# DESIGN REPORT

## MCDA5540

TRISHLA SHAH


**Prepared By:**

Bhagya Shree (A00431152)

Madeleine Min Jing Leong (A00430926)

Mohammad Nawaz (A00428036)

Rishab Gupta (A00429019)

# Table of Contents

## Executive Summary

A relational database schema is designed for Halifax Science Library (HSL) to fulfil its requirements for the purpose of maintaining good and efficient operating performance. This design has three aspects as followings:

1. record all articles for each magazine
2. record transactions' history
3. record monthly expenses

To perform a good design for the schema, an Enhanced Entity Relational (EER) Diagram is drawn according to the data requirements, and the cardinalities for every entity are identified. Then a mapping of EER to relational database schema is done. Additionally, query optimization is addressed to justify our model design is cost effective model.

## Tools Used

EER Diagram : Online Diagramming tool, creately.com

Relational Database Schema : Online Database Modelling tool, ERDPlus.com

Query Optimization : MySQL Workbench 8.0 CE

# Enhanced Entity Relational (EER) Diagram

Before creating database for HSL, an enhanced entity relation (EER) diagram is first drawn as it gives better graphic illustration on every required entity and the relationships between them, and leads to the ability of creating a better database design from the EER diagram. Bellow are the EER diagrams created with fulfilling HSL's data gathering requirements.

