Engineering and Technology Journal e-ISSN: 2456-3358

Volume 11 Issue 03 March-2026, Page No.-9173-9178

DOI: 10.47191/etj/v11i03.06, I.F. – 8.595

Application of Constraint Database for Validation Synchronization Between Front End and Back End in Waste Customer Data Management Information System

Muhammad Sholeh[1*], Suraya[2*], Dina Andayati[3]

[1]Informatics Study Program, Faculty of Science and Information Technology, AKPRIND University Indonesia

[2]Retail Management, Faculty of Business and Communication, AKPRIND University Indonesia

[3]Computer Systems Engineering Study Program, Faculty of Science and Information Technology, AKPRIND University Indonesia

ABSTRACT: The management of waste customer data carried out by the Margo Rukun Bantul Association requires an integrated application to support effective recording, billing, and payment processes. The developed application is designed with consideration for data consistency between the front end and back end layers in order to maintain information consistency across all system components. This study aims to analyze and apply various constraints to the database that serve to maintain data integrity and synchronization in the customer management information system. The research methods include data requirements analysis, relational entity model (ERD) design, and the implementation of constraints such as primary keys, foreign keys, enums, default values, triggers, and index optimization in the database. The implementation results show that the application of database constraints can reduce input errors from the front end, ensure the interrelationship between customer, billing, and payment tables, and enable automatic payment status updates through triggers. In addition, the application of double validation between the front end and back end ensures that the stored data is always consistent, valid, and in accordance with applicable business rules. The results of this study indicate that the proper implementation of constraints not only improves the accuracy and efficiency of the system, but also strengthens the security and reliability of the waste customer data management service at the Margo Rukun Association

KEYWORDS: database constraints, data validation, system synchronization, front end, back end,

I. INTRODUCTION

Accurate and integrated data management is an important element in supporting the operational efficiency of an organization, including the Margo Rukun Association. The Margo Rukun Association, as a community-based waste management unit, plays a strategic role in waste transportation services. Customer administration, billing, and payment processes are still carried out manually. This type of management often faces obstacles related to data duplication, inconsistencies between systems, and delays in transaction status updates. These problems can be addressed by implementing information technology-based applications. Applications can implement integrated systems for customer management, billing, and other management processes.

The development of a reliable information system does not only focus on the application being built but is also greatly influenced by the design of a reliable database system (1), (2). A database system not only stores information but also automatically enforces validation rules through constraints (3), (4). The constraint rules required in database design include primary key, foreign key, enum, default value, and trigger. These constraints serve to maintain the integrity of the relationship between tables and ensure data consistency when inputting, updating, or deleting information. The application of these constraints allows the back-end system to reject invalid data sent from the front-end, while strengthening validation both from the application side and within the database system (5), (6). The use of triggers on transaction tables can improve the efficiency and accuracy of business processes, for example, automatically updating payment status after the total payment meets the invoice value. The use of triggers no longer depends on manual updates from the application used by users, but can reduce the risk of recording errors and improve the reliability of financial reports (7),(8)

An information system is a combination of technology, data, procedures, and human resources used to collect, process, store, and disseminate information to support decision-making processes within an organization (9),(10). Information systems can be applied to all organizations, including environmental management such as waste banks. Information systems in waste management can be used to record customer transactions, manage services, and support

administrative processes efficiently and transparently (11),(12).

Information systems are classified into several types based on their development platform, namely web-based systems, desktop-based systems, mobile-based systems, and cloud-based systems. According to Pressman (13),, the selection of a development platform must consider aspects of accessibility, ease of maintenance, resource efficiency, and the scale of system usage. Web-based information systems are the most widely used type because they can be accessed through a web browser without requiring additional software installation on the user's side. Data and business processes are run on the server, so users only need an internet connection to access the service. This system is suitable for organizations that require extensive data integration.

The research aims to design and implement a customer data management information system by applying various types of database constraints to ensure synchronization and validation between the front end and back end. Through this approach, it is hoped that a more efficient, secure, and consistent system will be created to support customer service management and financial transactions in the waste bank environment

II. METHOD

The research conducted is applied research with a software engineering approach that focuses on the development and application of constraints in databases to maintain synchronization between the front end and back end in customer waste data management information systems. The approach used is descriptive and experimental, where the research process is carried out through the stages of requirements analysis, system design, implementation, and testing of the developed system.

The system development method used is the prototyping method, which allows direct user involvement in the system design and evaluation process (14), (15). The stages of this method include: (1) analysis of system and user requirements; (2) design of data models using Entity Relationship Diagrams (ERDs) and user interface designs; (3) implementation of a web-based system using the PHP programming language and MariaDB database with the application of constraints such as primary keys, foreign keys, enums, default values, triggers, and index optimization; (4) system testing for functionality and data validation; and (5) evaluation based on user feedback for system refinement.

Data analysis was conducted using simple qualitative and quantitative methods. Qualitative analysis was used to assess the suitability of the system to user needs and the effectiveness of constraint implementation in maintaining data integrity, while quantitative analysis was conducted by comparing input error rates, system response speeds, and data consistency before and after constraint implementation. The implementation was carried out on hardware in the form of a laptop with Intel i5 processor and 8 GB RAM specifications, and using supporting software such as XAMPP, Visual Studio Code, PHP, and MariaDB

III. RESULT

A. Database Design

The results of this study are the implementation of a web-based waste customer data management information system with a client–server architecture that integrates the front end and back end layers synchronously. The system was developed using the PHP programming language with a MariaDB database equipped with various constraints to ensure data integrity and consistency. The database structure consists of five main tables, namely customers, service_types, bills, payments, subscription_types, and users. Each table is designed with the application of primary keys, foreign keys, enums, default values, and triggers according to the requirements of the relationship between entities. Figure 1, relationship between tables in the waste customer management database

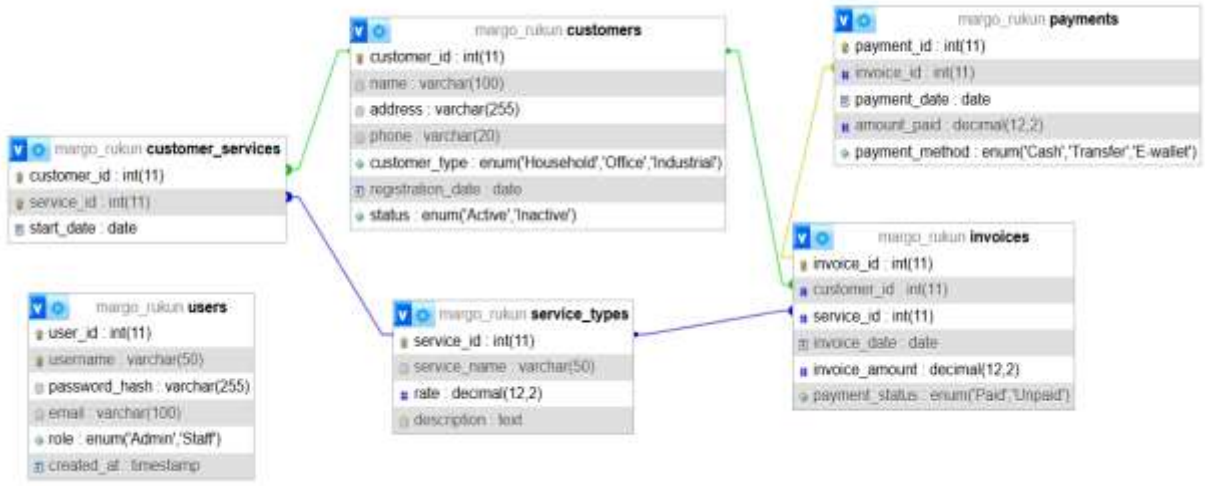

Figure 1. Relationships between tables in the waste customer management database

B. Main Components (Entities and Attributes)

The process of storing data in a database is stored in tables arranged according to the needs of the data to be stored. Data is stored in tables with consideration given to the normalization process.

1. service_types Table

Function: Stores a list of waste service types and their rates.

Structure:

- service_id INT(11) PRIMARY KEY AUTO_INCREMENT
- service_name VARCHAR(50) NOT NULL rate DECIMAL(12,2) NOT NULL description TEXT

Constraints:

- PRIMARY KEY on service_id
- AUTO_INCREMENT to generate automatic IDs
- NOT NULL constraint on service_name and rate

2. Table customers

Function: Stores customer data and membership status.

