4111_s21_hw1_programming

September 22, 2021

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.

<span style="font-size: 20pt; line-height: 1.2"; > Chan_I Hun, ic2552

0.1 Introduction

0.1.1 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.

0.1.2 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

0.1.3 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.

1 Part A: Written

1. What is a database management system?

A database management system (DBMS) is a software system designed for users to have deal with data in a database; users can manipulate, define, manage, retrieve data, and control access to the database.

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

   A primary key is a column or a group of columns together that uniquely identifies a row in a database. It is important because, since it is unique, it allows users to identify and operate on the record that primary key is assigned to.

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

   SQL is a query (programming) language used to manage data in a relational DBMS; MySQL Server is a DBMS used for warehousing, logging data records etc.; finally, DataGrip is an IDE for data management, it provides an interface for developers to work with.

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

   1. one-to-one relationship: for a record in one table, it is only related to only one record in the other table. 2. one-to-many relationship: for a record in one table, it can be related to one ore more than one record in the other table. 3. many-to-one relationship: for many records in one table, they are all related to one record in another table. 4. many-to-many relationship: for each record of the first table, they can be related to one or more records in the second table; and for each record in the second table, they can be related to one or more records in the second table.

5. What is an ER model?

   An ER model stands for the Entity-Relation model, which was developed to faciliate database design. It is used to describe the relationships between data elements. The ER model employs the concepts of entity sets, relationship sets, and attributes, which are used to represent the overall logic 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.

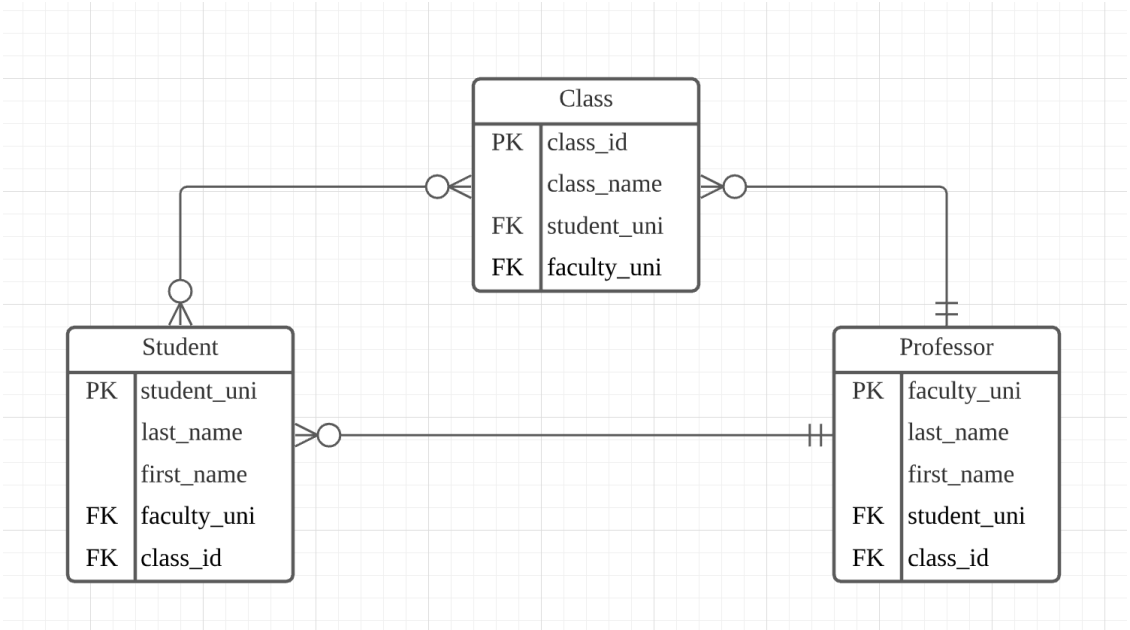
- In this example, explicitly show an example of a primary-key, foreign key, one-to-many relationship, and many-to-many relationship.

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.

[1]:
```
from IPython.display import Image
Image("./Lucid.png")
```

[1]:



2 Part B: Relational Algebra

You will use the online relational calculator, 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 =  building='Watson'  building='Taylor' (classroom)
```

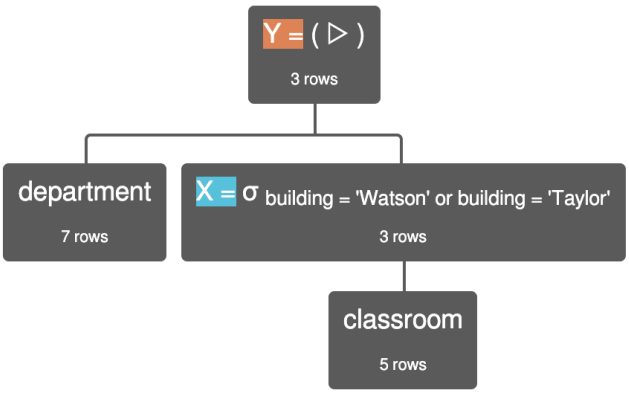
/* (2) Set Y = The Anti-Join of department and X */

```
Y = (department  X)
```

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

```
Y
```

[2]:
```
Image("./ra.png")
```
[2]:

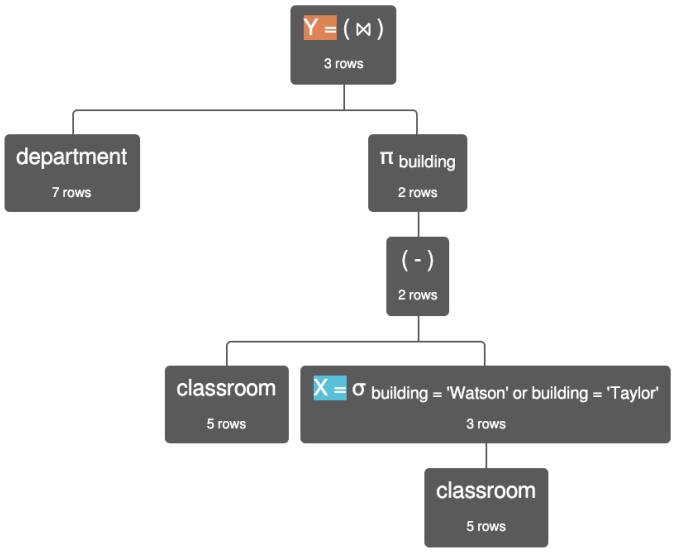


(department ▷ σ building = 'Watson' or building = 'Taylor' (classroom))

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.

[3]:
```
Image("./query.png")
```
[3]:



(department ⋈ π building (classroom - σ building = 'Watson' or building = 'Taylor' (classroom)))

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

3 Part C: Data Clean Up

3.1 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.

[8]:
```
# You will need to follow instructions from HW 0 to make a new schema, import␣
↪the data.
# Connect to the unclean schema below by setting the database host, user ID and␣
↪password.
```

```
%load_ext sql
%sql mysql+pymysql://root:dbuserdbuser@localhost/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.

3.1.1 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)

3.1.2 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

3.1.3 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!

3.2 People Table

3.2.1 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

[11]:
```
%sql SHOW KEYS FROM people WHERE Key_name = 'PRIMARY'
```

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

[11]:
```
[('people', 0, 'PRIMARY', 1, 'playerID', 'A', 19879, None, None, '', 'BTREE',
'', '', 'YES', None)]
```

3.2.2 1) Convert all empty strings to NULL

Explanation We want to convert all empty strings to NULL because NULL values are more efficiently stored; also we can't operate on NULL values, but we can on empty strings, which are of size zero. When a string is empty in the database, it means that no value exists for that record, thus we should use NULL to represent this.

Change I went through all the columns possibly containing an empty string in the people table to update all empty strings to the datatype NULL.

SQL

```
UPDATE
    people
SET
    playerID = IF(playerID = '', NULL, playerID),
    birthYear = IF(birthYear = '', NULL, birthYear),
    birthMonth = IF(birthMonth = '', NULL, birthMonth),
```

```
    birthDay = IF(birthDay = '', NULL, birthDay),
    birthCountry = IF(birthCountry = '', NULL, birthCountry),
    birthState = IF(birthState = '', NULL, birthState),
    birthCity = IF(birthCity = '', NULL, birthCity),
    deathYear = IF(deathYear = '', NULL, deathYear),
    deathMonth = IF(deathMonth = '', NULL, deathMonth),
    deathDay = IF(deathDay = '', NULL, deathDay),
    deathCountry = IF(deathCountry = '', NULL, deathCountry),
    deathState = IF(deathState = '', NULL, deathState),
    deathCity = IF(deathCity = '', NULL, deathCity),
    nameFirst = IF(nameFirst = '', NULL, nameFirst),
    nameLast = IF(nameLast = '', NULL, nameLast),
    nameGiven = IF(nameGiven = '', NULL, nameGiven),
    weight = IF(weight = '', NULL, weight),
    height = IF(height = '', NULL, height),
    bats = IF(bats = '', NULL, bats),
    throws = IF(throws = '', NULL, throws),
    debut = IF(debut = '', NULL, debut),
    finalGame = IF(finalGame = '', NULL, finalGame),
    retroID = IF(retroID = '', NULL, retroID),
    bbrefID = IF(bbrefID = '', NULL, bbrefID);
```

Tests

[12]:
```
%sql SELECT * FROM people WHERE birthState = ""
```

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

[12]:
```
[]
```

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

Explanation We want the column datatypes to be the appropriate values so upon retrieval, manipulation etc. of the records, we can apply the right operations on them; for example, we can do average on a column of integers, but not if their datatype is TEXT.

Change I went through all the datatypes in each column of the people table using DataGrip and changed them to the appropriate datatypes.

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;

alter table people modify deathYear int null;

alter table people modify deathMonth int null;

alter table people modify deathDay int 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;
```

3.2.4 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 We want to add the isDead column because it will be easier to determine if the player is dead or alive in future instances, we don't want to look up the player's birthdate or make guesses about the player's lifespan when we want to handle his/her data.

Change I first altered the people table by adding the isDead column of datatype char(1) and defined to be not null. I then updated the isDead column based on if deathYear is null or not, if deathYear is null, it means that the player is alive, else it means the player is dead.

SQL

```
alter table people
    add isDead char(1) not null;

update people
     set isDead = (CASE
                         WHEN deathYear is null
                              THEN 'N'
                            WHEN deathYear is not null
                              THEN 'Y'
                            END);
```

Tests

[14]:
```
%sql SELECT * FROM people WHERE isDead = "N" limit 10
```

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

[14]:
```
[('aardsda01', 1981, 12, 27, 'USA', 'CO', 'Denver', None, None, None, None,
None, None, 'David', 'Aardsma', 'David Allan', 215, 75, 'R', 'R', '2004-04-06',
'2015-08-23', 'aardd001', 'aardsda01', 'N', None, datetime.datetime(1981, 12,
27, 0, 0)),
 ('aaronha01', 1934, 2, 5, 'USA', 'AL', 'Mobile', None, None, None, None, None,
None, 'Hank', 'Aaron', 'Henry Louis', 180, 72, 'R', 'R', '1954-04-13',
'1976-10-03', 'aaroh101', 'aaronha01', 'N', None, datetime.datetime(1934, 2, 5,
0, 0)),
 ('aasedo01', 1954, 9, 8, 'USA', 'CA', 'Orange', None, None, None, None, None,
None, 'Don', 'Aase', 'Donald William', 190, 75, 'R', 'R', '1977-07-26',
'1990-10-03', 'aased001', 'aasedo01', 'N', None, datetime.datetime(1954, 9, 8,
0, 0)),
 ('abadan01', 1972, 8, 25, 'USA', 'FL', 'Palm Beach', None, None, None, None,
None, None, 'Andy', 'Abad', 'Fausto Andres', 184, 73, 'L', 'L', '2001-09-10',
'2006-04-13', 'abada001', 'abadan01', 'N', None, datetime.datetime(1972, 8, 25,
0, 0)),
 ('abadfe01', 1985, 12, 17, 'D.R.', 'La Romana', 'La Romana', None, None, None,
None, None, None, 'Fernando', 'Abad', 'Fernando Antonio', 235, 74, 'L', 'L',
'2010-07-28', '2019-09-28', 'abadf001', 'abadfe01', 'N', None,
datetime.datetime(1985, 12, 17, 0, 0)),
 ('abbotgl01', 1951, 2, 16, 'USA', 'AR', 'Little Rock', None, None, None, None,
```

```
None, None, 'Glenn', 'Abbott', 'William Glenn', 200, 78, 'R', 'R', '1973-07-29',
'1984-08-08', 'abbog001', 'abbotgl01', 'N', None, datetime.datetime(1951, 2, 16,
0, 0)),
 ('abbotje01', 1972, 8, 17, 'USA', 'GA', 'Atlanta', None, None, None, None,
None, None, 'Jeff', 'Abbott', 'Jeffrey William', 190, 74, 'R', 'L',
'1997-06-10', '2001-09-29', 'abboj002', 'abbotje01', 'N', None,
datetime.datetime(1972, 8, 17, 0, 0)),
 ('abbotji01', 1967, 9, 19, 'USA', 'MI', 'Flint', None, None, None, None, None,
None, 'Jim', 'Abbott', 'James Anthony', 200, 75, 'L', 'L', '1989-04-08',
'1999-07-21', 'abboj001', 'abbotji01', 'N', None, datetime.datetime(1967, 9, 19,
0, 0)),
 ('abbotku01', 1969, 6, 2, 'USA', 'OH', 'Zanesville', None, None, None, None,
None, None, 'Kurt', 'Abbott', 'Kurt Thomas', 180, 71, 'R', 'R', '1993-09-07',
'2001-04-13', 'abbok002', 'abbotku01', 'N', None, datetime.datetime(1969, 6, 2,
0, 0)),
 ('abbotky01', 1968, 2, 18, 'USA', 'MA', 'Newburyport', None, None, None, None,
None, None, 'Kyle', 'Abbott', 'Lawrence Kyle', 200, 76, 'L', 'L', '1991-09-10',
'1996-08-24', 'abbok001', 'abbotky01', 'N', None, datetime.datetime(1968, 2, 18,
0, 0))]
```

3.2.5 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 We want to add a deathDate and birthDate column so we gain information on when the player is born/dead, this helps in certain cases, such as when we want to generate a wiki for the player which includes the player's birthdate and deathdate.

Change I first added the birthDate and deathDate columns with the datatype of DATETIME, the records are set to NULL if birthDate/deathDate information isn't available.

SQL

```sql
alter table people
    add deathDate datetime null;

alter table people
    add birthDate datetime null;

update people
    set deathDate = cast((concat(cast(deathYear as char(4)), '-', cast(deathMonth as char(2)),

update people
    set birthDate = cast((concat(cast(birthYear as char(4)), '-', cast(birthMonth as char(2)),
```

Tests

[15]:
```
%sql SELECT deathDate FROM people WHERE deathDate >= '2005-01-01' ORDER BY␣
↪deathDate ASC LIMIT 10;
```

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

[15]:
```
[(datetime.datetime(2005, 1, 4, 0, 0),),
 (datetime.datetime(2005, 1, 7, 0, 0),),
 (datetime.datetime(2005, 1, 9, 0, 0),),
 (datetime.datetime(2005, 1, 10, 0, 0),),
 (datetime.datetime(2005, 1, 21, 0, 0),),
 (datetime.datetime(2005, 1, 22, 0, 0),),
 (datetime.datetime(2005, 1, 31, 0, 0),),
 (datetime.datetime(2005, 2, 4, 0, 0),),
 (datetime.datetime(2005, 2, 8, 0, 0),),
 (datetime.datetime(2005, 2, 11, 0, 0),)]
```

[16]:
```
%sql SELECT birthDate FROM people WHERE birthDate <= '1965-01-01' ORDER BY␣
↪birthDate ASC LIMIT 10;
```

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

[16]:
```
[(datetime.datetime(1820, 4, 17, 0, 0),),
 (datetime.datetime(1824, 10, 5, 0, 0),),
 (datetime.datetime(1832, 9, 17, 0, 0),),
 (datetime.datetime(1832, 10, 23, 0, 0),),
 (datetime.datetime(1835, 1, 10, 0, 0),),
 (datetime.datetime(1836, 2, 29, 0, 0),),
 (datetime.datetime(1837, 12, 26, 0, 0),),
 (datetime.datetime(1838, 3, 10, 0, 0),),
 (datetime.datetime(1838, 7, 16, 0, 0),),
 (datetime.datetime(1838, 8, 27, 0, 0),)]
```

3.2.6 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 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,
    debut        varchar(255) null,
    finalGame    varchar(255) null,
    retroID      varchar(255) null,
    bbrefID      varchar(255) null,
    isDead       char         not null,
    deathDate    datetime     null,
    birthDate    datetime     null
);
```

3.3 Batting Table

3.3.1 1) Convert all empty strings to NULL

Explanation We want to convert all empty strings to NULL because NULL values are more efficiently stored; also we can't operate on NULL values, but we can on empty strings, which are of size zero. When a string is empty in the database, it means that no value exists for that record, thus we should use NULL to represent this.

Change I went through all the columns possibly containing an empty string in the batting table to update all empty strings to the datatype NULL.

SQL

```
update batting set yearID=NULL where yearID = "";
update batting set stint=NULL where stint = "";
update batting set teamID=NULL where teamID = "";
update batting set lgID=NULL where lgID = "";
update batting set G=NULL where G = "";
update batting set AB=NULL where AB = "";
update batting set R=NULL where R = "";
update batting set H=NULL where H = "";
update batting set 2B=NULL where 2B = "";
```

```
update batting set 3B=NULL where 3B = "";
update batting set HR=NULL where HR = "";
update batting set RBI=NULL where RBI = "";
update batting set SB=NULL where SB = "";
update batting set CS=NULL where CS = "";
update batting set BB=NULL where BB = "";
update batting set SO=NULL where SO = "";
update batting set IBB=NULL where IBB = "";
update batting set HBP=NULL where HBP = "";
update batting set SH=NULL where SH = "";
update batting set SF=NULL where SF = "";
update batting set GIDP=NULL where GIDP = "";
```

Tests

[18]:
```
%sql SELECT count(*) FROM lahmansdb_to_clean.batting where RBI is NULL;
```

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

[18]:
```
[(756,)]
```

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

Explanation We want the column datatypes to be the appropriate values so upon retrieval, manipulation etc. of the records, we can apply the right operations on them; for example, we can do average on a column of integers, but not if their datatype is TEXT.

Change I went through all the datatypes in each column of the people table using DataGrip and changed them to the appropriate datatypes.

SQL

```
alter table batting modify playerID varchar(9) not null;

alter table batting modify yearID smallint not null;

alter table batting modify stint smallint not 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;
```

```sql
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;
```

3.3.3 3) Add a Primary Key

Two options for the Primary Key:

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

Choice Covering Key (Index): playerID, yearID, stint, teamID

Explanation I chose the covering key because this is a key that is bigger than it needs to be, so that when we have the information at hand (which may be used more commonly), having a covering key is more efficient.

SQL

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

Test

[19]:
```
%sql SHOW KEYS FROM batting WHERE Key_name = 'PRIMARY' and Column_name =␣
↪'playerID'
```

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

[19]:
```
[('batting', 0, 'PRIMARY', 1, 'playerID', 'A', 19978, None, None, '', 'BTREE',
'', '', 'YES', None)]
```

3.3.4 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 We want foreign key on playerID to demonstrate the relationship between the people and batting tables, foreign keys also add integrity to the database, so when a player is deleted from people, it will also be deleted in batting.

Change *Put your answer in this cell*

SQL

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

Tests

[20]:
```
%sql Select playerID from batting where playerID not in (select playerID from␣
↪people);
```

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

[20]:
```
[]
```

3.3.5 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 batting
(
    playerID varchar(9) not null,
    yearID   smallint   not null,
    stint    smallint   not null,
    teamID   char(3)    not 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, teamID),
    constraint batting_people_playerID_fk
        foreign key (playerID) references people (playerID)
);
```

4 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.

[3]:
```
%reload_ext sql
%sql mysql+pymysql://root:dbuserdbuser@localhost/lahmansbaseballdb
```

4.1 Question 0

What is the average salary in baseball history?

[4]:
```
%sql select avg(salary) from lahmansbaseballdb.salaries
```

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

[4]:
```
[(2085634.053125473,)]
```

[ ]:
```

```

4.2 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

[31]:
```
%sql select P.nameFirst, P.nameLast, P.playerID, C.schoolID, C.yearID, P.
↪birthState from lahmansbaseballdb.people P, lahmansbaseballdb.collegeplaying␣
↪C where P.nameFirst = 'Sam' and P.birthCountry = 'USA' and P.playerID = C.
↪playerID limit 10
```

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

[31]:
```
[('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')]
```

[3]:
```
%%sql
```

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

[3]:
```
[('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')]
```

4.3 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.

[ ]:
```
%sql update lahmansbaseballdb.schools set name_full='Columbia University in the␣
↪city of New York' where name_full='Columbia University';
```

[5]:
```
%sql select * from lahmansbaseballdb.schools where schoolID = 'columbia'
```

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

[5]:
```
[('columbia', 'Columbia University in the city of New York', 'New York', 'NY',
'USA')]
```

[6]:
```

```

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

[6]:
```
[('columbia', 'Columbia University in the City of New York', 'New York', 'NY',
'USA')]
```

5 Part E: CSVDataTable

5.1 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 contains playerID and collied01, field_list contains nameFirst and nameLast

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

   Check if there is already a PK with the same name, because PKs are supposed to be unique.

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

   If the row with that Pk value exists, because you can't perform UPDATE when such value doesn't exist.

5.2 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

5.3 iii. Testing

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

============================= test session starts ============================== collecting … collected 2 items

tests/csv_table_tests.py::tests_people tests/csv_table_tests.py::tests_batting

============================== 2 passed in 4.33s ===============================

Process finished with exit code 0 PASSED [50%] 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 []

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'}]

delete_by_key(): Known Record 1

delete_by_key(): Unkown Record 0

delete_by_template(): Known Template 1

find_by_primary_key() after deletion: should return empty list []

find_by_template() after deletion: should return empty list []

insert(): previously deleted player with playerID aardsda01 [{'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'}]

insert(): new player with playerID abcd01 [{‘playerID’: ‘abcd01’, ‘birthYear’: ‘1990’, ‘birthMonth’: ‘10’, ‘birthDay’: ‘24’, ‘birthCountry’: ‘USA’, ‘birthState’: ’‘, ’birthCity’: ’‘, ’deathYear’: ’‘, ’deathMonth’: ’‘, ’deathDay’: ’‘, ’deathCountry’: ’‘, ’deathState’: ’‘, ’deathCity’: ’‘, ’nameFirst’: ‘Kamado’, ‘nameLast’: ‘Tanjiro’, ‘nameGiven’: ‘Bob Tanjiro’, ‘weight’: ‘200’, ‘height’: ‘75’, ‘bats’: ‘R’, ‘throws’: ‘R’, ‘debut’: ‘2000-01-01’, ‘finalGame’: ‘2020-12-31’, ‘retroID’: ‘abcd01’, ‘bbrefID’: ‘abcd01’}]

insert(): player that already exists, should print statement stating that player already exists player already exists

update_by_template(): known template, should return 1 1 [{‘playerID’: ‘aardsda01’, ‘birthYear’: ‘1981’, ‘birthMonth’: ‘12’, ‘birthDay’: ‘27’, ‘birthCountry’: ‘China’, ‘birthState’: ‘Guangdong’, ‘birthCity’: ‘Zhuhai’, ‘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() unknown: should return 0 and print player doesn’t exist player doesn’t exist 0

update_by_key(): should return 1 1 [{‘playerID’: ‘abcd01’, ‘birthYear’: ‘1990’, ‘birthMonth’: ‘10’, ‘birthDay’: ‘24’, ‘birthCountry’: ‘China’, ‘birthState’: ‘Guangdong’, ‘birthCity’: ‘Zhuhai’, ‘deathYear’: ’‘, ’deathMonth’: ’‘, ’deathDay’: ’‘, ’deathCountry’: ’‘, ’deathState’: ’‘, ’deathCity’: ’‘, ’nameFirst’: ‘Kamado’, ‘nameLast’: ‘Tanjiro’, ‘nameGiven’: ‘Bob Tanjiro’, ‘weight’: ‘200’, ‘height’: ‘75’, ‘bats’: ‘R’, ‘throws’: ‘R’, ‘debut’: ‘2000-01-01’, ‘finalGame’: ‘2020-12-31’, ‘retroID’: ‘abcd01’, ‘bbrefID’: ‘abcd01’}] PASSED [100%]

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 []

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’}]

delete_by_key(): Known Record 1

delete_by_key(): Unkown Record 0

delete_by_template(): Known Template 1

find_by_primary_key() after deletion: should return empty list []

find_by_template() after deletion: should return empty list []

insert(): previously deleted player with playerID abercda01 [{‘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’}]

insert(): new player with primary key (abcd01, 1999, 1) [{‘playerID’: ‘abcd01’, ‘yearID’: ‘1999’, ‘stint’: ‘1’, ‘teamID’: ‘TRO’, ‘lgID’: ‘NA’, ‘G’: ‘100’, ‘AB’: ‘100’, ‘R’: ‘100’, ‘H’: ‘100’, ‘2B’: ‘100’,

‘3B’: ‘100’, ‘HR’: ‘100’, ‘RBI’: ‘100’, ‘SB’: ‘100’, ‘CS’: ‘100’, ‘BB’: ‘100’, ‘SO’: ‘100’, ‘IBB’: ’‘, ’HBP’: ’‘, ’SH’: ’‘, ’SF’: ’‘, ’GIDP’: ’ ’}]

insert(): player that already exists, should print statement stating that player already exists player already exists

update_by_template(): known template, update all players with teamID TRO and HR = 0 should return 20 20 [{‘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’: ‘100’, ‘CS’: ‘100’, ‘BB’: ‘0’, ‘SO’: ‘0’, ‘IBB’: ’‘, ’HBP’: ’‘, ’SH’: ’‘, ’SF’: ’‘, ’GIDP’: ‘0’}, {‘playerID’: ‘zettlge01’, ‘yearID’: ‘1872’, ‘stint’: ‘1’, ‘teamID’: ‘TRO’, ‘lgID’: ‘NA’, ‘G’: ‘25’, ‘AB’: ‘114’, ‘R’: ‘25’, ‘H’: ‘29’, ‘2B’: ‘9’, ‘3B’: ‘0’, ‘HR’: ‘0’, ‘RBI’: ‘21’, ‘SB’: ‘100’, ‘CS’: ‘100’, ‘BB’: ‘0’, ‘SO’: ‘2’, ‘IBB’: ’‘, ’HBP’: ’‘, ’SH’: ’‘, ’SF’: ’‘, ’GIDP’: ‘1’}, {‘playerID’: ‘nelsoca01’, ‘yearID’: ‘1872’, ‘stint’: ‘1’, ‘teamID’: ‘TRO’, ‘lgID’: ‘NA’, ‘G’: ‘4’, ‘AB’: ‘20’, ‘R’: ‘2’, ‘H’: ‘7’, ‘2B’: ‘0’, ‘3B’: ‘0’, ‘HR’: ‘0’, ‘RBI’: ‘4’, ‘SB’: ‘100’, ‘CS’: ‘100’, ‘BB’: ‘0’, ‘SO’: ‘2’, ‘IBB’: ’‘, ’HBP’: ’‘, ’SH’: ’‘, ’SF’: ’‘, ’GIDP’: ‘0’}, {‘playerID’: ‘mcatebu01’, ‘yearID’: ‘1872’, ‘stint’: ‘1’, ‘teamID’: ‘TRO’, ‘lgID’: ‘NA’, ‘G’: ‘25’, ‘AB’: ‘126’, ‘R’: ‘30’, ‘H’: ‘28’, ‘2B’: ‘3’, ‘3B’: ‘1’, ‘HR’: ‘0’, ‘RBI’: ‘15’, ‘SB’: ‘100’, ‘CS’: ‘100’, ‘BB’: ‘3’, ‘SO’: ‘2’, ‘IBB’: ’‘, ’HBP’: ’‘, ’SH’: ’‘, ’SF’: ’‘, ’GIDP’: ‘2’}, {‘playerID’: ‘martiph01’, ‘yearID’: ‘1872’, ‘stint’: ‘1’, ‘teamID’: ‘TRO’, ‘lgID’: ‘NA’, ‘G’: ‘25’, ‘AB’: ‘119’, ‘R’: ‘27’, ‘H’: ‘36’, ‘2B’: ‘2’, ‘3B’: ‘1’, ‘HR’: ‘0’, ‘RBI’: ‘14’, ‘SB’: ‘100’, ‘CS’: ‘100’, ‘BB’: ‘0’, ‘SO’: ‘1’, ‘IBB’: ’‘, ’HBP’: ’‘, ’SH’: ’‘, ’SF’: ’‘, ’GIDP’: ‘0’}, {‘playerID’: ‘kingst01’, ‘yearID’: ‘1872’, ‘stint’: ‘1’, ‘teamID’: ‘TRO’, ‘lgID’: ‘NA’, ‘G’: ‘25’, ‘AB’: ‘128’, ‘R’: ‘33’, ‘H’: ‘39’, ‘2B’: ‘8’, ‘3B’: ‘0’, ‘HR’: ‘0’, ‘RBI’: ‘20’, ‘SB’: ‘100’, ‘CS’: ‘100’, ‘BB’: ‘1’, ‘SO’: ‘2’, ‘IBB’: ’‘, ’HBP’: ’‘, ’SH’: ’‘, ’SF’: ’‘, ’GIDP’: ‘1’}, {‘playerID’: ‘kingma01’, ‘yearID’: ‘1872’, ‘stint’: ‘1’, ‘teamID’: ‘TRO’, ‘lgID’: ‘NA’, ‘G’: ‘3’, ‘AB’: ‘11’, ‘R’: ‘0’, ‘H’: ‘0’, ‘2B’: ‘0’, ‘3B’: ‘0’, ‘HR’: ‘0’, ‘RBI’: ‘1’, ‘SB’: ‘100’, ‘CS’: ‘100’, ‘BB’: ‘0’, ‘SO’: ‘1’, ‘IBB’: ’‘, ’HBP’: ’‘, ’SH’: ’‘, ’SF’: ’‘, ’GIDP’: ‘0’}, {‘playerID’: ‘hodesch01’, ‘yearID’: ‘1872’, ‘stint’: ‘1’, ‘teamID’: ‘TRO’, ‘lgID’: ‘NA’, ‘G’: ‘13’, ‘AB’: ‘62’, ‘R’: ‘17’, ‘H’: ‘15’, ‘2B’: ‘3’, ‘3B’: ‘0’, ‘HR’: ‘0’, ‘RBI’: ‘10’, ‘SB’: ‘100’, ‘CS’: ‘100’, ‘BB’: ‘1’, ‘SO’: ‘0’, ‘IBB’: ’‘, ’HBP’: ’‘, ’SH’: ’‘, ’SF’: ’‘, ’GIDP’: ‘1’}, {‘playerID’: ‘forceda01’, ‘yearID’: ‘1872’, ‘stint’: ‘1’, ‘teamID’: ‘TRO’, ‘lgID’: ‘NA’, ‘G’: ‘25’, ‘AB’: ‘130’, ‘R’: ‘40’, ‘H’: ‘53’, ‘2B’: ‘11’, ‘3B’: ‘0’, ‘HR’: ‘0’, ‘RBI’: ‘19’, ‘SB’: ‘100’, ‘CS’: ‘100’, ‘BB’: ‘1’, ‘SO’: ‘0’, ‘IBB’: ’‘, ’HBP’: ’‘, ’SH’: ’‘, ’SF’: ’‘, ’GIDP’: ‘1’}, {‘playerID’: ‘bellast01’, ‘yearID’: ‘1872’, ‘stint’: ‘1’, ‘teamID’: ‘TRO’, ‘lgID’: ‘NA’, ‘G’: ‘23’, ‘AB’: ‘115’, ‘R’: ‘22’, ‘H’: ‘30’, ‘2B’: ‘4’, ‘3B’: ‘0’, ‘HR’: ‘0’, ‘RBI’: ‘17’, ‘SB’: ‘100’, ‘CS’: ‘100’, ‘BB’: ‘0’, ‘SO’: ‘0’, ‘IBB’: ’‘, ’HBP’: ’‘, ’SH’: ’‘, ’SF’: ’‘, ’GIDP’: ‘2’}, {‘playerID’: ‘allisdo01’, ‘yearID’: ‘1872’, ‘stint’: ‘1’, ‘teamID’: ‘TRO’, ‘lgID’: ‘NA’, ‘G’: ‘23’, ‘AB’: ‘114’, ‘R’: ‘23’, ‘H’: ‘35’, ‘2B’: ‘4’, ‘3B’: ‘2’, ‘HR’: ‘0’, ‘RBI’: ‘20’, ‘SB’: ‘100’, ‘CS’: ‘100’, ‘BB’: ‘1’, ‘SO’: ‘3’, ‘IBB’: ’‘, ’HBP’: ’‘, ’SH’: ’‘, ’SF’: ’‘, ’GIDP’: ‘2’}, {‘playerID’: ‘mcmuljo01’, ‘yearID’: ‘1871’, ‘stint’: ‘1’, ‘teamID’: ‘TRO’, ‘lgID’: ‘NA’, ‘G’: ‘29’, ‘AB’: ‘136’, ‘R’: ‘38’, ‘H’: ‘38’, ‘2B’: ‘0’, ‘3B’: ‘5’, ‘HR’: ‘0’, ‘RBI’: ‘32’, ‘SB’: ‘100’, ‘CS’: ‘100’, ‘BB’: ‘8’, ‘SO’: ‘6’, ‘IBB’: ’‘, ’HBP’: ’‘, ’SH’: ’‘, ’SF’: ’‘, ’GIDP’: ‘2’}, {‘playerID’: ‘mcgeami01’, ‘yearID’: ‘1871’, ‘stint’: ‘1’, ‘teamID’: ‘TRO’, ‘lgID’: ‘NA’, ‘G’: ‘29’, ‘AB’: ‘148’, ‘R’: ‘42’, ‘H’: ‘39’, ‘2B’: ‘4’, ‘3B’: ‘0’, ‘HR’: ‘0’, ‘RBI’: ‘12’, ‘SB’: ‘100’, ‘CS’: ‘100’, ‘BB’: ‘6’, ‘SO’: ‘0’, ‘IBB’: ’‘, ’HBP’: ’‘, ’SH’: ’‘, ’SF’: ’‘, ’GIDP’: ‘0’}, {‘playerID’: ‘kingst01’, ‘yearID’: ‘1871’, ‘stint’: ‘1’, ‘teamID’: ‘TRO’, ‘lgID’: ‘NA’, ‘G’: ‘29’, ‘AB’: ‘144’, ‘R’: ‘45’, ‘H’: ‘57’, ‘2B’: ‘10’, ‘3B’: ‘6’, ‘HR’: ‘0’, ‘RBI’: ‘34’, ‘SB’: ‘100’, ‘CS’: ‘100’, ‘BB’: ‘1’, ‘SO’: ‘1’, ‘IBB’: ’‘, ’HBP’: ’‘, ’SH’: ’‘, ’SF’: ’‘, ’GIDP’: ‘0’}, {‘playerID’: ‘flynncl01’, ‘yearID’: ‘1871’, ‘stint’: ‘1’, ‘teamID’: ‘TRO’, ‘lgID’: ‘NA’, ‘G’: ‘29’, ‘AB’: ‘142’, ‘R’: ‘43’, ‘H’: ‘48’, ‘2B’: ‘6’, ‘3B’: ‘1’, ‘HR’: ‘0’, ‘RBI’: ‘27’, ‘SB’: ‘100’, ‘CS’: ‘100’, ‘BB’: ‘4’, ‘SO’: ‘2’, ‘IBB’: ’‘, ’HBP’: ’‘, ’SH’: ’‘, ’SF’: ’‘, ’GIDP’: ‘1’}, {‘playerID’: ‘flowedi01’, ‘yearID’: ‘1871’, ‘stint’: ‘1’, ‘teamID’: ‘TRO’, ‘lgID’: ‘NA’, ‘G’: ‘21’, ‘AB’: ‘105’, ‘R’: ‘39’, ‘H’: ‘33’, ‘2B’: ‘5’, ‘3B’: ‘4’, ‘HR’: ‘0’, ‘RBI’: ‘18’, ‘SB’: ‘100’, ‘CS’: ‘100’, ‘BB’: ‘4’, ‘SO’: ‘0’, ‘IBB’: ’‘, ’HBP’: ’‘, ’SH’: ’‘, ’SF’: ’‘, ’GIDP’: ‘0’}, {‘playerID’: ‘cravebi01’,

‘yearID’: ‘1871’, ‘stint’: ‘1’, ‘teamID’: ‘TRO’, ‘lgID’: ‘NA’, ‘G’: ‘27’, ‘AB’: ‘118’, ‘R’: ‘26’, ‘H’: ‘38’, ‘2B’: ‘8’, ‘3B’: ‘1’, ‘HR’: ‘0’, ‘RBI’: ‘26’, ‘SB’: ‘100’, ‘CS’: ‘100’, ‘BB’: ‘3’, ‘SO’: ‘0’, ‘IBB’: ’‘, ’HBP’: ’‘, ’SH’: ’‘, ’SF’: ’‘, ’GIDP’: ‘3’}, {‘playerID’: ‘connone01’, ‘yearID’: ‘1871’, ‘stint’: ‘1’, ‘teamID’: ‘TRO’, ‘lgID’: ‘NA’, ‘G’: ‘7’, ‘AB’: ‘33’, ‘R’: ‘6’, ‘H’: ‘7’, ‘2B’: ‘0’, ‘3B’: ‘0’, ‘HR’: ‘0’, ‘RBI’: ‘2’, ‘SB’: ‘100’, ‘CS’: ‘100’, ‘BB’: ‘0’, ‘SO’: ‘0’, ‘IBB’: ’‘, ’HBP’: ’‘, ’SH’: ’‘, ’SF’: ’‘, ’GIDP’: ‘0’}, {‘playerID’: ‘bellast01’, ‘yearID’: ‘1871’, ‘stint’: ‘1’, ‘teamID’: ‘TRO’, ‘lgID’: ‘NA’, ‘G’: ‘29’, ‘AB’: ‘128’, ‘R’: ‘26’, ‘H’: ‘32’, ‘2B’: ‘3’, ‘3B’: ‘3’, ‘HR’: ‘0’, ‘RBI’: ‘23’, ‘SB’: ‘100’, ‘CS’: ‘100’, ‘BB’: ‘9’, ‘SO’: ‘2’, ‘IBB’: ’‘, ’HBP’: ’‘, ’SH’: ’‘, ’SF’: ’‘, ’GIDP’: ‘2’}, {‘playerID’: ‘beaveed01’, ‘yearID’: ‘1871’, ‘stint’: ‘1’, ‘teamID’: ‘TRO’, ‘lgID’: ‘NA’, ‘G’: ‘3’, ‘AB’: ‘15’, ‘R’: ‘7’, ‘H’: ‘6’, ‘2B’: ‘0’, ‘3B’: ‘0’, ‘HR’: ‘0’, ‘RBI’: ‘5’, ‘SB’: ‘100’, ‘CS’: ‘100’, ‘BB’: ‘0’, ‘SO’: ‘0’, ‘IBB’: ’‘, ’HBP’: ’‘, ’SH’: ’‘, ’SF’: ’‘, ’GIDP’: ‘0’}]

update_by_template() unknown: should return 0 and print player doesn’t exist player doesn’t exist 0

update_by_key(): should return 1 1 [{‘playerID’: ‘abcd01’, ‘yearID’: ‘1999’, ‘stint’: ‘1’, ‘teamID’: ‘LAL’, ‘lgID’: ‘NA’, ‘G’: ‘100’, ‘AB’: ‘100’, ‘R’: ‘100’, ‘H’: ‘100’, ‘2B’: ‘100’, ‘3B’: ‘100’, ‘HR’: ‘100’, ‘RBI’: ‘100’, ‘SB’: ‘100’, ‘CS’: ‘100’, ‘BB’: ‘100’, ‘SO’: ‘100’, ‘IBB’: ’‘, ’HBP’: ’‘, ’SH’: ’‘, ’SF’: ’‘, ’GIDP’: ’ ’}]

[ ]: