Design and Implementation of a Relational Database for Laptop Specifications and Customer Reviews

Venkata Sai Bhargav Dhara (UB ID: 50559703)*
Thanmaya Sri Sigireddi (UB ID: 50559907)*
Lakshmi Mounika Masimukku (UB ID: 50560106)*

***Abstract*—This project presents the design and implementation of a relational database system for storing and analyzing laptop specifications and customer reviews. The objective is to create a normalized database schema that supports efficient querying, ensures data integrity through constraints, and optimizes performance via indexing strategies. The dataset, comprising over 5,000 laptop records with more than 20 attributes each, is sourced from a CSV file and undergoes extensive preprocessing and transformation before being loaded into the database. Advanced SQL functions and stored procedures are developed to enhance data retrieval capabilities. This report details the problem statement, target users, use case scenarios, schema design, normalization process, data transformation, SQL implementation, query execution analysis, constraints, indexing, challenges encountered, and potential future work.**

I. Executive Summary

The proliferation of laptops in the consumer market necessitates a structured approach to managing and analyzing extensive specifications and customer feedback data. This project addresses this need by developing a relational database system that efficiently stores detailed laptop information and associated customer reviews. The scope includes data preprocessing, schema design adhering to normalization principles, implementation of SQL queries for data manipulation and retrieval, performance optimization through indexing, development of advanced SQL functions, and analysis of query execution plans. The outcome is a robust database that facilitates complex queries and supports various stakeholders such as manufacturers, retailers, consumers, and data analysts in making informed decisions.

II. Problem Statement

Designing a relational database capable of handling diverse and voluminous data related to laptop specifications and customer reviews presents significant challenges. An Excel file is insufficient due to limitations in handling large datasets, lack of support for complex queries, and inability to enforce data integrity constraints. The database must support queries involving multiple tables, provide fast data retrieval, and maintain data consistency through constraints and normalization. Additionally, the system should accommodate updates, deletions, and insertions, reflecting the dynamic nature of the laptop market. The goal is to create a scalable and efficient database system that overcomes these challenges.

III. Target Users and Use Case Scenarios

A. Target Users

- **Manufacturers**: To analyze competitor specifications, identify market trends, and understand customer feedback.
- **Retailers**: To manage inventory, assist customers in product selection, and optimize stock based on popular models.
- **Consumers**: To compare laptop models based on specifications and reviews, aiding in purchase decisions.
- **Data Analysts**: To perform market analysis, identify consumer preferences, and predict future trends.

B. Use Case Scenarios

- **Scenario 1**: A retailer queries the database to find laptops within a specific price range, with at least 16 GB of RAM and an SSD of 512 GB or more, to recommend to a customer.
- **Scenario 2**: A manufacturer analyzes customer reviews associated with their latest laptop model to identify common complaints or praises.
- **Scenario 3**: A data analyst performs a trend analysis on the popularity of different GPU brands over the past year.
- **Scenario 4**: A consumer searches for laptops with specific processor specifications and reads detailed product information and reviews.

IV. ER Diagram and Schema Design

The database schema is derived from an Entity-Relationship (ER) model that captures the entities involved and their relationships. The primary entities and relationships are depicted in Figure 1.

