NTNU
Kunnskap for en bedre verden

Norwegian University of Science and Technology

Faculty of Information Technology and Electrical Engineering

Department of Computer Science

Database Project – Spring 2025

IDATG2204 – Database Systems

ElectroMart: Design and Implementation of a Relational Database-Driven E-commerce Platform

Group Members

Name	Candidate Number
Abdulsamad Sheikh	10335
Nourdin Ben Karroum	10218
Bilal Rasulovich Mataev	10340
Mohammad Rdwan Alhammod	10352
Ole Bjørn Halvorsen	10358

May 22, 2025

Table of Contents

List of Figures

List of Tables

1 Introduction

This project document describes the design, development, and implementation of ElectroMart, a relational database-driven e-commerce platform. As a newly established company in electronics sales, ElectroMart aims to offer a wide range of products, including smartphones, laptops, tablets, cameras, home appliances, and accessories. The main focus of this project is to establish a robust and efficient database architecture that can support a user-friendly online shopping experience for the company's customers.

The task as database engineers involves designing and implementing a fully functional relational database, as well as developing basic frontend and backend components for the website. Although the project places the greatest emphasis on database knowledge, the development of supporting web application functionality will be necessary to demonstrate the database's capabilities in a realistic scenario.

The project goals include:

- **Database Design:** Develop a comprehensive relational database schema for storing information about products, users, orders, payments, and other relevant entities. This includes preparing an EER diagram, logical model, and normalizing tables to 1NF, 2NF, 3NF, and BCNF.
- **Backend Development:** Implement the server-side of the website to handle user authentication, product management, order processing, and other necessary logic.
- **Frontend Development:** Develop an intuitive user interface for product browsing, shopping cart management, and order completion.
- **Database Implementation:** Implement the designed database schema using a relational database management system (RDBMS), focusing on correct keys, integrity constraints, and efficient data structures.
- **Documentation:** Deliver a thorough report covering the database schema, APIs, frontend components, as well as instructions for running and testing the system. The report will also include an overview of assumptions made during the project.

This report will detail each phase of the project, from the initial requirements and design decisions to the final implementation and testing of the ElectroMart platform.

2 Individual Contributions

This project was a collaborative effort, with tasks distributed among group members to ensure comprehensive coverage of all project aspects. The following outlines the primary responsibilities

of each team member:

- **Abdulsamad Sheikh (Group Leader):** Oversaw the project's overall direction and progress. Led the database design phase, including the conceptual and logical modeling, and ensured the integrity of the normalization process. Also contributed to the final report compilation and review.
- **Nourdin Ben Karroum:** Focused on the backend development, including the implementation of API endpoints to interact with the database, handling business logic for order processing, user authentication, and product management.
- **Bilal Rasulovich Mataev:** Primarily responsible for the detailed database modeling tasks. This included creating the EER diagram, translating it into the logical model, and thoroughly documenting the normalization steps from 1NF to BCNF.
- **Mohammad Rdwan Alhammod:** Led the frontend development efforts. Designed and implemented the user interface for browsing products, managing the shopping cart, user registration, and viewing order history, ensuring a user-friendly experience.
- **Ole Bjørn Halvorsen:** Contributed significantly to the documentation and report writing. This involved compiling various sections, ensuring consistency in style and content, writing user guides, and preparing the project for final submission. He also took part in the testing phase.

Regular team meetings and shared documentation ensured that all members were aligned and could contribute to areas outside their primary responsibilities as needed.

3 Assumptions

During the design and development of the ElectroMart platform, several assumptions were made to define the scope and guide the implementation. These include:

- **Single Payment per Order:** The system assumes that each order will be paid for with a single payment transaction. Complex scenarios like partial payments or multiple payment methods for a single order are not within the current scope.
- **Optional Phone Number:** User registration can be completed without providing a phone number; it is an optional field.
- **Address Validity:** The system assumes that users will input valid and complete shipping and billing addresses. No external address validation service is integrated.
- **Product Images:** While the schema supports an 'ImageURL' for products, the actual hosting and management of image files are considered external to the core database functionality. The URL is assumed to point to a valid image resource.
- **Stock Management:** The system operates on a basic stock management principle where

an order cannot be placed for a product with zero stock. Backorder functionality is not implemented.

- **Currency:** All monetary values (prices, totals, etc.) are assumed to be USD. Multi-currency support is not implemented.
- **User Roles:** The current design primarily focuses on customer users. Admin or staff roles with different permissions and views are not explicitly detailed in this phase but can be an extension.
- **Security of Hashed Passwords:** The 'PasswordHash' field implies that passwords are securely hashed before storage. The specific hashing algorithm and salt management are crucial implementation details assumed to follow best practices but are not detailed in the schema definition itself.
- **TransactionID Uniqueness:** If a 'TransactionID' from a payment gateway is provided, it is assumed to be unique across all payments.

These assumptions helped in streamlining the development process. Future iterations of the project might revisit these assumptions to enhance the platform's capabilities.

4 Project Overview

ElectroMart is an initiative to establish a new player in the electronics product market through an online e-commerce solution. The company's vision is to offer a wide and varied range of electronics, spanning from personal devices like smartphones and laptops to larger household appliances and specialized accessories. The core of ElectroMart's strategy is to deliver a seamless, efficient, and customer-friendly shopping experience.

To realize this vision, the development of a solid technological backbone is crucial. This project specifically focuses on the design and implementation of this backbone, with the main emphasis on the relational database that will drive the entire platform. The mission, given to us as database engineers, is to construct a database that is not only robust and scalable but also optimized for the performance required by a modern e-commerce application. Inspiration is drawn from established e-commerce sites like Power, Elkjøp, Komplett, and Zalando to understand best practices and market expectations.

The project encompasses the entire system development lifecycle, from requirements analysis and conceptual design to implementation and testing. Although the main evaluation will be on the database aspects – including schema normalization, integrity, and efficiency – the development of basic frontend and backend functionality is an integral part of the task. These components will serve as a means to demonstrate the database's functionality in practice and to validate that the design choices support the necessary business processes.

The central entities that the database will model include, but are not limited to:

- **Product:** Detailed information about each electronic product offered, including name, description, price, stock status, brand, category, and specifications.
- **Category:** Structuring of products into logical groups for easier navigation and filtering.
- **Brand:** Information about manufacturers and brands.
- **User:** Management of customer information, including account details, authentication, and personal preferences.
- **Order:** Tracking of customer orders from placement to delivery, including order date, total amount, and status.
- **OrderItem:** Details for each product within a specific order, such as quantity and subtotal.
- **Payment:** Handling of payment information related to orders, including method, amount, and transaction status.

Throughout the project, we will emphasize documenting assumptions, design choices, and technical solutions. This will culminate in a comprehensive project report and a fully functional prototype of the ElectroMart platform.

4.1 Application UI and Deployment

The ElectroMart platform boasts a user-friendly interface designed for a seamless shopping experience. Figure 1 shows the homepage, which provides an overview of featured products and categories. Figure 2 displays the user's order history page, allowing customers to track their past purchases.

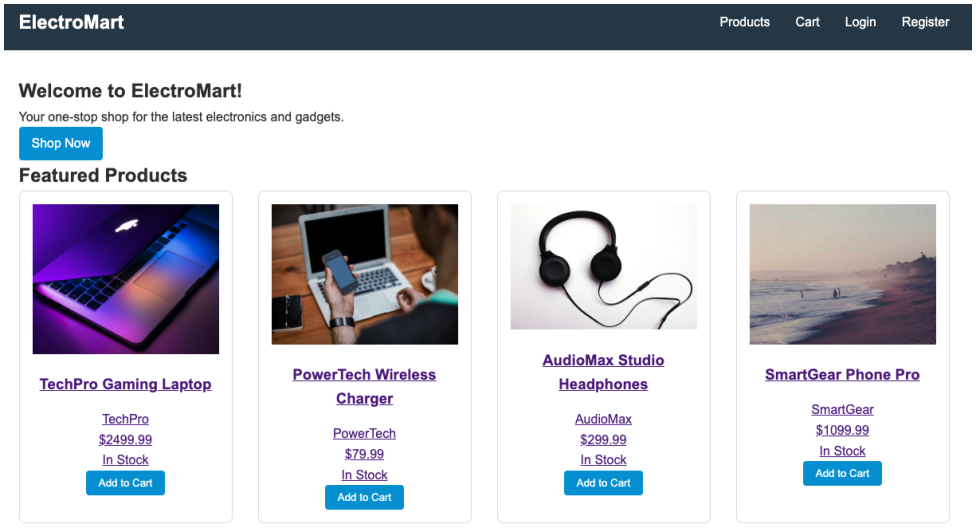

Figure 1: ElectroMart Homepage

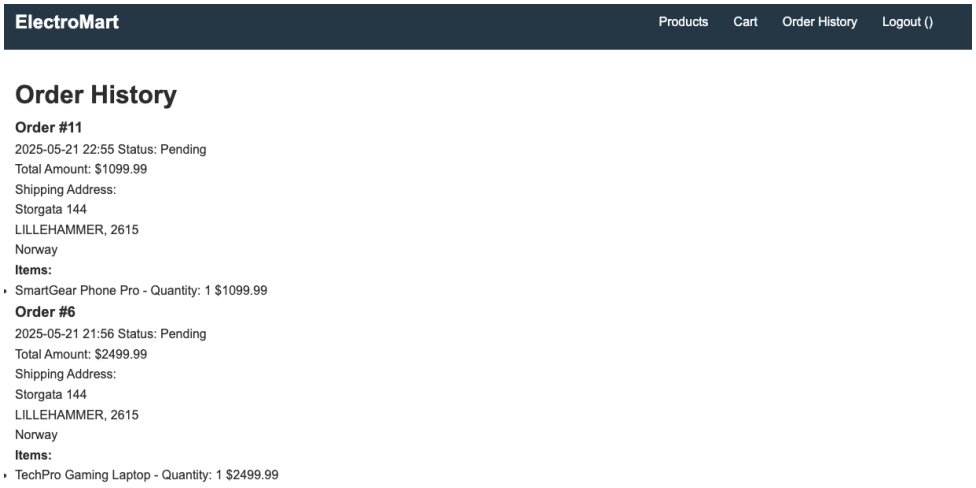

Figure 2: ElectroMart Order History

The application is deployed using Railway.app, a modern platform for infrastructure deployment. We've also (entirely legitimately, we promise) secured the prestigious domain ntno.no for our application — no resemblance to any well-known Norwegian university intended, of course! This setup ensures robust performance and scalability for *ElectroMart*.

5 Database Design

This section describes the design process for the ElectroMart database. The goal has been to develop a robust, efficient, and scalable relational database schema that can handle information related to products, users, orders, payments, and other relevant entities for the e-commerce platform. The design process has followed standard methodologies, including requirements analysis, conceptual modeling with an EER diagram, logical modeling, and a thorough normalization process to ensure data integrity and reduce redundancy.

5.1 Requirements Analysis

A thorough requirements analysis is the foundation for successful database design. For the ElectroMart project, the requirements are primarily derived from the project description, which outlines the functionality of an e-commerce platform for electronics. Below are the key functional and data-related requirements that have guided the database design:

5.1.1 Functional Requirements

The system must support the following core functionalities:

- **Product Catalog:** Users should be able to browse, search for, and view detailed information about a wide range of electronic products.
- **Categorization and Filtering:** Products should be organized into categories and filterable based on attributes such as brand, price, etc.
- **User Registration and Authentication:** Customers should be able to create a user account, log in, and manage their profile information.
- **Shopping Cart:** Users should be able to add products to a shopping cart, change quantities, and remove products before checkout.
- **Order Processing:** The system must handle the entire order process, from the user placing an order, to payment and status updates (e.g., "Processing", "Shipped", "Delivered").
- **Order History:** Registered users should be able to view a history of their past orders.
- **Inventory Management:** The system must keep track of the stock levels for each product and prevent sales of out-of-stock items (or inform about backorders).
- **Payment Processing:** Although a full-fledged payment gateway is outside the main scope of the database focus, the database must be able to store information about payment transactions related to orders.

5.1.2 Data-Related Requirements

Based on functional requirements and the proposed entities, the following data-related requirements have been identified:

Product:

- Each product must have a unique identifier (`ProductID`).
- Necessary attributes include name, detailed description, price, stock quantity, image URL (optional), date added, and last updated.
- Price should be a decimal number with two decimal places of accuracy.
- Stock quantity should be an integer, non-negative.
- A product must belong to one category (`CategoryID` - FK).
- A product must belong to one brand (`BrandID` - FK).

Category:

- Each category must have a unique identifier (`CategoryID`).
- Necessary attributes include name (unique) and an optional description.

Brand:

- Each brand must have a unique identifier (`BrandID`).
- Necessary attributes include name (unique) and an optional description.

User (AppUser):

- Each user must have a unique identifier (`UserID`).
- Necessary attributes for authentication: username (unique) and password hash.
- Personal information: email (unique), first name, last name.
- Address information: address line 1, address line 2 (optional), city, postal code, country.
- Optional contact information: phone number.
- Registration date should be stored.

Order (CustomerOrder):

- Each order must have a unique identifier (`OrderID`).
- An order must be associated with one user (`UserID` - FK).
- Necessary attributes: order date, total amount for the order, order status (e.g., "Pending", "Processing", "Shipped", "Delivered", "Cancelled").
- Delivery address for the order: address line 1, address line 2 (optional), city, postal code, country. This is stored per order to handle cases where the user's default address changes after ordering, or for gifts sent to other addresses.

OrderItem:

- Each order item must have a unique identifier (`OrderItemID`).
- An order item must belong to one order (`OrderID` - FK).
- An order item must refer to one product (`ProductID` - FK).
- Necessary attributes: quantity of units ordered for the product, unit price at the time of order (to handle price changes over time), and subtotal (quantity * unit price).
- The combination of `OrderID` and `ProductID` must be unique per order item to avoid duplication of the same product on the same order (unless there is a business rule that allows it, but typically this is represented by increased quantity).

Payment:

- Each payment transaction must have a unique identifier (`PaymentID`).
- A payment must be associated with one order (`OrderID` - FK, and should be unique to ensure one payment per order, or handle partial payments more complexly if necessary).
- Necessary attributes: payment date, payment method (e.g., "Credit Card", "Vipps"), amount paid, payment status (e.g., "Pending", "Completed", "Failed").
- A transaction ID from the payment provider can be stored (optional, unique if provided).