Structure:

- customer_id INT(11) PRIMARY KEY AUTO_INCREMENT
- name VARCHAR(100) NOT NULL
- address VARCHAR(255) NOT NULL
- phone VARCHAR(20)
- customer_type ENUM('Household','Office','Industrial') DEFAULT 'Household'
- registration_date DATE DEFAULT CURDATE()
- status ENUM('Active','Inactive') DEFAULT 'Active'

Constraints:

- PRIMARY KEY on customer_id
- AUTO_INCREMENT to generate automatic IDs
- NOT NULL constraint on name and address
- ENUM constraint on customer_type and status
- DEFAULT values for customer_type, registration_date, and status

3. Invoices Table

Function: Stores billing data for each customer.

Structure:

- invoice_id INT(11) PRIMARY KEY AUTO_INCREMENT
- customer_id INT(11) NOT NULL
- service_id INT(11) NOT NULL
- invoice_date DATE DEFAULT CURDATE()
- invoice_amount DECIMAL(12,2) NOT NULL
- payment_status ENUM('Paid','Unpaid') DEFAULT 'Unpaid'

Constraints:

- PRIMARY KEY on invoice_id
- FOREIGN KEY (customer_id) REFERENCES customers(customer_id)
- FOREIGN KEY (service_id) REFERENCES service_types(service_id)
- NOT NULL constraint on customer_id, service_id, invoice_amount
- ENUM constraint on payment_status
- DEFAULT values for invoice_date and payment_status

4. Table payments

Function: Records payments made by customers.

Structure:

- payment_id INT(11) PRIMARY KEY AUTO_INCREMENT
- invoice_id INT(11) NOT NULL
- payment_date DATE DEFAULT CURDATE()
- amount_paid DECIMAL(12,2) NOT NULL
- payment_method ENUM('Cash','Transfer','E-wallet') DEFAULT 'Cash'

Constraints:

- PRIMARY KEY on payment_id
- FOREIGN KEY (invoice_id) REFERENCES invoices(invoice_id)
- NOT NULL constraint on invoice_id and amount_paid
- ENUM constraint on payment_method
- DEFAULT values for payment_date and payment_method

5. Table customer_services

Function: Stores the relationship between customers and the services they use.

Structure:

- customer_id INT(11) NOT NULL
- service_id INT(11) NOT NULL
- start_date DATE DEFAULT CURDATE()

Constraints:

- PRIMARY KEY (customer_id, service_id) - Composite Primary Key
- FOREIGN KEY (customer_id) REFERENCES customers(customer_id)
- FOREIGN KEY (service_id) REFERENCES service_types(service_id)
- DEFAULT value for start_date

6. Table users

Function: Manages user access to the system.

Structure:

- user_id INT(11) PRIMARY KEY AUTO_INCREMENT
- username VARCHAR(50) NOT NULL UNIQUE
- password_hash VARCHAR(255) NOT NULL
- email VARCHAR(100)
- role ENUM('Admin','Staff') DEFAULT 'Staff'
- created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP

Constraints:

- PRIMARY KEY on user_id
- UNIQUE constraint on username
- NOT NULL constraint on username and password_hash
- ENUM constraint on role
- DEFAULT values for role and created_at

C. Trigger Implementation

An important aspect of the waste customer data management information system is maintaining data integrity and consistency through the application of triggers in the database. Triggers function as an automatic mechanism that is run by the database system whenever there is a change in data, such as the addition or update of payment transactions (16). The use of triggers is designed to minimize errors caused by manual processes and ensure synchronization between interrelated tables, particularly between payment and billing tables.

With the implementation of the after_payment_insert trigger, the system automatically calculates the total payments received for each bill, then updates the status_bayar column in the billing table to "Paid" if the total payment has reached the bill amount. This mechanism enables real-time status updates without requiring additional action from the application user. Thus, triggers not only improve the operational efficiency of the system, but also strengthen data validation and integrity on the back end, so that the data displayed on the front end is always accurate, up-to-date, and in accordance with the actual transaction conditions. The trigger implementation script is

```
DELIMITER //
CREATE TRIGGER after_payment_insert
AFTER INSERT ON payment
FOR EACH ROW
BEGIN
DECLARE total_payment DECIMAL(12,2);
DECLARE total_bill DECIMAL(12,2);
-- Calculate the total payment received for a specific bill
SELECT SUM(payment_amount) INTO total_payment
FROM payment
WHERE invoice_id = NEW.invoice_id;
-- Retrieve the total invoice amount
SELECT invoice_amount INTO total_invoice
FROM invoice
WHERE invoice_id = NEW.invoice_id;
-- Checking if total_payment >= total_invoice, change status to 'Paid'
IF total_payment >= total_invoice THEN
UPDATE invoice
SET payment_status = 'Paid'
WHERE invoice_id = NEW.invoice_id;
ELSE
UPDATE invoice
SET payment_status = 'Not Paid'
WHERE invoice_id = NEW.invoice_id;
END IF;
END //
DELIMITER ;
```

The explanation of the trigger script is presented in Table 1.

Table 1. Explanation of the parts and functions of the trigger used.

Part	Function
AFTER INSERT ON Payment	This trigger is executed after new payment data is saved.
For Each Row	Applies to each row of data entered (not the entire table at once).
DECLARE Total_Payment, Total_Invoice	Stores the temporary calculation results.
SELECT SUM(Payment_Amount)	Calculates the total payment for the related invoice.
SELECT Invoice_Amount	Retrieves the total invoice amount from the invoice table.
IF total_payment >= total_invoice THEN	Decision logic: if the payment is sufficient, change the status to Paid.
UPDATE invoice SET payment_status = ...	Changes the status in the invoice table according to the calculation results.

Synchronization of validation between the front end and back end is done with two layers of verification. Front-end validation includes checking input formats, selecting available values, and requiring certain fields to be filled in before data is sent to the server. Back-end validation is performed through database constraints, which ensure that the data received is in accordance with the established relational structure and rules. This layered validation approach has proven effective in preventing data duplication and input errors that previously occurred frequently.

D. Discussion

The results of this test prove that the database system has met the reliability and integrity standards required for system operations. The implementation of strict constraints ensures data consistency, while the use of triggers can improve

operational efficiency and minimize human error. The testing process for constraints and triggers includes

1. Foreign Key Constraint Test
   - creating an invoice for a non-existent customer

     INSERT INTO invoices (customer_id, service_id, invoice_amount) VALUES (999, 1, 5000.00);

The test results are presented in Figure 2

Figure 2. Test results by creating an invoice for a non-existent customer

2. Testing TRIGGER after_invoice_insert
   - Create a new invoice and check auto-populate customer_services

     -- Insert new invoice
     INSERT INTO invoices (customer_id, service_id, invoice_amount)
     VALUES (1, 2, 30000.00);
     -- Check customer_services
     SELECT * FROM customer_services
     WHERE customer_id = 1 AND service_id = 2;

Test results are shown in Figure 3

Figure 3. Test results for the after_invoice_insert trigger

3. Testing TRIGGER after_payment_insert - STATUS PAID
   - Perform the payment settlement process and change the status to "Paid"

     -- Check invoice status before payment
     SELECT invoice_id, invoice_amount, payment_status
     FROM invoices WHERE invoice_id = 380;
     -- Insert full payment
     INSERT INTO payments (invoice_id, amount_paid)
     VALUES (380, 30000.00);

     -- Check status after payment
     SELECT invoice_id, invoice_amount, payment_status
     FROM invoices WHERE invoice_id = 380

Test results are presented in Figure 4

Figure 4. Test TRIGGER after_payment_insert - STATUS PAID

4. Testing CASCADE DELETE
   - Testing by deleting customer and checking related data

     -- Check data before delete
     SELECT COUNT(*) AS invoice_count
     FROM invoices WHERE customer_id = 53;
     -- Delete customer
     DELETE FROM customers WHERE customer_id = 53;
     -- Check invoices after delete
     SELECT COUNT(*) AS invoice_count
     FROM invoices WHERE customer_id = 53;

The test results show that the contrains and trigger processes can prevent the addition, deletion, and modification of data without considering the relationship between data in other tables. The test results are shown in Table 3

Table 3. Results of various tests conducted

Component	Status	Description
Foreign Key Constraints	Successful	Prevents Orphan Data
NOT NULL Constraints	Successful	Enforce Required Fields
UNIQUE Constraints	Successful	Prevents Data Duplication
Trigger: After_Invoice_Insert	Successful	Auto-Update Customer_Services
Trigger: After_Payment_Insert	Successful	Auto-Update Payment Status
CASCADE DELETE	Successful	Maintain Referential Integrity

V. CONCLUSIONS

The results of testing conducted on the implementation of constraints and triggers in the waste management database system can perform automated processes well. The process of creating constraints, such as the use of foreign keys, not null, unique, and cascade delete, can be effective in maintaining

database structural integrity by preventing invalid data, duplication, and relational inconsistencies. In addition to constraints, the implementation of triggers, including after_invoice_insert and after_payment_insert, can automate data update processes, such as documenting customer service history and updating payment status in real-time. These results not only demonstrate data consistency in database normalization, but also the system's ability to support efficient data operations with minimal manual errors and good scalability for future development.