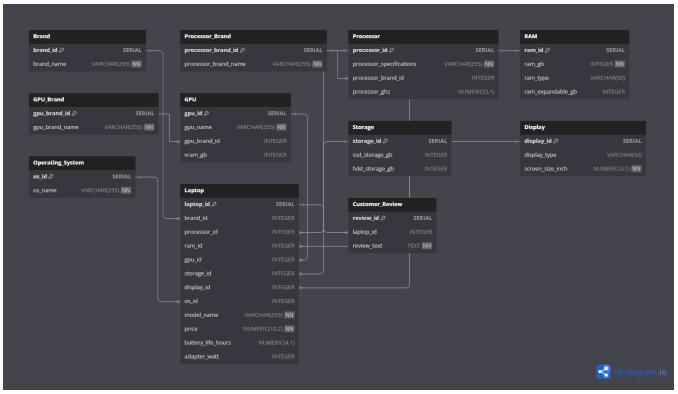

Fig. 1: Entity-Relationship Diagram of the Laptop Database

A. Entities and Attributes

- **Brand**: *brand_id* (PK), *brand_name*
- **Processor_Brand**: *processor_brand_id* (PK), *processor_brand_name*
- **Processor**: *processor_id* (PK), *processor_specifications*, *processor_brand_id* (FK), *processor_ghz*
- **RAM**: *ram_id* (PK), *ram_gb*, *ram_type*, *ram_expandable_gb*
- **GPU_Brand**: *gpu_brand_id* (PK), *gpu_brand_name*
- **GPU**: *gpu_id* (PK), *gpu_name*, *gpu_brand_id* (FK), *vram_gb*
- **Storage**: *storage_id* (PK), *ssd_storage_gb*, *hdd_storage_gb*
- **Display**: *display_id* (PK), *display_type*, *screen_size_inch*
- **Operating_System**: *os_id* (PK), *os_name*
- **Laptop**: *laptop_id* (PK), *brand_id* (FK), *processor_id* (FK), *ram_id* (FK), *gpu_id* (FK), *storage_id* (FK), *display_id* (FK), *os_id* (FK), *model_name*, *price*, *battery_life_hours*, *adapter_watt*
- **Customer_Review**: *review_id* (PK), *laptop_id* (FK), *review_text*

B. Relationships

- Each **Laptop** is associated with one **Brand**, **Processor**, **RAM**, **GPU**, **Storage**, **Display**, and **Operating_System**.
- Each **Processor** is associated with one **Processor_Brand**.
- Each **GPU** is associated with one **GPU_Brand**.
- Each **Customer_Review** is associated with one **Laptop**.

V. Normalization and Dependencies

A. Functional Dependencies

For each relation, the functional dependencies are analyzed to ensure compliance with Boyce-Codd Normal Form (BCNF).

- **Brand**: *brand_id* $\rightarrow$ *brand_name*
- **Processor_Brand**: *processor_brand_id* $\rightarrow$ *processor_brand_name*
- **Processor**: *processor_id* $\rightarrow$ *processor_specifications*, *processor_brand_id*, *processor_ghz*
- **RAM**: *ram_id* $\rightarrow$ *ram_gb*, *ram_type*, *ram_expandable_gb*
- **GPU_Brand**: *gpu_brand_id* $\rightarrow$ *gpu_brand_name*
- **GPU**: *gpu_id* $\rightarrow$ *gpu_name*, *gpu_brand_id*, *vram_gb*
- **Storage**: *storage_id* $\rightarrow$ *ssd_storage_gb*, *hdd_storage_gb*
- **Display**: *display_id* $\rightarrow$ *display_type*, *screen_size_inch*
- **Operating_System**: *os_id* $\rightarrow$ *os_name*
- **Laptop**: *laptop_id* $\rightarrow$ *brand_id*, *processor_id*, *ram_id*, *gpu_id*, *storage_id*, *display_id*, *os_id*, *model_name*, *price*, *battery_life_hours*, *adapter_watt*
- **Customer_Review**: *review_id* $\rightarrow$ *laptop_id*, *review_text*

B. Normalization Decisions

All relations are in BCNF because:

- Each non-trivial functional dependency has a superkey on the left side.
- There are no partial or transitive dependencies in the relations.

Maintaining BCNF ensures minimal redundancy and enhances data integrity. During the normalization process, attributes were allocated to appropriate tables to eliminate anomalies and redundancy.

VI. Dataset Description

A. Data Source

The dataset is sourced from a publicly available CSV file containing raw laptop specifications and customer reviews. It comprises over 5,000 records, each with more than 20 attributes, including brand, model name, price, processor specifications, RAM details, storage capacities, display information, GPU details, operating system, and customer reviews.

