Zac Foteff: CPSC 321, Fall - HW#1

Part 1:

36/40

Q1:

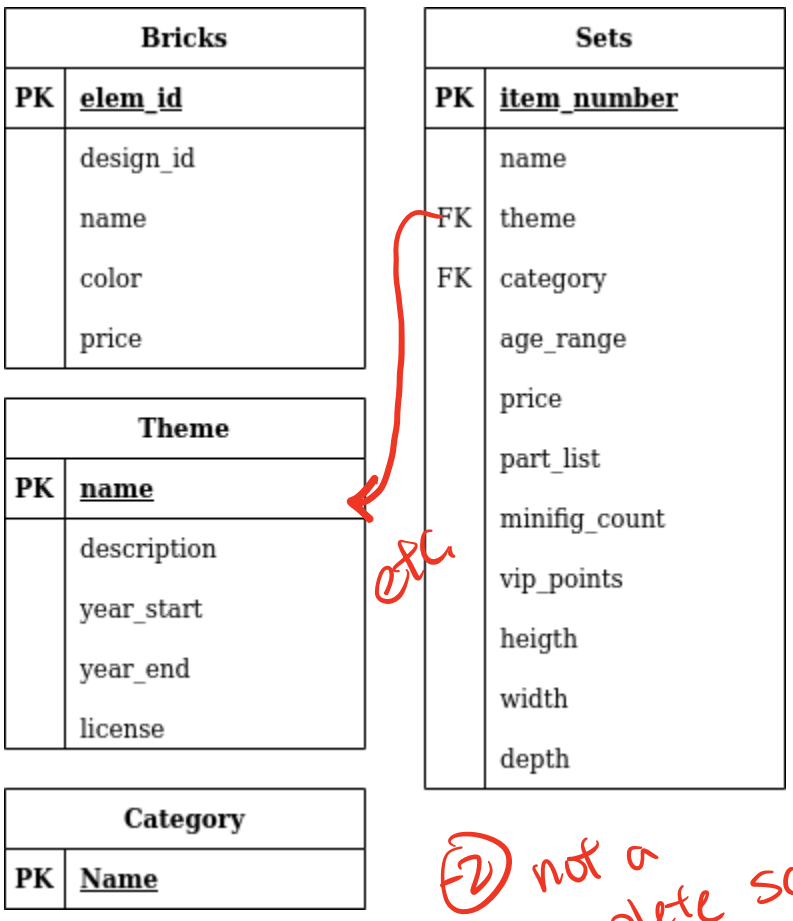
Primary Key
Foreign Key

Bricks(**elem_id**, design_id, name, color, price)
Sets(**item_number**, name, *theme* *category*, age_range, price, part_list, minifig_count, vip_points, height, width, depth) Theme(**name**, description, year_start, year_end, license)
Category(**name**)

can be many

can be many

Q2:



-2 set production years

-2 identical bricks

-2 not a complete schema diagram w/out connections

Q3:

Bricks

elem_id	design_id	name	color	price
101	2048	2x2	Red	0.75
102	3842	2x1	Blue	0.43
103	6785	8x1 flat	Grey	0.99
104	6785	5x5 flat	Blue	1.23
105	1046	2x4	Red	0.81
106	2453	2x8	Red	1.02
107	3331	4x16 flat	White	1.21
108	9830	1x2 corner	Black	0.32

Categories

name
Space
Pirates
City

Sets

name	theme	category	age_range	price	minifig_count	vip_points	height	width	depth
Jabba's Palace	Star Wars	Space	8-18	120	8	350	6	12	8
Space Shuttle	Real-World	Space	8-18	85	4	400	8	3	3
Death Star	Star Wars	Space	14-80	301	23	1500	12	12	12
Captain Jack's Ship	Pirates of the Caribbean	Pirates	8-18	75	6	350	8	16	6
Gas Station	Lego City	City	8-18	60	4	300	14	12	12
Lego City Minfig Pack	Lego City	City	8-18	15	4	75	NULL	NULL	NULL

Theme

name	description	year_start	year_end	license
Star Wars	From Disney's Star Wars	2001	2030	Disney
Pirates of the Caribbean	From Disney's Pirates	2005	2023	Disney
Lego City	Lego's Lego City	1990	2025	NULL
Real-World	Collection of real-world objects	1990	2025	NULL

Q4:

I would rate my design very highly. I believe designing the schema around 4 different table reduces redundancy and make it easy to understand. Since a majority of the key components of the Lego database involve unique identifiers, it seemed most prudent to design each relation with a singular primary key rather that composite keys. Additionally, the interconnectivity of these relations calls for certain tables, such as the Sets, to have multiple foreign keys for retrieving data that could be considered redundant. I think some improvement could be made in order to integrate the Category table into the Theme or Set tables, however the possible amount of categories made it seem prudent to create a table specifically to contain that data.

Part 2:

Q1:

I would like to modify an existing project I have been working on for some time, but have never been able to create an effective database application for. This project is an expense calculator written in Python that allows a user to track their expenses over a period of time. The current database component only stores single transactions, track a user's current balance, and output a history of every transactions that has happened in the application. I would like to expand the database functionality of this application in multiple ways. I would

like the user to be able to filter results of their history using tags, or other indicators that would be an added attribute to every transactions. Additionally, providing the user some visualization and analysis of their financial data would be a beneficial feature to learn how to implement. Furthermore, giving users the feature to search for specific transactions out of their expansive transaction history by constructing more efficient queries.

good

Q2:

Generally, I envision users opening this application and being prompted by a login screen. New users will be further prompted to enter their current cash on hand as a start for their account balance, while existing users will have their data loaded into the application. The user can then enter in their new transactions and have the database back end automatically process and store their transactions in the proper locations in the database. Users can then view a history of their transactions from the past 3 days and reference an older purchase from weeks ago if they need to. The user can then finish out by viewing visualizations of their spending habits and balance over a period of time.

good start

For HW2 try to flesh out the details ...

GUI? Other?