REFERENCES

1. Kalsum Siregar U, Arbaim Sitakar T, Haramain S, Nur Salamah Lubis Z, Nadhirah U, Sains dan Teknologi F. Pengembangan database Management system menggunakan My SQL. SAINTEK: Jurnal Sains, Teknologi & Komputer. 2024;1(1):8–12.
2. Yulherniawati, Ikhsan A. Perancangan Basis Data Untuk Pengembangan Sistem Informasi Akademik Berbasis Web Jurusan Teknologi Informasi Politeknik Negeri Padang. Jurnal Teknik Industri. 2013;2(1):15–6.
3. Kurniawan RA, Widyatama AW, Setiani H, Ikram MW, Arridho MN, Kurniawan AT, et al. Pentingnya Menggunakan Check Constraints Dalam Desain Database Sistem Pendukung Keputusan. Journal of Information System Management (JOISM). 2021;3(1):8–11.
4. Rahma FI, Agustin T, James RM, Utami E. Implementasi Constraint CHECK Pada Basis Data Aplikasi LaundryPOS Dalam Aspek Kebenaran Data. Creative Information Technology Journal. 2021;7(2):133.
5. Renaisan P, Astawa IGS, Giri GAVM. Desain Frontend Aplikasi Website Panggil Tukang Terpercaya Di Bali. Jurnal Pengabdian Informatika [Internet]. 2023;1(3):623–8. Available from: https://ejournal1.unud.ac.id/index.php/jupita/article/view/263
6. Hadianto RMGD, Zubaidi A. Pengembangan Back-End Pada Aplikasi Presensi Dan Website Pendataan Umkm Ntb Mall. Jurnal Begawe Teknologi Informasi (JBegaTI). 2024;5(2):170–81.
7. Thaariq MRA, Hamzah A, Raharjo S. Analisis Perbandingan Aspek Keamanan Basis Data Menggunakan Skema dan Tanpa Skema di PostgreSQL. Jnanaloka. 2024;51–61.
8. Lesmono A, Ghozali K, Indrawanti AS. Integrasi Data pada Aplikasi Desktop. Jurnal Teknik ITS. 2022;11(2):67–72.
9. Wahono HTT. Peran Sistem Informasi Manajemen Dalam Meningkatkan. Jurnal Filsafat, Sains, Teknologi, dan Sosial Budaya. 2024;30(4):97–110.
10. Aidah, Arifudin O, Ibrahim T. Pengembangan Sistem Informasi Manajemen Dalam Dunia Pendidikan. Jurnal Tahsinia. 2024;5(6):966–77.
11. Afuan L, Nofiyati N, Umayah N. Rancang Bangun Sistem Informasi Bank Sampah di Desa Paguyangan. Edumatic: Jurnal Pendidikan Informatika. 2021;5(1):21–30.
12. Atin S, Mutia S, Widayanti A, Yatawa HS, Rafdhi AA, Afrianto I, et al. Website-Based Information System Design for Waste Banks. IJIS Indonesian Journal on Information System. 2022;7(April):59–70.
13. Pressman RS, Maxim BR. Software Engineering: A Practitioner's Approach. McGraw-Hill Education; 2020.
14. Fridayanthie EW, Tsabitah T. Penerapan Metode Prototype Pada Perancangan Sistem Informasi Penggajian Karyawan (Persis Gawan) Berbasis Web. Paradigma. 2021;23(2):151–7.
15. Taufiq R, Prianggodo DY, Sugiani Y, Andini N, Informatika T, Teknik F, et al. PENGGUNAAN METODE PROTOTYPE PADA PENGEMBANGAN SISTEM. JIKA (Jurnal of Informatics). 2023;7(4):431–9.
16. Sentosa S, Simanullang J, Sembiring RSB, Ginting VS, Tanady D. Penanganan Data Integrity Dalam Object Relation Database Management System. Jurnal Kolaboratif Sains. 2025;8(1):922–7.