COMS W4111-002 (Spring 2021) Introduction to Databases

Homework 1: Programming - 10 Points

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

Kao_Shuoting, sk4920

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 > Download as > PDF via latex (.PDF) (Use the "File" menu option on the Jupyter notebook tool bar, not the option on the browser tool bar).
2. Upload .pdf and .ipynb to GradeScope
3. Upload CSVDataTable.py and CSVDataTable_Tests.py

This assignment is due January 29, 11:59 pm EDT

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?

   Basically, it is an interface program that allows users to define, describe, and manipulate data efficiently and conveniently.

   Quoted from the textbook page 1, 'A database-management system (DBMS) is a collection of interrelated data and a set of programs to access those data. The collection of data, usually referred to as the database, contains information relevant to an enterprise. The primary goal of a DBMS is to provide a way to store and retrieve database information that is both convenient and efficient.'

2. What is a foreign key?

   If two entities have a relationship, foreign-key is the attribute(s) whose value must occur in a row in the reference table. Foreign key could be multiple columns.

   Quoted from the textbook page 45, 'A foreign-key constraint from attribute(s) A of relation r1 to the primary-key B of relation r2 states that on any database instance, the value of A for each tuple in r1 must also be the value of B for some tuple in r2. Attribute set A is called a foreign key from r1, referencing r2.'

3. What is a primary key?

   Attribute(s) that forms a unique value in an entity set is a primary key. The primary key cannot be NULL.

4. What are 4 different types of DBMS relationships, give a brief explanaition for each?
   A. *One-to-one. Suppose two entity sets (A and B) have a one-to-one relationship. In that case, it means that any entity in A has at most one associated with an entity in B. Vice versa, any entity in B has at most one association with the one in A.*
   B. *One-to-many. If two entity sets (A and B) have a one-to-many relationship, any entity in A has zero or many associations with entities in B. But, any entity in B has at most one association with the one in A.*
   C. *Many-to-one. If two entity sets (A and B) have a many-to-one relationship, it means that any entity in A has at most one associated with an entity in B. But, any entity in B has zero or many associations with entities in A.*
   D. *Many-to-many. Suppose two entity sets (A and B) have a many-to-many relationship. In that case, any entity in A has zero or many associated with entities in B, and Vice versa.*

Refer to the textbook 6.4.

5. What is an ER model?

   A logical level data modeling that uses a diagram, incorporating entity sets, relationship sets, and attributes, to depict the organization of a database.

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.

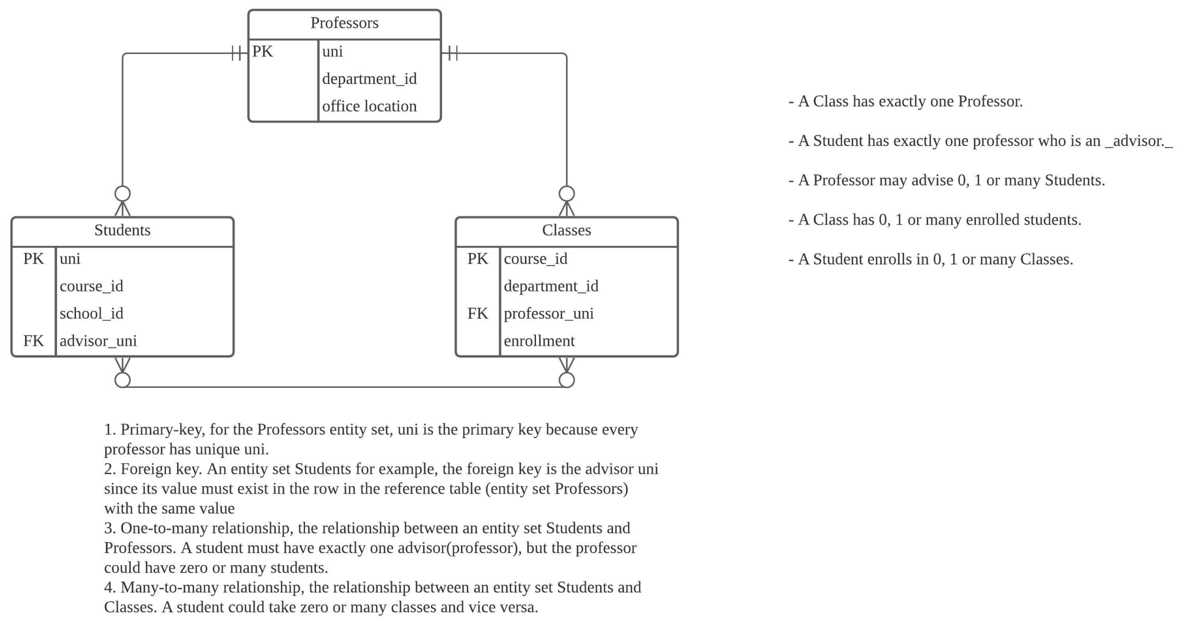
- You can define what you think are common attributes for each of the entity types. Do not define more than 5 or 6 attributes per entity type.

- 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.



- 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.

1. Primary-key, for the Professors entity set, uni is the primary key because every professor has unique uni.
2. Foreign key. An entity set Students for example, the foreign key is the advisor uni since its value must exist in the row in the reference table (entity set Professors) with the same value
3. One-to-many relationship, the relationship between an entity set Students and Professors. A student must have exactly one advisor(professor), but the professor could have zero or many students.
4. Many-to-many relationship, the relationship between an entity set Students and Classes. A student could take zero or many classes and vice versa.

Part B: Relational Algebra

You will use [the online relational calculator (https://dbis-uibk.github.io/relax/landing)](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. */

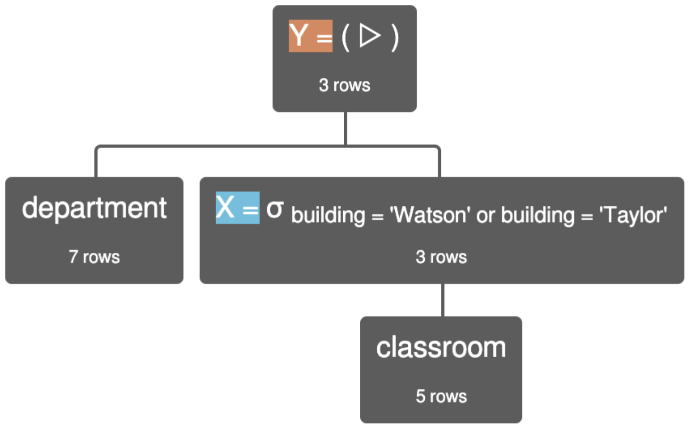
```
X = σ building='Watson' ∨ building='Taylor' (classroom)
```

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

```
Y = (department ▷ X)
```

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

```
Y
```

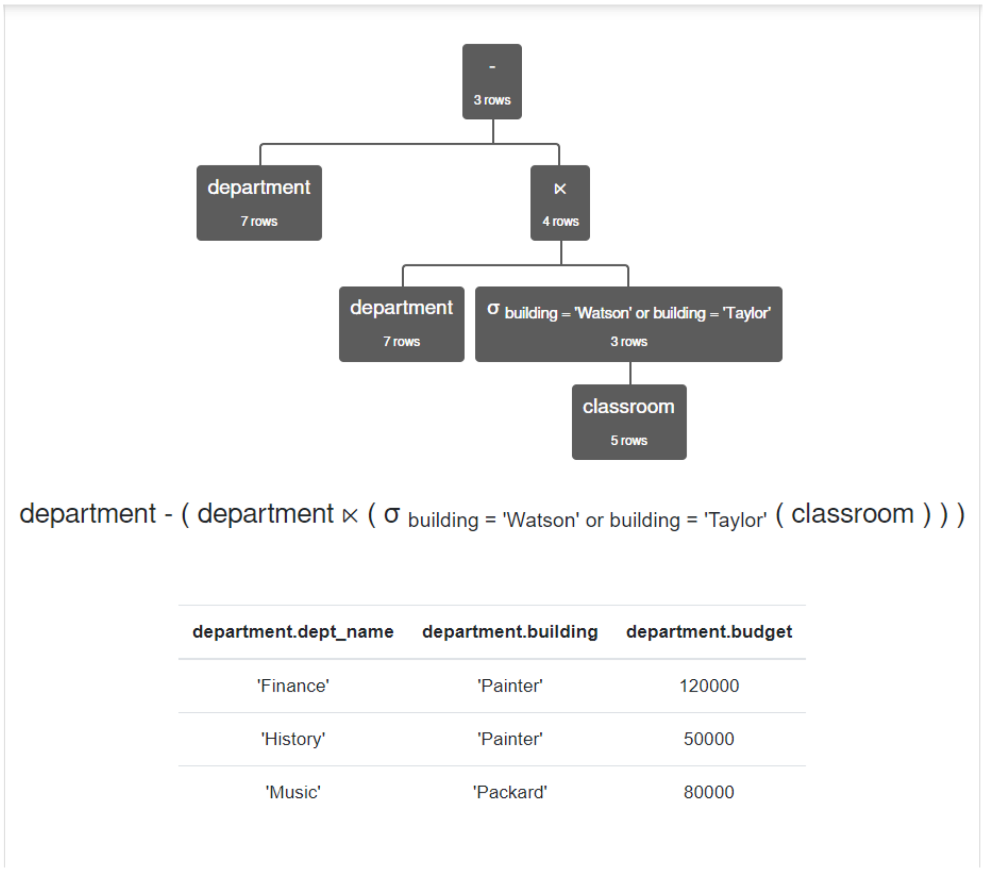


$$(\text{department} \triangleright \sigma_{\text{building} = \text{'Watson'} \text{ or } \text{building} = \text{'Taylor'}} (\text{classroom}))$$

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.

Y = department-department⋉(X)



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

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

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 [75]:
```
# You will need to follow instructions from HW 0 to make a new schema, impor
# Connect to the unclean schema below by setting the database host, user ID
%load_ext sql
%sql mysql+pymysql://admin:orphanage73@database-1.cie2eqwscgmp.us-east-2.rds
%sql use lahmansdb_to_clean
```

```
The sql extension is already loaded. To reload it, use:
  %reload_ext sql
   mysql+pymysql://admin:***@database-1.cie2eqwscgmp.us-east-2.rds.amazonaws.co
m/lahmansbaseballdb
 * mysql+pymysql://admin:***@database-1.cie2eqwscgmp.us-east-2.rds.amazonaws.co
m/lahmansdb_to_clean
0 rows affected.
```

Out[75]:
```
[]
```

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

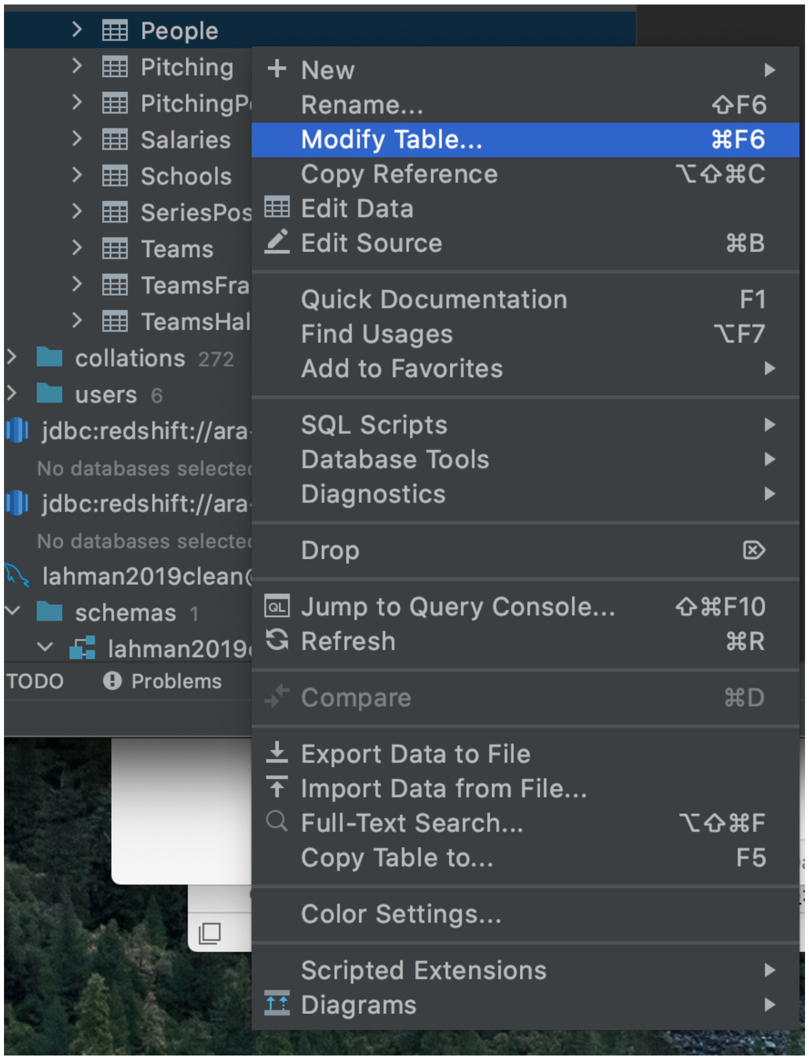
1. Empty strings to NULLs
2. Column typing
3. Primary Key
4. Foreign Key

How to make the changes:

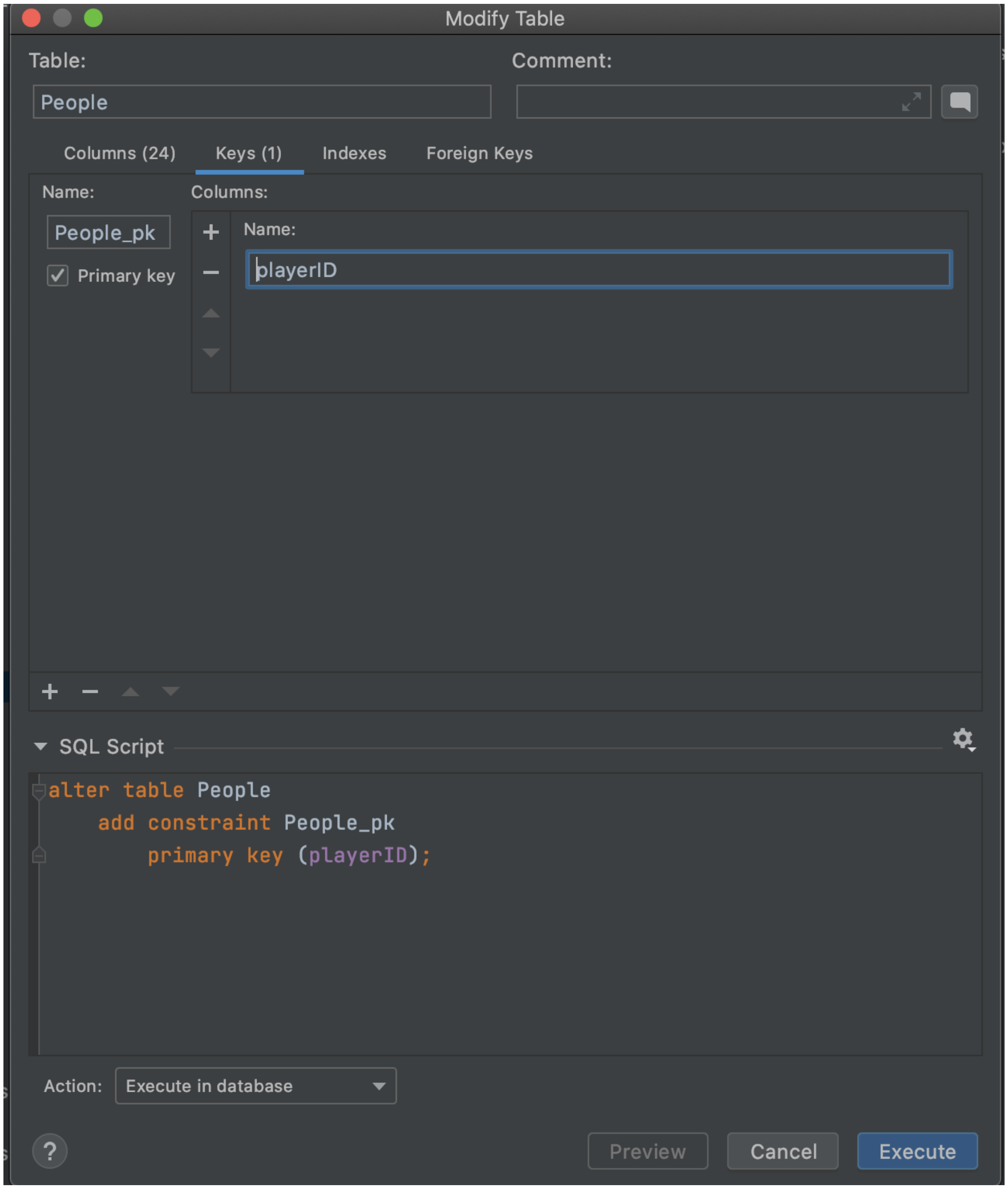
Using the Table Editor:

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



```
alter table People
    add constraint People_pk
        primary key (playerID);
```

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 [76]:
```
%sql SHOW KEYS FROM people WHERE Key_name = 'PRIMARY'

```

```
   mysql+pymysql://admin:***@database-1.cie2eqwscgmp.us-east-2.rds.amazonaws.co
m/lahmansbaseballdb
 * mysql+pymysql://admin:***@database-1.cie2eqwscgmp.us-east-2.rds.amazonaws.co
m/lahmansdb_to_clean
1 rows affected.
```

Out[76]:

Table	Non_unique	Key_name	Seq_in_index	Column_name	Collation	Cardinality	Sub_part	Pac
people	0	PRIMARY	1	playerID	A	19971	None	N

1) Convert all empty strings to NULL

Explanation

Empty strings have to be cleaned up to NULL because data operation with NULL value is correct but incorrect for an empty string. For example, if we perform an average of two people's weight, which are 150 and '', we will get 75, and this is unexpected. But with NULL, we get the expected result, which is 150.

Change

I update the value from all empty strings to NULL in the people table and have to go through every column to see if any value is equal to an empty string. If there is an empty string, we replace it with NULL. This procedure should go through every column of people table.

SQL

```
UPDATE
    people
SET
    birthYear = CASE birthYear WHEN '' THEN NULL ELSE birthYear END,
    birthMonth = CASE birthMonth WHEN '' THEN NULL ELSE birthMonth END,
    birthDay = CASE birthDay WHEN '' THEN NULL ELSE birthDay END,
    birthCountry = CASE birthCountry WHEN '' THEN NULL ELSE birthCountry
END,
    birthState = CASE birthState WHEN '' THEN NULL ELSE birthState END,
    birthCity = CASE birthCity WHEN '' THEN NULL ELSE birthCity END,
    deathYear = CASE deathYear WHEN '' THEN NULL ELSE deathYear END,
    deathMonth = CASE deathMonth WHEN '' THEN NULL ELSE deathMonth END,
    deathDay = CASE deathDay WHEN '' THEN NULL ELSE deathDay END,
    deathState = CASE deathState WHEN '' THEN NULL ELSE deathState END,
    deathCity = CASE deathCity WHEN '' THEN NULL ELSE deathCity END,
    nameFirst = CASE nameFirst WHEN '' THEN NULL ELSE nameFirst END,
    nameLast = CASE nameLast WHEN '' THEN NULL ELSE nameLast END,
    nameGiven = CASE nameGiven WHEN '' THEN NULL ELSE nameGiven END,
    weight = CASE weight WHEN '' THEN NULL ELSE weight END,
    height = CASE height WHEN '' THEN NULL ELSE height END,
    bats = CASE bats WHEN '' THEN NULL ELSE bats END,
    throws = CASE throws WHEN '' THEN NULL ELSE throws END,
    debut = CASE debut WHEN '' THEN NULL ELSE debut END,
    finalGame = CASE finalGame WHEN '' THEN NULL ELSE finalGame END,
    retroID = CASE retroID WHEN '' THEN NULL ELSE retroID END,
    bbrefID = CASE bbrefID WHEN '' THEN NULL ELSE bbrefID END;
```

Tests

In [16]:
```
%sql SELECT * FROM people WHERE birthState = '';
```

```
 * mysql+pymysql://admin:***@database-1.cie2eqwscgmp.us-east-2.rds.amazonaws.co
m/lahmansdb_to_clean
0 rows affected.
```

Out[16]:

playerID	birthYear	birthMonth	birthDay	birthCountry	birthState	birthCity	deathYear	deathMonth

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

VARCHAR, DATETIME, ETC)

Explanation

We have to convert the datatypes to suitable values to operate the data as we expect. For instance, if we would like to compare the birthYear of two people with text datatype alphabetically, then '999' > '1999' would be true, but this is not what we expect.

Change

Alter the columns in the table people. For example, converting text datatype of birthday, birthYear, weight, and height to int. The details of all alternations are listed in the SQL commands.

SQL

```
alter table people modify birthYear int null;

alter table people modify birthMonth nvarchar(2) null;

alter table people modify birthDay nvarchar(2) 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;

alter table people modify deathYear int null;

alter table people modify deathMonth nvarchar(2) null;

alter table people modify deathDay nvarchar(2) null;

alter table people modify deathCountry varchar(255) null;

alter table people modify deathState varchar(255) null;

alter table people modify deathCity varchar(255) null;

alter table people modify nameFirst varchar(255) null;

alter table people modify nameLast varchar(255) null;

alter table people modify nameGiven varchar(255) null;

alter table people modify weight int null;
alter table people modify height int null;

alter table people modify bats varchar(255) null;

alter table people modify throws varchar(255) null;

alter table people modify debut varchar(255) null;

alter table people modify finalGame varchar(255) null;

alter table people modify retroID varchar(255) null;

alter table people modify bbrefID 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

Assumed that a person is dead if the deadthYear value is not NULL, else a person is not dead (the information is not available.)

Change

Add a new column isDead with fixed length datatype and update it based on the value in column deathYear.

SQL

```
alter table people add isDead char(1) null;
Update people set isDead = 'Y' where deathYear is not null;
update people set isDead = 'N' where deathYear is null;
```

Tests

In [17]:
```
%sql SELECT * FROM people WHERE isDead = 'N' limit 10;
```

```
 * mysql+pymysql://admin:***@database-1.cie2eqwscgmp.us-east-2.rds.amazonaws.co
m/lahmansdb_to_clean
10 rows affected.
```

Out[17]:

playerID	birthYear	birthMonth	birthDay	birthCountry	birthState	birthCity	deathYear	deathM
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.

Explanation

In order to simplify the database and achieve higher storage efficiency, it is better to merge birthYear, birthMonth and birthDay into one column. The same for deathYear, deathMonth and deadthDay.

Change

Add new columns deathDate and birthDate. Fill in the date information from deathYear, deathMonth, deathDay, birthYear, birthMonth and birthDay.

SQL

```
alter table people add deathDate datetime null;
alter table people add birthDate datetime null;
update people set deathDate = str_to_date(concat(deathYear,right(concat(
'0',deathMonth), 2),right(concat('0',deathDay), 2)),'%Y%m%d') where deat
hYear is not null;
update people set birthDate = str_to_date(concat(birthYear,right(concat(
'0',birthMonth), 2),right(concat('0',birthDay), 2)),'%Y%m%d') where birt
hYear is not null;
```

Tests

In [18]:
```
%sql SELECT deathDate FROM people WHERE deathDate >= '2005-01-01' ORDER BY d
```

```
 * mysql+pymysql://admin:***@database-1.cie2eqwscgmp.us-east-2.rds.amazonaws.co
m/lahmansdb_to_clean
10 rows affected.
```

Out[18]:

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 [19]:
```
%sql SELECT birthDate FROM people WHERE birthDate <= '1965-01-01' ORDER BY b

```

```
 * mysql+pymysql://admin:***@database-1.cie2eqwscgmp.us-east-2.rds.amazonaws.co
m/lahmansdb_to_clean
10 rows affected.
```

Out[19]:

birthDate
1820-04-17 00:00:00
1824-10-05 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
1838-08-27 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   varchar(2) charset utf8 null,
    birthDay     varchar(2) charset utf8 null,
    birthCountry varchar(255)            null,
    birthState   varchar(255)            null,
    birthCity    varchar(255)            null,
    deathYear    int                     null,
    deathMonth   varchar(2) charset utf8 null,
    deathDay     varchar(2) charset utf8 null,
    deathCountry varchar(255)            null,
    deathState   varchar(255)            null,
    deathCity    varchar(255)            null,
    nameFirst    varchar(255)            null,
    nameLast     varchar(255)            null,
    nameGiven    varchar(255)            null,
    weight       int                     null,
    height       int                     null,
    bats         varchar(255)            null,
    throws       varchar(255)            null,
    debut        varchar(255)            null,
    finalGame    varchar(255)            null,
    retroID      varchar(255)            null,
    bbrefID      varchar(255)            null,
    isDead       char                    null,
    deathDate    datetime                null,
    birthDate    datetime                null
);
```

Batting Table

1) Convert all empty strings to NULL

Explanation

Empty strings have to be cleaned up to NULL because data operation with NULL value is correct but could be incorrect for an empty string. For example, if we perform an average of two people's HR, which are 10 and '', we will get 5, and that is unexpected. But with NULL, we will have the expected result, which is 10.

Change

I update the value from all empty strings to NULL in the people table and have to go through every column to see if any value is equal to an empty string. If there is an empty string, we replace it with NULL. This procedure should go through every column of people table.

SQL

```
update batting
SET
    yearID = CASE yearID WHEN '' THEN NULL ELSE yearID END,
    stint = CASE stint WHEN '' THEN NULL ELSE stint END,
    teamID = CASE teamID WHEN '' THEN NULL ELSE teamID END,
    lgID = CASE lgID WHEN '' THEN NULL ELSE lgID END,
    G = CASE G WHEN '' THEN NULL ELSE G END,
    AB = CASE AB WHEN '' THEN NULL ELSE AB END,
    R = CASE R WHEN '' THEN NULL ELSE R END,
    H = CASE H WHEN '' THEN NULL ELSE H END,
    2B = CASE 2B WHEN '' THEN NULL ELSE 2B END,
    3B = CASE 3B WHEN '' THEN NULL ELSE 3B END,
    HR = CASE HR WHEN '' THEN NULL ELSE HR END,
    RBI = CASE RBI WHEN '' THEN NULL ELSE RBI END,
    SB = CASE SB WHEN '' THEN NULL ELSE SB END,
    CS = CASE CS WHEN '' THEN NULL ELSE CS END,
    BB = CASE BB WHEN '' THEN NULL ELSE BB END,
    SO = CASE SO WHEN '' THEN NULL ELSE SO END,
    IBB = CASE IBB WHEN '' THEN NULL ELSE IBB END,
    HBP = CASE HBP WHEN '' THEN NULL ELSE HBP END,
    SH = CASE SH WHEN '' THEN NULL ELSE SH END,
    SF = CASE SF WHEN '' THEN NULL ELSE SF END,
    GIDP = CASE GIDP WHEN '' THEN NULL ELSE GIDP END;
```

Tests

In [12]:
```
%sql SELECT count(*) FROM lahmansdb_to_clean.batting where RBI is NULL;
```

```
 * mysql+pymysql://admin:***@database-1.cie2eqwscgmp.us-east-2.rds.amazonaws.co
m/lahmansdb_to_clean
1 rows affected.
```

Out[12]:

count(*)
756

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

Explanation

We have to convert the datatypes to suitable values to operate the data as we expect. For instance, if we would like to compare the HR of two people with text datatype, then '9' > '19' would be true, but this is not what we expect.

Change

Alter the columns in the table batting. For example, converting text datatype of stint, G, H to smallint. The details of all alternations are listed in the SQL commands.

SQL

```
alter table batting modify playerID varchar(9) null;

alter table batting modify yearID smallint null;

alter table batting modify stint smallint null;

alter table batting modify teamID char(3) null;

alter table batting modify lgID char(2) null;

alter table batting modify G smallint null;

alter table batting modify AB smallint null;

alter table batting modify R smallint null;

alter table batting modify H smallint null;

alter table batting modify `2B` smallint null;

alter table batting modify `3B` smallint null;

alter table batting modify HR smallint null;

alter table batting modify RBI smallint null;

alter table batting modify SB smallint null;

alter table batting modify CS smallint null;

alter table batting modify BB smallint null;

alter table batting modify SO smallint null;

alter table batting modify IBB smallint null;

alter table batting modify HBP smallint null;

alter table batting modify SH smallint null;
alter table batting modify SF smallint null;

alter table batting modify GIDP smallint 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 Composite Key because we can uniquely identify rows within our data. A primary key is also an index, which allows us to locate data faster.

Explanation

I added a Primary Key on the playerID, yearID, stint column.

SQL

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

Test

In [20]:
```
%sql SHOW KEYS FROM batting WHERE Key_name = 'PRIMARY' and Column_name = 'pl
```

```
 * mysql+pymysql://admin:***@database-1.cie2eqwscgmp.us-east-2.rds.amazonaws.co
m/lahmansdb_to_clean
1 rows affected.
```

Out[20]:

Table	Non_unique	Key_name	Seq_in_index	Column_name	Collation	Cardinality	Sub_part	Pac
batting	0	PRIMARY	1	playerID	A	18383	None	N

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

There two ways to address this issue.

1. Insert these two people into table people, then put a foreign key in table batting to refer people to build a referential integrity constraint.
2. Delete these two people in the battling table, then put a foreign key in table batting to refer people to build a referential integrity constraint.

Or we implement a foreign key ahead of the issue.

Change

1. Either insert these two people into table people or delete them from table batting.
2. Add foreign key, refer to column playerID in the table people, in the column playerID in the table batting

SQL

```
#insert or delete here
alter table batting
    add constraint batting_people_playerID_fk
        foreign key (playerID) references people (playerID);
```

Tests

In [37]:
```
%sql Select playerID from batting where playerID not in (select playerID fro
```

```
 * mysql+pymysql://admin:***@database-1.cie2eqwscgmp.us-east-2.rds.amazonaws.co
m/lahmansbaseballdb
   mysql+pymysql://admin:***@database-1.cie2eqwscgmp.us-east-2.rds.amazonaws.co
m/lahmansdb_to_clean
0 rows affected.
```

Out[37]:

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(9) not null,
    yearID   smallint   not null,
    stint    smallint   not null,
    teamID   char(3)    null,
    lgID     char(2)    null,
    G        smallint   null,
    AB       smallint   null,
    R        smallint   null,
    H        smallint   null,
    `2B`     smallint   null,
    `3B`     smallint   null,
    HR       smallint   null,
    RBI      smallint   null,
    SB       smallint   null,
    CS       smallint   null,
    BB       smallint   null,
    SO       smallint   null,
    IBB      smallint   null,
    HBP      smallint   null,
    SH       smallint   null,
    SF       smallint   null,
    GIDP     smallint   null,
    primary key (playerID, yearID, stint),
    constraint batting_people_playerID_fk
        foreign key (playerID) references lahmansdb_to_clean.people (pla
yerID)
);
```

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.

Question 0

What is the highest salary in baseball history?

In [62]:
```
%load_ext sql
%sql mysql+pymysql://admin:orphanage73@database-1.cie2eqwscgmp.us-east-2.rds
%sql use lahmansbaseballdb

```

```
The sql extension is already loaded. To reload it, use:
  %reload_ext sql
 * mysql+pymysql://admin:***@database-1.cie2eqwscgmp.us-east-2.rds.amazonaws.co
m/lahmansbaseballdb
   mysql+pymysql://admin:***@database-1.cie2eqwscgmp.us-east-2.rds.amazonaws.co
m/lahmansdb_to_clean
0 rows affected.
```

Out[62]:
```
[]
```

In [63]:
```
%sql select max(salary) from salaries;
```

```
 * mysql+pymysql://admin:***@database-1.cie2eqwscgmp.us-east-2.rds.amazonaws.co
m/lahmansbaseballdb
   mysql+pymysql://admin:***@database-1.cie2eqwscgmp.us-east-2.rds.amazonaws.co
m/lahmansdb_to_clean
1 rows affected.
```

Out[63]:

max(salary)
33000000.0

Question 1

Create a Table of all players with a first name of John who were born in the United States and played at Fordham university.

Include their first name, last name, playerID, and birth state.

Hint: Use a Join between People and CollegePlaying

In [70]:
```
%sql create table JOHNS(\
playerID varchar(9) null,\
`birth state` varchar(255) null,\
last_name varchar(255) null,\
first_name varchar(255) null\
);
%sql insert into JOHNS(playerID, `birth state`, last_name, first_name) SELEC

```

```
 * mysql+pymysql://admin:***@database-1.cie2eqwscgmp.us-east-2.rds.amazonaws.co
m/lahmansbaseballdb
   mysql+pymysql://admin:***@database-1.cie2eqwscgmp.us-east-2.rds.amazonaws.co
m/lahmansdb_to_clean
(pymysql.err.OperationalError) (1050, "Table 'JOHNS' already exists")
[SQL: create table JOHNS( playerID varchar(9) null, `birth state` varchar(255)
null, last_name varchar(255) null, first_name varchar(255) null );]
(Background on this error at: http://sqlalche.me/e/13/e3q8) (http://sqlalche.m
e/e/13/e3q8))
 * mysql+pymysql://admin:***@database-1.cie2eqwscgmp.us-east-2.rds.amazonaws.co
m/lahmansbaseballdb
   mysql+pymysql://admin:***@database-1.cie2eqwscgmp.us-east-2.rds.amazonaws.co
m/lahmansdb_to_clean
2 rows affected.
```

Out[70]:
```
[]
```

In [73]:
```
%sql select playerID,birthState,nameLast,nameFirst from people where birthCo
```

```
 * mysql+pymysql://admin:***@database-1.cie2eqwscgmp.us-east-2.rds.amazonaws.co
m/lahmansbaseballdb
   mysql+pymysql://admin:***@database-1.cie2eqwscgmp.us-east-2.rds.amazonaws.co
m/lahmansdb_to_clean
2 rows affected.
```

Out[73]:

playerID	birthState	nameLast	nameFirst
butlejo01	MA	Butler	John
walshjo02	PA	Walsh	John

Question 2

Update all entries with full_name Columbia University to 'Columbia University!' in the Schools table. Then select the row.

In [35]:
```
%sql update schools set name_full = 'Columbia University!' where schoolID =
```

```
 * mysql+pymysql://admin:***@database-1.cie2eqwscgmp.us-east-2.rds.amazonaws.co
m/lahmansbaseballdb
   mysql+pymysql://admin:***@database-1.cie2eqwscgmp.us-east-2.rds.amazonaws.co
m/lahmansdb_to_clean
1 rows affected.
```

Out[35]:
```
[]
```

In [36]:
```
%sql select * from schools where schoolID = 'columbia';
```

```
 * mysql+pymysql://admin:***@database-1.cie2eqwscgmp.us-east-2.rds.amazonaws.co
m/lahmansbaseballdb
   mysql+pymysql://admin:***@database-1.cie2eqwscgmp.us-east-2.rds.amazonaws.co
m/lahmansdb_to_clean
1 rows affected.
```

Out[36]:

schoolID	name_full	city	state	country
columbia	Columbia University!	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 is ['collied01'] and field_list is ['nameFirst', 'nameLast']

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

   Use find_by_primary_key(PK) to check if the data to be inserted has a key value that is already in the table because the key value is not allowed to duplicate in the table.

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

   We should check if the key set of new values is a subset of the keys in the table or not. If yes, update. Else, raise error("bad column")

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:

**Note some characters which leads to a pdf conversion failure is not included.

Test people

find_by_primary_key(): Known Record without projection[{'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(): Known Record with projection [{'playerID': 'aardsda01', 'nameFirst': 'David', 'nameLast': 'Aardsma', 'nameGiven': 'David Allan', 'weight': '215', 'height': '75', 'bats': 'R', 'throws': 'R'}]

find_by_primary_key(): Unknown Record []

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 []

insert(): insert a Known record
C:\Users\savik\Desktop\Columbia\DB\HW1\4111_s21_hw1_programming_CSVDataTable\src\CSVD
UserWarning: insert: key value already exists warnings.warn("insert: key value already exists")

insert(): insert 1st Unknown record query the 1st inserted record [{'playerID': 'insert_test1',
'birthYear': '1981'}]

insert(): insert 2nd Unknown record query the 2nd inserted record [{'playerID': 'insert_test2',
'birthYear': '1914'}]

delete_by_key(): delete 1st inserted record by key: 1 rows affected query the deleted record []

delete_by_template(): delete 2nd inserted record by template: 1 rows affected query the deleted
record []

update_by_key(): update these rows by key [{'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'}] Update birthYear to 1345, birthMonth to '1' Check results again [{'playerID':
'aardsda01', 'birthYear': '1345', 'birthMonth': '1', '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'}]

update_by_template(): update these rows by template [{'playerID': 'aardsda01', 'birthYear': '1345',
'birthMonth': '1', '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'}] Update birthYear to 1999, birthState to 'TX' Check results again [{'playerID':
'aardsda01', 'birthYear': '1999', 'birthMonth': '1', 'birthDay': '27', 'birthCountry': 'USA', 'birthState':
'TX', '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'}]

Process finished with exit code 0
```

Test batting

```
find_by_primary_key(): Known Record with a composite primary key playerID,yearID,teamID
without projection [{'playerID': 'bealsto01', 'yearID': '1871', 'stint': '1', 'teamID': 'WS3', 'lgID': 'NA', 'G':
'10', 'AB': '36', 'R': '6', 'H': '7', '2B': '0', '3B': '0', 'HR': '0', 'RBI': '1', 'SB': '2', 'CS': '0', 'BB': '2', 'SO': '0',
'IBB': '', 'HBP': '', 'SH': '', 'SF': '', 'GIDP': '2'}]

find_by_primary_key(): Known Record with a composite primary key playerID,yearID,teamID with
projection [{'G': '10', 'AB': '36', 'R': '6', 'H': '7'}]

find_by_primary_key(): Unknown Record []

find_by_template(): Known Template [{'playerID': 'bergmo01', 'yearID': '1937', 'stint': '1', 'teamID':
'BOS', 'lgID': 'AL', 'G': '47', 'AB': '141', 'R': '13', 'H': '36', '2B': '3', '3B': '1', 'HR': '0', 'RBI': '20', 'SB': '0',
'CS': '0', 'BB': '5', 'SO': '4', 'IBB': '', 'HBP': '0', 'SH': '2', 'SF': '', 'GIDP': ''}, {'playerID': 'eggleda01',
'yearID': '1877', 'stint': '1', 'teamID': 'CHN', 'lgID': 'NL', 'G': '33', 'AB': '136', 'R': '20', 'H': '36', '2B': '3',
'3B': '0', 'HR': '0', 'RBI': '20', 'SB': '', 'CS': '', 'BB': '1', 'SO': '5', 'IBB': '', 'HBP': '', 'SH': '', 'SF': '', 'GIDP':
''}]

find_by_template(): Unknown Template []

insert(): insert a Known record
C:\Users\savik\Desktop\Columbia\DB\HW1\4111_s21_hw1_programming_CSVDataTable\src\CSVD
UserWarning: insert: key value already exists warnings.warn("insert: key value already exists")

insert(): insert 1st Unknown record query the 1st inserted record [{'playerID': 'insert_test1', 'yearID':
'2001', 'teamID': 'TRO'}]

insert(): insert 2nd Unknown record query the 2nd inserted record [{'playerID': 'insert_test2',
'yearID': '2011', 'teamID': 'NY2'}]

delete_by_key(): delete 1st inserted record by key: 1 rows affected query the deleted record []

delete_by_template(): delete 2nd inserted record by template: 1 rows affected query the deleted
record []

update_by_key(): update these rows by key [{'playerID': 'bealsto01', 'yearID': '1871', 'stint': '1',
'teamID': 'WS3', 'lgID': 'NA', 'G': '10', 'AB': '36', 'R': '6', 'H': '7', '2B': '0', '3B': '0', 'HR': '0', 'RBI': '1',
'SB': '2', 'CS': '0', 'BB': '2', 'SO': '0', 'IBB': '', 'HBP': '', 'SH': '', 'SF': '', 'GIDP': '2'}] Update HR to 12, SO
to '7' Check results again [{'playerID': 'bealsto01', 'yearID': '1871', 'stint': '1', 'teamID': 'WS3', 'lgID':
'NA', 'G': '10', 'AB': '36', 'R': '6', 'H': '7', '2B': '0', '3B': '0', 'HR': '12', 'RBI': '1', 'SB': '2', 'CS': '0', 'BB': '2',
'SO': '7', 'IBB': '', 'HBP': '', 'SH': '', 'SF': '', 'GIDP': '2'}]

update_by_template(): update these rows by template [{'playerID': 'bergmo01', 'yearID': '1937',
'stint': '1', 'teamID': 'BOS', 'lgID': 'AL', 'G': '47', 'AB': '141', 'R': '13', 'H': '36', '2B': '3', '3B': '1', 'HR': '0',
'RBI': '20', 'SB': '0', 'CS': '0', 'BB': '5', 'SO': '4', 'IBB': '', 'HBP': '0', 'SH': '2', 'SF': '', 'GIDP': ''},
{'playerID': 'eggleda01', 'yearID': '1877', 'stint': '1', 'teamID': 'CHN', 'lgID': 'NL', 'G': '33', 'AB': '136',
'R': '20', 'H': '36', '2B': '3', '3B': '0', 'HR': '0', 'RBI': '20', 'SB': '', 'CS': '', 'BB': '1', 'SO': '5', 'IBB': '', 'HBP':
'', 'SH': '', 'SF': '', 'GIDP': ''}] Update 2B to 6, BB to '11' Check results again [{'playerID': 'bergmo01',
'yearID': '1937', 'stint': '1', 'teamID': 'BOS', 'lgID': 'AL', 'G': '47', 'AB': '141', 'R': '13', 'H': '36', '2B': '6',
'3B': '1', 'HR': '0', 'RBI': '20', 'SB': '0', 'CS': '0', 'BB': '11', 'SO': '4', 'IBB': '', 'HBP': '0', 'SH': '2', 'SF': '',
'GIDP': ''}, {'playerID': 'eggleda01', 'yearID': '1877', 'stint': '1', 'teamID': 'CHN', 'lgID': 'NL', 'G': '33',
'AB': '136', 'R': '20', 'H': '36', '2B': '6', '3B': '0', 'HR': '0', 'RBI': '20', 'SB': '', 'CS': '', 'BB': '11', 'SO': '5',
'IBB': '', 'HBP': '', 'SH': '', 'SF': '', 'GIDP': ''}]
```

```
Process finished with exit code 0
```