B. Data Processing and Transformation

Data processing involves several steps:

- **Loading Raw Data**: The raw CSV data is loaded into a temporary table named *raw_laptop_data*.
- **Data Cleaning**: Null and empty values are handled by assigning default values or setting them to `NULL`.
- **Data Transformation**: Regular expressions are used to extract numeric values from strings (e.g., extracting ”16” from ”16GB RAM”). Units are converted where necessary (e.g., battery life from minutes to hours).
- **Creating Cleaned Table**: A new table, *fixed_augmented_laptop_data*, is created to store the cleaned and transformed data.
- **Inserting into Normalized Tables**: Data is inserted into the normalized tables, establishing relationships via foreign keys.

C. Sample Data Rows

An excerpt from the *fixed_augmented_laptop_data* table is presented in Table I.

TABLE I: Sample Transformed Data

Brand	Model_Name	Price	Processor_Specifications	RAM (GB)
Dell	XPS 13	999.99	Intel Core i7-10710U	16
HP	Spectre x360	1199.99	Intel Core i7-1065G7	16
Lenovo	ThinkPad X1	1299.99	Intel Core i5-10210U	8

VII. SQL Implementation

A. Table Creation

Tables are created using `CREATE TABLE` statements with appropriate data types and constraints. An example is provided below:

Listing 1: Creating the Brand Table

```
CREATE TABLE Brand (
    brand_id SERIAL PRIMARY KEY,
    brand_name VARCHAR(255) NOT NULL UNIQUE
);
```

B. Data Insertion

Data is inserted into tables using `INSERT INTO` statements after processing. For instance:

Listing 2: Inserting Data into the Brand Table

```
INSERT INTO Brand (brand_name)
SELECT DISTINCT Brand
FROM fixed_augmented_laptop_data
WHERE Brand IS NOT NULL
ORDER BY Brand;
```

C. Advanced SQL Functions and Stored Procedures

To enhance the functionality of the database, several SQL functions and stored procedures are developed.

1) Function: Get Popular Laptops: This function retrieves laptops based on popularity, considering the number of reviews and average ratings.

Listing 3: Function to Get Popular Laptops

```
CREATE OR REPLACE FUNCTION get_popular_laptops(limit_count INTEGER DEFAULT 10)
RETURNS TABLE (
    laptop_id INTEGER,
    model_name VARCHAR,
    brand_name VARCHAR,
    price NUMERIC,
    review_count BIGINT,
    avg_rating NUMERIC,
    specs_summary TEXT
) AS $$
BEGIN
    RETURN QUERY
    -- Function logic
END;
$$ LANGUAGE plpgsql;
```

2) Function: Search Laptops: This function allows users to search for laptops based on various criteria.

Listing 4: Function to Search Laptops

```
CREATE OR REPLACE FUNCTION search_laptops(
    search_term VARCHAR,
    min_price NUMERIC = 0,
    max_price NUMERIC = 999999,
    p_brand_name VARCHAR = NULL,
    ram_size INTEGER = NULL
)
RETURNS TABLE (
    laptop_id INTEGER,
    model_name VARCHAR,
    brand_name VARCHAR,
    price NUMERIC,
    ram_gb INTEGER,
    processor_specs VARCHAR,
    gpu_name VARCHAR
) AS $$
BEGIN
    RETURN QUERY
    -- Function logic
END;
$$ LANGUAGE plpgsql;
```

3) Function: Get Laptop Details: This function retrieves detailed information about a specific laptop.

Listing 5: Function to Get Laptop Details

```
CREATE OR REPLACE FUNCTION get_laptop_details(p_la
RETURNS TABLE (
    laptop_id INTEGER,
    model_name VARCHAR,
    brand_name VARCHAR,
    price NUMERIC,
    ram_gb INTEGER,
    ram_type VARCHAR,
    processor_specs VARCHAR,
    processor_brand VARCHAR,
    gpu_name VARCHAR,
    gpu_brand VARCHAR,
    storage_ssd INTEGER,
    storage_hdd INTEGER,
    display_type VARCHAR,
    screen_size NUMERIC,
    os_name VARCHAR,
    battery_life NUMERIC,
    adapter_watt INTEGER
) AS $$
BEGIN
    RETURN QUERY
    -- Function logic
END;
$$ LANGUAGE plpgsql;
```

D. Data Manipulation Queries

Various SQL queries are executed for data manipulation:

Listing 6: Inserting a New Laptop Record

```
INSERT INTO Laptop (
    brand_id, processor_id, ram_id, gpu_id,
    display_id, os_id, model_name, price,
    battery_life_hours, adapter_watt
)
VALUES (
    1, 2, 3, 4, 5,
    6, 7, 'Inspiron 15', 799.99,
    10.0, 65
);
```

Listing 7: Updating the Price of a Laptop

```
UPDATE Laptop
SET price = 749.99
WHERE model_name = 'Inspiron 15';
```

Listing 8: Deleting a Customer Review

```
DELETE FROM Customer_Review
WHERE review_id = 100;
```

4) Select Queries: **Query 1:** Retrieve laptops with at least 16 GB of RAM.

```
SELECT
    l.model_name,
    b.brand_name,
    r.ram_gb
FROM
    Laptop l
JOIN
    Brand b ON l.brand_id = b.brand_id
JOIN
    RAM r ON l.ram_id = r.ram_id
WHERE
    r.ram_gb >= 16;
```

Query 2: List laptops with SSD storage greater than 512 GB.

```
SELECT
    l.model_name,
    s.ssd_storage_gb
FROM
    Laptop l
JOIN
    Storage s ON l.storage_id = s.storage_id
WHERE
    s.ssd_storage_gb > 512;
```

Query 3: Find average price of laptops by brand.

```
SELECT
    b.brand_name,
    AVG(l.price) AS average_price
FROM
    Laptop l
JOIN
    Brand b ON l.brand_id = b.brand_id
GROUP BY
    b.brand_name
ORDER BY
    average_price DESC;
```

Query 4: Retrieve laptops with specific processor specifications using a subquery.

```
SELECT
    model_name, price
FROM
    Laptop
WHERE
    processor_id IN (
        SELECT processor_id
        FROM Processor
        WHERE processor_specifications LIKE '%i7%'
    );
```

VIII. Query Execution Analysis

A. Problematic Queries

Three queries exhibited performance issues due to large data volumes and lack of appropriate indexing.

```
SELECT
    l.model_name,
    b.brand_name,
    l.price,
    r.ram_gb,
    s.ssd_storage_gb
FROM
    Laptop l
JOIN
    Brand b ON l.brand_id = b.brand_id
JOIN
    RAM r ON l.ram_id = r.ram_id
JOIN
    Storage s ON l.storage_id = s.storage_id
WHERE
    l.price > 1500 AND
    r.ram_gb >= 32 AND
    s.ssd_storage_gb >= 1024;
```

Issue: The query performs full table scans due to the absence of indexes on *price*, *ram_gb*, and *ssd_storage_gb*.

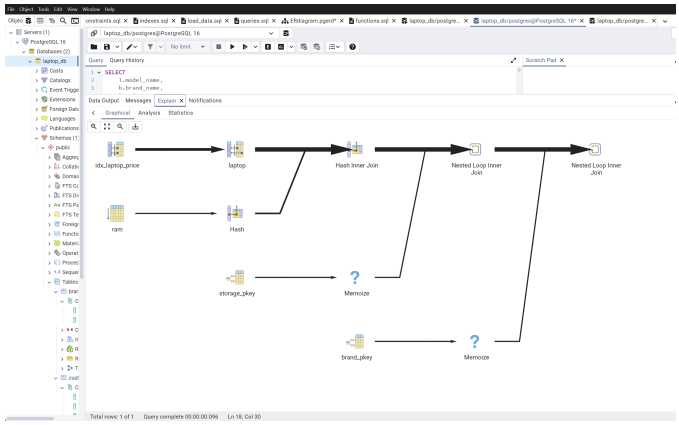

Fig. 2: Execution Plan for Query 1

```
SELECT
    c.review_text
FROM
    Customer_Review c
JOIN
    Laptop l ON c.laptop_id = l.laptop_id
WHERE
    l.model_name = 'XPS■13';
```

Issue: Slow performance due to lack of index on *model_name* in the *Laptop* table.

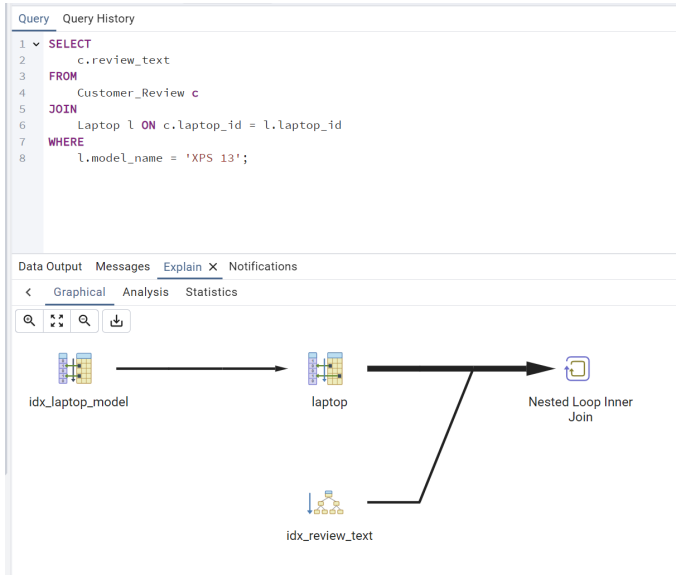

Fig. 3: Execution Plan for Query 2

```
SELECT
    gb.gpu_brand_name,
    AVG(l.price) AS average_price
FROM
    Laptop l
JOIN
    GPU g ON l.gpu_id = g.gpu_id
JOIN
    GPU_Brand gb ON g.gpu_brand_id = gb.gpu_brand_
GROUP BY
    gb.gpu_brand_name;
```

Issue: Performance degradation due to lack of indexes on foreign keys and grouping columns.

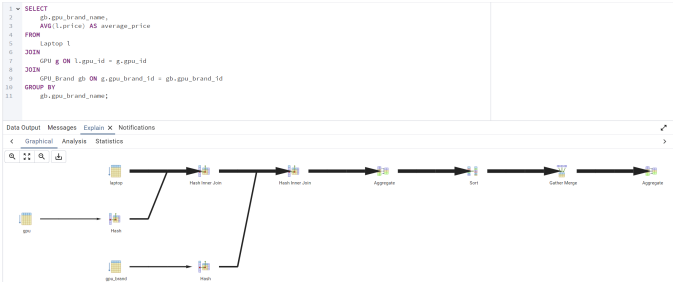

Fig. 4: Execution Plan for Query 3

B. Optimizations Performed

- **Index Creation**: Created indexes on *Laptop(price)*, *RAM(ram_gb)*, *Storage(ssd_storage_gb)*, *Laptop(model_name)*, and relevant foreign key columns.
- **Query Refactoring**: Rewrote queries to minimize subqueries and leverage indexed columns.
- **Execution Plan Analysis**: Used `EXPLAIN` to analyze query plans and ensure indexes were utilized.

IX. Constraints and Indexing

A. Primary and Foreign Keys

Primary keys (PK) and foreign keys (FK) are defined for all tables to enforce entity integrity and referential integrity, respectively. For example:

```
CREATE TABLE Laptop (
    laptop_id SERIAL PRIMARY KEY,
    brand_id INTEGER REFERENCES Brand(brand_id),
    -- Other columns and constraints
);
```

B. Constraints

Constraints are implemented to maintain data integrity:

- **NOT NULL**: Ensures essential fields are not left empty.
- **UNIQUE**: Enforces uniqueness of values in a column.
- **CHECK**: Validates data based on specified conditions. For instance, in the *RAM* table:

```
CONSTRAINT chk_ram_size CHECK (ram_gb > 0),
CONSTRAINT chk_ram_expandable CHECK (ram_expandable_gb >= 0)
```

C. Indexing Strategy

Indexes are created on columns frequently used in `WHERE`, `JOIN`, and `GROUP BY` clauses to optimize query performance. Examples include:

- **Index on Price**: `CREATE INDEX idx_laptop_price ON Laptop(price);`
- **Index on RAM Size**: `CREATE INDEX idx_ram_size ON RAM(ram_gb);`
- **Composite Index**: For queries involving multiple columns.

```
CREATE INDEX idx_laptop_components ON
Laptop(processor_id, ram_id, gpu_id);
```

X. Challenges and Solutions

A. Data Quality Issues

The raw dataset contained missing values, inconsistent formatting, and invalid data types, leading to constraint violations during insertion.

B. Solutions Implemented

- **Enhanced Data Cleaning**: Improved data transformation scripts to handle missing and invalid values appropriately.
- **Default Values**: Assigned sensible default values where applicable to comply with constraints.
- **Error Logging**: Implemented logging mechanisms to capture and analyze records that failed insertion.
- **Constraint Adjustment**: Re-evaluated certain constraints to balance data integrity with practical data considerations.

XI. Future Work

Potential enhancements include:

- **User Interface Development**: Creating a web-based interface for easier data interaction and visualization.
- **Stored Procedures and Triggers**: Implementing advanced database functionalities for automation and enforcing complex business rules.
- **Machine Learning Integration**: Applying predictive analytics to forecast market trends and consumer behavior.
- **Scalability Improvements**: Optimizing the database for larger datasets and higher concurrent user loads.

XII. Conclusion

The project successfully designed and implemented a normalized relational database for storing and analyzing laptop specifications and customer reviews. Through meticulous schema design, adherence to normalization principles, effective data transformation, strategic indexing, and the development of advanced SQL functions and triggers, the database provides efficient and reliable data storage and retrieval. The system addresses the initial problem statement by offering capabilities beyond those of an Excel file, supporting complex queries, ensuring data integrity, and facilitating updates. Future work aims to enhance the system's functionality and user accessibility.

References

[1] A. Pall, "Database Normalization," *Andrei Pall's Blog*, 2019. [Online]. Available: https://andreipall.github.io/sql/database-normalization/. [Accessed: Dec. 2, 2024].

[2] E. Zimanyi, "Temporal Aggregates and Temporal Universal Quantification in Standard SQL," *ACM SIGMOD Record*, vol. 35, no. 2, pp. 16–21, Jun. 2006. [Online]. Available: https://dl.acm.org/doi/10.1145/1147376.1147380. [Accessed: Dec. 2, 2024].

[3] J. Kossmann, T. Papenbrock, and F. Naumann, "Data Dependencies for Query Optimization: A Survey," *The VLDB Journal*, vol. 31, pp. 1–22, 2022. [Online]. Available: https://link.springer.com/article/10.1007/s00778-021-00676-3. [Accessed: Dec. 2, 2024].

[4] E. Pollack, "Query Optimization Techniques in SQL Server: Database Design and Architecture," *SQL Shack*, Jul. 13, 2018. [Online]. Available: https://www.sqlshack.com/query-optimization-techniques-in-sql-server-database-design-and-architecture/. [Accessed: Dec. 2, 2024].

[5] "Normalization in SQL (1NF - 5NF): A Beginner's Guide," *DataCamp*, May 28, 2024. [Online]. Available: https://www.datacamp.com/tutorial/normalization-in-sql. [Accessed: Dec. 2, 2024].