COMS W4111: Introduction to Databases
Spring 2024, Sections 002/V02

Homework 2: Nonprogramming

Introduction

This notebook contains HW2 Nonprogramming. **Only students 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.

Setup

SQL Magic

In [1]:
```
%load_ext sql
```

You may need to change the password below.

In [2]:
```
%sql mysql+pymysql://root:dbuserdbuser@localhost
```

In [3]:
```
%sql SELECT 1
```

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

Out[3]:

1
1

Python Libraries

In [4]:
```
import os

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

You may need to change the password below.

In [5]:
```
engine = create_engine("mysql+pymysql://root:dbuserdbuser@localhost")
```

Load Data

- We're going to load data into a new database called `s24_lahmans_hw2`
- The data is stored as CSV files in the `data/` directory.

In [6]:
```
%sql DROP SCHEMA IF EXISTS s24_lahmans_hw2
%sql CREATE SCHEMA s24_lahmans_hw2
```

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

Out[6]:
```
[]
```

In [7]:
```
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 o
```

```
    :param table_name: The name of the table to create. If the name is None, the function uses the name o
        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 [8]:
```
data_dir = "data"
csv_files = [
    "People.csv",
    "Appearances.csv",
    "Batting.csv",
    "Pitching.csv",
    "Teams.csv",
    "Managers.csv",
]
schema = "s24_lahmans_hw2"

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

```
Loaded file: People.csv
Loaded file: Appearances.csv
Loaded file: Batting.csv
Loaded file: Pitching.csv
Loaded file: Teams.csv
Loaded file: Managers.csv
```

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
- You will fix these issues

In [9]:
```
%sql USE s24_lahmans_hw2
```

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

Out[9]:
```
[]
```

In [10]:
```
# Visualize people
%sql DESCRIBE People
```

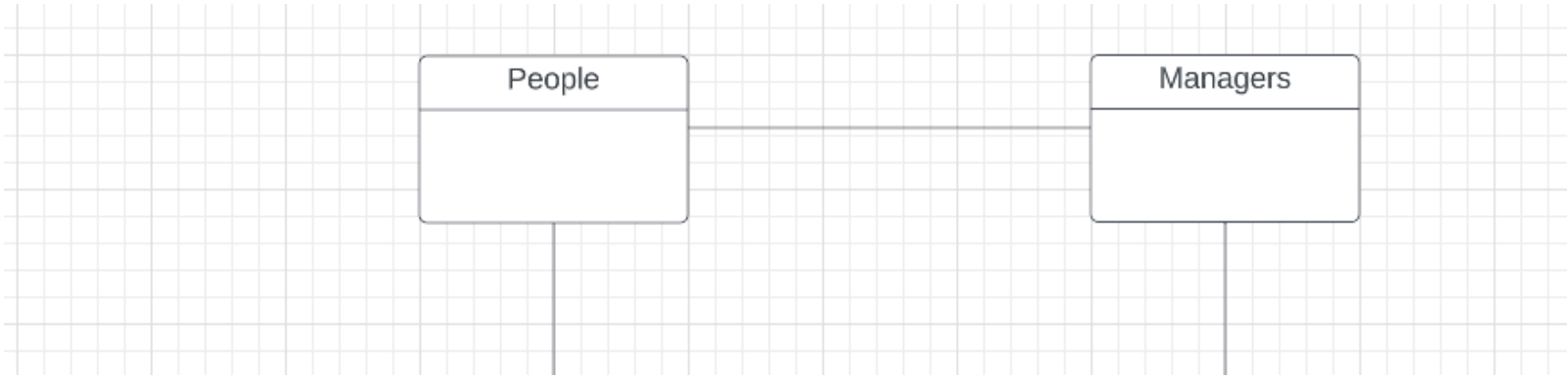
```
 * mysql+pymysql://root:***@localhost
24 rows affected.
```

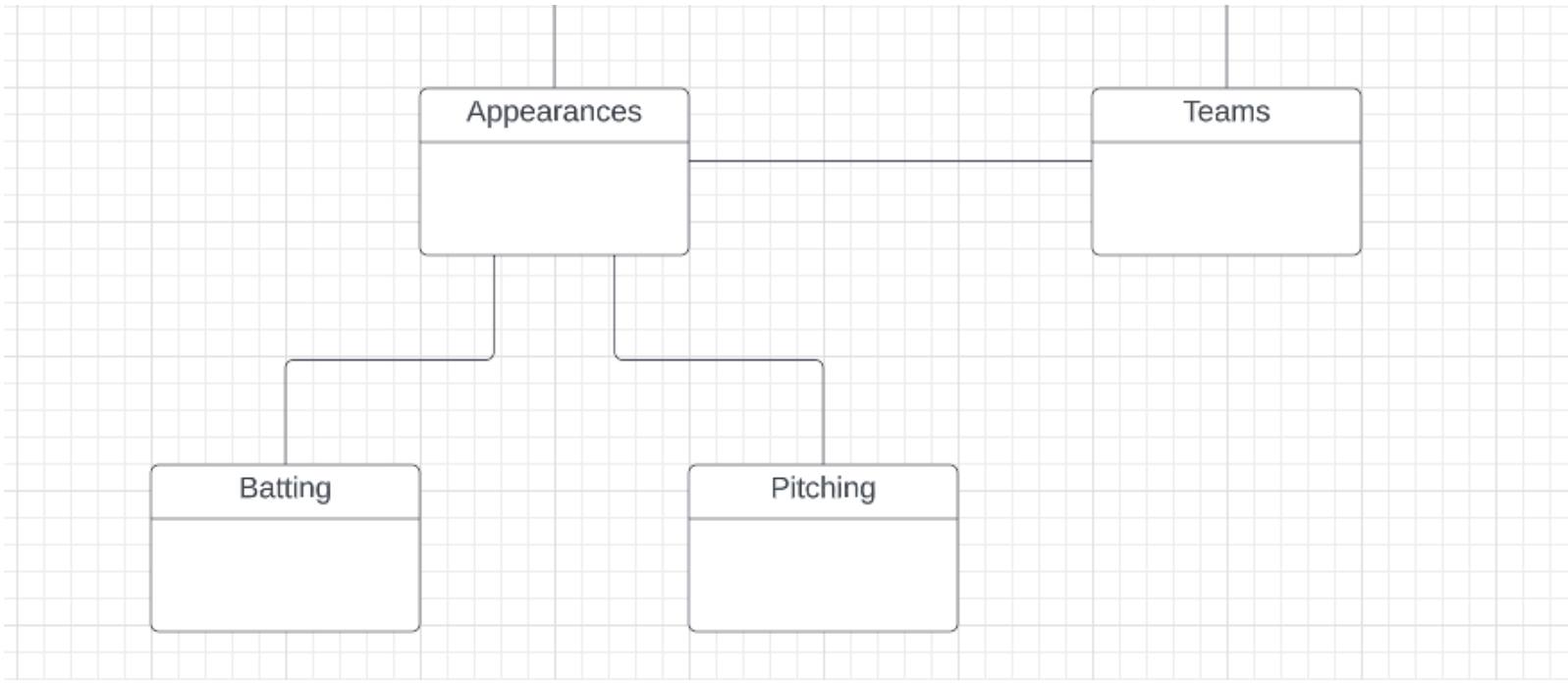
Out[10]:

Field	Type	Null	Key	Default	Extra
playerID	text	YES		None	
birthYear	double	YES		None	
birthMonth	double	YES		None	
birthDay	double	YES		None	
birthCountry	text	YES		None	
birthState	text	YES		None	
birthCity	text	YES		None	
deathYear	double	YES		None	
deathMonth	double	YES		None	

deathDay	double	YES	None
deathCountry	text	YES	None
deathState	text	YES	None
deathCity	text	YES	None
nameFirst	text	YES	None
nameLast	text	YES	None
nameGiven	text	YES	None
weight	double	YES	None
height	double	YES	None
bats	text	YES	None
throws	text	YES	None
debut	text	YES	None
finalGame	text	YES	None
retroID	text	YES	None
bbrefID	text	YES	None

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





Lahmans Database

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
);
```

1. Convert `playerID`, `retroID`, and `bbrefID` to **minimally sized** `CHAR`
   A. 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)
   B. `playerID`, `retroID`, and `bbrefID` may have different minimal sizes
   C. You don't need to show how you got the minimal sizes
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 [32]:
```
%%sql

SELECT MAX(LENGTH(playerID)) FROM people;

SELECT MAX(LENGTH(retroID)) FROM people;

SELECT MAX(LENGTH(bbrefID)) FROM people;

SELECT distinct(throws) FROM people;
```

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

Out[32]:

throws
R
L
None
S

In [17]:
```
%%sql

ALTER TABLE People
MODIFY COLUMN playerID CHAR(9),
MODIFY COLUMN retroID CHAR(8),
MODIFY COLUMN bbrefID CHAR(9);


ALTER TABLE People
MODIFY COLUMN birthYear INT,
MODIFY COLUMN birthMonth INT,
MODIFY COLUMN birthDay INT,
MODIFY COLUMN deathYear INT,
MODIFY COLUMN deathMonth INT,
MODIFY COLUMN deathDay INT,
MODIFY COLUMN weight INT,
MODIFY COLUMN height INT;


ALTER TABLE People
```

```
MODIFY COLUMN bats ENUM('R', 'L', 'B'),
MODIFY COLUMN throws ENUM('R', 'L', 'S');


ALTER TABLE People
ADD COLUMN dateOfBirth DATE,
ADD COLUMN dateOfDeath DATE;

UPDATE People
SET dateOfBirth = STR_TO_DATE(CONCAT(birthYear, '-', birthMonth, '-', birthDay), '%Y-%m-%d'),
    dateOfDeath = STR_TO_DATE(CONCAT(deathYear, '-', deathMonth, '-', deathDay), '%Y-%m-%d');


ALTER TABLE People
MODIFY COLUMN debut DATE,
MODIFY COLUMN 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[17]: []

In [18]:
```
%%sql
select dateOfBirth, dateOfDeath from People limit 10;
```

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

Out[18]:

dateOfBirth	dateOfDeath
1981-12-27	None
1934-02-05	2021-01-22
1939-08-05	1984-08-16
1954-09-08	None

1972-08-25	None
1985-12-17	None
1850-11-04	1905-05-17
1877-04-15	1957-01-06
1869-11-11	1962-06-11
1866-10-14	1926-04-27

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
);
```

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 [19]:
```
%%sql

SELECT MAX(LENGTH(playerID)) FROM managers;

SELECT MAX(LENGTH(teamID)) FROM managers;

SELECT MAX(LENGTH(lgID)) FROM managers;
```

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

Out[19]:

MAX(LENGTH(lgID))
2

In [20]:
```
%%sql

ALTER TABLE managers
MODIFY COLUMN playerID CHAR(9),
MODIFY COLUMN teamID CHAR(3),
MODIFY COLUMN lgID CHAR(2);

# Convert `yearID` to `CHAR(4)`
ALTER TABLE managers
MODIFY COLUMN yearID CHAR(4);

# Convert `plyrMgr` to `BOOLEAN`. This may require creating a temporary column.
ALTER TABLE managers
ADD COLUMN boolPlayer BOOLEAN;

UPDATE managers
SET boolPlayer = IF(plyrMgr = 'Y', TRUE, FALSE);

ALTER TABLE managers
DROP COLUMN plyrMgr;

# Convert the name of the `boolPlayer` column to `plyrMgr`.
```

```
ALTER TABLE managers
CHANGE COLUMN boolPlayer plyrMgr BOOLEAN;

select * from managers limit 5;
```

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

Out[20]:

playerID	yearID	teamID	lgID	inseason	G	W	L	rank	plyrMgr
wrighha01	1871	BS1	None	1	31	20	10	3	1
woodji01	1871	CH1	None	1	28	19	9	2	1
paborch01	1871	CL1	None	1	29	10	19	8	1
lennobi01	1871	FW1	None	1	14	5	9	8	1
deaneha01	1871	FW1	None	2	5	2	3	8	1

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

MySQL displays YEAR values in YYYY format, with a range of 1901 to 2155. In our dataset, we have some yearID values less than 1901, such as 1871. Therefore, we cannot represent those values if we use yearID, which is why we instead use CHAR(4) as it has more flexibility and allows us to encode years earlier than 1901.

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
);
```

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 [21]:
```
%%sql

SELECT MAX(LENGTH(playerID)) FROM appearances;

SELECT MAX(LENGTH(teamID)) FROM appearances;

SELECT MAX(LENGTH(lgID)) FROM appearances;
```

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

Out[21]:

MAX(LENGTH(lgID))
2

In [22]:
```
%%sql

ALTER TABLE appearances
MODIFY COLUMN playerID CHAR(9),
MODIFY COLUMN teamID CHAR(3),
MODIFY COLUMN lgID CHAR(2);

# Convert `yearID` to `CHAR(4)`
ALTER TABLE appearances
MODIFY COLUMN yearID CHAR(4);
```

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

Out[22]:
```
[]
```

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
);
```

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 [23]:
```
%%sql

ALTER TABLE batting
MODIFY COLUMN playerID CHAR(9),
MODIFY COLUMN teamID CHAR(3),
MODIFY COLUMN lgID CHAR(2);

# Convert `yearID` to `CHAR(4)`
ALTER TABLE batting
MODIFY COLUMN yearID CHAR(4);
```

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

Out[23]:
```
[]
```

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
```

```
    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
);
```

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 [24]:
```
%%sql

ALTER TABLE pitching
MODIFY COLUMN playerID CHAR(9),
MODIFY COLUMN teamID CHAR(3),
MODIFY COLUMN lgID CHAR(2);

# Convert `yearID` to `CHAR(4)`
ALTER TABLE pitching
MODIFY COLUMN yearID CHAR(4);
```

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

Out[24]:
```
[]
```

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
);
```

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 [25]:
```
%%sql

SELECT Max(LENGTH(teamID)) FROM teams;

SELECT Max(LENGTH(lgID)) FROM teams;

SELECT Max(LENGTH(franchID)) FROM teams;

SELECT Max(LENGTH(teamIDBR)) FROM teams;
```

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

Out[25]:

Max(LENGTH(teamIDBR))
3

In [26]:
```
%%sql

ALTER TABLE teams
MODIFY COLUMN teamID CHAR(3),
MODIFY COLUMN lgID CHAR(2),
```

```
MODIFY COLUMN franchID CHAR(3),
MODIFY COLUMN teamIDBR CHAR(3);

# Convert `yearID` to `CHAR(4)`
ALTER TABLE teams
MODIFY COLUMN yearID CHAR(4);
```

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

Out[26]:
```
[]
```

Primary Keys

- You will now add primary keys to the tables
- The PKs for the tables are
  - People: `playerID`
  - Managers: `(playerID, yearID, inseason)`
  - Appearances: `(playerID, yearID, teamID)`
  - Batting: `(playerID, yearID, stint)`
  - Pitching: `(playerID, yearID, stint)`
  - Teams: `(teamID, yearID)`
- Write and execute statements showing why `(playerID, yearID, teamID)` is a valid PK for Appearances
  - You should show that the PK is non-null for all rows and unique across all rows

In [22]:
```
%%sql

# Show the above two side by side
SELECT COUNT(*) AS count_all_rows, COUNT(DISTINCT playerID, yearID, teamID) AS count_distinct_rows FROM a
```

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

Out[22]:

count_all_rows	count_distinct_rows
110422	110422

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

In [33]:
```
%%sql

ALTER TABLE people
ADD PRIMARY KEY (playerID);

ALTER TABLE managers
ADD PRIMARY KEY (playerID, yearID, inseason);

ALTER TABLE appearances
ADD PRIMARY KEY (playerID, yearID, teamID);

ALTER TABLE batting
ADD PRIMARY KEY (playerID, yearID, stint);

ALTER TABLE pitching
ADD PRIMARY KEY (playerID, yearID, stint);

ALTER TABLE teams
ADD PRIMARY KEY (teamID, yearID);
```

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

Out[33]:
```
[]
```

Foreign Keys

- You will now add foreign keys to the tables
- **The conceptual ER diagram above should indicate to you which tables are related by foreign keys**
  - You need to figure out which table in a relationship has the foreign key
- Write and execute statements showing why `Appearances.playerID` is a valid FK referencing `People.playerID`
  - You should show that all the values in `Appearances.playerID` appear in `People.playerID`

In [34]:
```
%%sql

SELECT a.playerID
FROM Appearances a
LEFT JOIN People p ON a.playerID = p.playerID
WHERE p.playerID IS NULL;
```

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

Out[34]:

playerID

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

In [36]:
```
%%sql
ALTER TABLE appearances
ADD CONSTRAINT appearance_people_fk
FOREIGN KEY(playerID)
REFERENCES people(playerID);

ALTER TABLE appearances
ADD CONSTRAINT appearance_teams_fk
FOREIGN KEY(teamID, yearID)
REFERENCES teams(teamID, yearID);
```

```
ALTER TABLE batting
ADD CONSTRAINT batting_appearances_fk
FOREIGN KEY (playerID, yearID, teamID)
REFERENCES appearances(playerID, yearID, teamID);

ALTER TABLE pitching
ADD CONSTRAINT pitching_appearances_fk
FOREIGN KEY (playerID, yearID, teamID)
REFERENCES appearances(playerID, yearID, teamID);

ALTER TABLE managers
ADD CONSTRAINT managers_teams_fk
FOREIGN KEY (teamID, yearID)
REFERENCES teams(teamID, yearID);

ALTER TABLE managers
ADD CONSTRAINT managers_people_fk
FOREIGN KEY (playerID)
REFERENCES people(playerID);
```

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

Out[36]: []

In [37]:
```
%%sql

SELECT TABLE_NAME, COLUMN_NAME, CONSTRAINT_NAME, REFERENCED_TABLE_NAME, REFERENCED_COLUMN_NAME
FROM INFORMATION_SCHEMA.KEY_COLUMN_USAGE
WHERE REFERENCED_TABLE_SCHEMA = 's24_lahmans_hw2' AND TABLE_NAME = 'appearances';
```

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

Out[37]:

TABLE_NAME	COLUMN_NAME	CONSTRAINT_NAME	REFERENCED_TABLE_NAME	REFERENCED_COLUMN_NAME

_	_	_	_	_	_	_
appearances	playerID	appearance_people_fk		people		playerID
appearances	teamID	appearance_teams_fk		teams		teamID
appearances	yearID	appearance_teams_fk		teams		yearID

SQL Queries

On-Base Percentage and Slugging

- The formula for `onBasePercentage` is

$$\frac{(H-2B-3B-HR)+2\times 2B+3\times 3B+4\times HR)}{AB} \tag{1}$$

- `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 [40]:
```
%%sql

SELECT
b.playerID,
p.nameFirst,
p.nameLast,
b.yearID,
b.stint,
b.H,
b.AB,
b.G,
CASE
    WHEN b.AB > 0 THEN
        ((b.H - b.`2b` - b.`3b` - b.hr) +
            b.`2B`*2 + 3*b.`3B` + 4 * b.HR) / b.AB
    ELSE 0
END AS obp
```

```
FROM Batting b
JOIN People p ON b.playerID = p.playerID
ORDER BY obp DESC, p.nameLast ASC
limit 10;
```

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

Out[40]:

playerID	nameFirst	nameLast	yearID	stint	H	AB	G	obp
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`
- A person in `People` was a manager if their `playerID` appears in `Managers`
- 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 [43]:
```
%%sql
-- This is the query that I used to answer the question later.

SELECT playerID, G_all AS careerPlayerGames
    FROM Appearances
    where G_all<1;


select
    p.playerID,
    p.nameFirst,
    p.nameLast,
    a.careerPlayerGames AS careerPlayerGames
from
people p
LEFT JOIN (
    SELECT playerID, SUM(G_all) AS careerPlayerGames
    FROM Appearances
    GROUP BY playerID
) a ON p.playerID = a.playerID
limit 5;
```

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

5 rows affected.

Out[43]:

playerID	nameFirst	nameLast	careerPlayerGames
aardsda01	David	Aardsma	331
aaronha01	Hank	Aaron	3298
aaronto01	Tommie	Aaron	437
aasedo01	Don	Aase	448
abadan01	Andy	Abad	15

In [44]:

```
%%sql

select
    p.playerID,
    p.nameFirst,
    p.nameLast,
    IFNULL(a.careerPlayerGames, 0) AS careerPlayerGames,
    IFNULL(m.careerManagerGames, 0) AS careerManagerGames
FROM People p
LEFT JOIN (
    SELECT playerID, SUM(G_all) AS careerPlayerGames
    FROM Appearances
    GROUP BY playerID
) a ON p.playerID = a.playerID
LEFT JOIN (
    SELECT playerID, SUM(G) AS careerManagerGames
    FROM Managers
    GROUP BY playerID
) m ON p.playerID = m.playerID
ORDER BY (IFNULL(a.careerPlayerGames, 0) + IFNULL(m.careerManagerGames, 0)) DESC
LIMIT 10;
```

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

Out[44]:

playerID	nameFirst	nameLast	careerPlayerGames	careerManagerGames
mackco01	Connie	Mack	724	7755
torrejo01	Joe	Torre	2209	4323
mcgrajo01	John	McGraw	1105	4769
bakerdu01	Dusty	Baker	2039	3704
harribu01	Bucky	Harris	1262	4410
larusto01	Tony	LaRussa	132	5248
durocle01	Leo	Durocher	1637	3739
pinielo01	Lou	Piniella	1747	3536
dykesji01	Jimmy	Dykes	2283	2962
clarkfr01	Fred	Clarke	2246	2829

- 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 [45]:
```
%%sql

select
    p.playerID,
    p.nameFirst,
    p.nameLast,
    IFNULL(a.careerPlayerGames, 0) AS careerPlayerGames,
    IFNULL(m.careerManagerGames, 0) AS careerManagerGames
FROM People p
LEFT JOIN (
    SELECT playerID, SUM(G_all) AS careerPlayerGames
    FROM Appearances
    GROUP BY playerID
) a ON p.playerID = a.playerID
LEFT JOIN (
    SELECT playerID, SUM(G) AS careerManagerGames
    FROM Managers
    GROUP BY playerID
) m ON p.playerID = m.playerID
WHERE m.careerManagerGames IS NULL
ORDER BY (IFNULL(a.careerPlayerGames, 0) + IFNULL(m.careerManagerGames, 0)) DESC
LIMIT 10;
```

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

Out[45]:

playerID	nameFirst	nameLast	careerPlayerGames	careerManagerGames
yastrca01	Carl	Yastrzemski	3308	0
aaronha01	Hank	Aaron	3298	0
henderi01	Rickey	Henderson	3081	0
musiast01	Stan	Musial	3026	0
murraed02	Eddie	Murray	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