5.1.3 Business Rules and Integrity Constraints

- Prices and quantities cannot be negative.
- Email addresses and usernames must be unique.
- An order cannot be placed for a product with zero stock, unless the system supports backorders.
- The total amount of an order should match the sum of the subtotals for all order items.
- Once an order is completed or cancelled, certain changes should be restricted.
- Deletion of a user, product, category, or brand should be handled with consideration for referential integrity (e.g., using `ON DELETE SET NULL` or `ON DELETE RESTRICT`, or "soft deletes").

These requirements form the basis for the conceptual and logical database design presented in the following sections. Any assumptions made beyond these explicit requirements will be documented.

5.2 Entity-Relationship Diagram (EER)

The Entity-Relationship Diagram (EER) for the ElectroMart database is presented in Figure 3. This diagram visually represents the entities, their attributes, and the relationships between

them. It serves as a conceptual blueprint for the database structure, detailing how different data elements are interconnected.

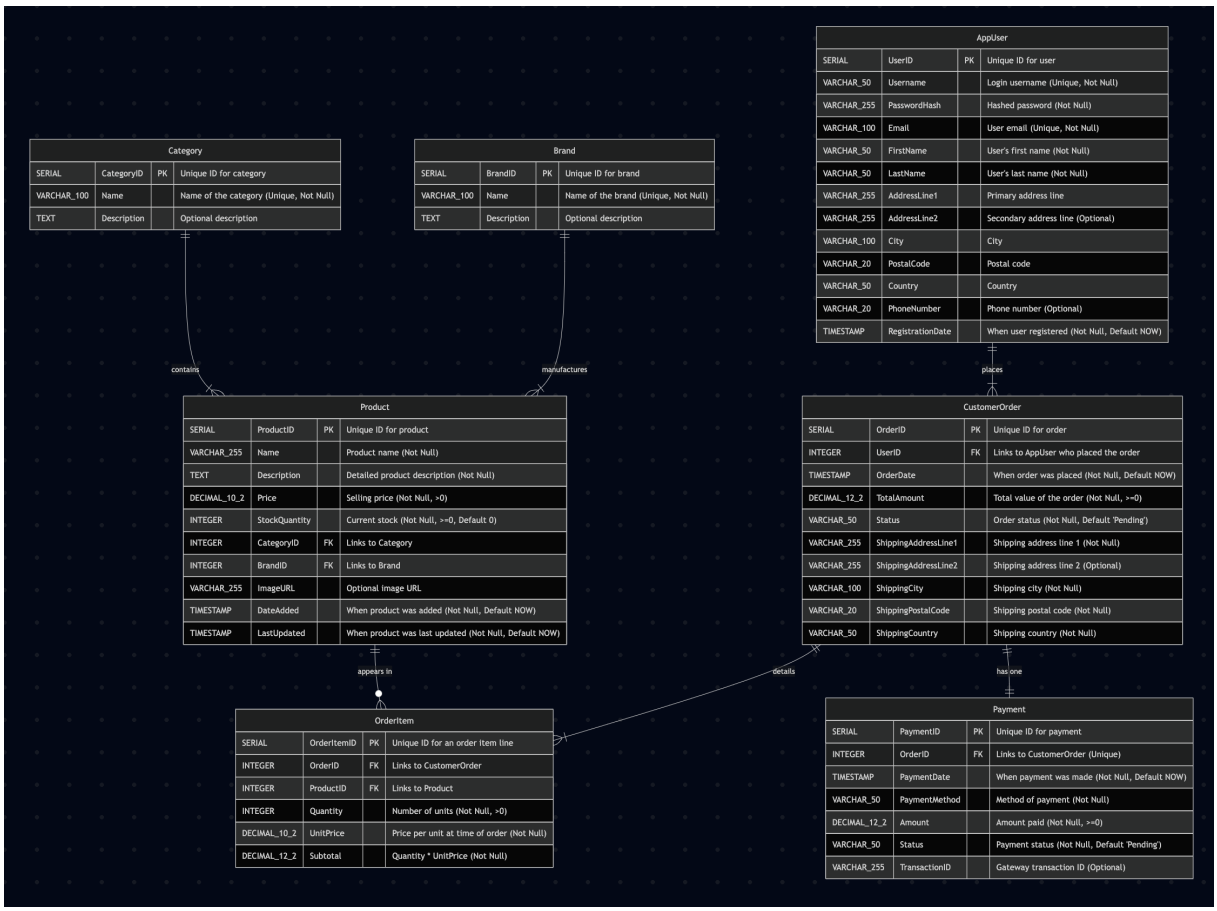

Figure 3: EER Diagram for ElectroMart

5.3 Logical Model

The logical model translates the conceptual EER diagram into a relational schema that can be implemented in a specific database management system. It defines the tables, columns, data types, primary keys, foreign keys, and other constraints. The logical model for the ElectroMart database is depicted in Figure 4.

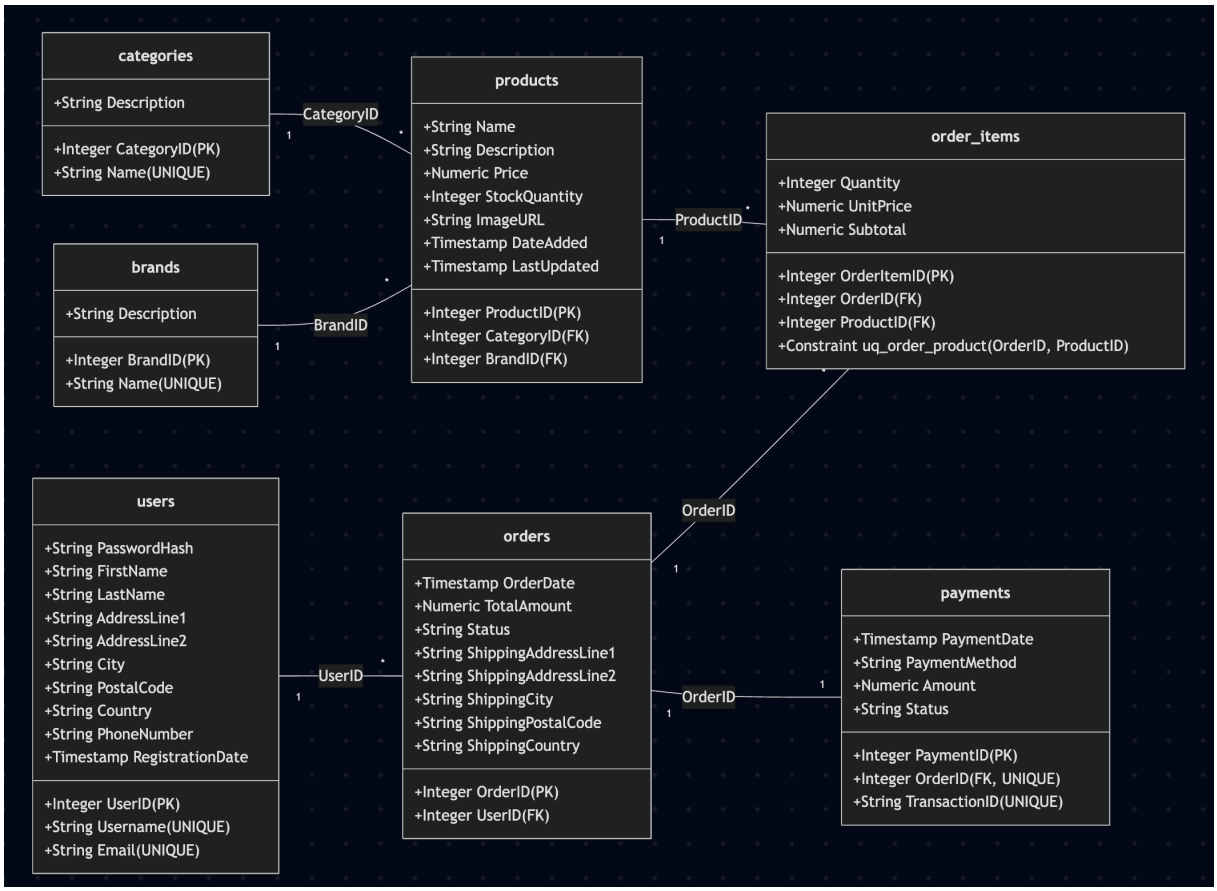

Figure 4: Logical Model for ElectroMart

5.4 Normalization Process

This subsection describes the normalization process applied to the database structure to ensure data integrity, reduce redundancy, and avoid anomalies. We will review First Normal Form (1NF), Second Normal Form (2NF), Third Normal Form (3NF), and Boyce-Codd Normal Form (BCNF). For each normal form, we will show examples of how the tables in the ElectroMart database meet the requirements.

5.4.1 First Normal Form (1NF)

Definition First Normal Form (1NF) requires that:

- All tables have a primary key.
- All columns contain atomic values (no repeating groups or arrays).
- All columns contain values of the same domain.
- The order of data does not matter.

Analysis of Current Schema

Primary Keys All tables in the schema have a primary key:

- `Category`: `CategoryID`
- `Brand`: `BrandID`
- `AppUser`: `UserID`
- `Product`: `ProductID`
- `CustomerOrder`: `OrderID`
- `OrderItem`: `OrderItemID`
- `Payment`: `PaymentID`

Atomic Values All columns contain atomic values:

- No arrays or repeating groups are present.
- Each column stores a single value.
- Complex data is properly decomposed into separate columns (e.g., address is split into multiple columns).

Domain Consistency All columns maintain consistent data types and domains:

- Text fields use appropriate VARCHAR lengths.
- Numeric fields use appropriate DECIMAL or INTEGER types.
- Timestamps use TIMESTAMP WITH TIME ZONE.
- Boolean values are represented using appropriate constraints.

Data Order Independence The schema is order-independent:

- All relationships are maintained through foreign keys.
- No implicit ordering is required for data integrity.
- Primary keys ensure unique identification regardless of order.

Conclusion The current database schema fully satisfies First Normal Form requirements. Each table has a proper primary key, all columns contain atomic values, data types are consistent, and the schema is order-independent.

Sample Data Illustrating 1NF Consider an initial unnormalized table for orders that might look like this before applying 1NF. This hypothetical table contains repeating groups for products.

Table 1: Unnormalized Order Data Example

OrderID	Customer	OrderDate	Products (ProductID, ProductName, Quantity, Price)
101	Malik Dagijevstein	2025-06-13	(P001, Laptop, 1, 1200.00), (P002, Mouse, 1, 25.00)
102	Harun Muratberg	2025-04-03	(P003, Keyboard, 2, 75.00)

To bring this to 1NF, we ensure atomicity by separating the repeating product information into its own table (e.g., 'OrderItems') and ensuring each main table has a primary key. Our schema already reflects this. For example, the 'products' table:

products (1NF Example - simplified) Each cell contains a single, atomic value. 'ProductID' is

Table 2: Simplified Products Table in 1NF

ProductID	Name	CategoryID	Price
P001	Gaming Laptop	C10 (Laptops)	1200.00
P002	Wireless Mouse	C12 (Acc.)	25.00
P003	Mech Keyboard	C12 (Acc.)	75.00

the primary key.

5.4.2 Second Normal Form (2NF)

Definition Second Normal Form (2NF) requires that:

- The database is in First Normal Form (1NF).
- All non-key attributes are fully functionally dependent on the primary key.
- No partial dependencies exist.

Analysis of Current Schema

Functional Dependencies Category Table

- `CategoryID` → `Name`, `Description`
- All attributes are fully dependent on the primary key.

Brand Table

- `BrandID` → `Name`, `Description`
- All attributes are fully dependent on the primary key.

AppUser Table

- `UserID` → `Username`, `PasswordHash`, `Email`, `FirstName`, `LastName`, `AddressLine1`, `AddressLine2`, `City`, `PostalCode`, `Country`, `PhoneNumber`, `RegistrationDate`
- All attributes are fully dependent on the primary key.

Product Table

- `ProductID` → `Name`, `Description`, `Price`, `StockQuantity`, `CategoryID`, `BrandID`, `ImageURL`, `DateAdded`, `LastUpdated`
- All attributes are fully dependent on the primary key.
- `CategoryID` and `BrandID` are foreign keys, not partial dependencies.

CustomerOrder Table

- `OrderID` → `UserID`, `OrderDate`, `TotalAmount`, `Status`, `ShippingAddressLine1`, `ShippingAddressLine2`, `ShippingCity`, `ShippingPostalCode`, `ShippingCountry`
- All attributes are fully dependent on the primary key.
- `UserID` is a foreign key, not a partial dependency.

OrderItem Table

- `OrderItemID` → `OrderID`, `ProductID`, `Quantity`, `UnitPrice`, `Subtotal`
- All attributes are fully dependent on the primary key.
- `OrderID` and `ProductID` are foreign keys, not partial dependencies.

Payment Table

- `PaymentID` → `OrderID`, `PaymentDate`, `PaymentMethod`, `Amount`, `Status`, `TransactionID`
- All attributes are fully dependent on the primary key.
- `OrderID` is a foreign key, not a partial dependency.

Partial Dependencies Check

- No composite primary keys exist in the schema.
- All tables use single-column primary keys.
- Therefore, no partial dependencies are possible.

Conclusion The current database schema fully satisfies Second Normal Form requirements. The schema is in 1NF, and all non-key attributes are fully functionally dependent on their respective primary keys. There are no partial dependencies present in the schema.

Sample Data Illustrating 2NF Imagine we had an ‘OrderItemDetails‘ table in 1NF with a composite primary key ‘(OrderID, ProductID)‘ and also included ‘ProductName‘ and ‘Product-

PriceAtSale‘.

OrderItemDetails (Hypothetical - Not 2NF if ProductName depends only on ProductID)

Table 3: Hypothetical OrderItemDetails Table (Not 2NF Example)

OrderID	ProductID	ProductName	ProductPriceAtSale	Quantity
101	P001	Gaming Laptop	1200.00	1
101	P002	Wireless Mouse	25.00	1
102	P001	Gaming Laptop	1200.00	2

Here, ‘ProductName‘ is partially dependent on ‘ProductID‘ alone, not the whole composite key ‘(OrderID, ProductID)‘. This violates 2NF. To achieve 2NF, ‘ProductName‘ would be moved to the ‘Products‘ table, where it is fully dependent on ‘ProductID‘ (the primary key of ‘Products‘). Our current ‘order_items‘ table avoids this by storing ‘UnitPrice‘ (which is specific to that order-product instance) and linking to ‘ProductID‘ for other product details.

order_items (2NF Compliant - from our schema) All non-key attributes (‘Quantity‘, ‘UnitPrice‘,

Table 4: Order Items Table in 2NF (from schema)

OrderItemID	OrderID	ProductID	Quantity	UnitPrice	Subtotal
OI001	101	P001	1	1200.00	1200.00
OI002	101	P002	1	25.00	25.00
OI003	102	P001	2	1200.00	2400.00

‘Subtotal‘) are fully dependent on the primary key ‘OrderItemID‘ (or the candidate key ‘(OrderID, ProductID)‘).

5.4.3 Third Normal Form (3NF)

Definition Third Normal Form (3NF) requires that:

- The database is in Second Normal Form (2NF).
- All attributes are directly dependent on the primary key.
- No transitive dependencies exist.

Analysis of Current Schema

Transitive Dependencies Check Category Table

- No transitive dependencies.

- All attributes directly depend on `CategoryID`.

Brand Table

- No transitive dependencies.
- All attributes directly depend on `BrandID`.

AppUser Table

- No transitive dependencies.
- All attributes directly depend on `UserID`.
- Address components are independent attributes, not transitively dependent.

Product Table

- No transitive dependencies.
- All attributes directly depend on `ProductID`.
- `CategoryID` and `BrandID` are foreign keys, not transitive dependencies.

CustomerOrder Table

- No transitive dependencies.
- All attributes directly depend on `OrderID`.
- Shipping address components are independent attributes.
- `UserID` is a foreign key, not a transitive dependency.

OrderItem Table

- No transitive dependencies.
- All attributes directly depend on `OrderItemID`.
- `Subtotal` is calculated from `Quantity` and `UnitPrice`, but this is a derived value, not a transitive dependency.
- `OrderID` and `ProductID` are foreign keys, not transitive dependencies.

Payment Table

- No transitive dependencies.
- All attributes directly depend on `PaymentID`.
- `OrderID` is a foreign key, not a transitive dependency.

Potential Transitive Dependencies Addressed

1. **Address Information**
   - In `AppUser` and `CustomerOrder`, address components are kept as separate columns.
   - This prevents transitive dependencies that would occur if address was stored as a

single field.

2. **Order Calculations**
   - `OrderItem.Subtotal` is stored but calculated from `Quantity` and `UnitPrice`.
   - This is a derived value, not a transitive dependency.
   - The calculation is maintained at the application level.
3. **User Information**
   - All user attributes directly depend on `UserID`.
   - No transitive dependencies between user attributes.

Conclusion The current database schema fully satisfies Third Normal Form requirements. The schema is in 2NF, and there are no transitive dependencies present. All attributes are directly dependent on their respective primary keys, and the schema properly handles derived values and foreign key relationships.

Sample Data Illustrating 3NF Consider a hypothetical 'ProductDetails' table (already in 2NF) where 'ProductID' is the key:

ProductDetails (Hypothetical - Not 3NF if BrandName depends on BrandID, which depends on ProductID)

Table 5: Hypothetical ProductDetails Table (Not 3NF Example)

ProductID	ProductName	Price	BrandID	BrandName
P001	Gaming Laptop	1200.00	B01	ElectroBrand
P002	Wireless Mouse	25.00	B02	TechGear
P003	Monitor	300.00	B01	ElectroBrand

In this example, 'ProductID' → 'BrandID', and 'BrandID' → 'BrandName'. This means 'BrandName' is transitively dependent on 'ProductID' via 'BrandID'. This violates 3NF. To achieve 3NF, 'BrandName' would be moved to a separate 'Brands' table where 'BrandID' is the primary key.

Our schema correctly implements this:

products (3NF Compliant - from our schema, simplified)

Table 6: Simplified Products Table in 3NF (from schema)

ProductID	Name	Price	BrandID
P001	Gaming Laptop	1200.00	B01
P002	Wireless Mouse	25.00	B02

brands (3NF Compliant - from our schema, simplified) Now, all non-key attributes in

Table 7: Simplified Brands Table in 3NF (from schema)

BrandID	Name
B01	ElectroBrand
B02	TechGear

'products' are directly dependent on 'ProductID', and all non-key attributes in 'brands' are directly dependent on 'BrandID'. No transitive dependencies involving non-key attributes exist.

5.4.4 Boyce-Codd Normal Form (BCNF)

Definition Boyce-Codd Normal Form (BCNF) is a higher level of normalization than 3NF. A table is in BCNF if and only if for every one of its non-trivial functional dependencies X → Y, X is a superkey—that is, X is either a candidate key or a superset of a candidate key.

Analysis of Current Schema To verify BCNF, we need to examine all functional dependencies in each table.

`categories` Table

- **Primary Key**: `CategoryID`
- **Functional Dependencies**:
  - `CategoryID` → `Name`, `Description`
- **Candidate Keys**: `CategoryID`, `Name` (since Name is UNIQUE)
- **Analysis**:
  - For `CategoryID` → `Name`, `Description`: `CategoryID` is a candidate key.
  - For `Name` → `CategoryID`, `Description`: `Name` is a candidate key.
- **Conclusion**: The `categories` table is in BCNF.

`brands` Table

- **Primary Key**: `BrandID`
- **Functional Dependencies**:
  - `BrandID` → `Name`, `Description`
- **Candidate Keys**: `BrandID`, `Name` (since Name is UNIQUE)
- **Analysis**:
  - For `BrandID` → `Name`, `Description`: `BrandID` is a candidate key.

  - For `Name` → `BrandID`, `Description`: `Name` is a candidate key.
- **Conclusion**: The `brands` table is in BCNF.

`users` Table

- **Primary Key**: `UserID`
- **Functional Dependencies**:
  - `UserID` → `Username`, `PasswordHash`, `Email`, `FirstName`, `LastName`, `AddressLine1`, `AddressLine2`, `City`, `PostalCode`, `Country`, `PhoneNumber`, `RegistrationDate`
- **Candidate Keys**: `UserID`, `Username`, `Email` (since Username and Email are UNIQUE)
- **Analysis**:
  - For `UserID` → All other attributes: `UserID` is a candidate key.
  - For `Username` → All other attributes: `Username` is a candidate key.
  - For `Email` → All other attributes: `Email` is a candidate key.
- **Conclusion**: The `users` table is in BCNF.

`products` Table

- **Primary Key**: `ProductID`
- **Functional Dependencies**:
  - `ProductID` → `Name`, `Description`, `Price`, `StockQuantity`, `CategoryID`, `BrandID`, `ImageURL`, `DateAdded`, `LastUpdated`
- **Candidate Keys**: `ProductID`
- **Analysis**:
  - For `ProductID` → All other attributes: `ProductID` is a candidate key.
- **Conclusion**: The `products` table is in BCNF.

`orders` Table

- **Primary Key**: `OrderID`
- **Functional Dependencies**:
  - `OrderID` → `UserID`, `OrderDate`, `TotalAmount`, `Status`, `ShippingAddressLine1`, `ShippingAddressLine2`, `ShippingCity`, `ShippingPostalCode`, `ShippingCountry`
- **Candidate Keys**: `OrderID`
- **Analysis**:
  - For `OrderID` → All other attributes: `OrderID` is a candidate key.
- **Conclusion**: The `orders` table is in BCNF.

`order_items` Table

- **Primary Key**: `OrderItemID`
- **Composite Key**: (`OrderID`, `ProductID`) is a UNIQUE constraint, making it a candidate key.
- **Functional Dependencies**:
  - `OrderItemID` → `OrderID`, `ProductID`, `Quantity`, `UnitPrice`, `Subtotal`
  - (`OrderID`, `ProductID`) → `OrderItemID`, `Quantity`, `UnitPrice`, `Subtotal`
- **Candidate Keys**: `OrderItemID`, (`OrderID`, `ProductID`)
- **Analysis**:
  - For `OrderItemID` → All other attributes: `OrderItemID` is a candidate key.
  - For (`OrderID`, `ProductID`) → `OrderItemID`, `Quantity`, `UnitPrice`, `Subtotal`: (`OrderID`, `ProductID`) is a candidate key.
- **Conclusion**: The `order_items` table is in BCNF.

`payments` Table

- **Primary Key**: `PaymentID`
- **Functional Dependencies**:
  - `PaymentID` → `OrderID`, `PaymentDate`, `PaymentMethod`, `Amount`, `Status`, `TransactionID`
  - `OrderID` → `PaymentID`, `PaymentDate`, `PaymentMethod`, `Amount`, `Status`, `TransactionID` (since OrderID is UNIQUE in payments)
  - `TransactionID` → `PaymentID`, `OrderID`, `PaymentDate`, `PaymentMethod`, `Amount`, `Status` (since TransactionID is UNIQUE)
- **Candidate Keys**: `PaymentID`, `OrderID`, `TransactionID`
- **Analysis**:
  - For each candidate key, it functionally determines all other attributes in the table.
- **Conclusion**: The `payments` table is in BCNF.

Overall Conclusion All tables in the ElectroMart database schema are in Boyce-Codd Normal Form (BCNF). For every non-trivial functional dependency X → Y, X is a superkey.

Sample Data Illustrating BCNF BCNF is a stricter version of 3NF. A table is in BCNF if for every non-trivial functional dependency X → Y, X is a superkey. Most 3NF tables are also in BCNF. Violations can occur with multiple overlapping candidate keys.

Consider a hypothetical (and slightly contrived for illustration) table ‘StudentCourseAdvisor‘:

StudentCourseAdvisor (Hypothetical - Potentially not BCNF) Assume functional dependencies:

- (`Student`, `Course`) → `Advisor` (A student has one advisor per course)

- `Advisor` → `Course` (An advisor advises for only one specific course - this is the contrived part for the example)

Candidate keys could be: (`Student`, `Course`) and (`Student`, `Advisor`).

Sample Data:

Table 8: Hypothetical StudentCourseAdvisor Table (Not BCNF Example)

Student	Course	Advisor
Alice	CS101	Prof. Smith
Bob	CS101	Prof. Smith
Alice	MA202	Prof. Doe

The dependency `Advisor` → `Course` violates BCNF because `Advisor` is not a superkey (it doesn't determine `Student`). To normalize to BCNF, we would decompose into:

StudentAdvisor (BCNF)

Table 9: Decomposed StudentAdvisor Table (BCNF)

Student	Advisor
Alice	Prof. Smith
Bob	Prof. Smith
Alice	Prof. Doe

AdvisorCourse (BCNF)

Table 10: Decomposed AdvisorCourse Table (BCNF)

Advisor	Course
Prof. Smith	CS101
Prof. Doe	MA202

Our ElectroMart schema, as analyzed, meets BCNF because for all identified functional dependencies, the determinant (LHS) is a superkey. For instance, in the `payments` table:

- `PaymentID` (PK) → all other attributes. `PaymentID` is a superkey.
- `OrderID` (Unique, Candidate Key) → all other attributes. `OrderID` is a superkey.
- `TransactionID` (Unique, Candidate Key) → all other attributes. `TransactionID` is a superkey.

This structure ensures BCNF compliance across all tables.

5.5 Database Schema (SQL)

The complete SQL schema for creating the ElectroMart database, including table definitions, constraints, and comments, is provided below. This schema is designed for PostgreSQL.

```sql
DROP TABLE IF EXISTS payments CASCADE;
DROP TABLE IF EXISTS order_items CASCADE;
DROP TABLE IF EXISTS orders CASCADE;
DROP TABLE IF EXISTS products CASCADE;
DROP TABLE IF EXISTS users CASCADE;
DROP TABLE IF EXISTS brands CASCADE;
DROP TABLE IF EXISTS categories CASCADE;

CREATE TABLE categories (
    CategoryID SERIAL PRIMARY KEY,
    Name VARCHAR(100) NOT NULL UNIQUE,
    Description TEXT
);
COMMENT ON TABLE categories IS 'Represents different categories of
    ↪ electronic products.';
COMMENT ON COLUMN categories.CategoryID IS 'Unique identifier for the
    ↪ category (e.g., 1, 2, 3...).';
COMMENT ON COLUMN categories.Name IS 'Name of the category (e.g., "
    ↪ Smartphones", "Laptops"). Must be unique.';
COMMENT ON COLUMN categories.Description IS 'A brief description of the
    ↪ category.';

CREATE TABLE brands (
    BrandID SERIAL PRIMARY KEY,
    Name VARCHAR(100) NOT NULL UNIQUE,
    Description TEXT
);
COMMENT ON TABLE brands IS 'Represents the brands or manufacturers of
    ↪ electronic products.';
COMMENT ON COLUMN brands.BrandID IS 'Unique identifier for the brand (e.g.,
    ↪  1, 2, 3...).';
COMMENT ON COLUMN brands.Name IS 'Name of the brand (e.g., "Apple", "
    ↪ Samsung"). Must be unique.';
COMMENT ON COLUMN brands.Description IS 'A brief description of the brand.'
    ↪ ;

CREATE TABLE users (
    UserID SERIAL PRIMARY KEY,
    Username VARCHAR(50) NOT NULL UNIQUE,
    PasswordHash VARCHAR(255) NOT NULL,
    Email VARCHAR(100) NOT NULL UNIQUE,
    FirstName VARCHAR(50) NOT NULL,
```

```
    LastName VARCHAR(50) NOT NULL,
    AddressLine1 VARCHAR(255),
    AddressLine2 VARCHAR(255),
    City VARCHAR(100),
    PostalCode VARCHAR(20),
    Country VARCHAR(50),
    PhoneNumber VARCHAR(20),
    RegistrationDate TIMESTAMP WITH TIME ZONE DEFAULT CURRENT_TIMESTAMP
);
COMMENT ON TABLE users IS 'Represents users (customers) of the website.';
COMMENT ON COLUMN users.UserID IS 'Unique identifier for the user.';
COMMENT ON COLUMN users.Username IS 'Unique username for login.';
COMMENT ON COLUMN users.PasswordHash IS 'Hashed version of the users
    ↪ password.';
COMMENT ON COLUMN users.Email IS 'Users email address (must be unique).';
COMMENT ON COLUMN users.RegistrationDate IS 'Timestamp when the user
    ↪ registered.';

CREATE TABLE products (
    ProductID SERIAL PRIMARY KEY,
    Name VARCHAR(255) NOT NULL,
    Description TEXT NOT NULL,
    Price NUMERIC(10, 2) NOT NULL,
    StockQuantity INTEGER NOT NULL DEFAULT 0,
    CategoryID INTEGER NOT NULL,
    BrandID INTEGER NOT NULL,
    ImageURL VARCHAR(255),
    DateAdded TIMESTAMP WITH TIME ZONE DEFAULT CURRENT_TIMESTAMP,
    LastUpdated TIMESTAMP WITH TIME ZONE DEFAULT CURRENT_TIMESTAMP,
    CONSTRAINT fk_category FOREIGN KEY (CategoryID) REFERENCES categories(
        ↪ CategoryID),
    CONSTRAINT fk_brand FOREIGN KEY (BrandID) REFERENCES brands(BrandID)
);
COMMENT ON TABLE products IS 'Represents the electronic products available
    ↪ for sale.';
COMMENT ON COLUMN products.ProductID IS 'Unique identifier for the product.
    ↪ ';
COMMENT ON COLUMN products.Price IS 'Selling price of the product.';
COMMENT ON COLUMN products.StockQuantity IS 'Number of units currently in
    ↪ stock.';
COMMENT ON COLUMN products.CategoryID IS 'Foreign key linking to the
    ↪ categories table.';
COMMENT ON COLUMN products.BrandID IS 'Foreign key linking to the brands
    ↪ table.';
COMMENT ON COLUMN products.DateAdded IS 'Timestamp when the product was
    ↪ added.';
COMMENT ON COLUMN products.LastUpdated IS 'Timestamp when the product was
```

```
    ↪ last updated.';

CREATE TABLE orders (
    OrderID SERIAL PRIMARY KEY,
    UserID INTEGER NOT NULL,
    OrderDate TIMESTAMP WITH TIME ZONE DEFAULT CURRENT_TIMESTAMP,
    TotalAmount NUMERIC(12, 2) NOT NULL,
    Status VARCHAR(50) NOT NULL DEFAULT 'Pending',
    ShippingAddressLine1 VARCHAR(255) NOT NULL,
    ShippingAddressLine2 VARCHAR(255),
    ShippingCity VARCHAR(100) NOT NULL,
    ShippingPostalCode VARCHAR(20) NOT NULL,
    ShippingCountry VARCHAR(50) NOT NULL,
    CONSTRAINT fk_user FOREIGN KEY (UserID) REFERENCES users(UserID)
);
COMMENT ON TABLE orders IS 'Represents individual orders placed by users.';
COMMENT ON COLUMN orders.OrderID IS 'Unique identifier for the order.';
COMMENT ON COLUMN orders.UserID IS 'Foreign key linking to the user who
    ↪ placed the order.';
COMMENT ON COLUMN orders.TotalAmount IS 'Total monetary value of the order.
    ↪ ';
COMMENT ON COLUMN orders.Status IS 'Current status of the order (e.g.,
    ↪ Pending, Shipped).';

CREATE TABLE order_items (
    OrderItemID SERIAL PRIMARY KEY,
    OrderID INTEGER NOT NULL,
    ProductID INTEGER NOT NULL,
    Quantity INTEGER NOT NULL,
    UnitPrice NUMERIC(10, 2) NOT NULL,
    Subtotal NUMERIC(12, 2) NOT NULL,
    CONSTRAINT fk_order FOREIGN KEY (OrderID) REFERENCES orders(OrderID) ON
        ↪  DELETE CASCADE,
    CONSTRAINT fk_product FOREIGN KEY (ProductID) REFERENCES products(
        ↪ ProductID),
    CONSTRAINT uq_order_product UNIQUE (OrderID, ProductID)
);
COMMENT ON TABLE order_items IS 'Represents the line items included in each
    ↪  order.';
COMMENT ON COLUMN order_items.OrderItemID IS 'Unique identifier for the
    ↪ order item entry.';
COMMENT ON COLUMN order_items.OrderID IS 'Foreign key linking to the orders
    ↪  table.';
COMMENT ON COLUMN order_items.ProductID IS 'Foreign key linking to the
    ↪ products table.';
COMMENT ON COLUMN order_items.Quantity IS 'Number of units of this product
    ↪ in the order.';
```

```sql
COMMENT ON COLUMN order_items.UnitPrice IS 'Price per unit at the time of order.';
COMMENT ON COLUMN order_items.Subtotal IS 'Calculated as Quantity * UnitPrice.';

CREATE TABLE payments (
    PaymentID SERIAL PRIMARY KEY,
    OrderID INTEGER NOT NULL UNIQUE,
    PaymentDate TIMESTAMP WITH TIME ZONE DEFAULT CURRENT_TIMESTAMP,
    PaymentMethod VARCHAR(50) NOT NULL,
    Amount NUMERIC(12, 2) NOT NULL,
    Status VARCHAR(50) NOT NULL DEFAULT 'Pending',
    TransactionID VARCHAR(255) UNIQUE,
    CONSTRAINT fk_payment_order FOREIGN KEY (OrderID) REFERENCES orders(OrderID) ON DELETE CASCADE
);
COMMENT ON TABLE payments IS 'Represents payments made for orders.';
COMMENT ON COLUMN payments.PaymentID IS 'Unique identifier for the payment.';
COMMENT ON COLUMN payments.OrderID IS 'Foreign key linking to the orders table.';
COMMENT ON COLUMN payments.PaymentMethod IS 'Method used for payment.';
COMMENT ON COLUMN payments.Amount IS 'Amount paid.';
COMMENT ON COLUMN payments.Status IS 'Status of the payment (e.g., Pending, Completed).';
COMMENT ON COLUMN payments.TransactionID IS 'Unique transaction identifier from the payment gateway, if applicable.';

-- Create indexes for better performance
CREATE INDEX idx_products_category ON products(CategoryID);
CREATE INDEX idx_products_brand ON products(BrandID);
CREATE INDEX idx_orders_user ON orders(UserID);
CREATE INDEX idx_order_items_order ON order_items(OrderID);
CREATE INDEX idx_order_items_product ON order_items(ProductID);
CREATE INDEX idx_payments_order ON payments(OrderID);
```

Code Listing 1: ElectroMart Database Schema

5.6 Database Hosting and Overview

The ElectroMart database is hosted on Railway.app, providing a flexible and scalable environment. Figure 5 shows an overview of the project deployment on Railway, while Figure 6 illustrates the PostgreSQL database structure within the Railway environment.

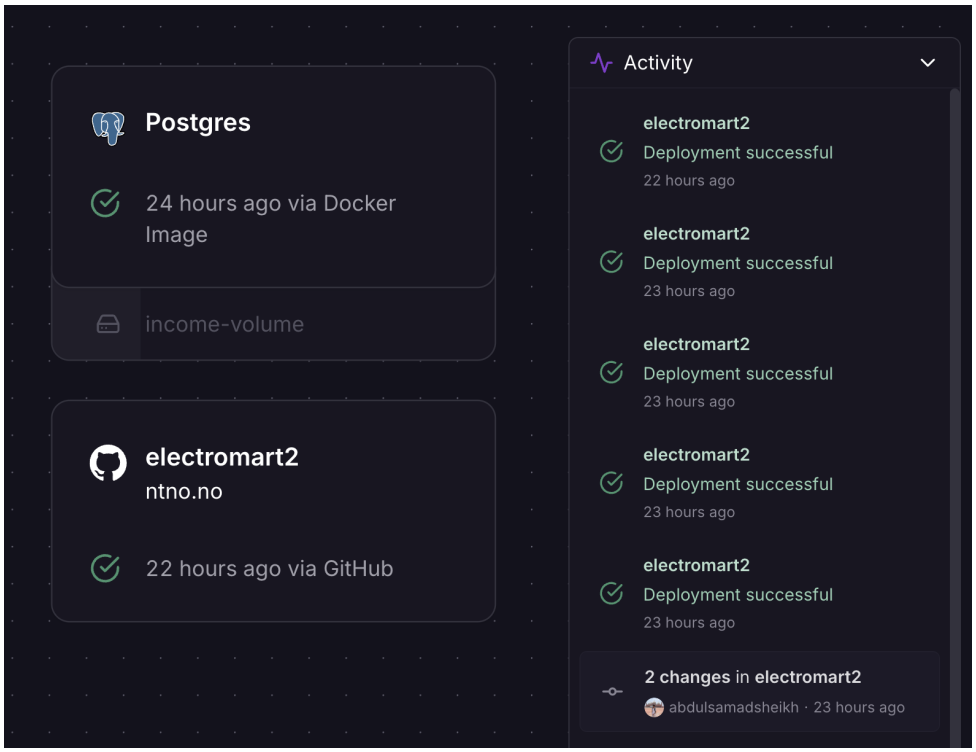

Figure 5: Project Overview on Railway.app

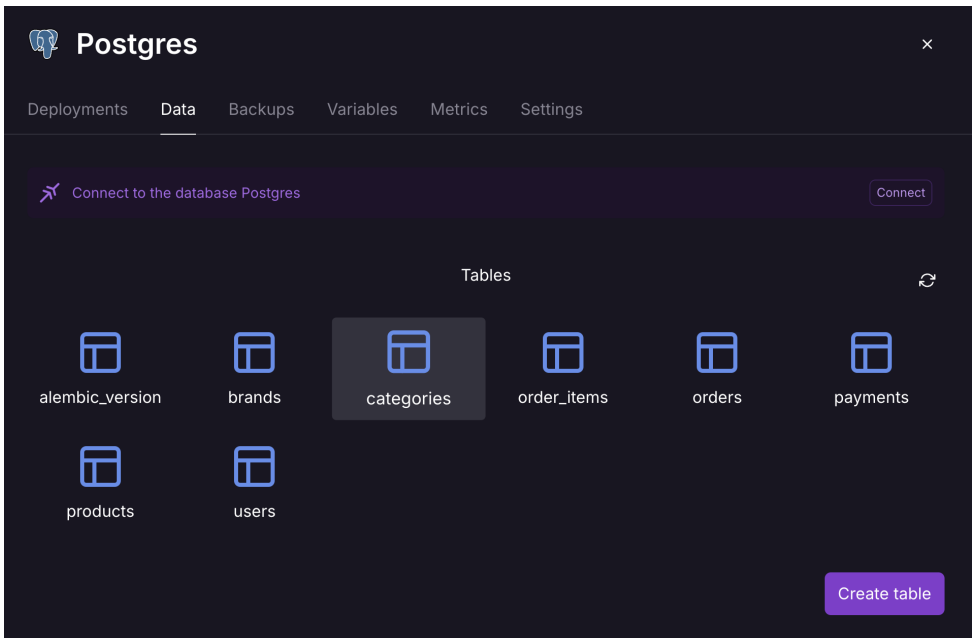

Figure 6: PostgreSQL Database Overview on Railway.app

6 Performance and Security Considerations

Ensuring adequate performance and security is critical for an e-commerce platform like ElectroMart. This section outlines some of the measures and considerations incorporated into the database design and overall system architecture.

6.1 Performance Optimization

Database performance is key to a responsive user experience. The following strategies have been employed or considered:

- **Indexing:** As shown in the SQL schema (Listing 1), indexes have been strategically created on foreign key columns and other frequently queried columns. For example:
  - `idx_products_category` on `products(CategoryID)`
  - `idx_products_brand` on `products(BrandID)`
  - `idx_orders_user` on `orders(UserID)`
  - `idx_order_items_order` on `order_items(OrderID)`
  - `idx_order_items_product` on `order_items(ProductID)`
  - `idx_payments_order` on `payments(OrderID)`

  These indexes significantly speed up join operations and filtering, which are common in e-commerce queries (e.g., retrieving all products in a category, or all orders for a user).
- **Normalization:** The normalization process up to BCNF (detailed in Section 5.4) reduces data redundancy. While highly normalized schemas can sometimes lead to more joins, they also ensure data integrity and can improve write performance by avoiding updating redundant data.
- **Data Types:** Appropriate data types have been chosen to optimize storage and performance (e.g., using ‘INTEGER‘ for IDs, ‘NUMERIC(10,2)‘ for prices, ‘VARCHAR‘ with reasonable length limits).
- **Query Optimization:** While not part of the schema itself, the application interacting with the database will be designed to use efficient queries, avoiding ‘SELECT *‘ where possible and utilizing ‘WHERE‘ clauses effectively.
- **Connection Pooling:** The backend application will use connection pooling to manage database connections efficiently, reducing the overhead of establishing new connections for each request.
- **Scalable Hosting:** The database is hosted on Railway.app, which offers scalable infrastructure, allowing resources to be adjusted based on demand.

6.2 Security Measures

Security is paramount, especially when handling user data and payment information. The following security measures are integral to the ElectroMart platform:

- **Password Hashing:** User passwords, as indicated by the ‘PasswordHash‘ column in the ‘users‘ table, are stored using strong, one-way hashing algorithms (e.g., bcrypt or Argon2) with unique salts for each user. Plaintext passwords are never stored.

- **Input Validation and Sanitization:** All user inputs, both on the frontend and backend, will be rigorously validated and sanitized to prevent common web vulnerabilities such as SQL Injection (SQLi) and Cross-Site Scripting (XSS). Parameterized queries (prepared statements) will be used for all database interactions to prevent SQLi.
- **HTTPS/TLS Encryption:** Communication between the client (browser) and the server, as well as between the application server and the database server (if over a network), will be encrypted using HTTPS/TLS to protect data in transit.
- **Principle of Least Privilege:** The database user account utilized by the application will be granted only the necessary permissions required for its operations (e.g., SELECT, INSERT, UPDATE, DELETE on specific tables), rather than full superuser privileges.
- **Regular Backups:** The database hosted on Railway.app will have regular automated backups configured to ensure data can be recovered in case of hardware failure or other disasters.
- **Dependency Management:** Software dependencies (backend libraries, frontend packages) will be kept up-to-date and monitored for known vulnerabilities.
- **Data Minimization:** Only necessary user data is collected and stored, adhering to data privacy principles.

While this project focuses primarily on database design, these broader security and performance considerations are crucial for a production-ready system.