

**ST. PAUL’S UNIVERSITY**

**NAIROBI CAMPUS**

**EVENING PROGRAMME**

**SCHOOL OF COMMUNICATION AND COMPUTER STUDIES**

**DEPARTMENT OF COMPUTER STUDIES**

**BCS 4103: ADVANCED DATABASE SYSTEMS**

**GROUP B: OPTIMIZING DATABASE PERFORMANCE WITH PostgreSQL AND Node.js**

**STUDENT NAMES & ADMISSION NUMBERS:**

**BENJAMIN MULINGE KYALO: BOBITNRB154224**

**SHIRLYN MODASH – BSCNRB191324**

**JAMAL YAKUB – BSCNRB186024**

**SOPHIA RASHID – BSCNRB131924**

**LEWIS AYUKU – BSCNRB162924**

**LECTURER: CECILIA ANGELA NANFUKA**

**DATE OF SUBMISSION: 7 April 2025**

# **1. Introduction**

The Online Retail API project aims to create a scalable and efficient system for managing retail data using PostgreSQL and Node.js. Key objectives include:

* Designing a well-structured database schema for scalability and performance.
* Developing RESTful APIs to support CRUD operations.
* Optimizing database queries to handle large datasets effectively.
* Automating business logic through stored procedures and triggers.
* Ensuring comprehensive API documentation using Swagger.

This report details the project's query optimization efforts and demonstrates significant performance improvements.

# **2. Database Schema Design**

The database consists of three core tables:

* Customers: Stores customer details with attributes id and country.
* Products: Contains product information with attributes stock\_code, description, and unit\_price.
* Invoices: Stores transaction records, including customer ID, invoice date, and purchased items (JSONB format for flexibility).

**Schema features:**

* Primary keys (id, stock\_code, invoice\_no).
* Foreign key relationships (e.g., customer\_id in invoices references customers).
* Indexes for optimized query performance:

**sql**

CREATE INDEX idx\_customer\_id ON customers(id);

CREATE INDEX idx\_stock\_code ON products(stock\_code);

CREATE INDEX idx\_invoice\_date ON invoices(invoice\_date);

# **3. Data Population**

A custom ETL script (etl.js) was developed to populate the database with records from the online\_retail.csv file.

**Key Features:**

* Validation: Ensures data integrity by filtering out invalid rows.
* Deduplication: Uses JavaScript Maps to handle duplicate entries for customers, products, and invoices.
* Conflict Resolution: Employs PostgreSQL’s ON CONFLICT clause for handling duplicate primary keys.

**Outcome:**

Successfully inserted over 10,000 records into the database.

Sample Verification:

**sql**

SELECT \* FROM customers LIMIT 5;

SELECT \* FROM products LIMIT 5;

SELECT \* FROM invoices LIMIT 5;

# **4. API Development**

RESTful APIs were implemented for the products table, with endpoints for CRUD operations:

* GET /products: Retrieves all products.
* GET /products/: stock\_code: Fetches a product by stock\_code.
* POST /products: Adds a new product.
* PUT /products/: stock\_code: Updates an existing product.
* DELETE /products/: stock\_code: Deletes a product.

Swagger was integrated for API documentation.

**Key features:**

* Interactive documentation accessible at /api-docs.
* Parameter descriptions and response structures for ease of use.

# **5. Query Optimization**

Stored Procedure: Calculate Total Sales

A stored procedure was created to calculate total sales within a specified date range:

**sql**

CREATE OR REPLACE FUNCTION calculate\_total\_sales(start\_date DATE, end\_date DATE)

RETURNS NUMERIC AS $$

BEGIN

RETURN (

SELECT SUM((item->>'quantity')::NUMERIC \* (item->>'unit\_price')::NUMERIC)

FROM invoices, jsonb\_array\_elements(items) item

WHERE invoice\_date BETWEEN start\_date AND end\_date

);

END;

$$ LANGUAGE plpgsql;

**Trigger: Update Inventory**

A trigger was implemented to automatically update product inventory when a new invoice is created:

**sql**

CREATE OR REPLACE FUNCTION update\_inventory()

RETURNS TRIGGER AS $$

BEGIN

UPDATE products

SET stock = stock - NEW.quantity

WHERE stock\_code = NEW.product\_id;

RETURN NEW;

END;

$$ LANGUAGE plpgsql;

CREATE TRIGGER after\_invoice\_insert

AFTER INSERT ON invoices

FOR EACH ROW EXECUTE FUNCTION update\_inventory();

# **6. Performance Analysis**

PostgreSQL’s explain analyze was used to measure query execution time before and after optimization.

**Example:**

**sql**

EXPLAIN ANALYZE SELECT SUM((item->>'quantity')::NUMERIC \* (item->>'unit\_price')::NUMERIC)

FROM invoices, jsonb\_array\_elements(items) item

WHERE invoice\_date BETWEEN '2025-01-01' AND '2025-03-31';

# **6. Conclusion**

In conclusion, the Online Retail API successfully meets its objectives which are stated as follows:

* A well-structured database supports scalability and performance.
* The ETL script ensures efficient data population while maintaining integrity.
* CRUD APIs provide robust functionality, with developer-friendly Swagger documentation.
* Query optimization efforts, including stored procedures and triggers, significantly enhance performance.

# **7. GitHub Link**

Find the link of our project below: