COMS W4111-002 (Fall 2021)
Introduction to Databases

Homework 1: Programming - 10 Points

Note: Please replace the information below with your last name, first name and UNI.

Sidharta, Aditya Kelvianto, aks2266

Introduction

Objectives

This homework has you practice and build skill with:

- PART A: (1 point) Understanding relational databases
- PART B: (1 point) Understanding relational algebra
- PART C: (1 point) Cleaning data
- PART D: (1 point) Performing simple SQL queries to analyze the data.
- PART E: (6 points) CSVDataTable.py

Note: The motivation for PART E may not be clear. The motivation will become clearer as the semester proceeds. The purpose of PART E is to get you started on programming in Python and manipulating data.

Submission

1. File > Print Preview > Download as PDF
2. Upload .pdf and .ipynb to GradeScope
3. Upload CSVDataTable.py and CSVDataTable_Tests.py

This assignment is due September 24, 11:59 pm ET

Collaboration

- You may use any information you get in TA or Prof. Ferguson's office hours, from lectures or from recitations.
- You may use information that you find on the web.
- You are NOT allowed to collaborate with other students outside of office hours.

Part A: Written

1. What is a database management system?

   Database Management System is the system in the application architecture that sits in the storage layer. It is responsible to store the data of an application. Moreover, it is also responsible to manipulate / transform the data and return it to the server layer whenever it receives a Database call from the backend system in the application architecture

2. What is a primary key and why is it important?

   Knowing the primary key means that you are able to identify the rest of the information of the tuple in the relation. Thus, Primary Key is chosen to identify each tuple in the relation. Primary key is also important for the case of deduplication, as primary key must be unique

3. Please explain the differences between SQL, MySQL Server and DataGrip?

   Structured Query Language is a type of language used for the purpose of defining data in database (DDL) or accessing and transforming the data that is currently stored in the database (DML)

   MySQL Server is a type of relational database management system, so it is a system that responsible for store, update, and output data as requested by the interacting system. The other system uses SQL in order to interact with MySQL Server

   DataGrip is an Integrated Development Environment for Database systems. It allows us to develop our SQL scripts more effectively whenever we want to access the data inside our MySQL Server

4. What are 4 different types of DBMS table relationships, give a brief explanaition for each?

   1.) One to One relationship means that each row in the left attribute / table only corresponds to either one row in the right table, or none.

   2.) One to Many relationship means that each row in the left attribute / table only corresponds to zero to more than one row in the right table.

   3.) Many to One relationship means that multiple rows with the same values in the left attribute / table only corresponds to zero to one row in the right table.

   4.) Many to Many relationship means that multiple rows with the same values in the left attribute / table corresponds to zero to more than one row in the right table.

5. What is an ER model?

   ER model is a way to model the relationship between our data using three basic concepts: entity, relationship, and attributes. Entity is the atomic subjects of our data. Relationship indicates how different objects of our data interact with one another. Attributes indicates the information available in our data for each of our entities /

objects.

6. Using Lucidchart draw an example of a logical ER model using Crow's Foot notation for Columbia classes. The entity types are:
    - Students, Professors, and Classes.
    - The relationships are:
        - A Class has exactly one Professor.
        - A Student has exactly one professor who is an *advisor.*
        - A Professor may advise 0, 1 or many Students.
        - A Class has 0, 1 or many enrolled students.
        - A Student enrolls in 0, 1 or many Classes.

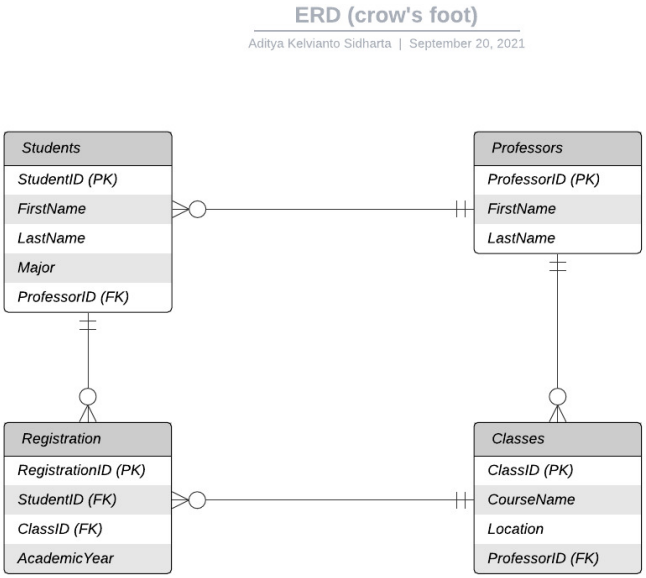
- You can define what you think are common attributes for each of the entity types. Do not define more than 5 or 6 attributes per entity type.

- In this example, explicitly show an example of a primary-key, foreign key, one-to-many relationship, and many-to-many relationship.

Notes:

- If you have not already done so, please register for a free account at Lucidchart.com. You can choose the option at the bottom of the left pane to add the ER diagram shapes.
- You can take a screen capture of you diagram and save in the zip directory that that contains you Jupyter notebook. Edit the following cell and replace "Boromir.jpg" with the name of the file containing your screenshot.

Use the following line to upload a photo of your Luicdchart.



Note:
There are two comments with regards to this diagram:

1.) I am assuming that Professor are able to teach zero to many classes

2.) I am adding another entity set, Registration, to handle the many-to-many relationship between student and class. Handling this without adding another entity will cause us to require composite key (ClassStudentID), which is certainly reduces the database performance and eligibility

Part B: Relational Algebra

You will use the online relational calculator (https://dbis-uibk.github.io/relax/landing), choose the “Karlsruhe University of Applied Sciences” dataset.

An anti-join is a form of join with reverse logic. Instead of returning rows when there is a match (according to the join predicate) between the left and right side, an anti-join returns those rows from the left side of the predicate for which there is no match on the right.

The Anti-Join Symbol is ▷.

Consider the following relational algebra expression and result.

/* (1) Set X = The set of classrooms in buildings Taylor or Watson. */

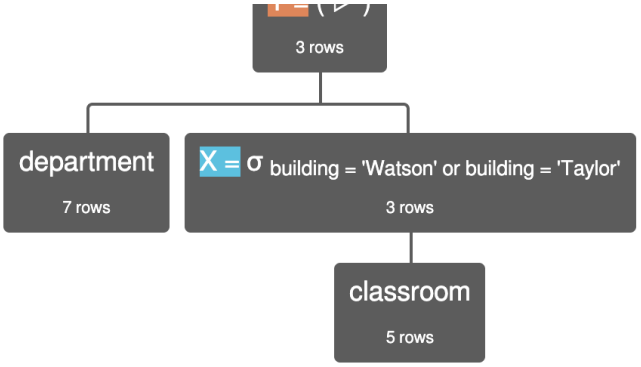
```
X = σ building='Watson' ∨ building='Taylor' (classroom)
```

/* (2) Set Y = The Anti-Join of department and X */

```
Y = (department ▷ X)
```

/* (3) Display the rows in Y. */

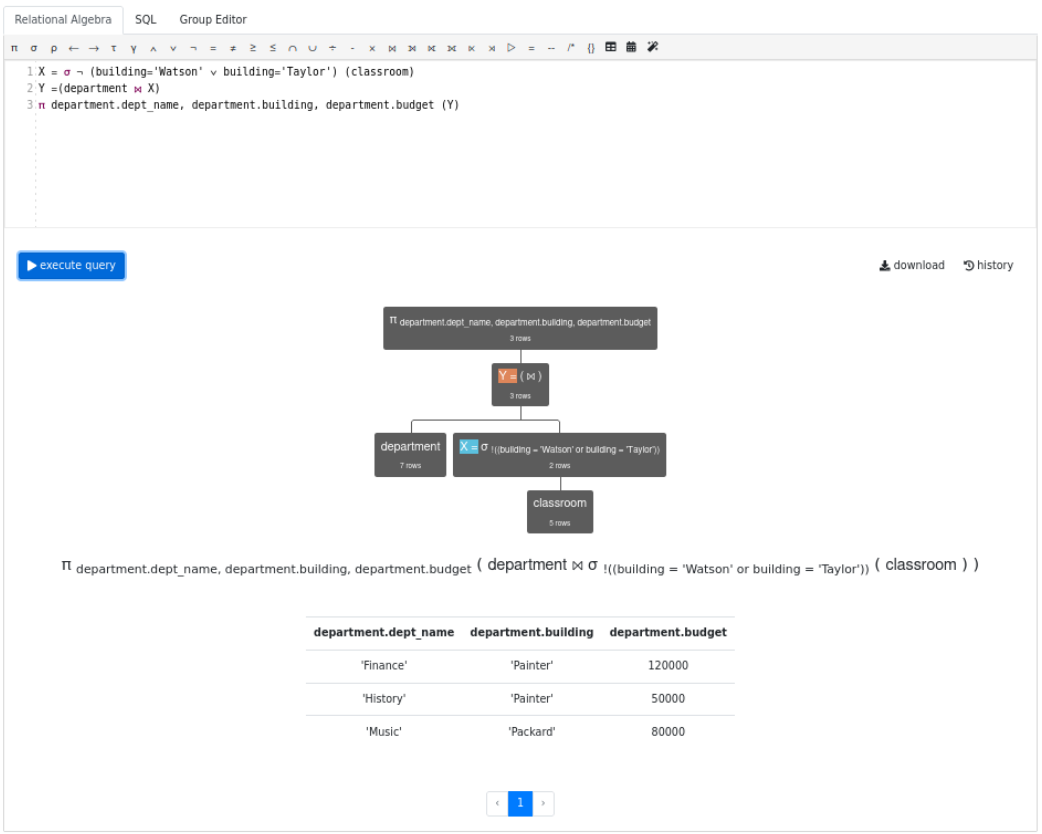
```
Y
```



$$(\text{department} \triangleright \sigma_{\text{building = 'Watson' or building = 'Taylor'}} (\text{classroom}))$$

department.dept_name	department.building	department.budget
'Finance'	'Painter'	120000
'History'	'Painter'	50000
'Music'	'Packard'	80000

1. Find an alternate expression to (2) that computes the correct answer given X. Display the execution of your query below.



Part C: Data Clean Up

Please note: You MUST make a new schema using the lahmansdb_to_clean.sql file provided in the data folder.

Use thelahmansdb_to_clean.sql file to make a new schema containing the raw data. The lahman database you created in Homework 0 has already been cleaned with all the constraints and will be used for Part D. Knowing how to clean data and add integrity constraints is very important which is why you go through the steps in part C.

TLDR: If you use the HW0 lahman schema for this part you will get a lot of errors and recieve a lot of deductions.

In [4]:
```
# You will need to follow instructions from HW 0 to make a new schema
# Connect to the unclean schema below by setting the database host, u
%load_ext sql
%sql mysql+pymysql://root:password@127.0.0.1/lahmansdb_to_clean
```

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

Data cleanup: For each table we want you to clean, we have provided a list of changes you have to make. You can reference the cleaned lahman db for inspiration and guidance, but know that there are different ways to clean the data and you will be graded for your choice rationalization. You should make these changes through DataGrip's workbench's table editor and/or using SQL queries. In this part you will clean two tables: People and Batting.

You must have:

- A brief explanation of why you think the change we requested is important.
- What change you made to the table.
- Any queries you used to make the changes, either the ones you wrote or the Alter statements provided by DataGrip's editor.
- Executed the test statements we provided
- The cleaned table's new create statement (after you finish all the changes)

Overview of Changes:

People Table

0. Primary Key (Explanation is given, but you still must add the key to your table yourself)
1. Empty strings to NULLs
2. Column typing
3. isDead column
4. deathDate and birthDate column

Batting Table

1. Empty strings to NULLs
2. Column typing

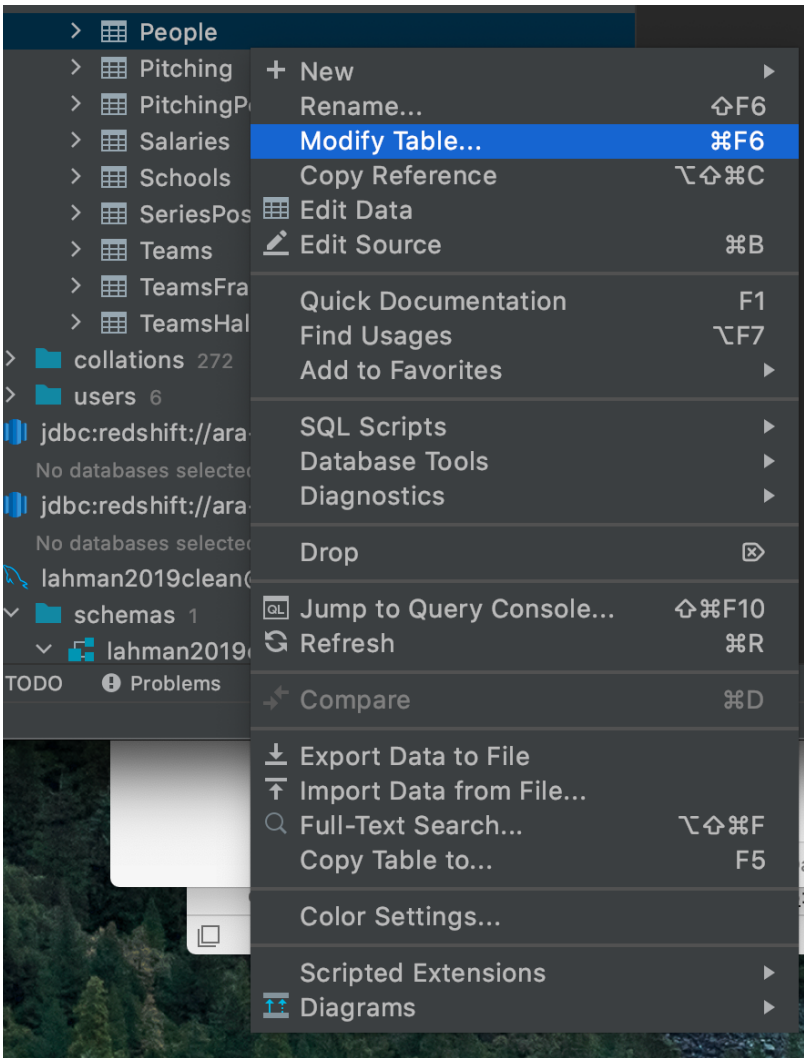
3. Primary Key
4. Foreign Key

How to make the changes:

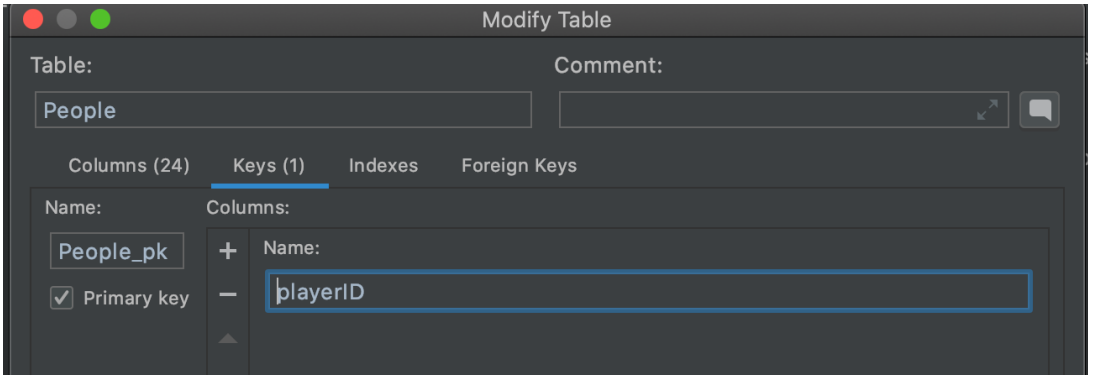
Using the Table Editor:

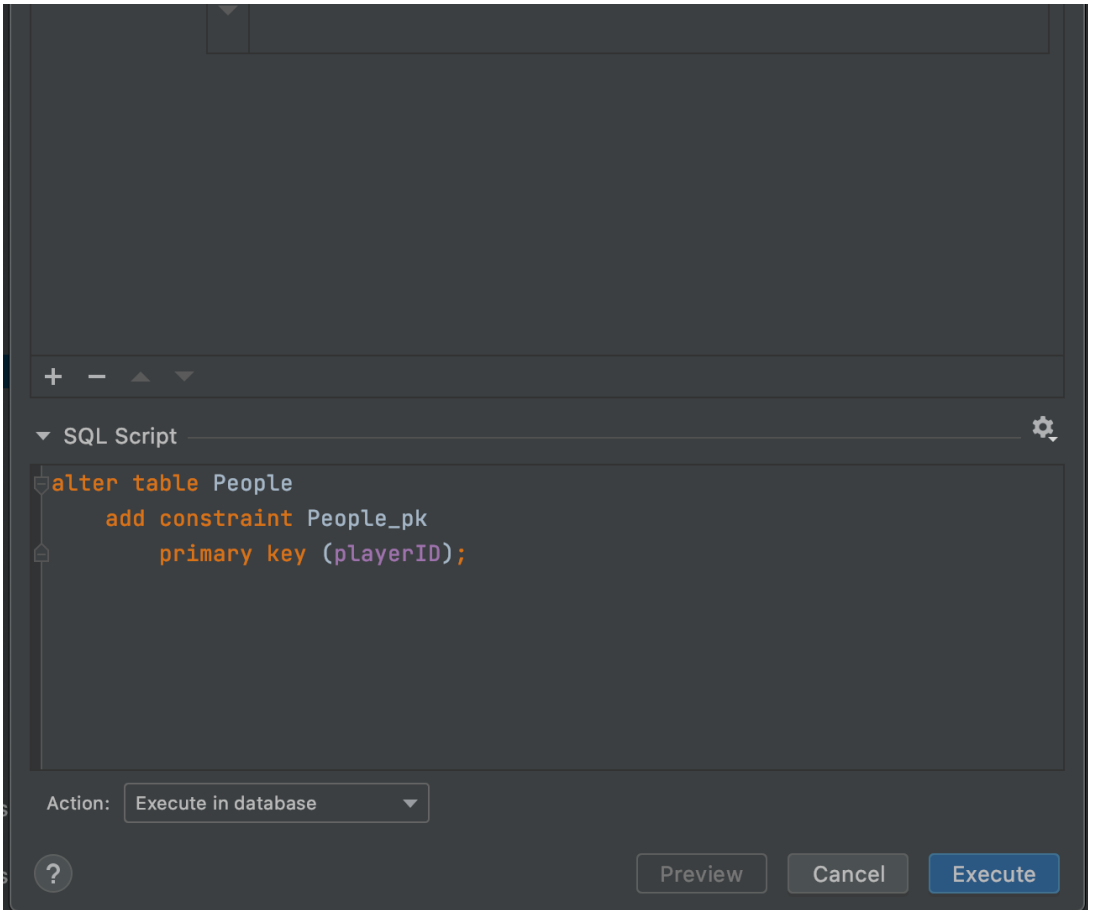
When you hit apply, a popup will open displaying the ALTER statments sql generates. Copy the sql provided first and paste it into this notebook. Then you can apply the changes. This means that you are NOT executing the ALTER statements through your notebook.

1. Right click on the table > Modify Table...



2. Keys > press the + button > input the parameters > Execute OR Keys > press the + button > input the parameters > copy and paste the script generated under "SQL Script" and paste into your notebook > Run the cell in jupyter notebook





Using sql queries:

Copy paste any queries that you write manually into the notebook as well!

People Table

0) EXAMPLE: Add a Primary Key

(Solutions are given but make sure you still do this step in workbench!)

Explanation

We want to add a Primary Key because we want to be able to uniquely identify rows within our data. A primary key is also an index, which allows us to locate data faster.

Change

I added a Primary Key on the playerID column and made the datatype VARCHAR(15)

Note: This is for demonstration purposes only. playerID **is not** a primary key for fielding.

SQL

```
ALTER TABLE `lahmansdb_to_clean`.`people`
CHANGE COLUMN `playerID` `playerID` VARCHAR(15) NOT NULL ,
ADD PRIMARY KEY (`playerID`);
```

Tests

In [5]:
```
%sql SHOW KEYS FROM people WHERE Key_name = 'PRIMARY'
```

```
 * mysql+pymysql://root:***@127.0.0.1/lahmansdb_to_clean
1 rows affected.
```

Out[5]:

Table	Non_unique	Key_name	Seq_in_index	Column_name	Collation	Cardinality	Sub_par
people	0	PRIMARY	1	playerID	A	20093	None

1) Convert all empty strings to NULL

Explanation

We want to make sure that undefined value is represented consistently. Having it as empty string to represent null makes it unclear whether its an undefined value, or it contains information of empty string

Change

Convert the following column to have NULL for all empty strings

birthYear birthMonth birthDay birthCountry birthState birthCity deathYear deathMonth deathDay deathCountry deathState deathCity nameFirst nameLast nameGiven weight height bats throws debut finalGame retroID bbrefID

SQL

```
UPDATE people
set birthYear = NULLIF(birthYear, '');
UPDATE people
set birthMonth = NULLIF(birthMonth, '');
UPDATE people
set birthDay = NULLIF(birthDay, '');
UPDATE people
set birthCountry = NULLIF(birthCountry, '');
UPDATE people
set birthState = NULLIF(birthState, '');
UPDATE people
set birthCity = NULLIF(birthCity, '');
UPDATE people
set deathYear = NULLIF(deathYear, '');
UPDATE people
set deathMonth = NULLIF(deathMonth, '');
UPDATE people
set deathDay = NULLIF(deathDay, '');
UPDATE people
set deathCountry = NULLIF(deathCountry, '');
UPDATE people
set deathState = NULLIF(deathState, '');
UPDATE people
set deathCity = NULLIF(deathCity, '');
UPDATE people
set nameFirst = NULLIF(nameFirst, '');
UPDATE people
set nameLast = NULLIF(nameLast, '');
UPDATE people
set nameGiven = NULLIF(nameGiven, '');
UPDATE people
set weight = NULLIF(weight, '');
```

Tests

In [6]:
```
%sql SELECT * FROM people WHERE birthState = ""
```

```
 * mysql+pymysql://root:***@127.0.0.1/lahmansdb_to_clean
0 rows affected.
```

Out[6]:

playerID	birthYear	birthMonth	birthDay	birthCountry	birthState	birthCity	deathYear	deatl

2) Change column datatypes to appropriate values (ENUM, INT, VARCHAR, DATETIME, ETC)

Explanation

Changing column datatype to the appropriate values allows us to 1.) Efficiently store the data 2.) Ensuring that future row have correct set of values 3.) Easier filtering and selecting, especially for number / dates

Change

Changing birthYear, birthMonth, birthDay, deathYear, deathMonth, deathDay, weight, height to INT

Changing birthCountry, birthState, birthCity, deathCountry, deathState, deathCity, nameFirst, nameLast, nameGiven, bats, throws, retroID, bbrefID to VARCHAR with 255 bytes

Changing bats, throws to DATE

SQL

```
alter table people modify birthYear INT null;

alter table people modify birthMonth INT null;

alter table people modify birthDay INT null;

alter table people modify birthCountry varchar(255) null;

alter table people modify birthState varchar(255) null;

alter table people modify birthCity varchar(255) null;
```

3) Add an isDead Column that is either 'Y' or 'N'

- Some things to think of: What data type should this column be? How do you know if the player is dead or not? Maybe you do not know if the player is dead.
- You do not need to make guesses about life spans, etc. Just apply a simple rule.

'Y' means the player is dead

'N' means the player is alive

Explanation

Adding isDead Column by checking the deathYear, deathMonth, and deathDay column. If any of them is not null, return 'Y' for the isDead column. Else, return 'N'

Change

Adding isDead Column to indicate whether the person has died

SQL

```
alter table people
    add isDead varchar(255) null;
UPDATE people
set isDead = IF((deathYear IS NOT NULL OR deathMonth IS NOT
NULL or deathDay IS NOT NULL), 'Y', 'N');
```

Tests

In [6]:
```
%sql SELECT * FROM people WHERE isDead = "N" limit 10
```

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

Out[6]:

playerID	birthYear	birthMonth	birthDay	birthCountry	birthState	birthCity	deathYear

aardsda01	1981	12	27	USA	CO	Denver	None
aaronha01	1934	2	5	USA	AL	Mobile	None
aasedo01	1954	9	8	USA	CA	Orange	None
abadan01	1972	8	25	USA	FL	Palm Beach	None
abadfe01	1985	12	17	D.R.	La Romana	La Romana	None
abbotgl01	1951	2	16	USA	AR	Little Rock	None
abbotje01	1972	8	17	USA	GA	Atlanta	None
abbotji01	1967	9	19	USA	MI	Flint	None
abbotku01	1969	6	2	USA	OH	Zanesville	None
abbotky01	1968	2	18	USA	MA	Newburyport	None

In [6]:
```
%sql SELECT * FROM people WHERE isDead = "N" limit 10
```

```
 * mysql+pymysql://dbuser:***@localhost/lahman2019clean
10 rows affected.
```

Out[6]:

playerID	birthYear	birthMonth	birthDay	birthCountry	birthState	birthCity	deathYear
aardsda01	1981	12	27	USA	CO	Denver	None
aaronha01	1934	2	5	USA	AL	Mobile	None
aasedo01	1954	9	8	USA	CA	Orange	None
abadan01	1972	8	25	USA	FL	Palm Beach	None
abadfe01	1985	12	17	D.R.	La Romana	La Romana	None
abbotgl01	1951	2	16	USA	AR	Little Rock	None
abbotje01	1972	8	17	USA	GA	Atlanta	None
abbotji01	1967	9	19	USA	MI	Flint	None
abbotku01	1969	6	2	USA	OH	Zanesville	None
abbotky01	1968	2	18	USA	MA	Newburyport	None

4) Add a deathDate and birthDate column

Some things to think of: What do you do if you are missing information? What datatype

should this column be?

