%%sql

select playerID,
       nameFirst,
       nameLast,
       coalesce(sum(G_all), 0) as careerPlayerGames,
       coalesce(sum(G), 0)     as careerManagerGames
from People
        left join Appearances using (playerID)
        left join Managers using (playerID)
group by 1, 2, 3
having careerManagerGames = 0
order by (careerPlayerGames + careerManagerGames) desc
limit 10
```

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

Out[163]:

playerID	nameFirst	nameLast	careerPlayerGames	careerManagerGames
yastrca01	Carl	Yastrzemski	3308	0
aaronha01	Hank	Aaron	3298	0
henderi01	Rickey	Henderson	3081	0
murraed02	Eddie	Murray	3026	0
musiast01	Stan	Musial	3026	0
ripkeca01	Cal	Ripken	3001	0
mayswi01	Willie	Mays	2992	0
bondsba01	Barry	Bonds	2986	0
winfida01	Dave	Winfield	2973	0
pujolal01	Albert	Pujols	2971	0

In [ ]: