4111_s21_hw1_programming

September 29, 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"; > Zeng_Xiangyi, xz2727

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 package designed to define, manipulate, retrieve and manage data in a database.

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

   A primary key is a subset of the attributes of entity that uniquely specify each element in the entity set. Using the primary key, you can easily identify and find unique element in an entity set, therefore inserting, updating, deleting or restoring data from entity set can be optimized by primary key.

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

   SQL is structural querry language, whereas MySQL Server is a database server program that manages access to the actual databases on disk or in the memory. DataGrip is a database management environment for developers and it is designed to query, create, and manage databases.

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

- One-to-One: **Each record of A table is related to only one record of B table and vice versa. Then A and B are in one-to-one relationship.**
- One-to-Many: **Each record in A table relates to many records in B table but each record of B table can only relate to one record of A table. Then A and B are in one-to-many relationship.**
- Many-to-One: **Each record in A table relates to many records in B table but each record of B table can only relate to one record of A table. Then B and A are in many-to-one relationship.**
- Many-to-Many: **Each record in A table relates to many records in B table and vice versa. Then A and B are in many-to-many relationship.**

5. What is an ER model?

   ER model stands for an Entity-Relationship model.It is a high-level data model. This model is used to define the data elements and relationship for a specified system.

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 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.

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
```

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

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.

[3]:
```
# 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:zxy3221915@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

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

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

```
[4]: [('people', 0, 'PRIMARY', 1, 'playerID', 'A', 19398, None, None, '', 'BTREE',
     '', '', 'YES', None)]
```

3.2.2 1) Convert all empty strings to NULL

Explanation We want to convert empty strings to NULL because that empty strings mean text value with length zero but NULL means absence of value which can be any type.

Change I updated empty strings of every columns of people to NULL.

SQL

```
update lahmansdb_to_clean.people
set playerID=NULL
where playerID='';
update lahmansdb_to_clean.people
```

```
set birthYear=NULL
where birthYear='';
update lahmansdb_to_clean.people
set birthMonth=NULL
where birthMonth='';
update lahmansdb_to_clean.people
set birthDay=NULL
where birthDay='';
update lahmansdb_to_clean.people
set birthCountry=NULL
where birthCountry='';
update lahmansdb_to_clean.people
set birthState=NULL
where birthState='';
update lahmansdb_to_clean.people
set birthCity=NULL
where birthCity='';
update lahmansdb_to_clean.people
set deathYear=NULL
where deathYear='';
update lahmansdb_to_clean.people
set deathMonth=NULL
where deathMonth='';
update lahmansdb_to_clean.people
set deathDay=NULL
where deathDay='';
update lahmansdb_to_clean.people
set deathCountry=NULL
where deathCountry='';
update lahmansdb_to_clean.people
set deathState=NULL
where deathState='';
update lahmansdb_to_clean.people
set deathCity=NULL
where deathCity='';
update lahmansdb_to_clean.people
set nameFirst=NULL
where nameFirst='';
update lahmansdb_to_clean.people
set nameLast=NULL
where nameLast='';
update lahmansdb_to_clean.people
set nameGiven=NULL
where nameGiven='';
update lahmansdb_to_clean.people
set weight=NULL
where weight='';
update lahmansdb_to_clean.people
```

```
set height=NULL
where height='';
update lahmansdb_to_clean.people
set bats=NULL
where bats='';
update lahmansdb_to_clean.people
set throws=NULL
where throws='';
update lahmansdb_to_clean.people
set debut=NULL
where debut='';
update lahmansdb_to_clean.people
set finalGame=NULL
where finalGame='';
update lahmansdb_to_clean.people
set retroID=NULL
where retroID='';
update lahmansdb_to_clean.people
set bbrefID=NULL
where bbrefID='';
```

Tests

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

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

```
[5]: []
```

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

Explanation We need datatypes for each column other than just text, so that we can add primary key or foreign key for the table.

Change I modified each column of peopel table to a appropriate datatype, such as birthYear of type int,birthCountry of type varchar(255) and debut pf type date.

SQL

```
alter table lahmansdb_to_clean.people
    modify column birthYear int,
    modify column birthMonth int,
    modify column birthDay int,
    modify column birthCountry varchar(255),
    modify column birthState varchar(255),
    modify column birthCity varchar(255),
    modify column deathYear int,
    modify column deathMonth int,
    modify column deathDay int,
    modify column deathCountry varchar(255),
    modify column deathState varchar(255),
    modify column deathCity varchar(255),
    modify column nameFirst varchar(255),
    modify column nameLast varchar(255),
    modify column nameGiven varchar(255),
    modify column weight int,
    modify column height int,
    modify column bats varchar(255),
    modify column throws varchar(255),
    modify column debut date,
    modify column finalGame date,
    modify column retroID varchar(255),
    modify column bbrefID varchar(255);
```

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 Adding an isDead Column as of datatype char(1), since its value can only be 'Y' or 'N', and we can know a player is dead or not from the 'deathYear' whether it's null or not.

Change I added isDead column to people table and update the values of this column based on the corresponding value of the deathYear column.

SQL

```
alter table lahmansdb_to_clean.people
add column isDead char(1) not null after birthCity;
update lahmansdb_to_clean.people
set isDead = 'Y' where deathYear is not null;
update lahmansdb_to_clean.people
set isDead = 'N' where deathYear is null ;
```

Tests

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

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

```
[7]: [('aardsda01', 1981, 12, 27, 'USA', 'CO', 'Denver', 'N', None, None, None, None,
     None, None, 'David', 'Aardsma', 'David Allan', 215, 75, 'R', 'R',
     datetime.date(2004, 4, 6), datetime.date(2015, 8, 23), 'aardd001', 'aardsda01'),
      ('aaronha01', 1934, 2, 5, 'USA', 'AL', 'Mobile', 'N', None, None, None, None,
     None, None, 'Hank', 'Aaron', 'Henry Louis', 180, 72, 'R', 'R',
     datetime.date(1954, 4, 13), datetime.date(1976, 10, 3), 'aaroh101',
     'aaronha01'),
      ('aasedo01', 1954, 9, 8, 'USA', 'CA', 'Orange', 'N', None, None, None, None,
     None, None, 'Don', 'Aase', 'Donald William', 190, 75, 'R', 'R',
     datetime.date(1977, 7, 26), datetime.date(1990, 10, 3), 'aased001', 'aasedo01'),
      ('abadan01', 1972, 8, 25, 'USA', 'FL', 'Palm Beach', 'N', None, None, None,
     None, None, None, 'Andy', 'Abad', 'Fausto Andres', 184, 73, 'L', 'L',
     datetime.date(2001, 9, 10), datetime.date(2006, 4, 13), 'abada001', 'abadan01'),
      ('abadfe01', 1985, 12, 17, 'D.R.', 'La Romana', 'La Romana', 'N', None, None,
     None, None, None, None, 'Fernando', 'Abad', 'Fernando Antonio', 235, 74, 'L',
     'L', datetime.date(2010, 7, 28), datetime.date(2019, 9, 28), 'abadf001',
     'abadfe01'),
      ('abbotgl01', 1951, 2, 16, 'USA', 'AR', 'Little Rock', 'N', None, None, None,
     None, None, None, 'Glenn', 'Abbott', 'William Glenn', 200, 78, 'R', 'R',
     datetime.date(1973, 7, 29), datetime.date(1984, 8, 8), 'abbog001', 'abbotgl01'),
      ('abbotje01', 1972, 8, 17, 'USA', 'GA', 'Atlanta', 'N', None, None, None, None,
     None, None, 'Jeff', 'Abbott', 'Jeffrey William', 190, 74, 'R', 'L',
     datetime.date(1997, 6, 10), datetime.date(2001, 9, 29), 'abboj002',
     'abbotje01'),
      ('abbotji01', 1967, 9, 19, 'USA', 'MI', 'Flint', 'N', None, None, None, None,
     None, None, 'Jim', 'Abbott', 'James Anthony', 200, 75, 'L', 'L',
     datetime.date(1989, 4, 8), datetime.date(1999, 7, 21), 'abboj001', 'abbotji01'),
      ('abbotku01', 1969, 6, 2, 'USA', 'OH', 'Zanesville', 'N', None, None, None,
     None, None, None, 'Kurt', 'Abbott', 'Kurt Thomas', 180, 71, 'R', 'R',
     datetime.date(1993, 9, 7), datetime.date(2001, 4, 13), 'abbok002', 'abbotku01'),
      ('abbotky01', 1968, 2, 18, 'USA', 'MA', 'Newburyport', 'N', None, None, None,
     None, None, None, 'Kyle', 'Abbott', 'Lawrence Kyle', 200, 76, 'L', 'L',
     datetime.date(1991, 9, 10), datetime.date(1996, 8, 24), 'abbok001',
     'abbotky01')]
```

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 If we miss some information for making the date then we should leave the value NULL. The datatype of the columns should be date.

Change I added two columns in people table and update the value of date based on the corresponding year,month and day if all information exist,if not leave the value NULL.

SQL

```
alter table lahmansdb_to_clean.people
add column deathDate date after isDead,
add column birthDate date after playerID;

update lahmansdb_to_clean.people
set birthDate = str_to_date(concat(convert(birthYear,char),',',convert(birthMonth,char),',',co
where birthYear is not null and birthMonth is not null  and birthDay is not null ;

update lahmansdb_to_clean.people
set deathDate = str_to_date(concat(convert(deathYear,char),',',convert(deathMonth,char),',',co
where deathYear is not null and deathMonth is not null and deathDay is not null ;
```

Tests

```
[10]: %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.
```

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

```
[11]: %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.
```

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

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 lahmansdb_to_clean.people
(
    playerID varchar(15) not null
        primary key,
    birthDate date null,
    birthYear int null,
    birthMonth int null,
    birthDay int null,
    birthCountry varchar(255) null,
    birthState varchar(255) null,
    birthCity varchar(255) null,
    isDead char not null,
    deathDate date 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 char null,
    throws char null,
    debut date null,
    finalGame date null,
    retroID varchar(255) null,
    bbrefID varchar(255) null
);
```

3.3 Batting Table

3.3.1 1) Convert all empty strings to NULL

Explanation We want to convert empty strings to NULL because that empty strings mean text value with length zero but NULL means absence of value which can be any type.

Change I updated the empty strings to NULL for each column of the batting table.

SQL

```sql
update lahmansdb_to_clean.batting
set playerID=NULL
where playerID='';
update lahmansdb_to_clean.batting
set yearID=NULL
where yearID='';
update lahmansdb_to_clean.batting
set stint=NULL
where stint='';
update lahmansdb_to_clean.batting
set teamID=NULL
where teamID='';
update lahmansdb_to_clean.batting
set lgID=NULL
where lgID='';
update lahmansdb_to_clean.batting
set G=NULL
where G='';
update lahmansdb_to_clean.batting
set AB=NULL
where AB='';
update lahmansdb_to_clean.batting
set R=NULL
where R='';
update lahmansdb_to_clean.batting
set H=NULL
where H='';
update lahmansdb_to_clean.batting
set 2B=NULL
where 2B='';
update lahmansdb_to_clean.batting
set 3B=NULL
where 3B='';
update lahmansdb_to_clean.batting
set HR=NULL
where HR='';
update lahmansdb_to_clean.batting
set RBI=NULL
where RBI='';
update lahmansdb_to_clean.batting
set SB=NULL
where SB='';
update lahmansdb_to_clean.batting
set CS=NULL
```

```
where CS='';
update lahmansdb_to_clean.batting
set BB=NULL
where BB='';
update lahmansdb_to_clean.batting
set SO=NULL
where SO='';
update lahmansdb_to_clean.batting
set IBB=NULL
where IBB='';
update lahmansdb_to_clean.batting
set HBP=NULL
where HBP='';
update lahmansdb_to_clean.batting
set SH=NULL
where SH='';
update lahmansdb_to_clean.batting
set SF=NULL
where SF='';
update lahmansdb_to_clean.batting
set GIDP=NULL
where GIDP='';
```

Tests

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

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

```
[16]: [(756,)]
```

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

Explanation We need datatypes for each column other than just text, so that we can add primary key or foreign key for the table.

Change I changed the data type of each column from text to the appropriate value, such as playerID of type varchar(9), teamID of type char(3) and IBB smallint.

SQL

```
alter table lahmansdb_to_clean.batting
    modify column playerID varchar(9),
    modify column yearID smallint,
    modify column stint smallint,
    modify column teamID char(3),
    modify column lgID char(2),
    modify column G smallint,
    modify column AB smallint,
    modify column R smallint,
    modify column H smallint,
    modify column 2B smallint,
    modify column 3B smallint,
    modify column HR smallint,
    modify column RBI smallint,
    modify column SB smallint,
    modify column CS smallint,
    modify column BB smallint,
    modify column SO smallint,
    modify column IBB smallint,
    modify column HBP smallint,
    modify column SH smallint,
    modify column SF smallint,
    modify column GIDP smallint;
```

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 I'll choose Composite key.

Explanation This composite key consisting of playerID,yearID and stint can uniquely identify each row in batting table.

SQL

```
alter table lahmansdb_to_clean.batting
add primary key (playerID,yearID,stint);
```

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', 19758, 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 could delete the two people's rows in batting table or add two people's new rows in people table.

Change I added a foreign key playerID in batting table referencing primary key playerID in people table.

SQL

```
alter table lahmansdb_to_clean.batting
add foreign key (playerID) references people(playerID);
```

Tests

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

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

```
[28]: []
```

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 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_ibfk_1
        foreign key (playerID) references lahmansdb_to_clean.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.

[1]:
```
%reload_ext sql
%sql mysql+pymysql://root:zxy3221915@localhost/lahmansbaseballdb
```

4.1 Question 0

What is the average salary in baseball history?

[3]:
```
%sql select avg(salary) from lahmansbaseballdb.salaries;
```

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

[3]:
```
[(2085634.053125473,)]
```

[4]:
```

```

```
 * 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

[5]:
```
%%sql select nameFirst,nameLast,playerID,schoolID,yearID,birthState
from lahmansbaseballdb.people
```

```
    join
        lahmansbaseballdb.collegeplaying
    using(playerID)
where nameFirst = 'Sam' and birthCountry = 'USA' limit 10;
```

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

```
[5]: [('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.

```
[8]: %%sql update lahmansbaseballdb.schools
     set name_full = 'Columbia University in the City of New York'
     where schoolID = 'columbia';
     select * from lahmansbaseballdb.schools where schoolID = 'columbia';
```

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

```
[8]: [('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: ['collied01']
   - field_lists: ['nameFist','nameLast']

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

   We should check whether the new Primary key is duplicate with the already existed primary keys of the table.

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

   We should check whether the values we want to update will create duplicated primary key with other rows in the table.

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:

```
/home/simon/Applications/anaconda3/envs/HW1/bin/python /home/simon/MS_CS/Databases/W4111F21/Ho
DEBUG:root:CSVDataTable.___init___: data = { "table_name": "People", "connect_info": { "di-
rectory": "/home/simon/MS_CS/Databases/W4111F21/HomeworkAssignments/HW1/4111_s21_hw1_program
"file_name": "People.csv" }, "key_columns": [ "playerID" ], "debug": true }

find_by_primary_key(): Known Record {'playerID': 'aardsda01', 'birthYear': '1981', 'birth-
Month': '12', 'birthDay': '27', 'birthCountry': 'USA', 'birthState': 'CO', 'birthCity': 'Denver',
'deathYear': '', 'deathMonth': '', 'deathDay': '', 'deathCountry': '', 'deathState': '', 'deathC-
ity': '', '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 DEBUG:root:CSVDataTable._load: Loaded 19619 rows
[{'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 row deleted

delete_by_key(): UnKnown Record 0 row deleted

delete_by_template(): Known Template 1527 rows deleted

delete_by_template(): UnKnown Template 0 rows deleted

update_by_key(): Known Record,no duplicate primary key 1 row updated

update_by_key(): UnKnown Record,no duplicate primary key 0 row updated

update_by_template(): Known Template,no duplicate primary key 100 rows updated

update_by_template(): UnKnown Template,no duplicate primary key 0 rows updated

insert(): no duplicate primary key None

update_by_key(): Known Record,duplicate primary key An error occurred: can't update with
duplicated primary key
```

```
update_by_key(): UnKnown Record,duplicate primary key 0 row updated

update_by_template(): Known Template,duplicate primary key An error occurred: can’t update
with duplicated primary key

update_by_template(): UnKnown Template,duplicate primary key 0 rows updated

insert(): duplicate primary key An error occurred: can’t insert with duplicated primary key

Process finished with exit code 0

DEBUG:root:CSVDataTable.___init___: data = { “table_name”: “Batting”, “connect_info”: {
“directory”: “/home/simon/MS_CS/Databases/W4111F21/HomeworkAssignments/HW1/4111_s21_hw1_progr
“file_name”: “Batting.csv” }, “key_columns”: [ “playerID”, “yearID”, “stint” ], “debug”: true }

find_by_primary_key(): Known Record DEBUG:root:CSVDataTable._load: Loaded 105861 rows
{‘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’: ‘addybo01’, ‘yearID’: ‘1871’, ‘stint’: ‘1’,
‘teamID’: ‘RC1’, ‘lgID’: ‘NA’, ‘G’: ‘25’, ‘AB’: ‘118’, ‘R’: ‘30’, ‘H’: ‘32’, ‘2B’: ‘6’, ‘3B’: ‘0’, ‘HR’:
‘0’, ‘RBI’: ‘13’, ‘SB’: ‘8’, ‘CS’: ‘1’, ‘BB’: ‘4’, ‘SO’: ‘0’, ‘IBB’: ’‘, ’HBP’: ’‘, ’SH’: ’‘, ’SF’: ’‘, ’GIDP’:
‘0’}]

find_by_template(): UnKnown Template []

delete_by_key(): Known Record 1 row deleted

delete_by_key(): UnKnown Record 0 row deleted

delete_by_template(): Known Template 5 rows deleted

delete_by_template(): UnKnown Template 0 rows deleted

update_by_key(): Known Record,no duplicate primary key 1 row updated

update_by_key(): UnKnown Record,no duplicate primary key 0 row updated

update_by_template(): Known Template,no duplicate primary key 82 rows updated

update_by_template(): UnKnown Template,no duplicate primary key 0 rows updated

insert(): no duplicate primary key None

update_by_key(): Known Record,duplicate primary key An error occurred: can’t update with
duplicated primary key

update_by_key(): UnKnown Record,duplicate primary key 0 row updated

update_by_template(): Known Template,duplicate primary key An error occurred: can’t update
with duplicated primary key

update_by_template(): UnKnown Template,duplicate primary key 0 rows updated

insert(): duplicate primary key An error occurred: can’t insert with duplicated primary key

Process finished with exit code 0
```