You have to create this column from other columns in the table.

The datatype of deathDate and birthDate column should be Date. this value will only be field if and only if the respective Year, Month, and Day column is filled.

Change

Adding deathDate and birthDate column using data from deathYear, deathMonth, deathDay, birthYear, birthMonth, and birthDay

SQL

```
alter table people
    add deathDate datetime null;

alter table people
    add birthDate datetimeD null;

UPDATE people
set deathDate = IF((deathYear IS NULL OR deathMonth IS NULL
or deathDay IS NULL),
    NULL,
    STR_TO_DATE(CONCAT(deathYear, '-', deathMonth, '-', deat
hDay), "%Y-%m-%d"));

UPDATE people
set birthDate = IF((birthYear IS NULL OR birthMonth IS NULL
or birthDay IS NULL),
    NULL,
    STR_TO_DATE(CONCAT(birthYear, '-', birthMonth, '-', birt
hDay), "%Y-%m-%d"));
```

Tests

In [9]:
```
%sql SELECT deathDate FROM people WHERE deathDate >= '2005-01-01' OR
```

```
 * mysql+pymysql://dbuser:***@localhost/lahman2019clean
10 rows affected.
```

Out[9]:

deathDate
2005-01-04 00:00:00
2005-01-07 00:00:00
2005-01-09 00:00:00

2005-01-10 00:00:00
2005-01-21 00:00:00
2005-01-22 00:00:00
2005-01-31 00:00:00
2005-02-04 00:00:00
2005-02-08 00:00:00
2005-02-11 00:00:00

In [10]:
```
%sql SELECT birthDate FROM people WHERE birthDate <= '1965-01-01' ORD
```

```
 * mysql+pymysql://dbuser:***@localhost/lahman2019clean
10 rows affected.
```

Out[10]:

birthDate
1820-04-17 00:00:00
1824-10-05 00:00:00
1832-02-25 00:00:00
1832-09-17 00:00:00
1832-10-23 00:00:00
1835-01-10 00:00:00
1836-02-29 00:00:00
1837-12-26 00:00:00
1838-03-10 00:00:00
1838-07-16 00:00:00

Final CREATE Statement

To find the create statement:

- Right click on the table name in workbench
- Select 'Copy to Clipboard'
- Select 'Create Statement'

The create statement will now be copied into your clipboard and can be pasted into the cell below.

```
create table lahmansdb_to_clean.people
(
    playerID varchar(15) not null
        primary key,
    birthYear int null,
    birthMonth int null,
    birthDay int null,
    birthCountry varchar(255) null,
    birthState varchar(255) null,
    birthCity varchar(255) null,
    deathYear int null,
    deathMonth int null,
    deathDay int null,
    deathCountry varchar(255) null,
    deathState varchar(255) null,
    deathCity varchar(255) null,
    nameFirst varchar(255) null,
    nameLast varchar(255) null,
    nameGiven varchar(255) null,
    weight int null,
    height int null,
    bats varchar(255) null,
    throws varchar(255) null
```

Batting Table

1) Convert all empty strings to NULL

Explanation

We want to make sure that undefined value is represented consistently. Having it as empty string to represent null makes it unclear whether its an undefined value, or it contains information of empty string

Change

Convert the following column to have NULL for all empty strings

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

SQL

```
UPDATE batting
set playerID = NULLIF(playerID, '');
UPDATE batting
set yearID = NULLIF(yearID, '');
UPDATE batting
set stint = NULLIF(stint, '');
UPDATE batting
set teamID = NULLIF(teamID, '');
UPDATE batting
set lgID = NULLIF(lgID, '');
UPDATE batting
set G = NULLIF(G, '');
UPDATE batting
set AB = NULLIF(AB, '');
UPDATE batting
set R = NULLIF(R, '');
UPDATE batting
set H = NULLIF(H, '');
UPDATE batting
set 2B = NULLIF(2B, '');
UPDATE batting
set 3B = NULLIF(3B, '');
UPDATE batting
set HR = NULLIF(HR, '');
UPDATE batting
set RBI = NULLIF(RBI, '');
UPDATE batting
set SB = NULLIF(SB, '');
UPDATE batting
set CS = NULLIF(CS, '');
UPDATE batting
set BB = NULLIF(BB, '');
UPDATE batting
set SO = NULLIF(SO, '');
UPDATE batting
set IBB = NULLIF(IBB, '');
UPDATE batting
set HBP = NULLIF(HBP, '');
UPDATE batting
set SH = NULLIF(SH, '');
UPDATE batting
set SF = NULLIF(SF, '');
```

Tests

In [8]:
```
%sql SELECT count(*) FROM batting where RBI is NULL;
```

```
 * mysql+pymysql://root:***@127.0.0.1/lahmansdb_to_clean
1 rows affected.
```

Out[8]:

count(*)

756

2) Change column datatypes to appropriate values (ENUM, INT, VARCHAR, DATETIME, ETC)

Explanation

Changing column datatype to the appropriate values allows us to 1.) Efficiently store the data 2.) Ensuring that future row have correct set of values 3.) Easier filtering and selecting, especially for number / dates

Change

Changing all columns to INT, except PlayerID, teamID, and lgID was changed to VARCHAR(255)

SQL

```
alter table batting modify playerID VARCHAR(255) null;

alter table batting modify yearID INT null;

alter table batting modify stint INT null;

alter table batting modify teamID VARCHAR(255) null;

alter table batting modify lgID VARCHAR(255) null;

alter table batting modify G INT null;

alter table batting modify AB INT null;

alter table batting modify R INT null;
```

3) Add a Primary Key

Two options for the Primary Key:

- Composite Key: playerID, yearID, stint
- Covering Key (Index): playerID, yearID, stint, teamID

Choice

We want to add a Primary Key because we want to be able to uniquely identify rows within our data. A primary key is also an index, which allows us to locate data faster.

Indexes are solely created for the purpose of making query faster

Explanation

I am making playerID, yearID, and stint as a composite key.

PlayerID, yearID, stint, and teamID are covering keys

SQL

```
create index batting_playerID_yearID_stint_teamID_index
    on batting (playerID, yearID, stint, teamID);

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

Test

In [11]:
```
%sql SHOW KEYS FROM batting WHERE Key_name = 'PRIMARY' and Column_nam
```

```
 * mysql+pymysql://root:***@127.0.0.1/lahmansdb_to_clean
1 rows affected.
```

Out[11]:

Table	Non_unique	Key_name	Seq_in_index	Column_name	Collation	Cardinality	Sub_par
batting	0	PRIMARY	1	playerID	A	19801	None

4) Add a foreign key on playerID between the People and Batting Tables

Note: Two people in the batting table do not exist in the people table. How should you handle this issue?

Explanation

Foreign key is used to link the data between different tables / relations

Change

Adding foreign key playerID in the batting table which references the playerID column in the people table

SQL

```
alter table batting
    add constraint batting_people_playerID_fk
        foreign key (playerID) references people (playerID);
```

Tests

In [13]:
```
%sql Select playerID from batting where playerID not in (select playe
```

```
 * mysql+pymysql://root:***@127.0.0.1/lahmansdb_to_clean
0 rows affected.
```

Out[13]:

playerID

Final CREATE Statement

To find the create statement:

- Right click on the table name in workbench
- Select 'Copy to Clipboard'
- Select 'Create Statement'

The create statement will now be copied into your clipboard and can be pasted into the cell

below.

```
create table lahmansdb_to_clean.batting
(
    playerID varchar(255) not null,
    yearID int not null,
    stint int not null,
    teamID varchar(255) null,
    lgID varchar(255) null,
    G int null,
    AB int null,
    R int null,
    H int null,
    `2B` int null,
    `3B` int null,
    HR int null,
    RBI int null,
    SB int null,
    CS int null,
    BB int null,
    SO int null,
    IBB int null,
    HBP int null,
    SH int null,
    SF int null,
    GIDP int null,
    primary key (playerID, yearID, stint),
    constraint batting_people_playerID_fk
        foreign key (playerID) references lahmansdb_to_clea
n.people (playerID)
);

create index batting_playerID_yearID_stint_teamID_index
    on lahmansdb_to_clean.batting (playerID, yearID, stint,
teamID);
```

Part D: SQL Queries

NOTE: You must use the CLEAN lahman schema provided in HW0 for the queries below to ensure your answers are consistent with the solutions.

In [15]:
```
%reload_ext sql
%sql mysql+pymysql://root:password@127.0.0.1/lahmansbaseballdb
```

Question 0

What is the average salary in baseball history?

In [16]:
```
%sql SELECT AVG(SALARY) FROM salaries
```

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

Out[16]:

AVG(SALARY)
2085634.053125473

Question 1

Select the players with a first name of Sam who were born in the United States and attended college.

Include their first name, last name, playerID, school ID, yearID and birth state. Limit 10

Hint: Use a Join between People and CollegePlaying

In [24]:
```
%%sql
SELECT nameFirst, nameLast, people.playerID, schoolID, yearID, birthS
FROM people
LEFT JOIN collegeplaying ON (people.playerID = collegeplaying.playerI
where nameFirst = "Sam"
and schoolID IS NOT NULL
LIMIT 10
```

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

Out[24]:

nameFirst	nameLast	playerID	schoolID	yearID	birthState
Sam	Barnes	barnesa01	auburn	1918	AL
Sam	Barnes	barnesa01	auburn	1919	AL
Sam	Barnes	barnesa01	auburn	1920	AL
Sam	Barnes	barnesa01	auburn	1921	AL
Sam	Bowens	bowensa01	tennst	1956	NC
Sam	Bowens	bowensa01	tennst	1957	NC
Sam	Bowens	bowensa01	tennst	1958	NC
Sam	Bowen	bowensa02	gacoast	1971	GA
Sam	Bowen	bowensa02	gacoast	1972	GA
Sam	Brown	brownsa01	grovecity	1899	PA

Question 2

Update all entries with full_name Columbia University to 'Columbia University in the City of New York' in the Schools table. Then select the row.

In [25]:
```
%%sql
```

```
UPDATE schools
SET name_full = "Columbia University in the City of New York"
WHERE name_full = "Columbia University";

SELECT *
FROM schools
WHERE name_full = "Columbia University in the City of New York";
```

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

Out[25]:

schoolID	name_full	city	state	country
columbia	Columbia University in the City of New York	New York	NY	USA

Part E: CSVDataTable

i. Conceptual Questions

The purpose of this homework is to teach you the behaviour of SQL Databases by asking you to implement functions that will model the behaviour of a real database with CSVDataTable. You will mimic a SQL Database using CSV files.

Read through the scaffolding code provided in the CSVDataTable folder first to understand and answer the following conceptual questions.

1. Given this SQL statement:

   ```
   SELECT nameFirst, nameLast FROM people WHERE playerID = collied01
   ```

   If you run find_by_primary_key() on this statement, what are key_fields and field_list?

   key_fields = ['collied01'], field_list = ['nameFirst', 'nameLast']

2. What should be checked when you are trying to INSERT a new row into a table with a PK?

   You need to check whether the primary key within the new row to be inserted exists in the table. If yes, abort the operation and return an exception

3. What should be checked when you are trying to UPDATE a row in a table with a PK?

   You need the check whether the update operation will cause a duplicate in the primary key columns. If yes, abort and cancel the operation, and return 0, as no rows are updated

ii. Coding

You are responsible for implementing and testing two classes in Python: CSVDataTable,

BaseDataTable. The python files and data can be found in the assignment under Courseworks.

We have already given you **find_by_template(self, template, field_list=None, limit=None, offset=None, order_by=None)** Use this as a jumping off point for the rest of your functions.

Methods to complete:

CSVDataTable.py

- find_by_primary_key(self, key_fields, field_list=None)
- delete_by_key(self, key_fields)
- delete_by_template(self, template)
- update_by_key(self, key_fields, new_values)
- update_by_template(self, template, new_values)
- insert(self, new_record) CSV_table_tests.py
- You must test all methods. You will have to write these tests yourself.
- You must test your methods on the People and Batting table.

If you do not include tests and tests outputs 50% of this section's points will be deducted at the start

iii. Testing

Please copy the text from the output of your tests and paste it below:

DEBUG:root:CSVDataTable.**init**: data = { "table_name": "People", "connect_info": { "directory": "/home/adityasidharta/columbia/database/hw1/hw1 /4111_s21_hw1_programming_CSVDataTable/data/Baseball", "file_name": "People.csv" }, "key_columns": ["playerID"], "debug": true } DEBUG:root:CSVDataTable._load: Loaded 19619 rows

find_by_primary_key(): Known Record {'playerID': 'aardsda01', 'birthYear': '1981', 'birthMonth': '12', 'birthDay': '27', 'birthCountry': 'USA', 'birthState': 'CO', 'birthCity': 'Denver', 'deathYear': '', 'deathMonth': '', 'deathDay': '', 'deathCountry': '', 'deathState': '', 'deathCity': '', 'nameFirst': 'David', 'nameLast': 'Aardsma', 'nameGiven': 'David Allan', 'weight': '215', 'height': '75', 'bats': 'R', 'throws': 'R', 'debut': '2004-04-06', 'finalGame': '2015-08-23', 'retroID': 'aardd001', 'bbrefID': 'aardsda01'}

find_by_primary_key(): Unknown Record None

find_by_template(): Known Template [{'playerID': 'aardsda01', 'birthYear': '1981', 'birthMonth': '12', 'birthDay': '27', 'birthCountry': 'USA', 'birthState': 'CO', 'birthCity': 'Denver', 'deathYear': '', 'deathMonth': '', 'deathDay': '', 'deathCountry': '', 'deathState': '', 'deathCity': '', 'nameFirst': 'David', 'nameLast': 'Aardsma', 'nameGiven': 'David Allan', 'weight': '215', 'height': '75', 'bats': 'R', 'throws': 'R', 'debut': '2004-04-06', 'finalGame': '2015-08-23', 'retroID': 'aardd001', 'bbrefID': 'aardsda01'}]

find_by_template(): Unknown Template []

delete_by_key(): Known Record 1

delete_by_key(): Unknown Record 0

delete_by_template(): Known Template 1

delete_by_template(): Unknown Template 0

update_by_key(): Known Record - Change nameGiven to add Dr. {'playerID': 'aaronha01', 'birthYear': '1934', 'birthMonth': '2', 'birthDay': '5', 'birthCountry': 'USA', 'birthState': 'AL', 'birthCity': 'Mobile', 'deathYear': '', 'deathMonth': '', 'deathDay': '', 'deathCountry': '', 'deathState': '', 'deathCity': '', 'nameFirst': 'Hank', 'nameLast': 'Aaron', 'nameGiven': 'Henry Louis', 'weight': '180', 'height': '72', 'bats': 'R', 'throws': 'R', 'debut': '1954-04-13', 'finalGame': '1976-10-03', 'retroID': 'aaroh101', 'bbrefID': 'aaronha01'} 1 {'playerID': 'aaronha01', 'birthYear': '1934', 'birthMonth': '2', 'birthDay': '5', 'birthCountry': 'USA', 'birthState': 'AL', 'birthCity': 'Mobile', 'deathYear': '', 'deathMonth': '', 'deathDay': '', 'deathCountry': '', 'deathState': '', 'deathCity': '', 'nameFirst': 'Hank', 'nameLast': 'Aaron', 'nameGiven': 'Dr. Henry Louis', 'weight': '180', 'height': '72', 'bats': 'R', 'throws': 'R', 'debut': '1954-04-13', 'finalGame': '1976-10-03', 'retroID': 'aaroh101', 'bbrefID': 'aaronha01'}

update_by_key(): Unknown Record 0

update_by_key(): Known Record, Problematic Primary Key {'playerID': 'aaronha01', 'birthYear': '1934', 'birthMonth': '2', 'birthDay': '5', 'birthCountry': 'USA', 'birthState': 'AL', 'birthCity': 'Mobile', 'deathYear': '', 'deathMonth': '', 'deathDay': '', 'deathCountry': '', 'deathState': '', 'deathCity': '', 'nameFirst': 'Hank', 'nameLast': 'Aaron', 'nameGiven': 'Dr. Henry Louis', 'weight': '180', 'height': '72', 'bats': 'R', 'throws': 'R', 'debut': '1954-04-13', 'finalGame': '1976-10-03', 'retroID': 'aaroh101', 'bbrefID': 'aaronha01'} 0 {'playerID': 'aaronha01', 'birthYear': '1934', 'birthMonth': '2', 'birthDay': '5', 'birthCountry': 'USA', 'birthState': 'AL', 'birthCity': 'Mobile', 'deathYear': '', 'deathMonth': '', 'deathDay': '', 'deathCountry': '', 'deathState': '', 'deathCity': '', 'nameFirst': 'Hank', 'nameLast': 'Aaron', 'nameGiven': 'Dr. Henry Louis', 'weight': '180', 'height': '72', 'bats': 'R', 'throws': 'R', 'debut': '1954-04-13', 'finalGame': '1976-10-03', 'retroID': 'aaroh101', 'bbrefID': 'aaronha01'}

update_by_template(): Known Record - Change nameGiven to add Dr. [{'playerID': 'aaronto01', 'birthYear': '1939', 'birthMonth': '8', 'birthDay': '5', 'birthCountry': 'USA', 'birthState': 'AL', 'birthCity': 'Mobile', 'deathYear': '1984', 'deathMonth': '8', 'deathDay': '16', 'deathCountry': 'USA', 'deathState': 'GA', 'deathCity': 'Atlanta', 'nameFirst': 'Tommie', 'nameLast': 'Aaron', 'nameGiven': 'Tommie Lee', 'weight': '190', 'height': '75', 'bats': 'R', 'throws': 'R', 'debut': '1962-04-10', 'finalGame': '1971-09-26', 'retroID': 'aarot101', 'bbrefID': 'aaronto01'}] 1 [{'playerID': 'aaronto01', 'birthYear': '1939', 'birthMonth': '8', 'birthDay': '5', 'birthCountry': 'USA', 'birthState': 'AL', 'birthCity': 'Mobile', 'deathYear': '1984', 'deathMonth': '8', 'deathDay': '16', 'deathCountry': 'USA', 'deathState': 'GA', 'deathCity': 'Atlanta', 'nameFirst': 'Tommie', 'nameLast': 'Aaron', 'nameGiven': 'Dr. Tommie Lee', 'weight': '190', 'height': '75', 'bats': 'R', 'throws': 'R', 'debut': '1962-04-10', 'finalGame': '1971-09-26', 'retroID': 'aarot101', 'bbrefID': 'aaronto01'}]

update_by_template(): Unknown Record 0

update_by_template(): Known Record, Problematic Primary Key [{'playerID': 'aaronto01', 'birthYear': '1939', 'birthMonth': '8', 'birthDay': '5', 'birthCountry': 'USA', 'birthState': 'AL', 'birthCity': 'Mobile', 'deathYear': '1984', 'deathMonth': '8', 'deathDay': '16', 'deathCountry': 'USA', 'deathState': 'GA', 'deathCity': 'Atlanta', 'nameFirst': 'Tommie', 'nameLast': 'Aaron', 'nameGiven': 'Dr. Tommie Lee', 'weight': '190', 'height': '75', 'bats': 'R', 'throws': 'R', 'debut': '1962-04-10', 'finalGame': '1971-09-26', 'retroID': 'aarot101', 'bbrefID': 'aaronto01'}] 0

[{'playerID': 'aaronto01', 'birthYear': '1939', 'birthMonth': '8', 'birthDay': '5', 'birthCountry': 'USA', 'birthState': 'AL', 'birthCity': 'Mobile', 'deathYear': '1984', 'deathMonth': '8', 'deathDay': '16', 'deathCountry': 'USA', 'deathState': 'GA', 'deathCity': 'Atlanta', 'nameFirst': 'Tommie', 'nameLast': 'Aaron', 'nameGiven': 'Dr. Tommie Lee', 'weight': '190', 'height': '75', 'bats': 'R', 'throws': 'R', 'debut': '1962-04-10', 'finalGame': '1971-09-26', 'retroID': 'aarot101', 'bbrefID': 'aaronto01'}]

insert(): Success 19617 None 19618

DEBUG:root:CSVDataTable.**init**: data = { "table_name": "Batting", "connect_info": { "directory": "/home/adityasidharta/columbia/database/hw1/hw1 /4111_s21_hw1_programming_CSVDataTable/data/Baseball", "file_name": "Batting.csv" }, "key_columns": ["playerID", "yearID", "stint"], "debug": true }

An error occurred: new_record have primary key values equal to the ones in the current table 19618

DEBUG:root:CSVDataTable._load: Loaded 105861 rows

find_by_primary_key(): Known Record {'playerID': 'abercda01', 'yearID': '1871', 'stint': '1', 'teamID': 'TRO', 'lgID': 'NA', 'G': '1', 'AB': '4', 'R': '0', 'H': '0', '2B': '0', '3B': '0', 'HR': '0', 'RBI': '0', 'SB': '0', 'CS': '0', 'BB': '0', 'SO': '0', 'IBB': '', 'HBP': '', 'SH': '', 'SF': '', 'GIDP': '0'}

find_by_primary_key(): Unknown Record None

find_by_template(): Known Template [{'playerID': 'abercda01', 'yearID': '1871', 'stint': '1', 'teamID': 'TRO', 'lgID': 'NA', 'G': '1', 'AB': '4', 'R': '0', 'H': '0', '2B': '0', '3B': '0', 'HR': '0', 'RBI': '0', 'SB': '0', 'CS': '0', 'BB': '0', 'SO': '0', 'IBB': '', 'HBP': '', 'SH': '', 'SF': '', 'GIDP': '0'}]

find_by_template(): Unknown Template []

delete_by_key(): Known Record 1

delete_by_key(): Unknown Record 0

delete_by_template(): Known Template 1

delete_by_template(): Unknown Template 0

update_by_key(): Known Record - Change teamID to Columbia {'playerID': 'allisar01', 'yearID': '1871', 'stint': '1', 'teamID': 'CL1', 'lgID': 'NA', 'G': '29', 'AB': '137', 'R': '28', 'H': '40', '2B': '4', '3B': '5', 'HR': '0', 'RBI': '19', 'SB': '3', 'CS': '1', 'BB': '2', 'SO': '5', 'IBB': '', 'HBP': '', 'SH': '', 'SF': '', 'GIDP': '1'} 1 {'playerID': 'allisar01', 'yearID': '1871', 'stint': '1', 'teamID': 'COLUMBIA', 'lgID': 'NA', 'G': '29', 'AB': '137', 'R': '28', 'H': '40', '2B': '4', '3B': '5', 'HR': '0', 'RBI': '19', 'SB': '3', 'CS': '1', 'BB': '2', 'SO': '5', 'IBB': '', 'HBP': '', 'SH': '', 'SF': '', 'GIDP': '1'}

update_by_key(): Unknown Record 0

update_by_key(): Known Record, Problematic Primary Key {'playerID': 'allisar01', 'yearID': '1871', 'stint': '1', 'teamID': 'COLUMBIA', 'lgID': 'NA', 'G': '29', 'AB': '137', 'R': '28', 'H': '40', '2B': '4', '3B': '5', 'HR': '0', 'RBI': '19', 'SB': '3', 'CS': '1', 'BB': '2', 'SO': '5', 'IBB': '', 'HBP': '', 'SH': '', 'SF': '', 'GIDP': '1'} 0 {'playerID': 'allisar01', 'yearID': '1871', 'stint': '1', 'teamID': 'COLUMBIA', 'lgID': 'NA', 'G': '29', 'AB': '137', 'R': '28', 'H': '40', '2B': '4', '3B': '5', 'HR': '0', 'RBI': '19', 'SB': '3', 'CS': '1', 'BB': '2', 'SO': '5', 'IBB': '', 'HBP': '', 'SH': '', 'SF': '', 'GIDP': '1'}

update_by_template(): Known Record - Change teamID to Columbia [{'playerID': 'allisdo01', 'yearID': '1871', 'stint': '1', 'teamID': 'WS3', 'lgID': 'NA', 'G': '27', 'AB': '133', 'R': '28', 'H': '44', '2B': '10', '3B': '2', 'HR': '2', 'RBI': '27', 'SB': '1', 'CS': '1', 'BB': '0', 'SO': '2', 'IBB': '', 'HBP': '', 'SH': '', 'SF': '', 'GIDP': '0'}] 1 [{'playerID': 'allisdo01', 'yearID': '1871', 'stint': '1', 'teamID': 'COLUMBIA', 'lgID': 'NA', 'G': '27', 'AB': '133', 'R': '28', 'H': '44', '2B': '10', '3B': '2', 'HR': '2', 'RBI': '27', 'SB': '1', 'CS': '1', 'BB': '0', 'SO': '2', 'IBB': '', 'HBP': '', 'SH': '', 'SF': '', 'GIDP': '0'}]

update_by_template(): Unknown Record 0

update_by_template(): Known Record, Problematic Primary Key [{'playerID': 'allisdo01', 'yearID': '1871', 'stint': '1', 'teamID': 'COLUMBIA', 'lgID': 'NA', 'G': '27', 'AB': '133', 'R': '28', 'H': '44', '2B': '10', '3B': '2', 'HR': '2', 'RBI': '27', 'SB': '1', 'CS': '1', 'BB': '0', 'SO': '2', 'IBB': '', 'HBP': '', 'SH': '', 'SF': '', 'GIDP': '0'}] 0 [{'playerID': 'allisdo01', 'yearID': '1871', 'stint': '1', 'teamID': 'COLUMBIA', 'lgID': 'NA', 'G': '27', 'AB': '133', 'R': '28', 'H': '44', '2B': '10', '3B': '2', 'HR': '2', 'RBI': '27', 'SB': '1', 'CS': '1', 'BB': '0', 'SO': '2', 'IBB': '', 'HBP': '', 'SH': '', 'SF': '', 'GIDP': '0'}]

insert(): Success 105859 None 105860

insert(): Duplicate Primary Key 105860 An error occurred: new_record have primary key

In [ ]: