CS 121 Final Project Reflection

Due: March 18th, 11:59PM PST

This reflection document will include written portions of the Final Project. Submit this as **reflection.pdf** with the rest of the required files for your submission on CodePost.

For ease, we have highlighted any parts which require answers in **blue**.

Student name(s): Gabriella Twombly, Ellen Min
Student email(s): gtwombly@caltech.edu, emin@caltech.edu

Part Lo. Introduction

Answer the following questions to introduce us to your database and application; you may pull anything relevant from your Project Proposal and/or README.

Database/Application Overview

What application did you design and implement? What was the motivation for your application? What was the dataset and rationale behind finding your dataset?

Database and Application Overview Answer (3-4 sentences) :

Have you ever wanted to learn where you get your LEGOs and how the LEGO store near you works? Our proposed database would be a LEGO store simulation. Client users will be able to make queries about the new sets available, how long it would take to build a specific set, the characteristics of different parts, and more while the admin user would act like the store manager able to manage inventory. Avid LEGO buyers often enjoy learning more information about the sets online now that LEGO has become more popular and more expensive. This database will support queries to do so. We also enjoy using LEGOs and this would be a cool way to explore the mechanics behind the business chain.

Data set (general or specific) Answer:

The dataset we plan on using is a Kaggle dataset on lego parts, themes, sets, and parts. We may need to find or compute additional data, such as prices or the time it takes to complete each set. We would also add data specific to this project, such as the inventory of each item. The data can be found here:

https://www.kaggle.com/datasets/rtatman/lego-database?select=parts.csv.

***Client user(s) Answer*:**

LEGO store clients are the intended user base. They would be able to make queries that assist them in making a decision for their purchase. This includes the price (or price per piece) of each set, how long it would take to complete each set, size of the completed set, and more.

They will also be able to filter to find sets that match their desired specifications, and request that new lego sets are added to the store inventory.

Admin user(s) Answer:

Admin users are the LEGO store employees or managers. They would be able to manage the inventory of the lego sets available in the store. This would involve decreasing inventory of each item when it's purchased or adding new sets when the store "receives" new inventory. Admin users can also approve client requests for new lego sets, reflecting interaction and communication between the client and admin.

Part A. ER Diagrams

As we've practiced these past few weeks, the ER model is essential for designing your database application, and we expect you to iterate upon your design as you work through the ER and implementation steps. In this answer, you should provide a full ER diagram of your system. Your grade will be based on correct representation of the ER model as well as readability, consistency, and organization.

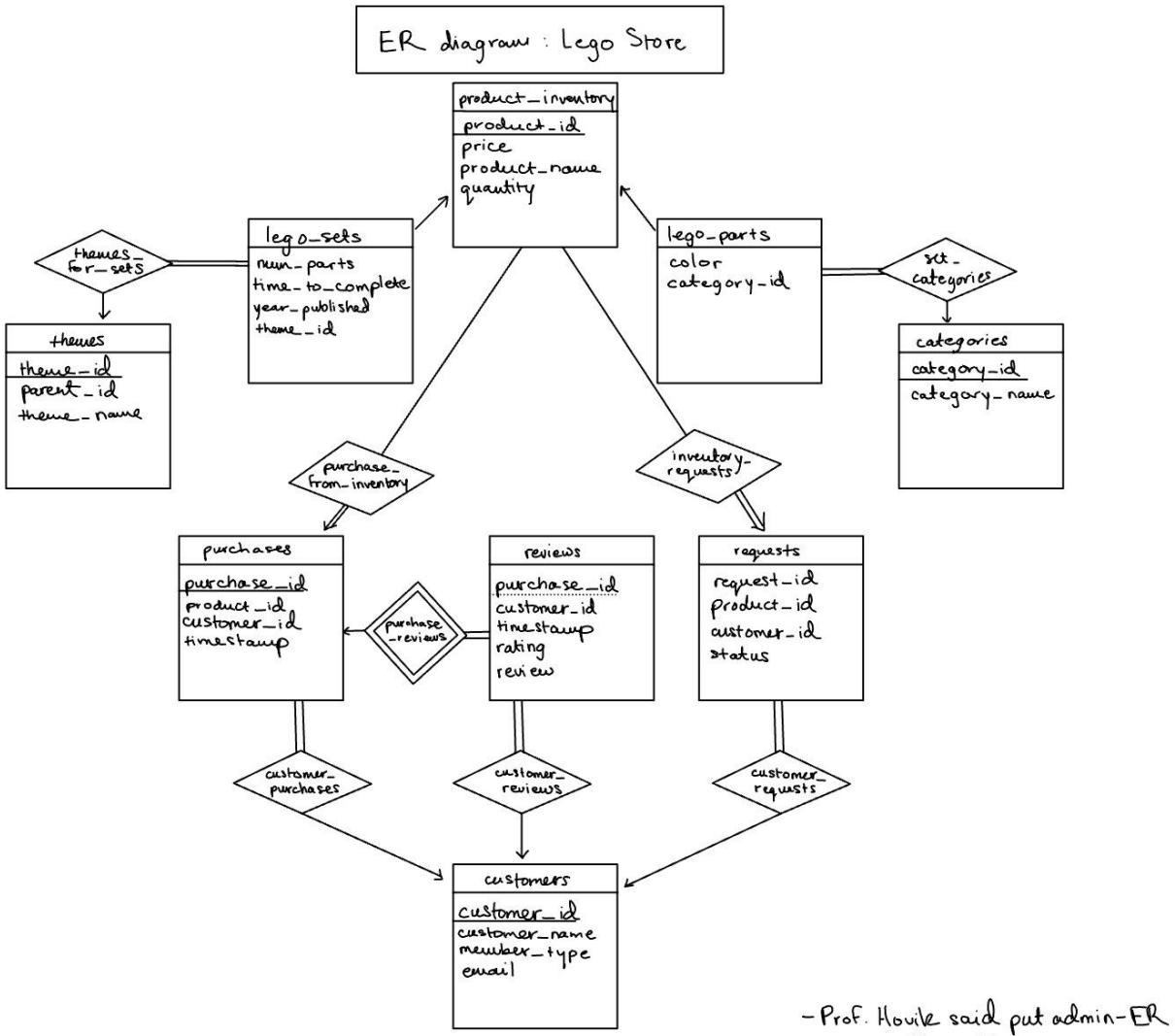
Notes: For this section **only**, we will allow (and encourage) students to share their diagrams on Discord (**#er-diagram-feedback**) to get feedback from other students on their ER diagrams given a brief summary of your dataset and domain requirements. This is offered as an opportunity to test your ER diagrams for accuracy and robustness, as another pair of eyes can sometimes catch constraints that are not satisfied or which are inconsistent with your specified domain requirements.

Requirements:

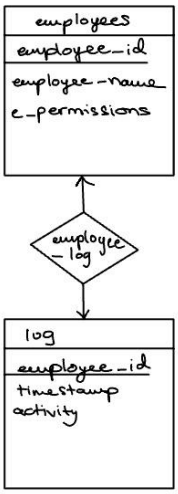
- Entity sets, relationship sets, and weak entity sets should be properly represented (also, do not use ER symbols not taught in class)
- Mapping cardinalities should be appropriate for your database schema, and in sync with your DDL
- Participation constraints should be appropriate and in sync with your DDL (total, partial, numeric)
- Use specialization where appropriate (e.g. *purchasers* and *travelers* inheriting from a *customers* specialization in A6)
- Do not use degrees greater than 3 in your relationships, do not use more than one arrow in ternary relationships.
- Use descriptive attributes appropriately
- Underline primary keys and dotted-underline discriminators
- Expectations from A6 still apply here
- Note: You do not need ER diagrams for views

ER Diagrams:

Main idea but made some design changes later on:



- Prof. Hovik said put admin-ER on the side
- will relate to requests



Final schema:
themes(theme_id, theme_name, parent_id)
categories(category_id, category_name)
product_inventory(product_id, product_price, product_name, quantity)
lego_sets(product_id, num_parts, time_to_complete, year_published, theme_id)
lego_parts(product_id, category_id)
discounts(member_type, discount_amount)
customers(customer_username, customer_name, email, member_type)
purchases(purchase_id, product_id, customer_username, purchase_item_total, purchase_time)
reviews(purchase_id, customer_username, review_time, rating, review)
requests(request_id, product_id, customer_username, request_status)
employees(employee_username, employee_first_name, employee_last_name, employee_permissions)
employee_log(request_id, employee_username, log_time, change_made)

Part B. DDL (Indexes)

As mentioned in Part B, you will need to add at least one index at the bottom of your **setup.sql** and show that it makes a performance benefit for some query(s).
Here, describe your process for choosing your index(es) and show that it is used by at least one query, which speeds up the performance of the same query on a version of the same table without that index. You may find lec14-analysis.sql and Lecture 14 slides on indexes useful for strategies to choose and test your indexes. **Remember that indexes are already created in MySQL for PKs and FKs, so you should not be recreating these.**

Index(es):

1. idx_theme_name on *themes(theme_name)*
2. idx_item_price on *product_inventory(product_price)*

Justification and Performance Testing Results:

We created *idx_theme_name* because retrieving the theme name of a theme, given the theme ID, is a common helper function and would be helpful given the number of theme IDs.

When we run a query to get the theme name and corresponding number of sets in each given theme, the performance is 0.0057 seconds without the index. With the index, it's 0.0051 seconds. This was a minor improvement, and we did struggle a little finding appropriate indices. This was because most of our queries used primary keys or foreign keys, so they were pretty fast already.

We then met with Professor Hovik, who suggested an index on price, which is an ordered numeric value. We did initially test an index on price but found it was slower (not being used). They suggested that we try various queries, including BETWEEN queries. While most of the BETWEEN queries ended up being similar, we found that the index sped up queries for higher/rare price points:

For the following query:

```
SELECT product_id, product_name, quantity FROM product_inventory
WHERE product_price=199;
```

We found that without the price index, it took 0.0025 seconds. With the index, it took 0.0004 seconds, which was a noticable improvement.

Part C. Functional Dependencies and Normal Forms

Requirements (from Final Project specification):

- Identify *at least 2 non-trivial functional dependencies* in your database
- Choose and justify your decision for the normal form(s) used in your database for at least 4 tables (if you have more, we will not require extra work, but will be more lenient with small errors). BCNF and 3NF will be the more common NF's expected, 4NF is also fine (but not 1NF).
    - Your justification will be strengthened with a discussion of your dataset breakdown, which we expect you to run into trade-offs of redundancy and performance.
- For two of your relations having at least 3 attributes (each) and at least one functional dependency, prove that they are in your chosen NF, using similar methods from A7.
    - If you have identified functional dependencies which are not preserved under a BCNF decomposition, this is fine
- Expectations from A7 still apply here.

Functional Dependencies:

1. *products(product_id → product_name, product_price, quantity)*
2. *customers(customer_username → customer_name, customer_email, member_type)*

Normal Forms Used (with Justifications):

1. **BCNF:** *themes(theme_id, theme_name, parent_id)*
   Functional dependencies: *theme_id → theme_name, parent_id*
   It made sense to have this table because we don't have very many attributes, just the primary key that identifies the theme, as well as two attributes that follow quite logically from it.
2. **BCNF:** *product_inventory(product_id, product_price, product_name, quantity)*
   Functional dependencies: *product_id → product_price, product_name, quantity*
   Because all of the three attributes besides *product_id* are dependent on product_id but not on each other, it again felt logical to put these together.
   While we could have, for example, split this into *product_names(product_id, product_name)* and *product_sale_inventory(product_id, product_price, quantity)*, we felt that this was unnecessary because we often search things by *product_name* and want to see its price. Having to join these tables constantly would be slow and unnecessary. We could have also done something like *product_sale_inventory(product_name, product_price, quantity)*. However, there could be multiple *product_id*s with the name *product_name*, such as if the two products are the same model but in different colors.
3. **BCNF:** *lego_parts(product_id, category_id)*
   Functional dependencies: *product_id → category_id*
   We noticed that *category_id* was unique to each lego part (and thus each *product_id*). We could have included *category_name* here as well, but because this is

only dependent on *category_id* but not necessarily *product_id*, we included that in a separate table that only includes *category_id* and *category_name*.

4. **BCNF:** *customers(customer_username, customer_name, customer_email, member_type)*
   Functional dependencies: *customer_username → customer_name, customer_email, member_type*
   One thing that we considered doing is also keeping track of the *discount_amount* here. *discount_amount* depends on whether the customer is a regular customer or a VIP customer. Because this is not dependent on the customer but rather what *type* of member they are, we created another table that related those two more directly called *discounts*.

NF Proof 1:

BCNF: *themes(theme_id, theme_name, parent_id)*
Functional dependencies: *theme_id → theme_name, parent_id*
Proof: Since we have dependencies *theme_name → theme_id*, we will show that *theme_id* is a superkey. We'll compute $(theme\_id)^+$. Initially, $\alpha^+ = theme\_id$. Since *theme_id → theme_name, parent_id*, we have $\alpha^+ = theme\_id, theme\_name, parent\_id$, which is our entire relation.

NF Proof 2:

BCNF: *customers(customer_username, customer_name, customer_email, member_type)*
Functional dependencies: *customer_username → customer_name, customer_email, member_type*
Proof: Similarly, given *customer_username → customer_name, customer_email, member_type*, we will show that *customer_username* is a superkey. We'll compute $(customer\_username)^+$. Initially, $\alpha^+ = customer\_username$. Since *customer_username → customer_name, customer_email, member_type*, we have $\alpha^+ = customer\_username, customer\_name, customer\_email, member\_type$, which is our entire relation.

Part G. Relational Algebra

Requirements (from Final Project specification, Part G):

- Minimum of 3 non-trivial queries (e.g. no queries simply in the form **SELECT <x> FROM <y>**)
- At least 1 group by with aggregation
- At least 3 joins (across a minimum of 2 queries)
- At least 1 update, insert, and/or delete
    - This may be equivalent to said SQL statements elsewhere (e.g. queries or procedural code), but are not required to be; in other words, you can write these independent of other sections
- Appropriate projection/extended projection use
- Computed attributes should be renamed appropriately
- Part of your grade will come from overall demonstration of relational algebra in the context of your schemas; obviously minimal effort will be ineligible for full credit; it is difficult to formally define "obviously minimal", but refer to A1 and the midterm for examples of what we're looking for
- Above each query, briefly describe what it is computing; we will use this to grade for correctness based on what the query is supposed to compute; lack of descriptions will result in deductions, since we have no idea otherwise of what the query is intended to do.

Below, provide each of your RA queries following their respective description.

1. Computes the number of sets for each theme.
   $\Pi_{\text{theme\_name, num\_sets}}(_{\text{theme\_name}}G_{\text{count-distinct(product\_id) as num\_sets}}(\text{themes} \bowtie \text{lego\_sets}))$

2. Computes the average rating for products that have received reviews.
   $\Pi_{\text{product\_name, avg\_rating}}(_{\text{product\_name}}G_{\text{avg(rating) as avg\_rating}}(\text{reviews} \rtimes (\sigma_{\text{purchases.product\_id=product\_inventory.product\_id}}(\text{purchases} \times \text{product\_inventory}))$

3. Lists each customer and amount of money they have spent.
   $\Pi_{\text{customer\_name, total\_spent}}(_{\text{customer\_name}}G_{\text{sum(produce\_price) as total\_spent}}(\sigma_{\text{customers.customer\_id=purchases.customer\_id}}(\text{customers} \times \text{purchases}) \rtimes \text{product\_inventory}))$

4. Decreases inventory for purchased items by 1.
   $\text{product\_inventory} \leftarrow \Pi_{\text{product\_id, price, product\_name, quantity - 1}}(\sigma_{\text{product\_id}}(\text{purchases} \rtimes \text{product\_inventory})) \cup \Pi_{\text{product\_id, price, product\_name, quantity}}(\sigma_{\text{product\_id}}(\text{product\_inventory} - \text{purchases}))$

Part L1. Written Reflection Responses

Challenges and Limitations

List any problems (at least one) that came up in the design and implementation of your database/application (minimum 2-3 sentences)

Answer:

- **Loading data:** We ran into multiple issues with the loading and value types of our database. We had duplicate rows and missing data. However, we partner-debugged the duplicate rows and added an OPTIONALLY ENCLOSED to catch all names in our categories. We manually added the data we wanted that was not available in the dataset we found (e.g. time_to_completion and price).
- **Normal forms:** We also did not initially have a 3NF database. To achieve a 3NF form database, we added a discounts table. This allows us to decompose the database more efficiently since the discount is only dependent on the amount.
- **Multiple Results:** Often we found that when writing queries to just show some sets to customers, we had to limit it to 1 for non-unique values.
- **Python Implementation:** We found that we had to commit the data to the database after making changes, and we found it challenging to call procedures correctly. We also made some minor changes to our database after seeing the Python implementation; for example, we originally had customer ID and employee ID but changed this to username so that we could correlate this with the username that people use to log into the application.
- **Other (minor) design choices:**
    - Condensed inventory and products because of one-to-one relationship / redundancy.
    - Added purchase_id to purchases because product_id might not be unique.
    - Changed E-R model to reflect that themes and categories are not weak entities because they can exist without having a set or part.
    - Split admin side into employee and employee log to keep persons and actions separate.
    - Kept customer_id in reviews for easier access to customer name.
    - Assumed that every purchase is for one item.
    - Edited product_id to be a number we assigned (sets are 1 - 11,000 and parts are like 100,000 to some higher number).
    - For inventory data and quantity, added in prices. Chose to make the earlier sets a random price between $2-12 and the laters sets starting in the year 2000 from $50-200 to reflect inflation and the popularity of legos. The individual parts are also a random value from $2-12.
    - Removed color because it is not a popular query.
    - Added a purchase_price in order to keep track of revenue made and discounts given.
    - Assumed employees are the only ones with access to employee login for app.py, this will allow them to be customers if they want.

Future work

If you are particularly eager for a certain application (have your own start-up in mind?), it is easy to over-scope a final project, especially one that isn't a term-long project. You can list any stretch goals you might have "if you had the time" which staff can help give feedback on prioritizing (2-3 sentences).

Answer:

- **Stretch goals:** A finished website and additional in-depth queries.
- **Website development:** We both have previous knowledge of website development either through CS 132 or self-study. We thought it might be cool to make a semi-accurate replica of the LEGO website with our own twist.
- **Additional Queries:** We wanted to implement our second stretch goal, additional queries, to allow for wider usage by the customer on our database. Realistically, the LEGO website and store have many paths for customers to proceed on. We were not able to create as many as we wanted due to time constraints, but it would be interesting to do eventually.
- **Edge cases/Debugging:** We want to include several constraints, such as allowing customers to only purchase a product if there is 1 or more in stock; checking for valid user input, etc. that we did not have time to implement. As a result, our program only works if the user gives it valid inputs.

Collaboration (Required for Partners)

This section is required for projects which involve partner work. Each partner should include 2-3 sentences identifying the amount of time they spent working on the project, as well as their specific responsibilities and overall experience working with a partner in this project.

Partner 1 Name: Gabriella Twombly
Partner 1 Responsibilities and Reflection:

I really enjoyed working on this project! Part of it was because database creation seems really fun to me, but also because Ellen is such a great partner. We met early during the week and came up with a plan for partnerwork time everyday with the intention of finishing the project before the due date. We worked on almost everything together to some degree in one of the Bechtel study rooms. We spent at least 3-4 hours everyday starting Saturday working together on the project. I focused on processing some of the data, the E-R models, the routines, the password setup, the permissions setup, the DDL, the python application, and writing up various parts of the report. We also worked on the ideal queries and UDFS for our database planning. We met with Prof. Hovik at least twice.

Partner 2 Name: Ellen Min
Partner 2 Responsibilities and Reflection:

I also enjoyed working with Twombly on this project! I feel like we got to explore practical applications of our homework and lectures in more creative and fun settings. We had a lot of fun researching lego sets and coming up with the data, and we started pretty early to reduce stress toward the end of the week. I feel like we worked really well together and were able to come up with a plan and work on our parts both together and individually.

We started early on the project and developed a timeline that ended the Thursday before the due date. We worked on the project for at least 3-4 hours daily starting the previous Saturday working together on the project in Bechtel. We met with Professor Hovik who was able to guide some of our design decisions. I worked on implementing the DDL, queries, and routines, writing down the algebra, loading the data, and developing the Python application.

Other comments

This is the first time CS 121 has had a Final Project, and we would appreciate your feedback on whether you would recommend this in future terms, as well as what you found most helpful, and what you might find helpful to change.

Answer:

We really enjoyed having the end goal in this class be a project. We were able to learn a lot by being so hands on, but we also had fun with it. The only suggestions we would make is that it would be helpful to add a possible timeline, to make sure that the final spec is working properly/edited, and to add a couple OH sessions for the final project. Otherwise, great job! Thanks so much.