Purpose

In today's society, the ability to store and manage data of multiple types is a necessity for every company. The primary way these companies store their valuable data is by utilizing relational database management systems, also known as RDMS. Structured Query Language, or SQL, one of the most popular languages for relational database management systems, allows us to easily store and manipulate tabular data.

Skills

This assignment will give you practice querying data from a MySQL database and retrieving the information through Python in order to manipulate it. Students will also be exposed to the structure and usefulness of the pymysql library, as well as gain insight on how to bridge the gap between Python and MySQL.

Knowledge

Along with the skills you will gain as you practice querying database tables for this assignment, you will also gain insight into the design of databases and the common ways that data can be organized for easy access. This knowledge of database design and structure will be useful when you encounter larger, more detailed databases in your future coursework and employment as well as when you are responsible for designing a database of your own.

Important

1. Due Date: **04/24/2022 at 11:59PM**
2. This homework is graded out of **100** points.
3. This is an individual assignment. You may collaborate with other students in this class. Collaboration means talking through problems, assisting with debugging, explaining a concept, etc. Students may only collaborate with fellow students currently taking CS 2316, the TA's and the instructor. You should not exchange code or write code for others.
4. For Help:
    - TA Helpdesk (Schedule posted on class website)
    - Email TA's or use Ed Discussion
    - How to Think Like a Computer Scientist
    - [https://pymysql.readthedocs.io/en/latest/]
    - CS2316 Handouts: MySQL, PyMySQL
5. Comment out or delete all your function calls. Only global variables, and comments are okay to be outside the scope of a function. When your code is run, all it should do is run without any errors.
6. Do not wait until the last minute to do this assignment in case you run into problems.
7. Read the entire specifications document before starting this assignment.
8. If your code cannot run because of an error, it is a 0%.

Introduction

You are an employee at the LEGO Company. Suddenly one day, all of your fellow coworkers met the fatal doom of stepping on legos, and had to take a leave from the company in order to recover from their paining feet. Now, you are tasked with understanding the LEGO database given to you so that you can help all of your customers who have lost their LEGO pieces! Here is some context about LEGOs that will prove useful in understanding the data. LEGO is a popular brand of toy building bricks. They are often sold in sets in order to build a specific object. Each set contains a number of parts in different shapes, sizes and colors. This database contains information on which parts are included in different LEGO sets.

Schema

The diagram below provides a visual for the entities and relationships created in the provided schema file.

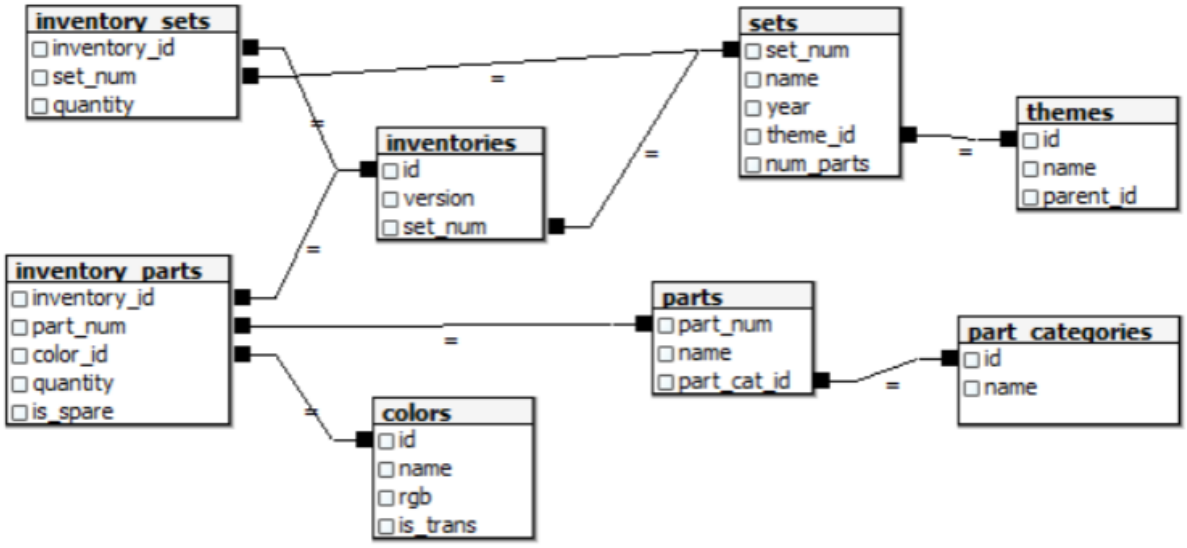

Table Information

The following are given details of information contained in each table within the database:

colors

- This table contains information on LEGO piece colors, including their unique IDs, RGB values, and whether or not they are transparent.

themes

- This table contains information on LEGO themes, including their unique ID numbers, names, and parent ids if they are a part of a larger theme.

part_categories

- This table contains information on the part category (what type of part it is), the category name and its unique ID.

sets

- This table contains information on LEGO sets, including the unique set ID number, the name of the set, the year it was released, the ID of the theme, and how many parts it includes.

inventories

- This table contains information on inventories, including each inventory's unique ID number, its version number, and the set ID of the set it holds.

inventory_sets

- This table contains information on what inventory holds which sets and the quantity of each set in the inventory.

parts

- This table contains information on lego parts, including a part's unique ID number, the name of the part, and the category that the part is in.

inventory_parts

- This table contains information on part inventories, including a unique ID number, the part number, the color of the part, how many are included, and whether it's a spare or not.

Database Creation and Population

Follow the corresponding instructions to create and populate your database.

1. Navigate to the HW04 folder on Canvas
2. Download the zip folder titled "lego_database.zip"
3. Extract this folder to your 2316 Homework folder
4. **Important: DO NOT open/edit/delete any files within this zip folder. Doing so may affect your ability to load the database.**
5. Open MySQL Workbench.
6. Open your "Local Instance" connection
7. Click File > Run SQL Script
8. Navigate to your 2316 Homework folder, then into the "lego_database" file (the extracted file, **not** the .zip)
9. Click the "lego-database-schema.sql" file, and select Open
10. Do not edit any of the fields, simply press "Run"

11. You should see a window titled "Run SQL Script"
    a. This window may freeze on "Finished Executing Script", allow a few minutes to populate the tables before cancelling.
12. After a few minutes, you will see "Operation completed successfully". You may close the window.
13. Next, open your command prompt and navigate to where you saved your HW04 files.
14. When you type "ls" (or "dir" depending on your OS), you should see "lego-database-schema.sql", "insert_lego_database.py", and a folder title "data".
15. You must now run the "insert_lego_database.py" file in the **exact** way:
    `python insert_lego_database.py root <pw>`
    Where <pw> is your **MySQL password**
16. Please wait while the file runs. You should see a success message after a few moments. You may now navigate back into your MySQL workbench, select the lego database, and begin HW04!

Query Formatting

As mentioned above, you are asked to write a few Python functions, each of which corresponds to a query. Some of the functions will take in parameters that will be used to construct the queries. For each function, you should assign your final query to the variable `query` as one single string. You should use the %s placeholder to incorporate the function parameters for the queries. **To see an example of utilizing the "%s" placeholder in your query, please see Sample Function query0.**

When running queries via PyMySQL, you should never use f-strings or string concatenation when integrating arbitrary values into your query string. Using the placeholder `%s` helps avoid SQL injection attacks on your databases.

Notice that we have provided you with these 3 lines of code in each function:

```
[1]  cursor.execute(query)
[2]  result = cursor.fetchall()
[3]  return result
```

Line [1]: a pymysql cursor executes the query passed in.
Line [2]: Because we are using the **pymysql.cursors.Cursor** cursorclass, the `.fetchall()` method of the cursor object retrieves the query result from MySQL server and returns the result as ***a tuple of tuples***.
Line [3]: return that tuple of tuples.

Do **NOT** change these 3 lines.

Do **NOT** hard code the resulting tuple of tuples from the query. We will manually check for hardcoding after the assignment is due and will deduct points accordingly (and harshly as this would be considered cheating).

Testing and Submitting

There are 3 ways you can test your code for HW04:

1. **[Recommended]** Write your queries in MySQL Workbench, and when your output matches the test cases, copy the query into your HW04.py file, then submit to gradescope when ready.
2. If you are familiar with interfacing into MySQL through the command line, you may test your queries in the terminal, then copy them into HW04.py when the output matches the test cases.
3. Write your queries in HW04.py, printing your outputs and running the file. This is a more time consuming method of testing queries.

Regardless of how you test your queries, you should copy them into HW04.py, which is the only file you will submit to Gradescope. **We will not grade any outputs from MySQL Workbench.**

Sample Function

Function name: `query0`
Parameter(s): `cursor` (pymysql.cursors.Cursor), `part_num` (int)
Return Type: tuple
Description:
`Select the name of a part given a specific part_num.`

Solution Query:

```
Query = 'SELECT name FROM parts WHERE part_num LIKE %s'
```

MySQL Query Output:

name
Baseplate 24 x 32

Python Test Case:

```
>>> cursor = create_cursor('localhost', 'root', user_password,
'lego')
>>> query0(cursor, 10)
('Baseplate 24 x 32')
```

Functions

Function name: `query1`
Parameter(s): `cursor` (pymysql.cursors.Cursor)
Return Type: tuple
Description:
Using the `inventory_parts` table, return the inventory id and part number for all parts with a color id of 20.

MySQL Query Output:

inventory_id	part_num
250	33230
250	4202
250	6162
1971	3899
1971	6176
2417	6161

Python Test Case:

```
>>> cursor = create_cursor('localhost', 'root', user_password,
'lego')
>>> query1(cursor)
((250, '33230'),
 (250, '4202'),
 (250, '6162'),
 (1971, '3899'),
 (1971, '6176'),
 (2417, '6161'))
```

Function name: `query2`
Parameter(s): `cursor` (pymysql.cursors.Cursor)
Return Type: tuple
Description:
Using the `sets` table, return the name and number of parts for all sets with a number of parts between 5000 and 6000.

MySQL Query Output:

name	num_parts
Millennium Falcon - UCS	5195
Taj Mahal	5922
Window Exploration Bag	5200
Star Wars / M&M Mosaic - Promo Set	5461

Python Test Case:

```
>>> cursor = create_cursor('localhost', 'root', user_password,
'lego')
>>> query2(cursor)
(('Millennium Falcon - UCS', 5195),
 ('Taj Mahal', 5922),
 ('Window Exploration Bag', 5200),
 ('Star Wars / M&M Mosaic - Promo Set', 5461))
```

Function name: `query3`
Parameter(s): `cursor` (pymysql.cursors.Cursor),
Return Type: tuple
Description:
`Using the `themes` table, return the top 5 names that have the parent id 5 ordered by descending id.`
MySQL Query Output:

name
Traffic
Robot
Riding…
Race
Off-Road

Python Test Case:

```
>>> cursor = create_cursor('localhost', 'root', user_password,
'lego')
>>> query3(cursor)
(('Traffic',), ('Robot',), ('Riding Cycle',), ('Race',),
('Off-Road',))
```

Function name: `query4`
Parameter(s): `cursor` (`pymysql.cursors.Cursor`), `color`
Return Type: tuple
Description:
Using the `inventory_parts` table, return row_id and part_num that have a color id given by **color** and quantity less than 4, sorted by inventory_id in descending order. Return only the first 10 rows.
MySQL Query Output:

row_id	part_num
99998	4349
99857	6141
99852	61409
99849	47457
99848	4740
99829	3031
99823	3021
99817	11477
99767	970c00
99764	6636

Python Test Case:

```
>>> cursor = create_cursor('localhost', 'root', user_password,
'lego')
>>> query4(cursor, 72)
((99998, '4349'), (99857, '6141'), (99852, '61409'),
 (99849, '47457'), (99848, '4740'), (99829, '3031'),
 (99823, '3021'), (99817, '11477'), (99767, '970c00'),
 (99764, '6636'))
```

Function name: `query5`
Parameter(s): `cursor` (pymysql.cursors.Cursor), `year`
Return Type: tuple
Description:
Using the 'sets' table, return the year and the number of sets that were created during that year. Only include years that occurred during or after the given **year**, and order by ascending years.
MySQL Query Output:

year	COUNT(set_num)
2000	327
2001	339
2002	447
2003	415
2004	371
2005	330
2006	283
2007	321
2008	349
2009	402
2010	444
2011	503

Python Test Case:

```
>>> cursor = create_cursor('localhost', 'root', user_password,
'lego')
>>> query5(cursor, 2000)
((2000, 327),
 (2001, 339),
 (2002, 447),
 (2003, 415),
 ..
 (2017, 296))
```

Function name: `query6`
Parameter(s): `cursor` (pymysql.cursors.Cursor)
Return Type: tuple
Description:
Using the 'inventory_parts' table, return the inventory_id and the sum of quantities that have the greatest total quantity of parts.

MySQL Query Output:

inventory_id	SUM(quantit...
76	5962

Python Test Case:

```
>>> cursor = create_cursor('localhost', 'root', user_password,
'lego')
>>> query6(cursor)
(76, 5962)
```

Function name: `query7`
Parameter(s): `cursor` (pymysql.cursors.Cursor)
Return Type: tuple
Description:
Using the 'themes' table, return each parent_id and the number of occurrences of each parent_id. Only return values where the parent_id begins with 4. Return the values in ascending order based on their number of occurrences.
Note: The ordering of parent_id's in your output might differ.
MySQL Query Output:

parent_id	count(parent_id)
478	1
435	1
484	1
454	1
408	2
475	2
494	2
497	2
400	3
458	4
425	6
465	8
482	10
411	13
443	13

Python Test Case:

```
>>> cursor = create_cursor('localhost', 'root', user_password,
lego)
>>> query7(cursor)
((478, 1),
 (435, 1),
 (484, 1),
 (454, 1),
 (408, 2),
 (475, 2),
 (494, 2),
 (497, 2),
 ...
 (411, 13),
 (443, 13))
```

Function name: `query8`
Parameter(s): `cursor` (pymysql.cursors.Cursor), `minimum` (int)
Return Type: tuple
Description:
In the 'inventory_sets' table, return each inventory_id and the number of occurrences of each inventory_id. Only return values where there inventory_id has more than 5 occurrences. Return the values in descending order based on their number of occurrences.
Note: The ordering of inventory_id's in your output might differ.
MySQL Query Output:

inventory_id	count(inventory_id)
2514	30
10024	24
6702	24
9926	24
263	24
11317	24
8348	24
8820	24
305	24
6174	24
7850	24
10770	24
1899	24
3162	24

...

5248	6
15955	6
6331	6
8471	6
1334	6
13380	6
5137	6
5645	6
2849	6
6544	6
5183	6
15060	6
10158	6
15594	6

Python Test Case:

```
>>> cursor = create_cursor('localhost', 'root', user_password,
'lego')
>>> query8(cursor)
((2514, 30),
 (10024, 24),
```

```
 (6702, 24),
 (9926, 24),
  ...
 (15594, 6))
```

Function name: `query9`
Parameter(s): `cursor` (`pymysql.cursors.Cursor`)
Return Type: tuple
Description:

For this question, focus on the "sets" and "themes" relations. Select themes (names and id) as well as the mean number of parts of all the sets for each theme. Make sure to round your averages to whole numbers. Order by the descending average number of parts. Then return the first 10 tuples (rows). Also, do not forget to exclude "Disney" because it is overrated.

MySQL Query Output:

id	mean_num_parts	name
155	2351	Modular Buildings
174	2200	Star Wars Episode 4/5/6
171	2130	Ultimate Collector Series
277	2100	Mosaic
104	2017	Town Plan
276	1717	Sculptures
485	1456	Ultimate Collector Series
514	1437	Creator
398	1387	FIRST LEGO League
156	1358	Mini

Python Test Case:

```
>>> cursor = create_cursor('localhost', 'root', user_password,
'lego')
>>> query9(cursor)
((155, Decimal('2351'), 'Modular Buildings'),
 (174, Decimal('2200'), 'Star Wars Episode 4/5/6'),
 (171, Decimal('2130'), 'Ultimate Collector Series'),
 (277, Decimal('2100'), 'Mosaic'),
 (104, Decimal('2017'), 'Town Plan'),
 (276, Decimal('1717'), 'Sculptures'),
 (485, Decimal('1456'), 'Ultimate Collector Series'),
 (514, Decimal('1437'), 'Creator'),
```

```
 (398, Decimal('1387'), 'FIRST LEGO League'),
 (156, Decimal('1358'), 'Mini'))
```

Function name: **query10**
Parameter(s): `cursor` (pymysql.cursors.Cursor),
Return Type: tuple
Description:
For this question, we focus on the "colors", and "inventory_parts" relations. We would like to find out the total number of pieces (with color RGB starting with "A") in inventory for each specific part.

Further, to be picky, we will only keep those parts whose total is an odd number. Return only the top 5 results when sorted in descending order based on the quantity sum.

MySQL Query Output:

part_num	SUM(inventory_parts.quant...
6141	1269
3004	827
3024	757
32123b	665
4274	639

Python Test Case:

```
>>> cursor = create_cursor('localhost', 'root', user_password,
'lego')
>>> query10(cursor)
(('6141', Decimal('1269')),
 ('3004', Decimal('827')),
 ('3024', Decimal('757')),
 ('32123b', Decimal('665')),
 ('4274', Decimal('639')))
```

Gradescope/Canvas Requirements

- **NO PRINT STATEMENTS.** This will break the autograder
- **NO FUNCTION CALLS** outside of function definitions. This will also break the autograder
- Make sure the file submitted is titled **HW04.py**
- Allowed imports: **pymysql**, **pprint**

Grading Rubric

query1	10	pts
query2	10	pts
query3	10	pts
query4	10	pts
query5	10	pts
query6	10	pts
query7	10	pts
query8	10	pts
query9	10	pts
query10	10	pts
Total	100/100	pts