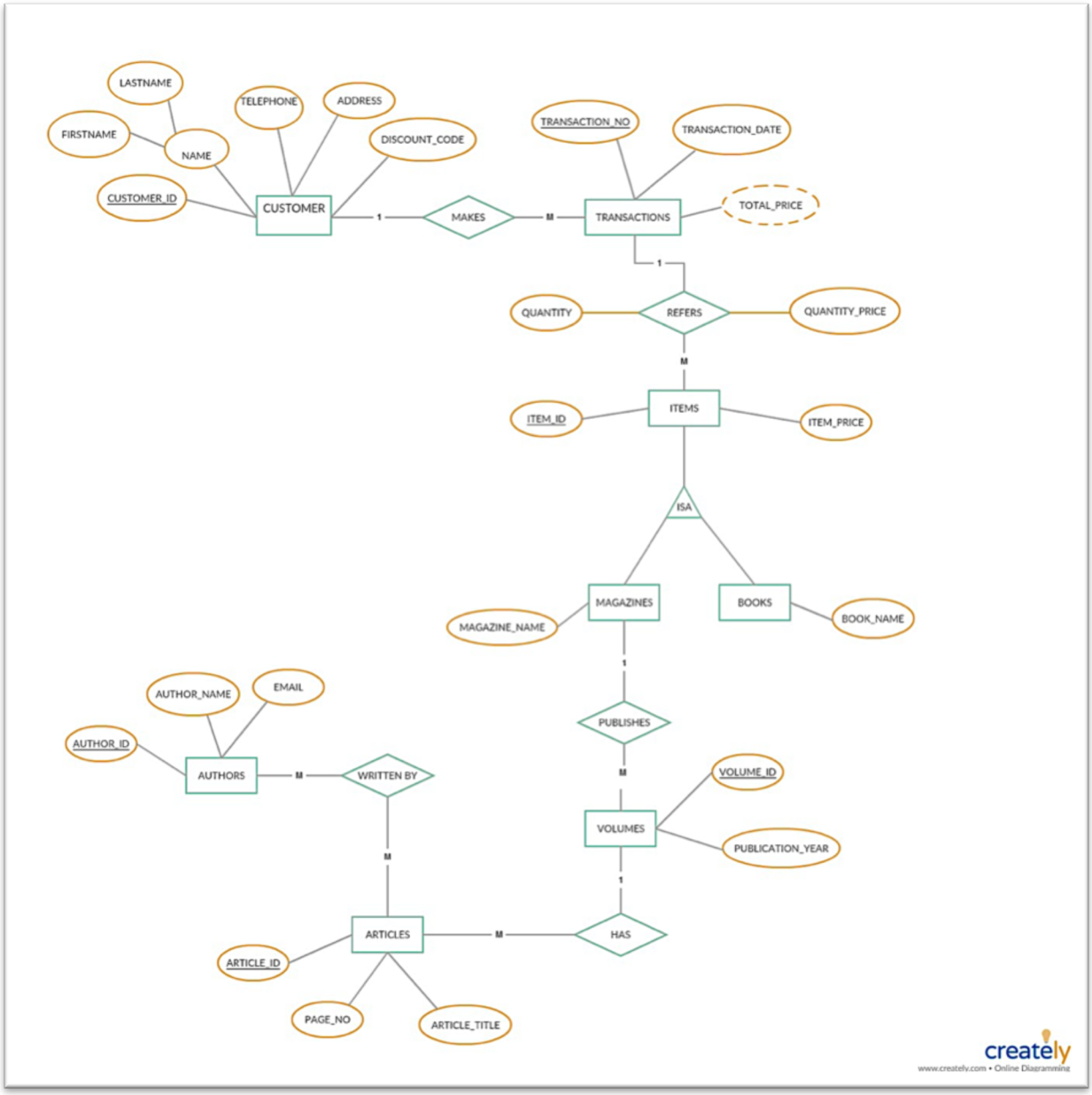


Figure 1: EER diagram for new library items, customers and transactions

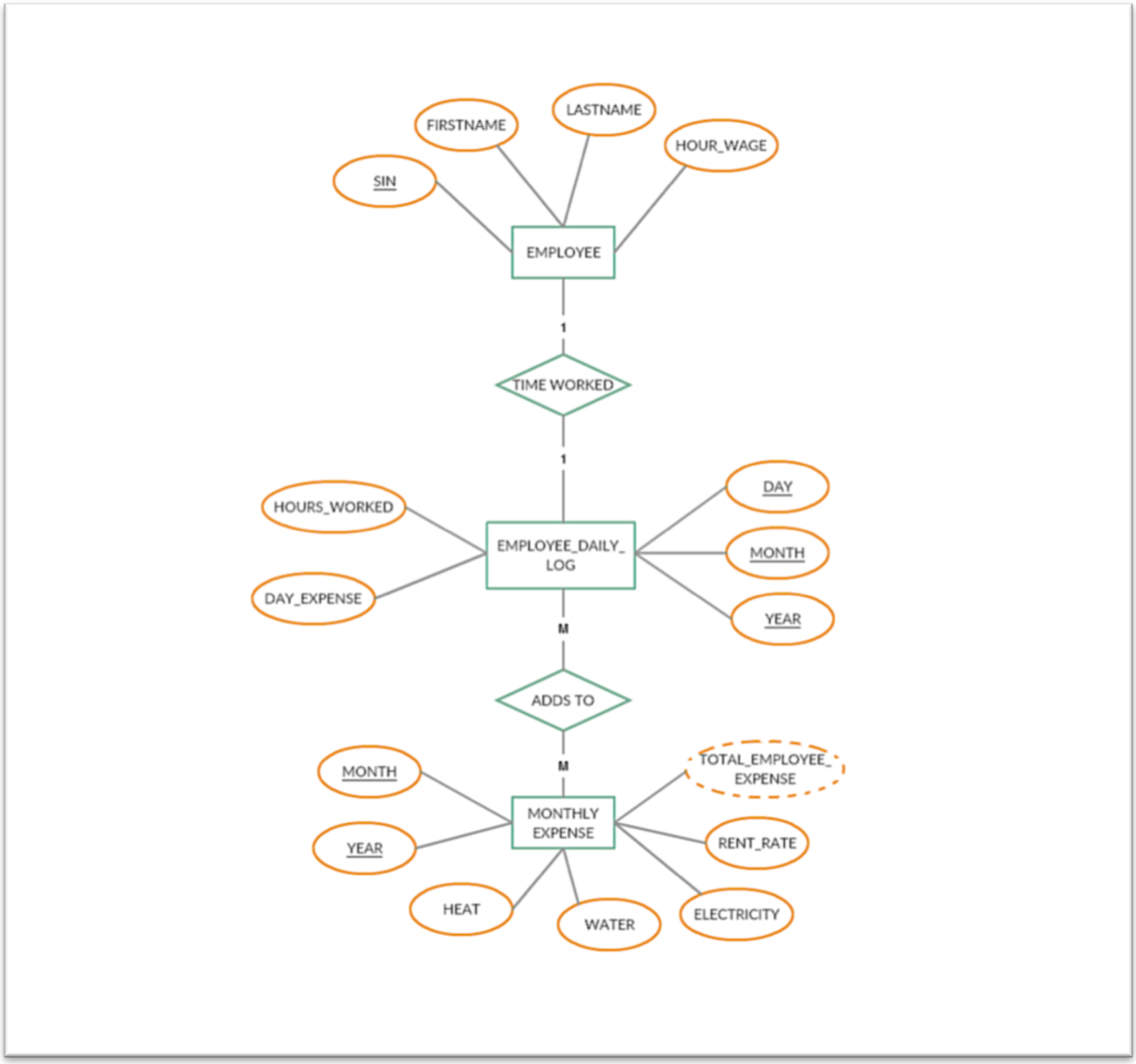


Figure 2: EER diagram for HSL monthly expenses

# Mapping of EER Diagram to Relational Database

As our EER diagram is drawn completely, we have to do mapping process from the EER diagram to relational database through table creation for each entity, attributes should become fields of tables, and declare primary key.

Below is the list of tables we created with their attributes:

- Each relation is in 3NF because it is in 2NF and no non-candidate-key attribute is transitively dependent on any candidate key (i.e., also on PK).
- Each relation is in BCNF because every determinant is a candidate key.

1. ITEM (Item_Id, Item_price)

2. MAGAZINE (Magazine, Magazine Name)
   Foreign Key MAGAZINE (Magazine_Id) references ITEM (Item_Id)

3. MAGAZINE_VOLUME (Volume_Id, Magazine_Id, Publication_Year)
   Foreign Key MAGAZINE_VOLUME (Magazine_Id) references ITEM (Item_Id)

4. ARTICLE (Article_Id, Article_title, Page_No, Volume_Id)
   Foreign Key ARTICLE (Volume_Id) references VOLUME (Volume_Id)

5. AUTHOR (Author_Id, Author_name, email)

6. ARTICLE_AUTHOR (Art_Auth_Id, Article_Id, Author_Id)
   Foreign Key ARTICLE_AUTHOR (Article_Id) references ARTICLE (Article_Id)
   Foreign Key ARTICLE_AUTHOR (Author_Id) references AUTHOR (Author_Id)

7. CUSTOMER (Customer_Id, Firstname, Lastname, Telephone, Address,
   Discount_Code)

8. TRANSACTION (Transaction_Number, Total_Price, Customer_Id)
   Foreign Key TRANSACTION (Customer_Id) references CUSTOMER (Customer_Id)

9. TRANSACTION_ITEM (Tran_Item_Id, Quantity, Quantity_price, Item_Id, Transaction_Number)
   Foreign Key TRANSACTION_ITEM (Item_Id) references ITEM (Item_Id)
   Foreign Key TRANSACTION_ITEM (Transaction_Number)
   references TRANSACTION (Transaction_Number)

10. EMPLOYEE (SIN, Firstname, Lastname, Hour_wage)

11. RENT (Year, Rent_Rate)

12. MONTHLY_EXPENSE (Month, Year, Heat, Water, Electricity, Total_employee_expense)
   Foreign Key MONTHLY_EXPENSE (Year) references RENT (Year)

13. EMPLOYEE_DAILY_LOG (Day, Month, Year, SIN, Hours_Worked, Day_expense)
   Foreign Key EMPLOYEE_DAILY_LOG (Month)
   references MONTHLY_EXPENSE (Month)
   Foreign Key EMPLOYEE_DAILY_LOG (Year)
   references MONTHLY_EXPENSE (Year)
   Foreign Key EMPLOYEE_DAILY_LOG (SIN)
   references EMPLOYEE (SIN)

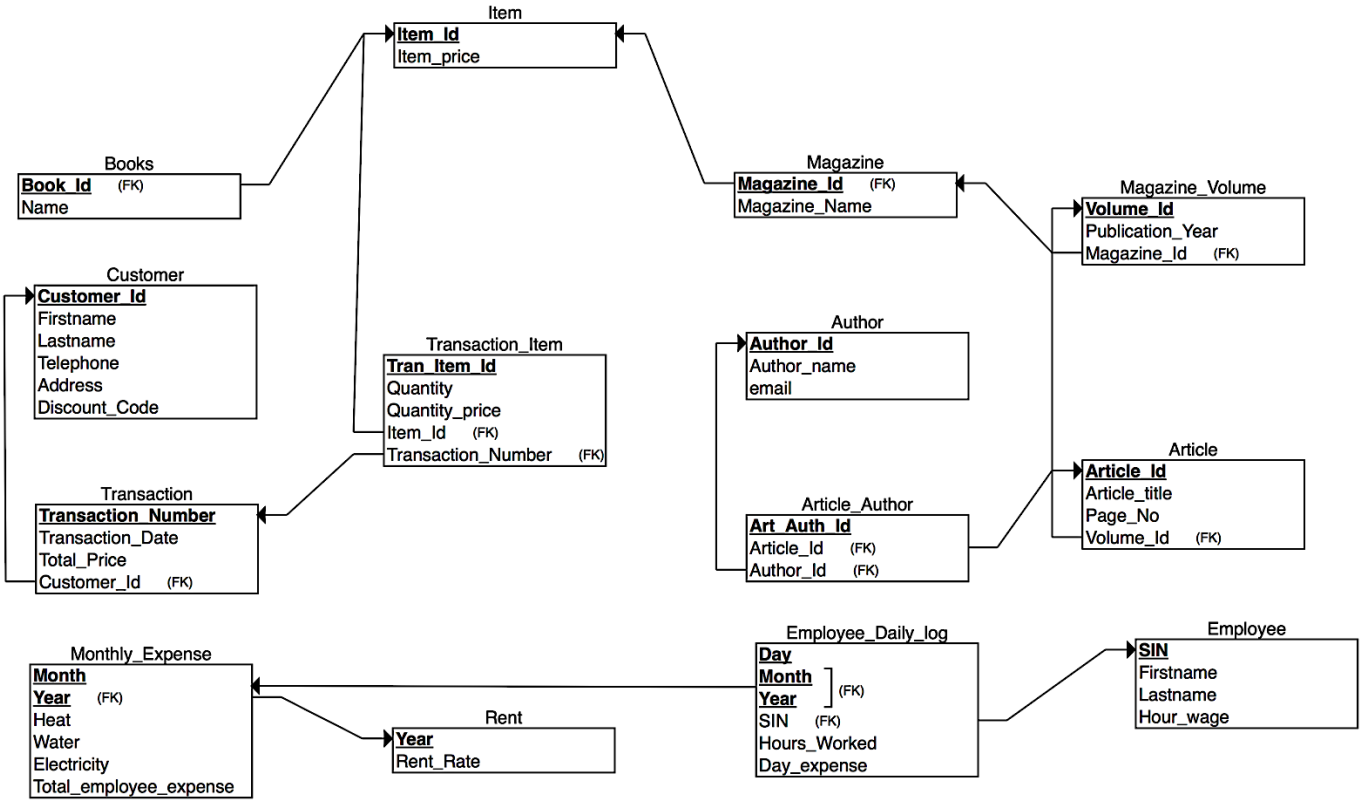


Figure 3: Relational database schema for Halifax Science Library

## Design Guidelines Followed for HSL Database Design

HSL database schema design has followed four guidelines as below:

1. Semantics of the Attributes

   The relational schema is formed with containing some meaning among the attributes for each table. For example, table 1 solely contains customer information details. It is easy for library employees to record or track specific customer from this table.

| Customer_Id | Firstname | Lastname | Telephone | Address | Discount_Code |
|---|---|---|---|---|---|
| | | | | | |

Table 1: Customer Table

2. Reducing the Redundant Value in Tuples

   Redundancy of Information is reduced by separating multiple entities in our relational model. Additionally, removing the redundancy in the tuples will prevent anomalies of update, insertion, deletion and modification in our model. For instance, instead of

having one table for author and article, and having same author for multiple articles, article and author entities are separated to different tables, and a new table named ARTICLE_AUTHOR is created so that the values will be mapped one-to-one and without losing any information.

3. Reducing Null Values in Tuples

   Grouping too many attributes may bring us many nulls in the tuples and the end user, HSL staffs will have problem understanding the meaning of the attributes. To avoid such problem, some attributes should be ungrouped. For example, article and author attributes are related as author writes article, but these two entities are separated in order to avoid having one record of containing article's values but not author's value when author can't be identified.

4. Disallowing Spurious Tuples

   Our schema is designed as such we have prevented the generation of spurious tuples and erroneous results after joining the tuples of our database, e.g. by joining the customer and transaction table with the equality condition of customer_id, there is no additional tuples created from new table as the customer_id attribute is used to join in which it is a primary key in customer table as well as a foreign key in transaction table.

# Optimization

Below are three steps to optimize our relational database in order to build a cost-effective relational model:

1. Use Joins (Inner Joins) to connect Foreign Key constraints rather than using WHERE:

   For example, in the Article and Article_Author Table, rather than using WHERE Article.Article_ID = Article_Author.Article_ID we should use an inner join on these fields to ensure better performance. This is because a WHERE clause will use a cross join, matching every possible outcome then filtering out those that we require.

2. Try to reduce usage of Wildcards:

   Wildcard usage for finding records is very inefficient as it can include any types of data containing the characters mentioned between '%'. However, in our database design for our queries we will try our best to limit this to only the end of the phrase, to ensure that we reduce the cost of matching from both sides of the phrase.

3. Indexing

Indexing allows the sorting of data on a column, so a query knows where exactly to search, thereby reducing the time and improving efficiency. The query will not need to run a FULL-TABLE-Scan, that is scanning each record until it finds the right one to output the results.

Since we propose to use joins instead of where, we need to take into consideration the SQL constraint of leaning on ONLY ONE index per joined table per query. Even though we could technically have more indexes it will not be beneficial in the joins. Therefore, we have to index only column against a table in our join to improve the efficiency of our database

We will target our indexing to be placed in tables which have a higher cardinality. This is to maximise our efficiency gain. An index for instance on a field which has 5 values like Discount code will not gain us much efficiency as opposed to placing an index on the Customer ID. This is because the B-Tree will be able to eliminate more records that do not match our criteria

We need to be mindful however that since indexing will speed up our SELECT and WHERE clauses, it will slow down our UPDATE and INSERT statements.

Below is an example to show using index is better than using full table scan to find author email.


Figure 4: FULL-TABLE-Scan and Index Search from AUTHOR table

# References


- *Informal Design Guidelines for Relational Schema*. [online] Idc-online. Available at: http://www.idc-online.com/technical_references/pdfs/information_technology/Informal_design_guidelines_for_relational_schema.pdf
- Ginsberg, J. (2018). *3 Things You Should Know About SQL Indexes*. [online] The Celerity Blog. Available at: http://blog.celerity.com/how-to-design-sql-indexes [Accessed 8 Nov. 2018].
- Winnard, M. (2018). *Anatomy of an SQL Index: What is an SQL Index*. [online] Use the Index. Available at: https://use-the-index-luke.com/sql/anatomy [Accessed 8 Nov. 2018].
- TP (2018). *SQL - Indexes*. [online] TP. Available at: https://www.tutorialspoint.com/sql/sql-indexes.htm [Accessed 8 Nov. 2018].