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

Homework 3

Introduction

- This notebook contains HW3. **Both Programming and Nonprogramming tracks should complete this homework.**
- You will submit **PDF and ZIP files** for this assignment. Gradescope will have two separate assignments for these.
- For the PDF:
  - The most reliable way to save as PDF is to go to your browser's menu bar and click `File -> Print`. Switch the orientation to landscape mode, and hit save.
  - **MAKE SURE ALL YOUR WORK (CODE AND SCREENSHOTS) IS VISIBLE ON THE PDF. YOU WILL NOT GET CREDIT IF ANYTHING IS CUT OFF.** Reach out for troubleshooting.
  - **MAKE SURE YOU DON'T SUBMIT A SINGLE PAGE PDF.** Your PDF should have multiple pages.
- For the ZIP:
  - Zip a folder containing this notebook and any screenshots.
  - You may delete any unnecessary files, such as caches.

Setup

In [146…

```
%load_ext sql
%sql mysql+pymysql://root:dbuserbdbuser@localhost
%sql SELECT 1
```

```
The sql extension is already loaded. To reload it, use:
  %reload_ext sql
   mysql+pymysql://general_user:***@localhost/s24_hw3
 * mysql+pymysql://root:***@localhost
   mysql+pymysql://root:***@localhost/s24_hw3
1 rows affected.
```

Out[146]:

1
1

In [147…

```
%%sql

drop schema if exists s24_hw3;
create schema s24_hw3;
use s24_hw3;
```

```
   mysql+pymysql://general_user:***@localhost/s24_hw3
 * mysql+pymysql://root:***@localhost
   mysql+pymysql://root:***@localhost/s24_hw3
1 rows affected.
1 rows affected.
0 rows affected.
```

Out[147]: []

In [148…

```
import copy
import math

import pandas
import pymysql
from sqlalchemy import create_engine

sql_conn = pymysql.connect(
    user="root",
    password="dbuserbdbuser",
    host="localhost",
    port=3306,
    cursorclass=pymysql.cursors.DictCursor,
    autocommit=True
)
engine = create_engine("mysql+pymysql://root:dbuserbdbuser@localhost")

cur = sql_conn.cursor()
res = cur.execute("SELECT 1")
res = cur.fetchall()
res
```

Out[148]:

```
[{'1': 1}]
```

Written

- As usual, try to keep things short. Do not bloviate.
- You may use external resources, but you should cite your sources.

W1

Explain and list some differences between

- RAM
- Solid state drives
- Hard drives

RAM: it is volatile memory used for temporarily storing data that a computer needs to access quickly while it is running. It is much faster than other types of storage but loses its data when the power is turned off.

Solid state drives: it is a type of non-volatile storage media that store data on flash memeory. It is faster than hard drives because it has no moving parts and access data electronically instead of phsically. It is more durable and consumes less power.

Hard drives: it uses magnetic storage to store and retrieve data using one or more rigid rapidlly rotating disks coated with magnetic material. It is genearlly cheaper per GB than SSDs but is slower, more prone to damage due to moving parts, and use more power.

W2

With regards to disk drives, define

- Seek time
- Rotational latency time
- Transfer time/data transfer rate

Seek time: it is the time it takes for a hard drive's read/write head to move to the area of the disk where the data is stored. It measures the speed at which the drive can locate a specific piece of stored information, typically measured in ms.

Rotational Latency time: it refers to the delay experienced while waiting for the desired sector of the disk to rotate into position under the read/write head after the head has been positiond over the correct track. it depends on the rotational speed of the disk, usually measured in ms as well, and is a factor in the overall access time alongside seek time.

Tranfer time/data transfer rate: it is the time it takes to actually read or write data to the disk once the head is in position. Transfer time is influenced by how fast the disk spins and the density of the data on the disk. Data transfer rate, often measured in megabytes per second (MB/s) or gigabits per second (Gbps), indicates the speed at which data can be read from or written the drive.

W3

Explain the concepts of

- Logical block addressing
- Cylinder-head-sector addressing

Logical block addreassing is a method that uses a linear addressing system for specifying locations on a storage device, simplifying the interface between the device and the system by abstracting the physical details.

Cylinder-head-sector addressing refers to a method that specifies data locations on a hard drive based on its physical geometry, using the cylinder, head, and sector to pinpoint data, which is less efficient for large-capacity drives compared to Logical block addreassing.

W4

Define and list some benefits of

- Fixed-length records
- Variable-length records
- Row-oriented storage
- Column-oriented storage

Fixed-length records have a consistent size for each record, simplifying storage allocation and access. Benefits include easier data retrieval and updates, as well as efficient use of indexes.

Variable-length records can vary in size, allowing for more flexible storage of data. Benefits include efficient storage space usage, as it avoids wasting space on unused portions of records, and the ability to store data of varying sizes without padding.

Row-oriented storage stores table data by row, making it optimal for transaction processing and CRUD operations (Create, Read, Update, Delete) on a complete record. Benefits include faster data retrieval and updates for individual records, and efficient processing of transactions.

Column-oriented storage stores table data by column rather than by row, optimizing for read-heavy operations and data analysis. Benefits include improved performance for queries accessing only a subset of columns, better compression rates due to column data similarity, and efficiency in aggregation and analytics operations.

W5

Explain and list some differences between

- RAID 0
- RAID 1
- RAID 5

RAID 0:it splits data evenly across two or more disks with no redundancy. It improves performance by increasing read/write speeds but offers no data protection; if one drive fails, all data is lost.

RAID 1:it involves duplicating data on two or more disks, providing high data protection since data is recoverable if one disk fails. However, it requires twice the storage capacity for the data and may have slower write performance due to the mirroring process.

RAID 5: it distributes data and parity information across three or more disks. It offers a balance between performance and data protection, allowing for the recovery of data if a single drive fails. RAID 5 provides better storage efficiency than RAID 1 but requires more complex read and write operations due to the parity calculation.

SQL

Overview

- The `data` directory contains a file `People.csv`. The columns are
  - `nameFirst`
  - `nameLast`
  - `birthYear`
  - `birthCountry`
  - `deathYear`
  - `deathCountry`
- For Nonprogramming students, note that this `People.csv` differs from the one you loaded in HW2. Do not mix the two files.
- **There is no one right answer for this section.** You can come up with and document your own design (as long as they satisfy the requirements).

Create Table

- Create a table based on the structure of `People.csv`
  - **You must add an additional attribute, `personID`, which has type char(9)**
    - `personID` should be the primary key of your table
  - `nameFirst` and `nameLast` cannot be null. The other (non-PK) columns can be null.
  - You should choose reasonable data types for the attributes
    - Do not use the `year` data type for `birthYear` or `deathYear`. The range for year is too small.
  - Your table will be empty for the next few sections. We will insert data later.

In [149…

```
%%sql

create table people (
    personID char(9) primary key,
    nameFirst varchar(64) not null,
    nameLast varchar(64) not null,
    birthYear char(4),
    birthCountry varchar(64),
    deathYear char(4),
    deathCountry varchar(64)
);
```

```
   mysql+pymysql://general_user:***@localhost/s24_hw3
 * mysql+pymysql://root:***@localhost
   mysql+pymysql://root:***@localhost/s24_hw3
0 rows affected.
```

Out[149]:

```
[]
```

Person ID Function

- `personID` is formed using the following rules:

1. The ID consists of three sections: `[lastSubstr][firstSubstr][number]`
2. `lastSubstr` is formed by lowercasing `nameLast`, then taking the first 5 letters. If `nameLast` is less than 5 letters, use the entire `nameLast`.
3. `firstSubstr` is formed by lowercasing `nameFirst`, then taking the first 2 letters. If `nameFirst` is less than 2 letters, use the entire `nameFirst`.
4. For a specific combination of `[lastSubstr][firstSubstr]`, `number` starts from 1 and increments. `number` should be padded to have length 2.
5. `nameFirst` and `nameLast` may contain periods ".", hyphens "-", and spaces " ". You should remove these characters from `nameFirst` and `nameLast` **before** doing the above substring processing.

- As an example, starting from an empty table, below is what `personID` would be assigned to the following names (assuming they were inserted in the order that they are shown)

nameFirst	nameLast	personID
Donald	Ferguson	fergudo01
David	Aardsma	aardsda01
Doe	Fergue	fergudo02
J. J.	Park	parkjj01

- Write a SQL function that generates a person ID using the above rules
  - You should determine what parameters and return type are needed
  - This function will be called by triggers in the next section. **It is up to you which logic you put in the function and which logic you put in the triggers.**
    - That is, if you plan to place the bulk of your logic in your triggers, then your function could be a few lines.
  - You may define helper functions
  - You may add additional attributes to your table if it helps

In [150…

```
%%sql

alter table people add nameSubstr char(7) not null;
alter table people add idNum int not null;

alter table people add rowCounter int unique;

create function stripString (string varchar(64))
returns varchar(64) deterministic
begin
    return replace(replace(replace(string, ".", ""), " ", ""), "'", "");
end;

create function generateNameSubstr (nameFirst varchar(64), nameLast varchar(64))
returns char(7) deterministic
begin

    declare firstSubstr char(2);
    declare lastSubstr char(5);

    set firstSubstr = left(lower(stripString(nameFirst)), 2);
    set lastSubstr = left(lower(stripString(nameLast)), 5);

    return concat(lastSubstr, firstSubstr);
end;

create function getNextNumber (_nameSubstr char(7))
returns int reads sql data
begin

    declare res int;

    select ifnull(max(p.idNum), 0) + 1 into res
    from people p
    where p.nameSubstr = _nameSubstr;

    return res;
end;

create function generatePersonID (nameSubstr char(7), idNum int)
returns char(9) deterministic
begin

    return concat(nameSubstr, lpad(idNum, 2, "0"));

end;
```

```
   mysql+pymysql://general_user:***@localhost/s24_hw3
 * mysql+pymysql://root:***@localhost
   mysql+pymysql://root:***@localhost/s24_hw3
0 rows affected.
0 rows affected.
0 rows affected.
0 rows affected.
0 rows affected.
0 rows affected.
0 rows affected.
```

Out[150]:

```
[]
```

Insert and Update Triggers

- We want to automatically generate `personID` using the function above whenever a row is inserted. The user should not need to manually specify it.
- Write a SQL trigger that runs every time a row is inserted
  - The trigger should generate a person ID for the row based on its `nameFirst` and `nameLast`; it should then set the `personID` for that row.
    - This should occur even if the user attempts to manually set `personID`. The user's value for `personID` is ignored.
    - You should call the function you wrote above
- Write another SQL trigger that runs every time a row is updated
  - There is no `immutable` keyword in MySQL; however, we can simulate immutability using a trigger. If the user attempts to modify `personID` directly, throw an exception.
  - If the user modifies `nameFirst` or `nameLast` such that the `personID` is no longer valid based on the rules in the previous section (specifically, if `[lastSubstr][firstSubstr]` is no longer the same as before), you should re-generate `personID` and re-set it.
    - You should call the function you wrote above
- **You are writing two SQL triggers for this section**

In [151…

```sql
%%sql

create trigger people_insert
before insert on people
for each row
begin

    declare _rowCounter int;

    set new.nameSubstr = generateNameSubstr(new.nameFirst, new.nameLast);
    set new.idNum = getNextNumber(new.nameSubstr);
    set new.personID = generatePersonID(new.nameSubstr, new.idNum);

    select ifnull(max(p.rowCounter), 0) + 1 into _rowCounter
    from people p;
    set new.rowCounter = _rowCounter;

end;

create trigger people_update
before update on people
for each row
begin

    declare newNameSubstr char(7);
    declare _rowCounter int;

    if new.personID <> old.personID then
        signal sqlstate "45000"
        set message_text = "cannot manually change personID", mysql_errno = 1001;
    end if;

    set newNameSubstr = generateNameSubstr(new.nameFirst, new.nameLast);
    if newNameSubstr <> old.nameSubstr then
        set new.nameSubstr = newNameSubstr;
        set new.idNum = getNextNumber(new.nameSubstr);
        set new.personID = generatePersonID(new.nameSubstr, new.idNum);
    end if;

    select ifnull(max(p.rowCounter), 0) + 1 into _rowCounter
    from people p;
    set new.rowCounter = _rowCounter;

end;
```

```
   mysql+pymysql://general_user:***@localhost/s24_hw3
 * mysql+pymysql://root:***@localhost
   mysql+pymysql://root:***@localhost/s24_hw3
0 rows affected.
0 rows affected.
```

Out[151]: []

Create and Update Procedures

- You must implement two stored procedures

1. `createPerson(nameFirst, nameLast, birthYear, birthCountry, deathYear, deathCountry, personID)`
   A. `personID` is an out parameter. It should be set to the ID generated for the person.
   B. All the other parameters are in paramaters
2. `updatePerson(personID, nameFirst, nameLast, birthYear, birthCountry, deathYear, deathCountry, newPersonID)`
   A. `newPersonID` is an out parameter. It should be set to the ID of the person after the update (even if it didn't change).
   B. All the other parameters are in parameters.
      a. `personID` is used to identify the row that the user wants to update. The other in parameters are the values that the user wants to set.
      b. **Ignore null in parameters.** Only update an attribute if the in parameter is non-null.

- Depending on how you implemented your triggers, these procedures could be as simple as calling `insert` / `update` and setting the out parameters

In [152…

```
%%sql

create function getPersonID ()
returns char(9) reads sql data
begin

    declare res char(9);

    select p.personID into res from people p
    order by p.rowCounter desc limit 1;

    return res;

end;

create procedure createPerson (
    in _nameFirst varchar(64),
    in _nameLast varchar(64),
    in _birthYear char(4),
    in _birthCountry varchar(64),
    in _deathYear char(4),
    in _deathCountry varchar(64),
    out _personID char(9)
)
begin

    insert into people (nameFirst, nameLast, birthYear, birthCountry, deathYear, deathCountry)
    values (_nameFirst, _nameLast, _birthYear, _birthCountry, _deathYear, _deathCountry);

    set _personID = getPersonID();

end;

create procedure updatePerson (
    in _personID char(9),
    in _nameFirst varchar(64),
    in _nameLast varchar(64),
    in _birthYear char(4),
    in _birthCountry varchar(64),
    in _deathYear char(4),
    in _deathCountry varchar(64),
    out _newPersonID char(9)
)
begin

    update people set
        nameFirst = ifnull(_nameFirst, nameFirst),
        nameLast = ifnull(_nameLast, nameLast),
        birthYear = ifnull(_birthYear, birthYear),
        birthCountry = ifnull(_birthCountry, birthCountry),
        deathYear = ifnull(_deathYear, deathYear),
        deathCountry = ifnull(_deathCountry, deathCountry)
    where personID = _personID;

    set _newPersonID = getPersonID();

end;
```

```
   mysql+pymysql://general_user:***@localhost/s24_hw3
 * mysql+pymysql://root:***@localhost
   mysql+pymysql://root:***@localhost/s24_hw3
0 rows affected.
0 rows affected.
0 rows affected.
```

Out[152]: []

Security

- You must create a new user `general_user` and use security to allow it to perform only `select` and `execute` operations (i.e., no `insert`, `delete`, and `update` operations)

In [153…

```
%%sql

drop user if exists "general_user"@"%";
create user "general_user"@"%" identified by "dbuserbdbuser";

revoke all privileges on *.* from "general_user"@"%";
grant select, execute on s24_hw3.* to "general_user"@"%";
```

```
   mysql+pymysql://general_user:***@localhost/s24_hw3
 * mysql+pymysql://root:***@localhost
   mysql+pymysql://root:***@localhost/s24_hw3
0 rows affected.
0 rows affected.
0 rows affected.
0 rows affected.
```

Out[153]: []

Inheritance Using Views

- A person can be a player or manager
  - That is, a player is-a person, and a manager is-a person
- Describe how you could implement this inheritance relationship given that you already have your `people` table
  - No code is necessary

Use views to implement inheritance. Create a view that represents the `player`, combining general people attributes with player-specific ones. Similarly, have a `manager` view, merging general people attributes with manager-specific ones. `player` would only contain rows that are players, and `manager` would only contain rows that are managers.

Data Insertion Testing

- The cells below load data from `People.csv` to your database
  - No code is required on your part. Make sure everything runs without error.

In [154…

```
# Load People.csv into a dataframe.
# You may see NaNs in the non-null columns. This is fine.

people_df = pandas.read_csv("data/People.csv")
people_df.head(10)
```

Out[154]:

	nameFirst	nameLast	birthYear	birthCountry	deathYear	deathCountry
0	Ed	White	1926.0	USA	1982.0	USA
1	Sparky	Adams	1894.0	USA	1989.0	USA
2	Bob	Johnson	1959.0	USA	NaN	NaN
3	Johnny	Ryan	1853.0	USA	1902.0	USA
4	Jose	Alvarez	1956.0	USA	NaN	NaN
5	Andrew	Brown	1981.0	USA	NaN	NaN
6	Chris	Johnson	1984.0	USA	NaN	NaN
7	Johnny	Johnson	1914.0	USA	1991.0	USA
8	Albert	Williams	1954.0	Nicaragua	NaN	NaN
9	Ed	Brown	NaN	USA	NaN	NaN

In [155…

```
def add_person(p):
    """
    p is a dictionary containing the column values for either a student or an employee.
    """

    cur = sql_conn.cursor()

    # This function changes the data, converting nan to None.
    # So, we make a copy and change the copy.
    p_dict = copy.copy(p)
    for k, v in p_dict.items():
        if isinstance(v, float) and math.isnan(v):
            p_dict[k] = None

    # This provides a hint for what your stored procedure will look like.
    res = cur.callproc("s24_hw3.createPerson",
                       # The following are in parameters
                        (p_dict['nameFirst'],
                        p_dict['nameLast'],
                        p_dict['birthYear'],
                        p_dict['birthCountry'],
                        p_dict['deathYear'],
                        p_dict['deathCountry'],
                        # The following are out parameters for personID.
                        None))

    # After the procedure executes, the following query will select the out values.
    res = cur.execute("""SELECT @_s24_hw3.createPerson_6""")
    result = cur.fetchall()

    sql_conn.commit()
    cur.close()
    return result[0]["@_s24_hw3.createPerson_6"]  # Return personID
```

- Below is the main data insertion logic
  - `add_person` calls your `createPerson` procedure
  - The `data` directory also contains a file `People_Ids.csv`, which is the expected `personID` for each row after it is inserted. We'll use this to check your `createPerson` implementation.

In [156…

```
%sql truncate table s24_hw3.people

expected_ids_df = pandas.read_csv("data/People-Ids.csv", header=None)
expected_ids = [e[0] for e in expected_ids_df.values.tolist()]
```

```
for i, (p, e_id) in enumerate(zip(people_df.to_dict(orient="records"), expected_ids)):
    p_id = add_person(p)
    assert p_id == e_id, \
    f"Row {i}: Expected {e_id}, but got {p_id} for {p['nameFirst']} {p['nameLast']}"

print("Successfully inserted all data")
```

```
   mysql+pymysql://general_user:***@localhost/s24_hw3
 * mysql+pymysql://root:***@localhost
   mysql+pymysql://root:***@localhost/s24_hw3
0 rows affected.
Successfully inserted all data
```

Data Updating Testing

- The following cells test your update trigger and `updatePerson` implementation
  - No code is required on your part. Make sure everything runs as expected.
  - The tests assume you just finished the Data Insertion Testing section. You may run into issues if you run the Data Updating Testing section multiple times without reseting your data.

In [157…

```
# Switch back to root
%sql mysql+pymysql://root:dbuserbdbuser@localhost/s24_hw3

def transform(d):
    # %sql returns dict of attributes to one-tuples.
    # This function extracts the values from the one-tuples.
    return {k: v[0] for k, v in d.items()}

def is_subset(d1, d2):
    # Checks if d1 is a subset of a d2
    for k, v in d1.items():
        if k not in d2 or str(d2[k]) != str(v):
            return False
    return True
```

In [158…

```
# Create new person to test on

%sql call createPerson("Babe", "Ruth", null, null, null, null, @ruthID)
res1 = %sql select * from people p where p.personID = @ruthID
res1_d = transform(res1.dict())
expected_d = dict(
    personID="ruthba01",
    nameFirst="Babe",
    nameLast="Ruth",
    birthYear=None,
    birthCountry=None,
    deathYear=None,
    deathCountry=None
)

print(res1)

assert is_subset(expected_d, res1_d), \
f"Row has unexpected value. Expected {expected_d}, but got {res1_d}"

print("Success")
```

```
   mysql+pymysql://general_user:***@localhost/s24_hw3
   mysql+pymysql://root:***@localhost
 * mysql+pymysql://root:***@localhost/s24_hw3
1 rows affected.
   mysql+pymysql://general_user:***@localhost/s24_hw3
   mysql+pymysql://root:***@localhost
 * mysql+pymysql://root:***@localhost/s24_hw3
1 rows affected.
+----------+-----------+----------+-----------+--------------+-----------+--------------+------------+------
-+------------+
| personID | nameFirst | nameLast | birthYear | birthCountry | deathYear | deathCountry | nameSubstr | idNum
| rowCounter |
+----------+-----------+----------+-----------+--------------+-----------+--------------+------------+------
-+------------+
| ruthba01 |    Babe   |   Ruth   |    None   |     None     |    None   |     None     |   ruthba   |   1
|    3215    |
+----------+-----------+----------+-----------+--------------+-----------+--------------+------------+------
-+------------+
Success
```

In [159…

```
# Update birth country and year
%sql call updatePerson(@ruthID, null, null, 1895, "USA", 1948, "USA", @ruthID)
res2 = %sql select * from people p where p.personID = @ruthID
res2_d = transform(res2.dict())
expected_d = dict(
    personID="ruthba01",
    nameFirst="Babe",
    nameLast="Ruth",
    birthYear=1895,
    birthCountry="USA",
    deathYear=1948,
    deathCountry="USA"
)

print(res2)

assert is_subset(expected_d, res2_d), \
f"Row has unexpected value. Expected {expected_d}, but got {res2_d}"

print("Success")
```

```
   mysql+pymysql://general_user:***@localhost/s24_hw3
   mysql+pymysql://root:***@localhost
 * mysql+pymysql://root:***@localhost/s24_hw3
1 rows affected.
   mysql+pymysql://general_user:***@localhost/s24_hw3
   mysql+pymysql://root:***@localhost
 * mysql+pymysql://root:***@localhost/s24_hw3
1 rows affected.
+----------+-----------+----------+-----------+--------------+-----------+--------------+------------+------
-+------------+
| personID | nameFirst | nameLast | birthYear | birthCountry | deathYear | deathCountry | nameSubstr | idNum
| rowCounter |
+----------+-----------+----------+-----------+--------------+-----------+--------------+------------+------
-+------------+
| ruthba01 |    Babe   |   Ruth   |    1895   |     USA      |    1948   |     USA      |   ruthba   |   1
|    3216    |
+----------+-----------+----------+-----------+--------------+-----------+--------------+------------+------
-+------------+
Success
```

In [160…

```
# Checking that null is a noop
%sql call updatePerson(@ruthID, null, null, null, null, null, null, @ruthID)
res3 = %sql select * from people p where p.personID = @ruthID
res3_d = transform(res3.dict())

print(res3)

assert is_subset(expected_d, res3_d), \
f"Row has unexpected value. Expected {expected_d}, but got {res3_d}"

print("Success")
```

```
   mysql+pymysql://general_user:***@localhost/s24_hw3
   mysql+pymysql://root:***@localhost
 * mysql+pymysql://root:***@localhost/s24_hw3
1 rows affected.
   mysql+pymysql://general_user:***@localhost/s24_hw3
   mysql+pymysql://root:***@localhost
 * mysql+pymysql://root:***@localhost/s24_hw3
1 rows affected.
+----------+-----------+----------+-----------+--------------+-----------+--------------+------------+------
-+------------+
| personID | nameFirst | nameLast | birthYear | birthCountry | deathYear | deathCountry | nameSubstr | idNum
| rowCounter |
+----------+-----------+----------+-----------+--------------+-----------+--------------+------------+------
-+------------+
| ruthba01 |    Babe   |   Ruth   |    1895   |      USA     |    1948   |      USA     |   ruthba   |   1
|    3217    |
+----------+-----------+----------+-----------+--------------+-----------+--------------+------------+------
-+------------+
Success
```

In [161…

```python
# Try to manually set personID
# Note: You should get an OperationalError. If you get an AssertionError, then
# your trigger is not doing its job.

res4 = %sql update people set personID = "dff9" where personID = "ruthba01"

assert res4 is None, "Your trigger should throw an exception"

print("Success")
```

```
   mysql+pymysql://general_user:***@localhost/s24_hw3
   mysql+pymysql://root:***@localhost
 * mysql+pymysql://root:***@localhost/s24_hw3
(pymysql.err.OperationalError) (1001, 'cannot manually change personID')
[SQL: update people set personID = "dff9" where personID = "ruthba01"]
(Background on this error at: https://sqlalche.me/e/20/e3q8)
Success
```

In [162…

```python
# Check that update trigger updates personID if name changes

%sql call updatePerson(@ruthID, "George", "Herman", 1920, "USA", 2005, "USA", @ruthID)
res5 = %sql select * from people p where p.personID = @ruthID
res5_d = transform(res5.dict())
expected_d = dict(
    personID="hermage01",
    nameFirst="George",
    nameLast="Herman",
    birthYear=1920,
    birthCountry="USA",
    deathYear=2005,
    deathCountry="USA"
)

print(res5)

assert is_subset(expected_d, res5_d), \
f"Row has unexpected value. Expected {expected_d}, but got {res5_d}"

print("Success")
```

```
   mysql+pymysql://general_user:***@localhost/s24_hw3
   mysql+pymysql://root:***@localhost
 * mysql+pymysql://root:***@localhost/s24_hw3
1 rows affected.
   mysql+pymysql://general_user:***@localhost/s24_hw3
   mysql+pymysql://root:***@localhost
 * mysql+pymysql://root:***@localhost/s24_hw3
1 rows affected.
+-----------+-----------+----------+-----------+--------------+-----------+--------------+------------+-----
--+-------------+
|  personID | nameFirst | nameLast | birthYear | birthCountry | deathYear | deathCountry | nameSubstr | idNu
m | rowCounter |
+-----------+-----------+----------+-----------+--------------+-----------+--------------+------------+-----
--+-------------+
| hermage01 |   George  |  Herman  |    1920   |     USA      |    2005   |     USA      |  hermage   |   1
|    3218    |
+-----------+-----------+----------+-----------+--------------+-----------+--------------+------------+-----
--+-------------+
Success
```

Security Testing

- Write and execute statements below to show that you set up the permissions for `general_user` correctly
  - You should show that `select` and `execute` work, but `insert`, `update`, and `delete` don't

In [163…

```
# Connect to database as general_user

%sql mysql+pymysql://general_user:dbuserbdbuser@localhost/s24_hw3
```

In [164…

```
%sql select * from people limit 5;
```

```
 * mysql+pymysql://general_user:***@localhost/s24_hw3
   mysql+pymysql://root:***@localhost
   mysql+pymysql://root:***@localhost/s24_hw3
5 rows affected.
```

Out[164]:

personID	nameFirst	nameLast	birthYear	birthCountry	deathYear	deathCountry	nameSubstr	idNum	rowCounter
abernte01	Ted	Abernathy	1921	USA	2001	USA	abernte	1	1816
abernte02	Ted	Abernathy	1933	USA	2004	USA	abernte	2	2489
abreujo01	Jose	Abreu	1987	Cuba	None	None	abreujo	1	1721
abreujo02	Joe	Abreu	1913	USA	1993	USA	abreujo	2	3169
adamsau01	Austin	Adams	1986	USA	None	None	adamsau	1	991

In [165…

```
%sql call updatePerson(@ruthID, null, "Ruth", null, null, null, null, @ruthID)
%sql select * from people where personID = @ruthID
```

```
 * mysql+pymysql://general_user:***@localhost/s24_hw3
   mysql+pymysql://root:***@localhost
   mysql+pymysql://root:***@localhost/s24_hw3
1 rows affected.
 * mysql+pymysql://general_user:***@localhost/s24_hw3
   mysql+pymysql://root:***@localhost
   mysql+pymysql://root:***@localhost/s24_hw3
1 rows affected.
```

Out[165]:

personID	nameFirst	nameLast	birthYear	birthCountry	deathYear	deathCountry	nameSubstr	idNum	rowCounter
hermage01	George	Herman	1920	USA	2005	USA	hermage	1	3218

In [166…

```
%sql insert into people (nameFirst, nameLast) values ("Dongzhou", "Li")
```

```
 * mysql+pymysql://general_user:***@localhost/s24_hw3
   mysql+pymysql://root:***@localhost
   mysql+pymysql://root:***@localhost/s24_hw3
(pymysql.err.OperationalError) (1142, "INSERT command denied to user 'general_user'@'localhost' for table 'p
eople'")
[SQL: insert into people (nameFirst, nameLast) values ("Dongzhou", "Li" )]
(Background on this error at: https://sqlalche.me/e/20/e3q8)
```

In [167…

```
%sql update people set nameFirst = "Dong" where personID = @ruthID
```

```
 * mysql+pymysql://general_user:***@localhost/s24_hw3
   mysql+pymysql://root:***@localhost
   mysql+pymysql://root:***@localhost/s24_hw3
(pymysql.err.OperationalError) (1142, "UPDATE command denied to user 'general_user'@'localhost' for table 'p
eople'")
[SQL: update people set nameFirst = "Dong" where personID = @ruthID]
(Background on this error at: https://sqlalche.me/e/20/e3q8)
```

In [168…

```
%sql delete from people where personID = @ruthID
```

```
 * mysql+pymysql://general_user:***@localhost/s24_hw3
   mysql+pymysql://root:***@localhost
   mysql+pymysql://root:***@localhost/s24_hw3
(pymysql.err.OperationalError) (1142, "DELETE command denied to user 'general_user'@'localhost' for table 'p
eople'")
[SQL: delete from people where personID = @ruthID]
(Background on this error at: https://sqlalche.me/e/20/e3q8)
```

GoT Data Visualization

Data Loading

- Run the cell below to create and insert data into GoT-related tables

In [169…

```
%sql mysql+pymysql://root:dbuserbdbuser@localhost/s24_hw3

for filename in [
    "episodes_basics", "episodes_characters", "episodes_scenes"
]:
    df = pandas.read_json(f"data/{filename}.json")
    df.to_sql(name=filename, schema="s24_hw3", con=engine, index=False, if_exists="replace")

print("Success")
```

```
Success
```

Overview

- In this section, you'll be combining SQL and Dataframes to create data visualizations
  - You may find this notebook helpful
  - You may also find the Pandas docs helpful
- **For all questions, you need to show the SQL output and the visualization generated from it.** See DV0 for an example.

DV0

- This question is an example of what is required from you
- Create a bar graph showing the amount of time each season ran for (in seconds)
- You should use the `episodes_scenes` table

- Note: `season_running_time <<` in the following cell saves the output of the SQL query into a local Python variable `season_running_time`

In [170…

```
%%sql

season_running_time <<

with one as (
    select seasonNum, episodeNum, sceneNum, sceneEnd, time_to_sec(sceneEnd) as sceneEndSeconds,
           sceneStart,  time_to_sec(sceneStart) as sceneStartSeconds,
            time_to_sec(sceneEnd)-time_to_sec(sceneStart) as sceneLengthSeconds
    from episodes_scenes
),
two as (
    select seasonNum, episodeNum, max(sceneEnd) as episodeEnd, max(sceneEndSeconds) as episodeEndSeconds
    from one
    group by seasonNum, episodeNum
),
    three as (
        select seasonNum, cast(sum(episodeEndSeconds) as unsigned) as totalSeasonSeconds,
               sec_to_time(sum(episodeEndSeconds)) as totalRunningTime
        from two
        group by seasonNum
    )
select * from three;
```

```
   mysql+pymysql://general_user:***@localhost/s24_hw3
   mysql+pymysql://root:***@localhost
 * mysql+pymysql://root:***@localhost/s24_hw3
8 rows affected.
Returning data to local variable season_running_time
```

In [171…

```
# You must show the SQL output

season_running_time = season_running_time.DataFrame()
season_running_time
```

Out[171]:

	seasonNum	totalSeasonSeconds	totalRunningTime
0	1	33143	0 days 09:12:23
1	2	31863	0 days 08:51:03
2	3	32541	0 days 09:02:21
3	4	32100	0 days 08:55:00
4	5	34003	0 days 09:26:43
5	6	34775	0 days 09:39:35
6	7	26675	0 days 07:24:35
7	8	25922	0 days 07:12:02

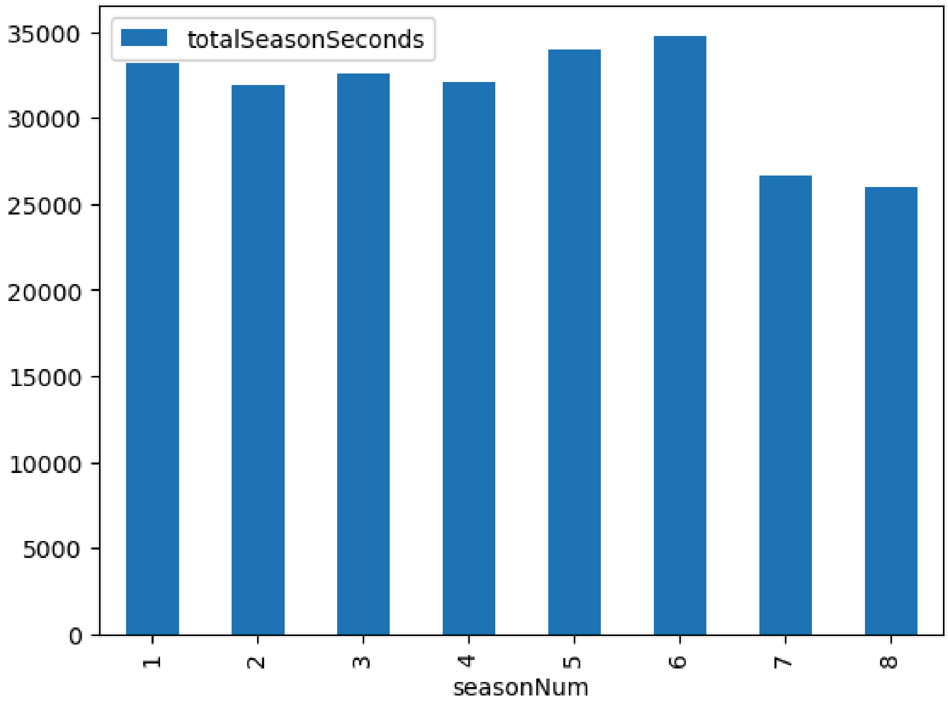
In [172…

```
# You must show the visualization

season_running_time[['seasonNum', 'totalSeasonSeconds']].plot.bar(x='seasonNum', y='totalSeasonSeconds')
```

Out[172]:

```
<AxesSubplot:xlabel='seasonNum'>
```



DV1

- Create a pie chart showing the proportion of episodes aired in each month (regardless of year)

- You should use the `episodes_basics` table
- As an example, your pie chart may look like this:

In [173…

```
%%sql

episodes_per_month <<

select month(episodeAirDate) as `month`, count(*) as episodeCount
from episodes_basics
group by `month`
order by `month`;
```

```
   mysql+pymysql://general_user:***@localhost/s24_hw3
   mysql+pymysql://root:***@localhost
 * mysql+pymysql://root:***@localhost/s24_hw3
6 rows affected.
Returning data to local variable episodes_per_month
```

In [174…

```
# SQL output

episodes_per_month = episodes_per_month.DataFrame()
episodes_per_month
```

Out[174]:

	month	episodeCount
0	3	2
1	4	21
2	5	28
3	6	15
4	7	3
5	8	4

In [175…

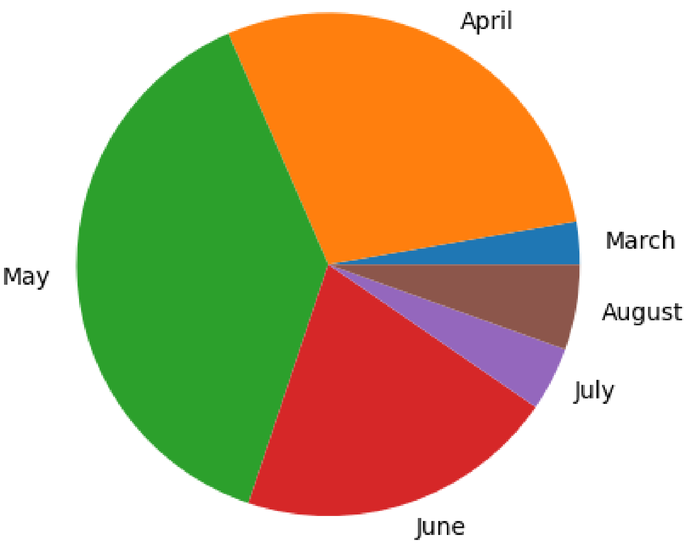
```
# TODO: visualization

import calendar

episodes_per_month.plot.pie(
    y="episodeCount",
    labels=[calendar.month_name[e] for e in list(episodes_per_month["month"])],
    ylabel="",
    legend=False,
)
```

Out[175]:

```
<AxesSubplot:>
```



DV2

- Create a bar chart showing the number of episodes that every location (not sublocation) appeared in
  - You are counting the number of episodes, not scenes. If a location appeared in multiple scenes in a single episode, that should increment your count only by one.
  - You should order your chart on the number of episodes descending, and you should only show the top 10 locations
- You should use the `episodes_scenes` table
- As an example, your bar chart may look like this:

In [176…

```
%%sql

location_episode_count <<

select sceneLocation as location, count(distinct concat(seasonNum, "_", episodeNum)) as episodeCount
from episodes_scenes
group by location
order by episodeCount desc
limit 10;
```

```
   mysql+pymysql://general_user:***@localhost/s24_hw3
   mysql+pymysql://root:***@localhost
 * mysql+pymysql://root:***@localhost/s24_hw3
10 rows affected.
Returning data to local variable location_episode_count
```

In [177…

```
# SQL output

location_episode_count = location_episode_count.DataFrame()
location_episode_count
```

Out[177]:

	location	episodeCount
0	The Crownlands	64
1	The North	54
2	The Wall	36
3	North of the Wall	35
4	The Riverlands	33
5	Meereen	24
6	Braavos	14
7	The Vale	11
8	Dorne	10
9	The Westerlands	9

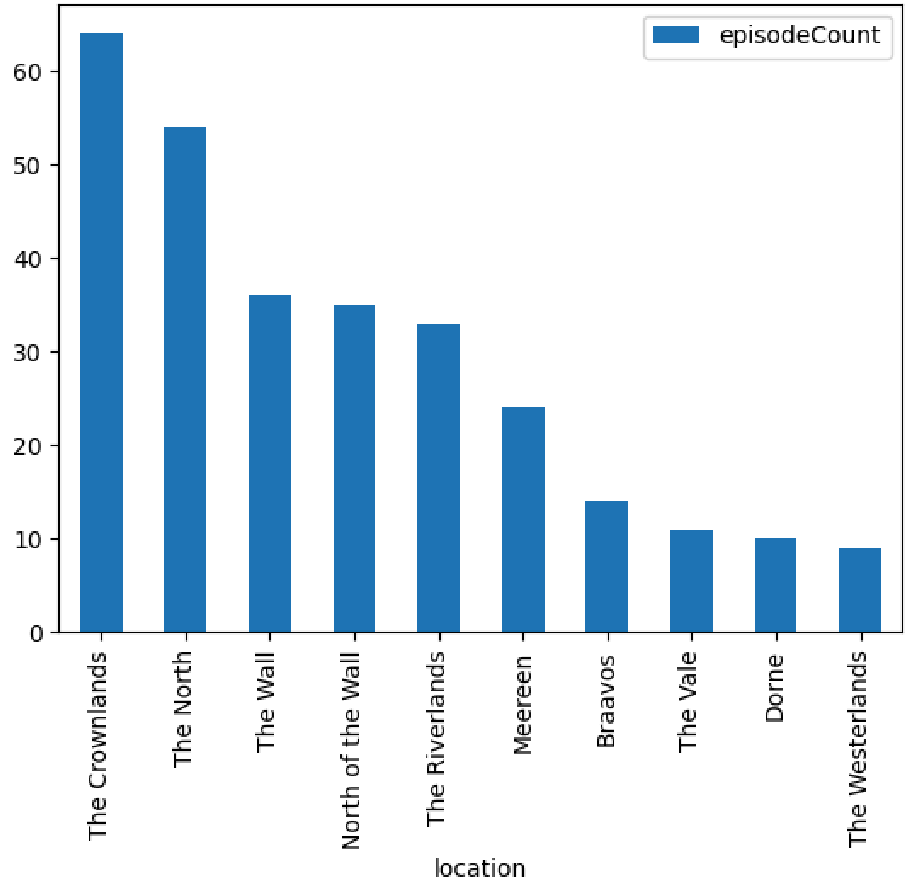
In [178…

```
# TODO: visualization

location_episode_count.plot.bar(x="location", y="episodeCount")
```

Out[178]:

```
<AxesSubplot:xlabel='location'>
```



DV3

- Create a scatter plot showing the relationship between the number of episodes (not scenes) a character appears in and their screen time (in seconds)
    - A character's screen time is the sum of the time lengths of all the scenes that the character appears in
- You should use the `episodes_characters` and `episodes_scenes` tables
- As an example, your scatter plot may look like this:

In [179…

```
%%sql

episode_count_screen_time <<

with characterEpisodeCount as (
    select characterName, count(distinct concat(seasonNum, "_", episodeNum)) as episodeCount
    from episodes_characters
    group by characterName
    order by episodeCount desc
),
characterScreenTime as (
    select characterName, sum(time_to_sec(sceneEnd) - time_to_sec(sceneStart)) as screenTime
    from episodes_characters join episodes_scenes using (seasonNum, episodeNum, sceneNum)
    group by characterName
)
select * from characterEpisodeCount a join characterScreenTime b using (characterName);
```

```
   mysql+pymysql://general_user:***@localhost/s24_hw3
   mysql+pymysql://root:***@localhost
 * mysql+pymysql://root:***@localhost/s24_hw3
577 rows affected.
Returning data to local variable episode_count_screen_time
```

In [180…

```
# SQL output
# Output is big, so just show first 10 rows

episode_count_screen_time = episode_count_screen_time.DataFrame()
episode_count_screen_time.head(10)
```

Out[180]:

	characterName	episodeCount	screenTime
0	Gared	1	362
1	Waymar Royce	1	306
2	Will	1	763
3	Wight Wildling Girl	2	51
4	White Walker	15	1557
5	Jon Snow	62	40365
6	Bran Stark	40	14346
7	Robb Stark	23	7721
8	Eddard Stark	12	8604
9	Catelyn Stark	26	9297

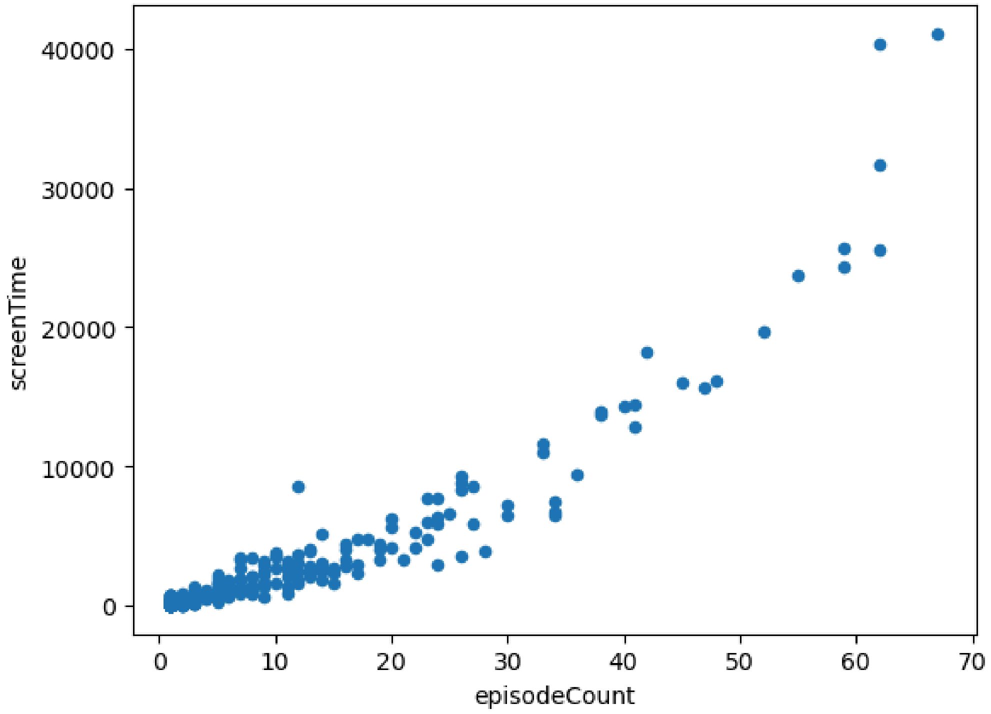
In [181…

```
# TODO: visualization

episode_count_screen_time.plot.scatter(x="episodeCount", y="screenTime")
```

Out[181]:

```
<AxesSubplot:xlabel='episodeCount', ylabel='screenTime'>
```



DV4

- Create a bar chart showing the number of exclusive characters in each season
  - An exclusive character is a character that appeared in only that season, no other season
  - You should order your chart on the number of exclusive characters descending
- You should use the `episodes_characters` table
  - You can assume `characterName` is unique across all characters. That is, a single name is one unique character.
- As an example, your bar chart may look like this:

In [182…

```
%%sql

season_exclusive_characters <<

select a.seasonNum, count(distinct a.characterName) as exclusiveCharacterCount
from episodes_characters a
left join episodes_characters b
```

```
on (a.characterName = b.characterName and a.seasonNum <> b.seasonNum)
where b.characterName is null
group by a.seasonNum
order by exclusiveCharacterCount desc;
```

```
   mysql+pymysql://general_user:***@localhost/s24_hw3
   mysql+pymysql://root:***@localhost
 * mysql+pymysql://root:***@localhost/s24_hw3
8 rows affected.
Returning data to local variable season_exclusive_characters
```

In [183…

```
# SQL output

season_exclusive_characters = season_exclusive_characters.DataFrame()
season_exclusive_characters
```

Out[183]:

	seasonNum	exclusiveCharacterCount
0	6	99
1	5	83
2	2	49
3	1	46
4	4	45
5	3	42
6	8	17
7	7	11

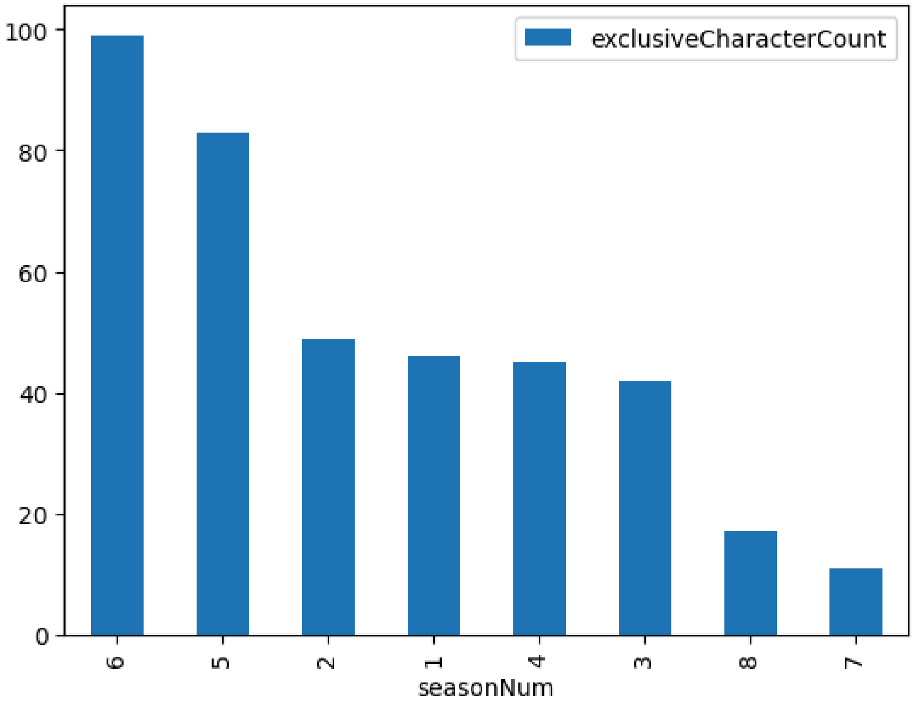
In [184…

```
# TODO: visualization

season_exclusive_characters.plot.bar(x="seasonNum", y="exclusiveCharacterCount")
```

Out[184]:

```
<AxesSubplot:xlabel='seasonNum'>
```



In [ ]: