Validasi dan Deteksi Kesalahan Data Relasional dengan *Hybrid Rule-based System*

Validation and Error Detection in Relational Data using a Hybrid Rule-based System

[1]Daniel Andrew Shane Chayono*, [2]Johan Jimmy Carter Tambotoh
[1,2,]Program Studi Sistem Informasi, Fakultas Teknologi Informasi, Universitas Kristen Satya Wacana
[1,2]Jl. Dr. O. Notohamidjodjo Blotongan, Sidorejo, Kota Salatiga, Jawa Tengah, Indonesia
*e-mail: 682022006@student.uksw.edu

(***received*:** 9 June 2026, ***revised*:** 20 June 2026, ***accepted*:** 21 June 2026)

Abstrak

Database relasional menjadi tulang punggung sistem informasi modern. Permasalahan kualitas data seperti duplikat, format tidak valid, nilai kosong, dan inkonsistensi antar tabel dapat menurunkan akurasi pengambilan keputusan secara signifikan. Rule-based validation memiliki keterbatasan dalam mendeteksi kesalahan ambigu seperti perbedaan penulisan nama akibat typo. Penelitian ini mengusulkan sistem Hybrid Rule-Based yang menggabungkan SQL trigger untuk mendeteksi kesalahan terstruktur dan fuzzy matching menggunakan Python RapidFuzz untuk mendeteksi kemiripan data pada database relasional multi tabel. Sistem diimplementasikan pada enam tabel utama dengan enam tabel karantina dan satu error_log terpusat. Pengujian menggunakan dataset sintetis 2.775 record yang tersebar di enam tabel, dibangkitkan secara terkontrol menggunakan script generate_dataset.py dengan berbagai jenis kesalahan yang disengaja disisipkan untuk memungkinkan verifikasi hasil deteksi secara tepat. Hasil menunjukkan 472 entri error terdeteksi dan 297 record dipindahkan ke tabel karantina. Rule-based berkontribusi 311 entri (65,9%) mencakup kesalahan format, nilai negatif, dan pelanggaran integritas referensial, sedangkan fuzzy matching mendeteksi 127 entri kesalahan semantik (112 duplikat nama customer, 7 nama produk mirip, 5 kategori bermasalah) yang tidak dapat ditangkap oleh aturan SQL apapun. Pada dataset pengujian ini, pendekatan hybrid mendeteksi 34,1% kesalahan tambahan dibandingkan jika hanya menggunakan rule-based saja.

Kata kunci: data quality, deteksi duplikat, fuzzy matching, SQL trigger, validasi data relasional

Abstract

Relational databases form the backbone of modern information systems. However, data quality issues, such as duplicate records, invalid formats, missing values, and cross-table inconsistencies, can significantly reduce the accuracy of data-driven decision-making. Conventional rule-based validation is effective for detecting structured errors but has limited capability in identifying ambiguous errors, such as typographical variations in entity names. This study proposes a hybrid rule-based system that combines SQL triggers for structured error detection with fuzzy matching using the RapidFuzz Python library to identify semantically similar records across multiple relational database tables. The proposed system was implemented using six primary database tables, six corresponding quarantine tables, and a centralized error_log table. The evaluation was conducted using a synthetic dataset containing 2,775 records distributed across the six tables. The dataset was systematically generated using the generate_dataset.py script, with various intentionally injected data quality issues to enable accurate verification of the detection results. The experimental results show that the proposed system detected 472 data quality issues, with 297 records automatically moved to the quarantine tables. The rule-based component identified 311 errors (65.9%), including format violations, negative values, and referential integrity violations. Meanwhile, the fuzzy matching component detected 127 semantic errors that could not be identified using SQL rules alone, including 112 duplicate customer names, 7 similar product names, and 5 inconsistent product categories. On the experimental dataset, the proposed hybrid approach detected 34.1% more data quality issues than a rule-based validation

 ISSN:2302-8149
e-ISSN:2540-9719

approach alone. These findings demonstrate that integrating rule-based validation with fuzzy matching substantially improves error detection capability in relational databases, particularly for semantic inconsistencies that are difficult to capture using conventional validation rules.

Keywords: *data quality, duplicate detection, fuzzy matching, relational database, SQL trigger*

1 Pendahuluan

Database relasional menjadi tulang punggung sistem informasi modern. Hampir seluruh aplikasi yang digunakan dalam kehidupan sehari-hari mulai dari platform e-commerce, sistem perbankan, hingga sistem informasi institusi menyimpan dan mengelola datanya menggunakan database relasional. Ketika sebuah sistem bergantung sepenuhnya pada data, kualitas data menjadi aspek yang tidak dapat diabaikan. Berbagai framework telah dikembangkan untuk mengukur dan memastikan kualitas data secara sistematis, namun masing-masing memiliki fokus dan cakupan yang berbeda tergantung konteks penggunaannya [1]. Data yang tidak akurat, terduplikasi, atau tidak konsisten dapat berujung pada keputusan bisnis yang keliru, laporan yang tidak dapat diandalkan, serta potensi kerugian finansial yang signifikan [2], [3]. Pemahaman yang komprehensif tentang dimensi kualitas data mencakup akurasi, konsistensi, kelengkapan, dan ketepatan waktu menjadi landasan penting dalam merancang sistem validasi yang efektif [4]. Lebih jauh, kualitas data tidak hanya ditentukan dari atribut intrinsiknya, tetapi juga dari kesesuaiannya dengan konteks penggunaan sehingga data yang sama dapat dinilai berkualitas berbeda tergantung pada tujuan penggunaannya [2], [5].

Inspirasi penelitian ini bermula dari pengalaman magang di sebuah perusahaan yang mengelola database pelanggan dalam skala besar. Selama proses magang, ditemukan bahwa sejumlah data pelanggan dalam sistem mengandung berbagai permasalahan: terdapat entri duplikat baik yang identik maupun yang hanya berbeda sedikit akibat kesalahan ketik, format nomor telepon yang tidak sesuai standar, serta data yang seharusnya terisi namun bernilai kosong. Permasalahan tersebut telah terakumulasi dalam sistem tanpa terdeteksi karena tidak tersedia mekanisme validasi yang memadai. Pendekatan siklus perbaikan data secara berkelanjutan diperlukan untuk memastikan kualitas data terjaga sepanjang siklus hidupnya dalam sistem informasi [6]. Kondisi ini mendorong pertanyaan mendasar: bagaimana membangun sistem yang mampu mendeteksi berbagai jenis kesalahan data secara otomatis dan komprehensif pada database relasional multi tabel?

Pendekatan validasi berbasis aturan (rule-based validation) merupakan metode yang telah lama diterapkan untuk menjaga integritas data. Implementasi melalui SQL constraint, trigger, dan stored procedure memungkinkan validasi berjalan secara otomatis setiap kali terjadi operasi manipulasi data [7], [8]. Metode ini efektif untuk mendeteksi kesalahan yang bersifat terstruktur dan dapat didefinisikan secara eksplisit, seperti format email tidak valid, nomor telepon terlalu pendek, atau pelanggaran foreign key. Namun, rule-based memiliki keterbatasan fundamental: ia tidak mampu mendeteksi kesalahan yang bersifat ambigu, seperti dua entri nama "Ahmad Fauzi" dan "Achmad Fauzi" yang merujuk pada orang yang sama namun ditulis berbeda [9].

Fuzzy matching merupakan pendekatan komplementer yang mengukur tingkat kemiripan antara dua string menggunakan algoritma seperti Levenshtein distance dan token based similarity. Dengan pendekatan ini, sistem dapat mendeteksi bahwa dua penulisan nama yang berbeda sesungguhnya merujuk pada entitas yang sama berdasarkan nilai kemiripan di atas threshold yang ditentukan [10]. Untuk implementasi, digunakan library Python RapidFuzz yang secara empiris terbukti memiliki performa lebih tinggi dibandingkan library sejenis [11].

Sejumlah penelitian terdahulu telah mengkaji permasalahan kualitas data dan deteksi duplikat, namun sebagian besar berfokus pada satu tabel atau satu metode validasi saja. Gao et al. [12] mengusulkan framework hybrid MLNClean yang menggabungkan teknik berbasis aturan dengan pendekatan statistik, namun penerapannya terbatas pada satu relasi tabel dan tidak mencakup mekanisme karantina data. Chaudhuri et al. [13] membahas teknik fuzzy match untuk pembersihan data secara online, tetapi fokusnya pada efisiensi algoritma bukan pada integrasi multi tabel. Stoikov et al. [14] mengkaji teknik record linkage lintas dataset namun dalam konteks warisan budaya, bukan sistem transaksional relasional. Penelitian yang menggunakan SQL trigger umumnya hanya menerapkannya sebagai constraint tambahan tanpa integrasi dengan deteksi semantik [8], [15], sedangkan penelitian fuzzy matching Python seperti Elmobark [11] dan Wibowo et al. [16] berfokus pada perbandingan algoritma, bukan pada arsitektur sistem validasi multi tabel yang terintegrasi.

 ISSN:2302-8149
e-ISSN:2540-9719

Dengan demikian, belum ditemukan penelitian yang secara spesifik menggabungkan SQL trigger BEFORE INSERT dengan fuzzy matching Python pada database relasional multi tabel mencakup seluruh aspek integritas terstruktur dan semantik sekaligus disertai mekanisme karantina terpusat yang mendukung traceability. Celah inilah yang menjadi kontribusi penelitian ini.

Penelitian ini mengusulkan sistem Hybrid Rule-Based yang bekerja dalam dua lapisan: lapisan pertama berupa SQL trigger yang berjalan otomatis pada setiap operasi INSERT untuk menangkap kesalahan terstruktur, dan lapisan kedua berupa script Python dengan RapidFuzz yang mendeteksi kemiripan data antar record. Data yang teridentifikasi bermasalah tidak dihapus secara permanen, melainkan dipindahkan ke tabel karantina untuk keperluan analisis lebih lanjut, sehingga mendukung prinsip traceability dalam data quality management [17], [18].

2 Tinjauan Literatur

Kualitas data mencakup beberapa dimensi utama akurasi, konsistensi, kelengkapan, dan ketepatan waktu yang saling bergantung dan berpengaruh pada keandalan sistem informasi [1], [2], [3]. Wang et al. [4] dan Merino et al. [5] memperluas perspektif ini dengan menyatakan bahwa kualitas data juga ditentukan oleh kesesuaiannya dengan konteks penggunaan, bukan hanya atribut intrinsik data itu sendiri. Dalam praktiknya, Yulianto [6] menekankan bahwa penjagaan kualitas data harus dilakukan secara berkelanjutan sebagai bagian dari siklus hidup sistem informasi, bukan sebagai aktivitas sekali jalan.

Mekanisme validasi berbasis aturan melalui SQL trigger dan constraint merupakan pendekatan yang paling banyak diterapkan untuk menjaga integritas data di level database. Aidjili [8] dan Eessaar [15] menunjukkan bahwa validasi di level database lebih andal karena tidak dapat di bypass oleh lapisan aplikasi manapun, meskipun banyak sistem nyata masih kekurangan constraint deklaratif yang kritis. Cruz-Filipe et al. [19] mengembangkan pendekatan active integrity constraints yang secara otomatis memicu aksi perbaikan saat pelanggaran terdeteksi. Khomh et al. [7] mengkonfirmasi melalui tinjauan sistematis bahwa rule-based tetap menjadi komponen utama dalam pipeline data cleaning, namun memiliki keterbatasan fundamental dalam menangani kesalahan semantik seperti variasi penulisan nama.

Untuk mengatasi keterbatasan tersebut, pendekatan fuzzy string matching telah banyak dikaji sebagai solusi deteksi duplikat semantik. Elmagarmid et al. [17] dan Chaudhuri et al. [13] meletakkan fondasi teknis deteksi duplikat berbasis kemiripan string, sementara Kaufman dan Klevs [10] serta Wibowo et al. [16] menunjukkan efektivitasnya pada data nama dengan variasi penulisan. Azeroual et al. [20] dan Stoikov et al. [14] memperluas cakupannya ke domain record linkage lintas dataset, dan Barlaug dan Gulla [21] mengkonfirmasi bahwa pendekatan berbasis kemiripan string tetap relevan dibandingkan neural network untuk skenario dengan kebutuhan interpretabilitas tinggi. Dari sisi implementasi, Elmobark [11] menunjukkan bahwa RapidFuzz secara empiris 40% lebih cepat dari fuzzywuzzy, menjadikannya pilihan optimal untuk dataset berskala besar.

Sejumlah penelitian mengusulkan pendekatan hybrid untuk menggabungkan kekuatan kedua metode. Gao et al. [12] dan Fu et al. [22] mengintegrasikan aturan eksplisit dengan machine learning, sementara Ilyas dan Rekatsinas [23] serta Ilyas dan Chu [18] menyimpulkan bahwa hubungan keduanya bersifat simbiotik dan telah menjadi tren utama riset data cleaning relasional. Chu et al. [9] dan Herschel et al. [24] menekankan pentingnya pipeline deteksi yang terintegrasi dengan mekanisme pencatatan jejak data (provenance) untuk mendukung auditabilitas. Cahyaningsih et al. [25] dan Dankar et al. [26] mengkonfirmasi bahwa pengukuran kualitas data yang sistematis termasuk penggunaan data sintetis terkontrol untuk pengujian menghasilkan sistem validasi yang lebih andal dan terverifikasi.

Dari kajian di atas, teridentifikasi bahwa penelitian yang mengintegrasikan SQL trigger dengan fuzzy matching Python pada database relasional multitabel secara komprehensif masih sangat terbatas. Sebagian besar pendekatan hybrid yang ada berfokus pada satu tabel atau menggunakan machine learning sebagai komponen kedua yang memerlukan data pelatihan. Penelitian ini mengisi celah tersebut dengan mengusulkan sistem yang menggabungkan SQL trigger untuk deteksi terstruktur secara real time dengan modul Python RapidFuzz untuk deteksi semantik dan relasional, dilengkapi mekanisme karantina terpusat yang mendukung traceability pada database relasional enam tabel.

 ISSN:2302-8149
e-ISSN:2540-9719

3 Metode Penelitian

3.1 Arsitektur Sistem

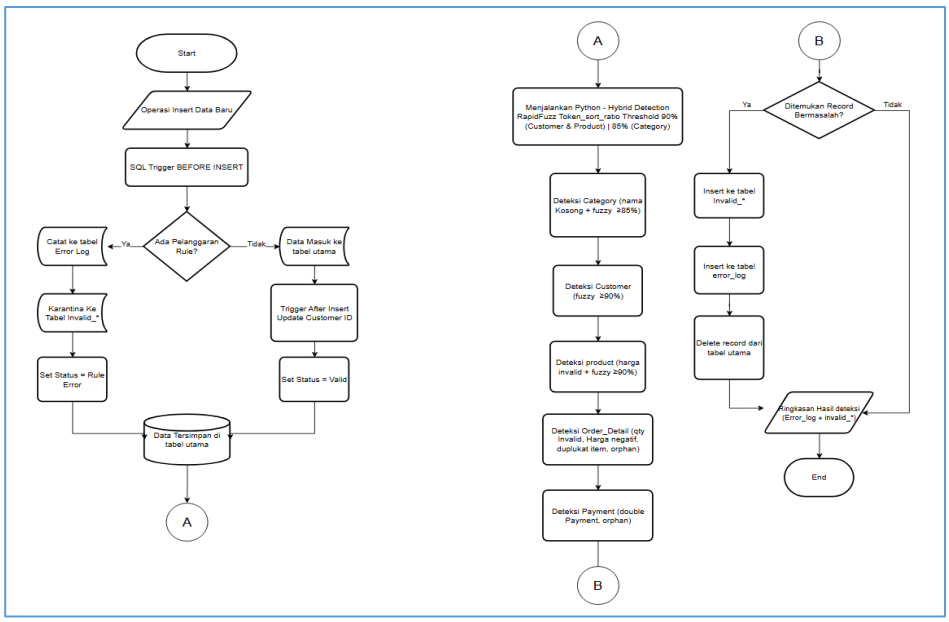
Sistem yang dirancang dalam penelitian ini terdiri dari dua komponen utama yang saling melengkapi. Komponen pertama adalah mekanisme rule-based validation yang diimplementasikan melalui SQL trigger pada database MariaDB. Komponen kedua adalah modul fuzzy matching yang dikembangkan menggunakan bahasa pemrograman Python dengan library RapidFuzz [11]. Kedua komponen beroperasi secara berurutan: trigger aktif secara real time pada setiap operasi INSERT, sementara modul Python dijalankan secara periodik untuk melakukan pemindaian menyeluruh terhadap data yang telah tersimpan.

Database relasional yang digunakan terdiri dari enam tabel utama: *category*, *product*, *customer*, *orders*, *order_detail*, dan *payment*. Struktur relasi antar tabel dapat dilihat pada Gambar 1. Setiap tabel utama memiliki pasangan tabel karantina masing-masing sebagai tempat penyimpanan data yang teridentifikasi bermasalah. Seluruh kejadian kesalahan dicatat secara terpusat dalam tabel *error_log* yang menyimpan informasi sumber tabel, jenis kesalahan, sumber deteksi, dan skor kemiripan untuk kesalahan fuzzy.

Gambar 1 Entity relationship diagram (ERD) database relasional sistem pengujian

Alur kerja sistem secara keseluruhan ditunjukkan pada Gambar 2 yang terdiri dari dua bagian: Bagian 1 menggambarkan jalur trigger SQL yang aktif real time setiap INSERT, dan Bagian 2 menggambarkan jalur modul Python yang dijalankan secara periodik. Kedua jalur saling melengkapi untuk mencakup dua kategori kesalahan yang berbeda: kesalahan terstruktur dan kesalahan semantik [9], [17].

 ISSN:2302-8149
e-ISSN:2540-9719



Gambar 2 Flowchart alur kerja sistem hybrid rule-based

3.2 Rule-Based Validation

Validasi berbasis aturan diimplementasikan melalui SQL trigger *BEFORE INSERT* pada masing-masing tabel utama. Trigger ini aktif secara otomatis sebelum setiap operasi INSERT dieksekusi, sehingga memungkinkan penangkapan kesalahan secara real time [7], [8]. Implementasi trigger pada sistem manajemen basis data relasional terbukti efektif dalam menegakkan aturan validasi secara otomatis pada level database, sehingga validasi berlaku konsisten terlepas dari antarmuka atau aplikasi yang mengakses data [8], [15]. Pendekatan serupa juga dikaji dalam bentuk active integrity constraints aturan yang tidak hanya mendeteksi pelanggaran tetapi juga secara otomatis memicu aksi perbaikan ketika kondisi integritas dilanggar [19]. Tabel 1 merangkum jenis validasi yang diterapkan pada setiap tabel. Sebagai ilustrasi, berikut adalah contoh implementasi trigger BEFORE INSERT pada tabel customer untuk validasi format nomor HP dan deteksi duplikat nomor HP:

```
CREATE TRIGGER before_insert_customer
BEFORE INSERT ON customer
FOR EACH ROW
BEGIN
  -- Validasi format nomor HP (hanya digit)
  IF NEW.no_hp REGEXP '[^0-9]' THEN
    INSERT INTO error_log (error_source, tabel_sumber,
      input_nama, input_email, error_type, error_message)
    VALUES ('RULE_BASED', 'customer', NEW.nama, NEW.email,
      'PHONE_FORMAT',
      CONCAT('Format HP tidak valid: ', NEW.no_hp));
  END IF;

  -- Validasi panjang nomor HP
  IF LENGTH(NEW.no_hp) < 10 THEN
    INSERT INTO error_log (error_source, tabel_sumber,
      input_nama, input_email, error_type, error_message)
    VALUES ('RULE_BASED', 'customer', NEW.nama, NEW.email,
      'PHONE_LENGTH',
      CONCAT('Nomor HP terlalu pendek: ', NEW.no_hp));
  END IF;

  -- Deteksi duplikat nomor HP
  IF EXISTS (SELECT 1 FROM customer
             WHERE no_hp = NEW.no_hp) THEN
    INSERT INTO error_log (error_source, tabel_sumber,
      input_nama, input_email, error_type, error_message)
    VALUES ('RULE_BASED', 'customer', NEW.nama, NEW.email,
      'DUPLICATE_PHONE',
      CONCAT('Nomor HP sudah terdaftar: ', NEW.no_hp));
  END IF;
END;
```

Gambar 3 Trigger Before Insert customer untuk validasi format nomor HP dan deteksi duplikat nomor HP

 ISSN:2302-8149
e-ISSN:2540-9719

Trigger di atas menunjukkan tiga karakteristik desain utama: (1) setiap kondisi diperiksa secara independen sehingga satu record dapat menghasilkan lebih dari satu entri error_log; (2) trigger tidak memblokir INSERT data tetap masuk ke tabel utama sementara pelanggaran dicatat (log only design); dan (3) nilai input_nama dan input_email disimpan pada saat BEFORE INSERT karena ID record belum tersedia pada tahap ini. Trigger serupa diimplementasikan pada lima tabel lainnya dengan aturan validasi yang disesuaikan mencakup pemeriksaan referensi foreign key pada tabel orders dan order_detail, validasi nilai numerik pada tabel product dan payment, serta deteksi duplikat eksak pada tabel orders berdasarkan kombinasi customer_id, tanggal, dan total harga. Seluruh kode trigger tersedia atas permintaan kepada penulis korespondensi.

Tabel 1 Aturan validasi rule-based dan fuzzy per tabel

Tabel	Jenis Validasi	Kode Error	Layer
category	Nama kategori kosong atau NULL	EMPTY_CATEGORY_NAME	Python (Pemeriksaan Kondisi)
category	Nama kategori mirip dengan yang sudah ada (≥85%)	DUPLICATE_CATEGORY_NAME	Python (Fuzzy String)
product	Nama produk kosong atau NULL	EMPTY_NAME	Rule-Based
product	Harga produk < 0 (negatif)	INVALID_PRICE	Rule-Based
product	Harga produk = 0 (nol)	INVALID_PRICE	Python (Relasional)
product	Stok produk negatif	INVALID_STOCK	Rule-Based
product	category_id tidak ada di tabel category	INVALID_CATEGORY_REF	Rule-Based
product	Nama produk mirip dengan yang sudah ada (≥90%)	DUPLICATE_PRODUCT_NAME	Python (Fuzzy String)
customer	Nama kosong atau NULL	EMPTY_NAME	Rule-Based
customer	Format email tidak valid	EMAIL_FORMAT	Rule-Based
customer	Nomor HP mengandung karakter non-digit	PHONE_FORMAT	Rule-Based
customer	Nomor HP < 10 digit	PHONE_LENGTH	Rule-Based
customer	Email sudah terdaftar (duplikat eksak)	DUPLICATE_EMAIL	Rule-Based
customer	Nomor HP sudah terdaftar (duplikat eksak)	DUPLICATE_PHONE	Rule-Based
customer	Tanggal lahir di masa depan	INVALID_BIRTHDATE	Rule-Based
customer	Nama mirip dengan customer lain (≥90%)	DUPLICATE_NAME	Python (Fuzzy String)
orders	customer_id tidak ada di tabel customer	INVALID_CUSTOMER_REF	Rule-Based
orders	Tanggal order di masa depan	FUTURE_ORDER_DATE	Rule-Based
orders	Order duplikat (customer + tanggal + total sama)	DUPLICATE_ORDER	Rule-Based + Python (Relasional)
order_detail	order_id tidak ada di tabel	INVALID_ORDER_REF	Rule-Based

 ISSN:2302-8149
e-ISSN:2540-9719

Tabel	Jenis Validasi	Kode Error	Layer
	orders		
order_detail	product_id tidak ada di tabel product	INVALID_PRODUCT_REF	Rule-Based
order_detail	Qty ≤ 0 atau negatif	INVALID_QTY	Rule-Based + Python (Relasional)
order_detail	Harga satuan negatif	INVALID_PRICE	Rule-Based + Python (Relasional)
order_detail	Product sama muncul lebih dari satu kali per order	DUPLICATE_ITEM	Rule-Based + Python (Relasional)
order_detail	order_id tidak ada di tabel orders (orphan)	ORPHAN_DETAIL	Python (Relasional)
payment	order_id tidak ada di tabel orders	INVALID_ORDER_REF	Rule-Based
payment	Jumlah pembayaran ≤ 0	INVALID_AMOUNT	Rule-Based
payment	Tanggal bayar di masa depan	FUTURE_PAYMENT_DATE	Rule-Based
payment	Order sudah memiliki pembayaran SUCCESS	DOUBLE_PAYMENT	Rule-Based + Python (Relasional)
payment	order_id tidak ada di tabel orders (orphan)	ORPHAN_PAYMENT	Python (Relasional)

Pada tabel customer, terdapat delapan aturan validasi yang mencakup pemeriksaan nama kosong, format email, format dan panjang nomor HP, duplikat email, duplikat nomor HP, serta validitas tanggal lahir. Aturan format nomor HP menggunakan ekspresi reguler untuk memastikan nilai hanya mengandung digit dengan panjang minimal 10 karakter sesuai standar nomor telepon Indonesia. Nomor HP yang mengandung karakter non-digit umumnya juga memicu aturan panjang secara bersamaan karena karakter non-digit menyebabkan panjang efektif menjadi kurang dari 10 digit. Dua aturan validasi lainnya, EMAIL_FORMAT dan DUPLICATE_EMAIL tidak menghasilkan entri pada tabel hasil pengujian karena dataset sintetis yang digunakan tidak menyisipkan data dengan format email tidak valid maupun email duplikat secara eksplisit. Kedua aturan tersebut tetap berfungsi dalam sistem dan akan aktif apabila data dengan kondisi tersebut diinput.

Pada tabel category dan product, validasi mencakup pemeriksaan nama kosong, referensi kategori yang valid, serta nilai numerik yang tidak boleh negatif. Kondisi pengecekan harga pada trigger adalah harga < 0 artinya harga bernilai nol tidak tertangkap trigger, melainkan dideteksi oleh modul Python.

Pada tabel orders, order_detail, dan payment, validasi difokuskan pada integritas relasional dan validitas nilai numerik. Trigger orders mendeteksi order duplikat berdasarkan kombinasi customer_id, tanggal, dan total harga yang identik. Trigger payment mendeteksi double payment dengan memeriksa apakah order yang bersangkutan sudah memiliki pembayaran berstatus SUCCESS sebelumnya.

Selain trigger BEFORE INSERT, sistem mengimplementasikan trigger AFTER INSERT pada setiap tabel utama. Trigger ini memperbarui kolom customer_id (atau record_id untuk tabel lain) pada tabel error_log setelah data berhasil tersimpan. Hal ini diperlukan karena pada saat BEFORE INSERT berjalan, nilai AUTO_INCREMENT belum tersedia. Dengan adanya AFTER INSERT, setiap entri error_log yang sebelumnya bernilai NULL diperbarui menggunakan ID yang baru terbentuk sehingga traceability antara error_log dan tabel sumber terjaga sepenuhnya [17].

 ISSN:2302-8149
e-ISSN:2540-9719

3.3 Fuzzy Matching

Modul fuzzy matching dikembangkan menggunakan Python dengan library RapidFuzz yang mengimplementasikan algoritma token_sort_ratio. Algoritma ini bekerja dengan mengurutkan token (kata) dalam sebuah string sebelum menghitung skor kemiripan, sehingga pasangan seperti "Budi Ahmad" dan "Ahmad Budi" tetap terdeteksi sebagai mirip meskipun urutan katanya berbeda [10]. Pendekatan ini lebih robust untuk nama orang Indonesia dibanding algoritma berbasis karakter murni seperti Levenshtein distance.

Threshold kemiripan yang ditetapkan berbeda untuk setiap tabel berdasarkan karakteristik datanya. Untuk tabel customer dan product, threshold ditetapkan sebesar 90% karena nama-nama tersebut umumnya lebih panjang sehingga false positive dapat diminimalkan pada nilai ini. Untuk tabel category, threshold diturunkan menjadi 85% mengingat nama kategori biasanya lebih singkat dan perbedaan satu atau dua karakter saja sudah cukup signifikan secara semantik [10].

Perlu dipahami bahwa fuzzy string matching hanya diterapkan pada tiga tabel yang memiliki kolom nama tekstual yang rentan variasi penulisan: category, customer, dan product. Tiga tabel lainnya orders, order_detail, dan payment tidak memiliki kolom nama yang perlu dibandingkan secara semantik. Kesalahan pada ketiga tabel tersebut bersifat struktural dan eksak: duplikat order dideteksi melalui kecocokan kombinasi customer_id + tanggal + total; duplikat item dideteksi melalui kecocokan order_id + product_id; dan double payment dideteksi melalui pemeriksaan status SUCCESS sebelumnya. Semua deteksi ini dilakukan oleh modul Python menggunakan query relasional, bukan algoritma kemiripan string.

Modul Python menjalankan enam langkah deteksi secara berurutan sesuai hirarki relasi antar tabel: (1) deteksi category, nama kosong dan nama mirip (fuzzy string); (2) deteksi customer, nama mirip (fuzzy string); (3) deteksi product, harga tidak valid dan nama mirip (fuzzy string); (4) deteksi order_detail, qty tidak valid, harga negatif, item duplikat, dan orphan detail (relasional); (5) deteksi orders, order duplikat identik (relasional); (6) deteksi payment, double payment dan orphan payment (relasional). Setiap record yang terdeteksi bermasalah dipindahkan ke tabel karantina yang sesuai dan dihapus dari tabel utama [9], [18].

Konsep orphan record merujuk pada record yang kehilangan referensi induknya karena record induk tidak ada atau telah dihapus dari tabel utama. Orphan order_detail adalah record di tabel order_detail yang memiliki order_id tidak ada di tabel orders, item pesanan yang tidak bisa dikaitkan ke transaksi manapun. Orphan payment adalah record di tabel payment yang memiliki order_id tidak ada di tabel orders pembayaran yang tidak bisa dikaitkan ke order manapun, yang berbahaya secara finansial karena uang tercatat masuk namun tidak ada transaksi yang membenarkan pembayaran tersebut. Label FUZZY pada kolom error_source di error_log untuk seluruh deteksi modul Python termasuk orphan record dan double payment merupakan keterbatasan desain skema yang ada: kolom error_source hanya memiliki dua nilai ENUM yaitu RULE_BASED dan FUZZY, sehingga semua deteksi oleh modul Python dilabeli FUZZY meskipun secara teknis sebagian di antaranya menggunakan query relasional eksak, bukan algoritma kemiripan string. Untuk pengembangan selanjutnya, disarankan menambahkan nilai ENUM ketiga yaitu PYTHON_RELATIONAL untuk membedakan kedua jenis deteksi Python secara lebih akurat.

Pemilihan RapidFuzz sebagai library implementasi didasarkan pada hasil perbandingan empiris yang menunjukkan bahwa RapidFuzz 40% lebih cepat dibanding library fuzzywuzzy yang paling umum digunakan, dengan efisiensi memori yang lebih baik pada dataset besar [11]. Hal ini penting mengingat algoritma fuzzy matching memiliki kompleksitas $O(n^2)$ setiap record dibandingkan dengan seluruh record lainnya sehingga performa library menjadi faktor kritis pada dataset dengan ratusan hingga ribuan record.

3.4 Mekanisme Karantina Data

Penelitian ini mengimplementasikan pola karantina data (data quarantine pattern) sebagai alternatif dari penghapusan permanen. Ketika sebuah record terdeteksi bermasalah baik oleh trigger SQL maupun modul Python record tersebut tidak langsung dihapus dari sistem, melainkan dipindahkan ke tabel karantina pasangannya dan dihapus dari tabel utama [17], [18]. Pendekatan ini mendukung prinsip traceability dalam data quality management: data yang pernah masuk ke sistem tetap dapat diaudit, dianalisis, dan berpotensi dipulihkan di kemudian hari. Desain ini secara khusus mengantisipasi kemungkinan false positive pada fuzzy string matching di mana sistem tidak dapat

 ISSN:2302-8149
e-ISSN:2540-9719

membedakan secara otomatis antara duplikat nyata dengan data yang hanya kebetulan memiliki nama mirip. Dengan menyimpan data yang terdeteksi ke tabel karantina alih-alih menghapusnya, administrator dapat melakukan review manual dan memulihkan data yang salah dikarantina apabila diperlukan. Integrasi antara constraint berbasis aturan dan pendekatan deduplication berbasis kemiripan telah menjadi salah satu tren utama dalam data cleaning pada sistem relasional [9], [18].

Setiap tabel utama memiliki tabel karantina pasangannya dengan prefiks invalid_, sebagaimana dirangkum pada Tabel 2. Tidak ada foreign key antara tabel karantina dan tabel utama karena record di tabel utama sudah dihapus pada saat karantina; relasi dijaga melalui kolom original_id sebagai soft reference. Kolom ini menyimpan ID asli record sebelum dihapus, sehingga administrator dapat melacak asal-usul setiap record yang dikarantina.

Proses karantina untuk record yang memiliki relasi ke tabel lain dilakukan dengan urutan yang ketat untuk menghindari pelanggaran foreign key constraint. Sebagai contoh, ketika sebuah record customer akan dikarantina, proses dilakukan dalam urutan: (1) hapus payment yang terkait, (2) hapus order_detail yang terkait, (3) hapus orders yang terkait, (4) insert ke invalid_customer, dan (5) hapus dari customer. Seluruh operasi ini dibungkus dalam satu transaksi sehingga bersifat atomik jika salah satu langkah gagal, seluruh proses di rollback.

Tabel 2 Ringkasan tabel karantina (invalid_*)

Tabel Karantina	Pasangan Tabel Utama	Kolom Khas & Keterangan
invalid_category	category	original_id, similarity_score menyimpan kategori dengan nama kosong atau mirip kategori yang sudah ada
invalid_customer	customer	original_id, similarity_score menyimpan customer dengan error format, duplikat eksak, maupun duplikat fuzzy nama
invalid_product	product	original_id, harga, similarity_score menyimpan produk dengan harga tidak valid atau nama duplikat fuzzy
invalid_orders	orders	original_id, total_harga menyimpan order dengan referensi customer tidak valid atau order duplikat identik
invalid_order_detail	order_detail	original_id, qty, harga_satuan menyimpan item detail dengan qty/harga tidak valid, item duplikat dalam satu order, atau orphan detail
invalid_payment	payment	original_id, jumlah menyimpan pembayaran dengan jumlah tidak valid, double payment, atau orphan payment

3.5 Struktur Tabel Error_log

Seluruh aktivitas deteksi baik dari trigger SQL maupun modul Python dicatat dalam satu tabel terpusat bernama error_log. Desain tabel ini menggunakan dua kolom kunci untuk mengakomodasi perbedaan waktu antara trigger BEFORE INSERT dan AFTER INSERT: kolom input_nama dan input_email menyimpan nilai yang diinput pada saat trigger BEFORE INSERT berjalan (ketika ID belum tersedia), sedangkan kolom customer_id diperbarui oleh AFTER INSERT menggunakan input_email sebagai kunci pencocokan [17]. Pencatatan jejak kesalahan dalam tabel error_log mengikuti prinsip data provenance yakni menyimpan informasi tentang asal-usul dan riwayat transformasi data yang memungkinkan rekonstruksi kronologi kesalahan serta mendukung auditabilitas sistem secara keseluruhan [24]. Struktur lengkap tabel error_log disajikan pada Tabel 3.

Tabel 3 Struktur tabel error_log

Kolom	Tipe Data	Nullable	Keterangan
id	INT AUTO_INCREMENT	Tidak	Primary key, identifikasi unik setiap entri log
customer_id	INT	Ya	ID customer; NULL saat BEFORE INSERT,

 ISSN:2302-8149
e-ISSN:2540-9719

Kolom	Tipe Data	Nullable	Keterangan
			diperbarui AFTER INSERT
error_source	ENUM	Tidak	Sumber deteksi: RULE_BASED (trigger) atau FUZZY (Python)
tabel_sumber	VARCHAR(50)	Tidak	Nama tabel asal: customer, product, orders, dst.
record_id	INT	Ya	ID record bermasalah (diisi setelah AFTER INSERT)
input_nama	VARCHAR(100)	Ya	Nilai nama yang diinput saat BEFORE INSERT pada tabel customer; NULL untuk tabel lain
input_email	VARCHAR(100)	Ya	Nilai email yang diinput saat BEFORE INSERT pada tabel customer; digunakan oleh AFTER INSERT trigger sebagai kunci pencocokan untuk mengisi customer_id; NULL untuk tabel lain karena tidak memiliki kolom email
error_type	VARCHAR(50)	Tidak	Kode jenis error: EMPTY_NAME, PHONE_FORMAT, DUPLICATE_NAME, dst.
error_message	TEXT	Ya	Pesan deskriptif, menyertakan nilai bermasalah
similarity_score	FLOAT	Ya	Skor kemiripan fuzzy string (0–100); hanya terisi untuk deteksi kemiripan nama pada tabel category, customer, dan product
created_at	TIMESTAMP	Tidak	Waktu pencatatan, default CURRENT_TIMESTAMP

Kolom similarity_score hanya terisi untuk error yang berasal dari deteksi kemiripan nama (fuzzy string matching) pada tabel category, customer, dan product. Untuk deteksi relasional pada tabel orders, order_detail, dan payment, kolom ini bernilai NULL karena tidak melibatkan perhitungan skor kemiripan string. Kolom error_source menunjukkan layer deteksi: nilai RULE_BASED untuk trigger SQL, dan FUZZY untuk seluruh deteksi oleh modul Python termasuk deteksi relasional yang secara teknis bukan fuzzy string matching.

3.6 Dataset Pengujian

Dataset pengujian dibangkitkan secara sintetis menggunakan script Python generate_dataset.py yang dirancang untuk menghasilkan data dengan berbagai jenis kesalahan yang terkontrol. Pembangkitan data sintetis dipilih karena memungkinkan pengendalian penuh atas jenis dan jumlah kesalahan yang disisipkan, sehingga hasil deteksi sistem dapat diverifikasi secara tepat [23]. Kualitas data sintetis dapat dievaluasi melalui beberapa dimensi termasuk fidelitas distribusi atribut dan konsistensi antar kolom untuk memastikan data yang dibangkitkan merepresentasikan karakteristik data nyata secara terkontrol [22], [26].

Untuk mensimulasikan kesalahan penulisan nama yang realistis, script menggunakan lima jenis mutasi: (1) typo_swap menukar dua karakter yang berdekatan (contoh: "Budi" → "Bdui"), (2) typo_missing_char menghilangkan satu karakter ("Santoso" → "Santso"), (3) typo_double_char menduplikasi satu karakter ("Wijaya" → "Wiijaya"), (4) nickname mengambil nama depan saja ("Budi Santoso" → "Budi"), dan (5) short_name mengubah nama depan menjadi inisial ("Budi Santoso" → "B. Santoso"). Variasi mutasi ini menghasilkan data yang cukup mirip untuk melewati validasi rule-based namun tetap terdeteksi oleh fuzzy matching [10], [16]. Komposisi lengkap dataset pengujian disajikan pada Tabel 4.

Tabel 4 Komposisi dataset pengujian

Tabel	Total Record	Data Bersih	Data Bermasalah	Jenis Kesalahan Disisipkan

 ISSN:2302-8149
e-ISSN:2540-9719

Tabel	Total Record	Data Bersih	Data Bermasalah	Jenis Kesalahan Disisipkan
category	10	5	5	Nama kosong (2), nama duplikat/typo (3)
product	138	128	10	Harga invalid (5), nama duplikat/typo (5)
customer	850	600	250	Fuzzy duplikat disisipkan (150), terdeteksi (112), phone format invalid (40), phone terlalu pendek (70: 30 murni + 40 overlap PHONE_FORMAT), lahir masa depan (20), nama kosong (10)
orders	408	398	10	Order duplikat identik (10)
order_detail	1.048	1.020	28	Qty nol/negatif (10), harga negatif (5), item duplikat (8), orphan (5)
payment	321	308	13	Double payment (8), orphan payment (5)

Dataset terdiri dari total 2.775 record yang tersebar di enam tabel utama. Tabel customer menjadi fokus utama pengujian dengan 850 record, di mana 250 di antaranya mengandung kesalahan yang sengaja disisipkan. Tabel product memiliki 138 record karena script generate_dataset menyisipkan kategori bertipe mirip (3 nama typo) sebelum membangkitkan produk sehingga terdapat 8 kategori valid yang masing-masing menghasilkan 16 produk bersih (128 total), ditambah 5 fuzzy duplikat dan 5 produk harga invalid. Dari 250 record customer bermasalah, 150 merupakan duplikat fuzzy yang disisipkan dengan cara memutasi nama customer yang sudah ada menggunakan salah satu dari lima fungsi mutasi secara acak. Tidak seluruh 150 record tersebut terdeteksi pada threshold 90%, karena beberapa fungsi mutasi terutama short_name yang menghasilkan format inisial seperti 'B. Santoso' menghasilkan skor kemiripan di bawah threshold. Perlu dicatat bahwa generate_dataset.py tidak menggunakan fixed random seed, sehingga hasil eksekusi tidak sepenuhnya reproducible jumlah record yang berhasil diinsert maupun yang terdeteksi dapat bervariasi antar eksekusi tergantung pada urutan acak pembangkitan data. Pada pengujian ini, angka yang dilaporkan merupakan hasil dari satu siklus eksekusi penuh. Untuk keperluan reproducibility, penelitian selanjutnya disarankan menambahkan random.seed() dengan nilai tetap di awal fungsi main() pada generate_dataset.py sebelum pemanggilan fungsi insert pertama, sehingga seluruh urutan acak dapat direproduksi secara identik. Dari 150 yang disisipkan, 112 berhasil terdeteksi pada pengujian ini, sebuah skenario yang merepresentasikan kondisi nyata di mana pengguna memasukkan nama yang sama dengan penulisan sedikit berbeda [20]. Selain itu, jumlah entri error_log (472) lebih besar dari jumlah record yang dikarantina (297) karena dua alasan: pertama, satu record dapat memicu lebih dari satu aturan validasi sekaligus misalnya nomor HP yang mengandung karakter non digit sekaligus memiliki panjang kurang dari 10 digit akan menghasilkan dua entri error_log (PHONE_FORMAT dan PHONE_LENGTH) dari satu record yang sama; kedua, trigger SQL pada beberapa tabel bersifat log only mencatat kesalahan ke error_log tanpa langsung memindahkan record, sementara pemindahan ke tabel karantina dilakukan oleh modul Python pada tahap berikutnya.

4 Hasil dan Pembahasan

4.1 Hasil Deteksi Keseluruhan

Pengujian sistem dilakukan dengan menjalankan modul rule-based (SQL trigger) dan modul Python secara berurutan terhadap dataset sintetis yang telah disiapkan menggunakan script generate_dataset.py. Trigger SQL aktif secara otomatis pada saat proses insert dataset, sementara script hybrid_detection.py dijalankan setelah seluruh data selesai diinput. Secara keseluruhan, sistem

 ISSN:2302-8149
e-ISSN:2540-9719

berhasil mencatat 472 entri pada tabel error_log dan memindahkan 297 record ke enam tabel karantina. Tabel 5 menyajikan rekapitulasi hasil deteksi secara lengkap per tabel, per metode, dan per jenis error.

Tabel 5 Rekapitulasi hasil deteksi sistem hybrid rule-based

Tabel	Metode	Jenis Error	Error Log	Karantina	Tabel Karantina
category	Python (Pemeriksaan Kondisi)	EMPTY_CATEGORY_NAME	2	2	invalid_category
category	Python (Fuzzy String)	DUPLICATE_CATEGORY_NAME	3	3	invalid_category
customer	Rule-Based	EMPTY_NAME	10	10	invalid_customer
customer	Rule-Based	PHONE_FORMAT	40	40	invalid_customer
customer	Rule-Based	PHONE_LENGTH	70	70	invalid_customer
customer	Rule-Based	DUPLICATE_PHONE	68	0	invalid_customer
customer	Rule-Based	INVALID_BIRTHDATE	20	20	invalid_customer
customer	Python (Fuzzy String)	DUPLICATE_NAME	112	112	invalid_customer
product	Rule-Based	INVALID_PRICE	4	4	invalid_product
product	Python (Relasional)	INVALID_PRICE	5	5	invalid_product
product	Python (Fuzzy String)	DUPLICATE_PRODUCT_NAME	7	4	invalid_product
orders	Rule-Based	DUPLICATE_ORDER	10	0	invalid_orders
orders	Python (Relasional)	DUPLICATE_ORDER	9	9	invalid_orders
order_detail	Rule-Based	INVALID_QTY	10	6	invalid_order_detail
order_detail	Rule-Based	INVALID_PRICE	42	3	invalid_order_detail
order_detail	Rule-Based	DUPLICATE_ITEM	8	3	invalid_order_detail
order_detail	Rule-Based	INVALID_ORDER	5	0	invalid_order_detail
order_detail	Python (Relasional)	INVALID_QTY	4	4	invalid_order_detail
order_detail	Python (Relasional)	INVALID_PRICE	3	3	invalid_order_detail
order_detail	Python (Relasional)	DUPLICATE_ITEM	6	0	invalid_order_detail
order_detail	Python (Relasional)	ORPHAN_DETAIL	3	3	invalid_order_detail

 ISSN:2302-8149
e-ISSN:2540-9719

Tabel	Metode	Jenis Error	Error Log	Karantina	Tabel Karantina
payment	Rule-Based	DOUBLE_PAYMENT	8	0	invalid_payment
payment	Rule-Based	DUPLICATE_PAYMENT	8	0	invalid_payment
payment	Rule-Based	INVALID_AMOUNT	3	0	invalid_payment
payment	Rule-Based	INVALID_ORDER	5	0	invalid_payment
payment	Python (Relasional)	DOUBLE_PAYMENT	2	2	invalid_payment
payment	Python (Relasional)	ORPHAN_PAYMENT	5	5	invalid_payment
-	**TOTAL**	**-**	**472**	297	**6 tabel**

Pada tabel customer, sistem berhasil mendeteksi dan mencatat 320 record bermasalah di error_log, yang sebagian besar disalin ke invalid_customer oleh trigger. Sebanyak 208 entri error_log dihasilkan oleh rule-based trigger dari 168 record unik, mencakup 10 nama kosong, 40 format nomor HP tidak valid, 70 nomor HP terlalu pendek (40 di antaranya juga melanggar format karakter sehingga memicu dua aturan sekaligus), 68 nomor HP terdaftar duplikat, dan 20 tanggal lahir di masa depan. Sisa 112 record merupakan duplikat nama yang terdeteksi oleh fuzzy string matching dengan threshold 90%. Dari 850 data customer yang diinput, 638 record tersisa dengan status VALID setelah seluruh proses deteksi selesai. Jumlah entri error_log customer (320) lebih besar dari jumlah unique record yang benar-benar meninggalkan tabel customer karena dua faktor: pertama, 40 record yang melanggar PHONE_FORMAT sekaligus memicu PHONE_LENGTH menghasilkan dua entri error_log per record; kedua, 68 entri DUPLICATE_PHONE mencatat customer dari pool data bersih yang nomor HP nya bertabrakan secara acak, namun status_validasi record tersebut tidak diperbarui sehingga masih terhitung VALID sehingga 850−638 = 212 record yang benar-benar keluar dari tabel customer.

Pada tabel category, modul Python mendeteksi 5 record bermasalah: 2 kategori dengan nama kosong dan 3 kategori dengan nama yang terlalu mirip dengan kategori yang sudah ada menggunakan fuzzy string matching threshold 85% [10]. Pada Tabel 5, baris EMPTY_CATEGORY_NAME berlabel 'Python (Pemeriksaan Kondisi)' untuk membedakannya dari baris fuzzy string matching deteksi nama kosong menggunakan pemeriksaan kondisi eksak (nama = NULL atau nama = '') bukan algoritma kemiripan string, sehingga pelabelannya dibedakan meskipun keduanya dijalankan oleh modul Python yang sama. Skor kemiripan: "Eletronik" ↔ "Elektronik" (94,7%), "Pakaiann" ↔ "Pakaian" (93,3%), dan "Maknan" ↔ "Makanan" (92,3%).

Pada tabel product, terdeteksi 16 entri di error_log dengan 8 record dipindahkan ke invalid_product. Rule-based trigger mencatat 4 produk dengan harga negatif saat INSERT (kondisi harga < 0). Modul Python kemudian mendeteksi 5 produk dengan harga nol yang lolos trigger (kondisi harga <= 0), serta 7 produk dengan nama mirip melalui fuzzy string matching termasuk satu pasang "Guling Cosmos 300W" ↔ "Guling Cosmos 500W" (94,4%) yang merupakan contoh false positive di mana perbedaan angka spesifikasi tidak dapat dibedakan oleh algoritma kemiripan string [22]. Dari 7 produk yang tercatat di error_log, hanya 4 yang berhasil dipindahkan ke invalid_product. Tiga sisanya tidak ditemukan saat proses karantina berjalan karena sudah lebih dulu dihapus melalui cascade delete yang dipicu oleh karantina category induknya menunjukkan bahwa urutan eksekusi deteksi antar tabel dapat mempengaruhi jumlah record yang akhirnya dikarantina.

Pada tabel order_detail, terdapat temuan yang patut dicatat: rule-based trigger mencatat 42 entri INVALID_PRICE, jauh di atas 5 record yang sengaja disisipkan. Investigasi menunjukkan bahwa hal ini merupakan konsekuensi logis dari desain trigger yang bersifat log only: produk dengan harga negatif tetap tersimpan di tabel product saat INSERT karena trigger hanya mencatat ke error_log tanpa memblokir data. Ketika generate_dataset membangkitkan order_detail, produk-produk tersebut masih ada di tabel dan ikut terpilih sebagai referensi harga_satuan. Setelah modul Python memindahkan produk invalid ke invalid_product, order_detail yang sebelumnya

 ISSN:2302-8149
e-ISSN:2540-9719

mereferensikan produk tersebut menjadi orphan dan terdeteksi pada tahap detect_orphan_records(). Ini menunjukkan bagaimana desain log-only pada trigger menciptakan jendela waktu antara pencatatan error dan pembersihan data yang dapat menghasilkan efek berantai di tabel hilir. Modul Python kemudian memindahkan total 16 record order_detail ke invalid_order_detail, mencakup item dengan qty tidak valid, harga negatif, item duplikat, dan orphan detail record dengan order_id yang tidak ada di tabel orders. Enam entri DUPLICATE_ITEM yang terdeteksi modul Python tidak menghasilkan karantina karena record duplikat tersebut sudah lebih dulu dihapus oleh rule-based trigger pada saat INSERT, sehingga saat Python menjalankan pemindahan, record tidak lagi ditemukan di tabel utama.

Pada tabel orders, rule-based trigger mencatat 10 order duplikat di error_log namun record tetap tersimpan di tabel utama karena trigger bersifat log only. Modul Python kemudian mendeteksi dan memindahkan 9 order duplikat ke invalid_orders menggunakan deteksi relasional (exact match kombinasi customer_id + tanggal + total). Perbedaan satu record disebabkan oleh cascade delete yang terjadi sebelum Python menjalankan deteksi.

Pada tabel payment, rule-based trigger mencatat 24 entri dari empat jenis error: 8 double payment, 8 duplicate payment, 3 jumlah tidak valid, dan 5 orphan payment. Kedua label ini merujuk pada dua pemeriksaan yang berbeda dalam satu modul deteksi: DOUBLE_PAYMENT mendeteksi order yang memiliki lebih dari satu pembayaran berstatus SUCCESS, sedangkan DUPLICATE_PAYMENT mendeteksi kombinasi order_id + jumlah + metode yang identik meskipun statusnya berbeda. Terdapat kemungkinan overlap antara keduanya pada record yang sama. Tiga entri INVALID_AMOUNT berasal dari orphan payment yang dibangkitkan dengan jumlah nol, sedangkan 5 entri INVALID_ORDER berasal dari orphan payment dengan order_id yang tidak ada di tabel orders keduanya merupakan efek samping dari proses insert orphan payment yang menonaktifkan FK check sementara, sehingga trigger tetap mencatat pelanggaran meskipun record berhasil masuk. Semua dicatat di error_log namun record tetap tersimpan. Modul Python selanjutnya mendeteksi 2 double payment tambahan dan 5 orphan payment pembayaran dengan order_id yang tidak ada di tabel orders, sehingga pembayaran tersebut tidak dapat dikaitkan ke transaksi manapun. Total 7 record dipindahkan ke invalid_payment.

4.2 Evaluasi Kinerja Sistem

Untuk mengevaluasi kinerja sistem secara formal, dihitung precision, recall, dan F1-score berdasarkan komposisi dataset yang telah diketahui secara terkontrol. Karena dataset sintetis dibangkitkan dengan jumlah kesalahan yang telah ditentukan sebelumnya, nilai true positive, false positive, dan false negative dapat dihitung secara langsung.

True positive (TP) adalah kesalahan yang sengaja disisipkan dan berhasil terdeteksi. False negative (FN) adalah kesalahan yang disisipkan namun tidak terdeteksi dalam pengujian ini total 78 FN terdiri dari: 38 duplikat nama customer yang skor kemiripannya di bawah threshold 90% akibat mutasi short_name, 39 kesalahan pada tabel orders dan order_detail yang lolos dari rule-based trigger karena disisipkan dengan menonaktifkan sql_mode sementara pada saat INSERT, sehingga trigger tidak dapat mengevaluasi kondisi validasi dan baru dideteksi oleh modul Python pada tahap berikutnya, dan 1 efek cascade pada Python relasional.

False positive (FP) adalah record yang dikarantina padahal tidak termasuk kesalahan yang disisipkan — dalam kasus ini hanya mencakup kasus Guling Cosmos 300W/500W dari Python fuzzy matching. Adapun 68 DUPLICATE_PHONE meskipun tercatat di error_log tidak dikategorikan sebagai FP karena record tersebut tidak dikarantina. Tabel 6 menyajikan metrik evaluasi per komponen sistem.

Tabel 6 Metrik evaluasi

Komponen	TP	FP	FN	Precision	Recall	F1-Score
Rule-Based	161	0	39	100,0%	80,5%	89,2%
Python Fuzzy	120	2	38	98,4%	75,9%	85,7%
Python Relasional	26	0	1	100,0%	96,3%	98,1%

 ISSN:2302-8149
e-ISSN:2540-9719

Komponen	TP	FP	FN	Precision	Recall	F1-Score
Keseluruhan	**307**	**2**	**78**	**99,4%**	**79,7%**	**88,5%**

Evaluasi formal menunjukkan precision 99,4%, recall 79,7%, dan F1-score 88,5% untuk keseluruhan sistem dengan false positive algoritmik hanya 2 record dari Python fuzzy matching (termasuk kasus Guling Cosmos 300W/500W), sementara 51 entri tambahan merupakan artefak desain sistem dan urutan eksekusi yang telah diidentifikasi sebagai keterbatasan penelitian. Di luar FP algoritmik tersebut, terdapat 51 entri yang muncul akibat karakteristik desain sistem: 37 INVALID_PRICE order_detail dari efek cascade log-only design, 8 DUPLICATE_PAYMENT dari overlap deteksi, dan 6 DUPLICATE_ITEM dari efek urutan eksekusi. Adapun 68 DUPLICATE_PHONE tidak termasuk dalam kategori ini karena record tersebut memang tidak dikarantina — hanya dicatat di error_log dengan status_validasi tetap VALID di tabel utama. Ketiganya bukan merupakan kesalahan algoritmik dan dapat dieliminasi dengan perbaikan desain dataset dan skema deteksi.

4.3 Efektivitas Pendekatan Hybrid

Perbandingan karakteristik antara rule-based validation dan modul Python disajikan pada Tabel 7. Kedua komponen memiliki kelebihan dan keterbatasan yang berbeda namun saling melengkapi, menghasilkan sistem deteksi yang lebih komprehensif dibandingkan penggunaan satu metode saja [9], [23]. Hubungan antara pendekatan rule-based dan machine learning dalam data cleaning bersifat simbiotik: ML dapat mengotomasi deteksi pola kesalahan kompleks, sementara aturan eksplisit tetap diperlukan untuk memastikan konsistensi yang dapat diaudit dan dijelaskan [12], [23].

Tabel 7 Perbandingan rule-based validation dan modul python

Aspek	Rule-Based Validation	Modul Python
Implementasi	SQL Trigger BEFORE INSERT pada MariaDB	Script Python hybrid_detection.py + library RapidFuzz
Waktu deteksi	Real-time aktif setiap operasi INSERT	Periodik dijalankan secara batch setelah data masuk
Jenis kesalahan	Terstruktur dan terdefinisi eksplisit	Semantik (fuzzy string) untuk category/customer/product; relasional (exact match) untuk orders/order_detail/payment
Contoh deteksi	Format HP tidak valid, FK violation, nilai negatif	Nama mirip akibat typo; orphan record; double payment
Algoritma deteksi	Pengecekan kondisi logika (regexp, EXISTS, perbandingan nilai)	token_sort_ratio (fuzzy string) atau query relasional (exact)
Threshold	Boolean melanggar aturan atau tidak	Skor ≥ 90% (customer & product), ≥ 85% (category); exact match untuk tabel lain
Mekanisme karantina	Log ke error_log; flag RULE_ERROR; Modul python memindahkan ke tabel invalid_*	Deteksi → log → pindahkan ke tabel invalid_*
Keterbatasan	Tidak dapat mendeteksi kesalahan semantik atau ambigu	Fuzzy string hanya relevan untuk kolom nama; deteksi relasional tidak menangkap variasi penulisan
Kontribusi error_log	311 entri (65,9% dari total 472)	161 entri (34,1%): 127 dari fuzzy string, 34 dari deteksi relasional

 ISSN:2302-8149
e-ISSN:2540-9719

Aspek	Rule-Based Validation	Modul Python
Cakupan tabel	Semua 6 tabel utama	Semua 6 tabel utama

Dari total 472 entri di error_log, rule-based menyumbang 311 entri (65,9%) yang mencakup kesalahan di semua enam tabel, sedangkan modul Python menyumbang 161 entri (34,1%) terdiri dari 127 entri dari fuzzy string matching pada tabel category, customer, dan product, serta 34 entri dari deteksi relasional pada tabel orders, order_detail, dan payment. Jika hanya mengandalkan rule-based, seluruh 112 duplikat nama customer, 7 produk nama mirip, dan 9 order duplikat yang diidentifikasi Python tidak akan terkarantina. Sebaliknya, jika hanya mengandalkan modul Python, 311 kesalahan terstruktur tidak akan terdeteksi pada saat data masuk. Meskipun secara teknis Python dapat melakukan pemeriksaan yang serupa validasi format, pengecekan duplikat eksak, maupun pelanggaran referensial perbedaan mendasarnya terletak pada enforcement level: trigger berjalan di dalam database engine dan tidak dapat di bypass oleh aplikasi manapun, sedangkan modul Python yang berjalan secara periodik rentan terhadap data yang masuk di luar jadwal eksekusinya. Keunggulan trigger bukan hanya soal kecepatan, melainkan soal jaminan bahwa setiap INSERT akan selalu diperiksa tanpa pengecualian [7], [8]. Oleh karena itu, kedua komponen tersebut memiliki peran yang saling melengkapi, dimana trigger bertugas menjaga integritas data pada saat data masuk ke database, sedangkan modul Python berfokus pada identifikasi anomali semantik dan relasional yang tidak dapat dideteksi secara efektif menggunakan aturan berbasis SQL.

Hasil pengujian juga mengungkap karakteristik penting. Threshold 85% pada category tepat menangkap typo 1–2 karakter (skor 92–95%) tanpa false positive. Threshold 90% pada customer menyaring 112 duplikat nama nyata, termasuk kasus nama identik dengan spasi ganda dan inisial berbeda. Satu temuan penting adalah terdeteksinya "Guling Cosmos 300W" ↔ "Guling Cosmos 500W" sebagai duplikat menunjukkan bahwa fuzzy string matching tidak mampu membedakan angka spesifikasi yang secara semantik membedakan dua produk. Hasil ini menunjukkan bahwa pendekatan fuzzy string matching bekerja berdasarkan kemiripan karakter antar string dan belum mempertimbangkan konteks atribut numerik yang terkandung dalam nama produk. Oleh karena itu, produk dengan merek yang sama namun memiliki spesifikasi berbeda masih berpotensi terdeteksi sebagai duplikat meskipun secara bisnis merupakan entitas yang berbeda. Deteksi duplikat berbasis kemiripan string merupakan salah satu pendekatan klasik dalam persoalan entity matching yakni mengidentifikasi record berbeda yang merujuk ke entitas nyata yang sama yang telah dikaji secara luas dari pendekatan rule-based hingga metode berbasis neural network [17], [21]. Ini mengindikasikan perlunya validasi lanjutan untuk domain produk dengan variasi spesifikasi numerik [10], [22].

Mekanisme karantina memberikan nilai tambah penting: 297 record bermasalah tersimpan di tabel invalid_* lengkap dengan jenis error, sumber deteksi, skor kemiripan untuk kasus fuzzy string, dan waktu karantina. Ini mendukung prinsip traceability dalam data quality management [17], [25]. seluruh data yang pernah ditolak sistem dapat diaudit kapan saja, lengkap dengan jejak deteksinya.

Secara keseluruhan, sistem hybrid berhasil mendeteksi 472 entri error dari enam tabel relasional dalam satu siklus eksekusi penuh dengan 297 record dikarantina. Hasil ini menunjukkan bahwa integrasi SQL trigger dan Python dengan RapidFuzz merupakan solusi data quality management yang efektif, komprehensif, dan dapat dilacak pada sistem database relasional multi tabel [9], [23].

5 Kesimpulan

Penelitian ini merancang dan mengimplementasikan sistem Hybrid Rule-Based untuk validasi dan deteksi kesalahan data pada database relasional multi tabel, menggabungkan SQL trigger BEFORE INSERT dengan modul Python RapidFuzz menggunakan algoritma token_sort_ratio. Pengujian pada dataset sintetis 2.775 record di enam tabel menghasilkan 472 entri error_log dan 297 record dikarantina. Rule-based berkontribusi 311 entri (65,9%) dan modul Python 161 entri (34,1%), terdiri dari 127 entri fuzzy string matching dan 34 entri deteksi relasional. Evaluasi formal menunjukkan precision 99,4%, recall 79,7%, F1-score 88,5% untuk keseluruhan sistem. Tanpa komponen Python, 127 kesalahan semantik tidak akan terdeteksi karena tidak melanggar constraint SQL apapun; tanpa trigger, 311 kesalahan terstruktur tidak akan tertangkap secara real time karena trigger memberikan jaminan enforcement di level database engine yang tidak dapat di bypass lapisan

 ISSN:2302-8149
e-ISSN:2540-9719

aplikasi manapun. Mekanisme karantina memastikan data bermasalah tersimpan lengkap dengan jenis error, sumber deteksi, dan skor kemiripan untuk mendukung prinsip traceability [17], [25]. Penelitian ini memiliki beberapa keterbatasan yang perlu diakui: (1) pengujian dilakukan pada dataset sintetis terkontrol sehingga generalisasi ke data produksi nyata memerlukan validasi lebih lanjut; (2) generate_dataset.py tidak menggunakan fixed random seed sehingga hasil tidak sepenuhnya reproducible; (3) label error_source hanya memiliki dua nilai ENUM (RULE_BASED dan FUZZY) sehingga deteksi relasional Python tidak dapat dibedakan dari deteksi fuzzy string secara skematis; dan (4) 68 DUPLICATE_PHONE tercatat di error_log akibat collision nomor acak tanpa fixed seed, namun tidak dikarantina karena bukan merupakan kesalahan yang disengaja disisipkan. Untuk penelitian selanjutnya, disarankan pengujian pada dataset nyata skala produksi, penambahan nilai ENUM PYTHON_RELATIONAL pada error_source, penerapan fixed random seed untuk reproducibility, serta eksplorasi metode yang mempertimbangkan konteks semantik atribut numerik untuk mengurangi false positive pada nama produk dengan variasi spesifikasi.

Referensi

[1] R. Miller, S. H. M. Chan, H. Whelan, and J. Gregório, "*A Comparison of Data Quality Frameworks: A Review*," *Big Data and Cognitive Computing*, Vol. 9, No. 4, Apr. 2025, DOI: 10.3390/bdcc9040093.

[2] M. Ibrahim, Y. Helmy, and D. Elzanfaly, "*Data Quality Dimensions, Metrics, and Improvement Techniques*," *Future Computing and Informatics Journal*, Vol. 6, No. 1, pp. 1–12, 2021, DOI: 10.54623/fue.fcij.6.1.3.

[3] F. Ridzuan and W. M. N. W. Zainon, "*A Review on Data Quality Dimensions for Big Data*," *Procedia Comput. SCI.*, Vol. 234, No. 1, pp. 341–348, 2024, DOI: 10.1016/j.procs.2024.03.008.

[4] J. Wang, Y. Liu, P. Li, Z. Lin, S. Sindakis, and S. Aggarwal, "*Overview of Data Quality: Examining the Dimensions, Antecedents, and Impacts of Data Quality*," *Journal of the Knowledge Economy*, Vol. 15, No. 1, pp. 1159–1178, Mar. 2024, DOI: 10.1007/s13132-022-01096-6.

[5] J. Merino, I. Caballero, B. Rivas, M. Serrano, and M. Piattini, "*A Data Quality in Use Model for Big Data*," *Future Generation Computer Systems*, Vol. 63, pp. 123–130, 2016, DOI: 10.1016/j.future.2015.11.024.

[6] A. Yulianto and Firmansyah, "*Data Improvement Life Cycle untuk Meningkatkan Kualitas Data: Studi Kasus Data Survey Kesehatan Mental*," *Remik*, Vol. 9, No. 2, pp. 474–483, 2025, DOI: 10.33395/remik.v9i2.14643.

[7] P.-O. Côté, A. Nikanjam, N. Ahmed, D. Humeniuk, and F. Khomh, "*Data Cleaning and Machine Learning: A Systematic Literature Review*," *Automated Software Engineering*, Vol. 31, No. 54, May 2024, DOI: 10.1007/s10515-024-00453-w.

[8] M. Aidjili, "Implementasi *Trigger* dan *View* untuk Mendukung Konsistensi dan Efisiensi Pengolahan Data pada Sistem Database (Study Kasus: Toko Nanda Pekalongan)," *j.komputer, j.informasi, j.teknologi*, Vol. 5, No. 2, pp. 1–12, Dec. 2025, DOI: 10.53697/jkomitek.v5i2.39.

[9] X. Chu, I. F. Ilyas, S. Krishnan, and J. Wang, "*Data Cleaning: Overview and Emerging challenges*," in *Proceedings of the ACM SIGMOD International Conference on Management of Data*, Association for Computing Machinery, Jun. 2016, pp. 2201–2206. DOI: 10.1145/2882903.2912574.

[10] A. R. Kaufman and A. Klevs, "*Adaptive Fuzzy String Matching: How to Merge Datasets with Only One (Messy) Identifying Field*," *Political Analysis*, Vol. 30, No. 4, pp. 590–596, Oct. 2022, DOI: 10.1017/pan.2021.38.

[11] N. Elmobark, "*A Comparative Analysis of Python Text Matching Libraries: A Multilingual Evaluation of Capabilities, Performance and Resource Utilization*," *International Journal of Environment, Engineering and Education*, Vol. 7, No. 1, pp. 48–60, Apr. 2025, DOI: 10.55151/ijeedu.v7i1.188.

[12] Y. Gao, C. Ge, X. Miao, H. Wang, B. Yao, and Q. Li, "*A Hybrid Data Cleaning Framework using Markov Logic Networks*," *arXiv preprint*, Mar. 2019, DOI: 10.48550/arXiv.1903.05826.

[13] S. Chaudhuri, K. Ganjam, V. Ganti, and R. Motwani, "*Robust and Efficient Fuzzy Match for Online Data Cleaning*," in *Proceedings of the 2003 ACM SIGMOD international conference on Management of data*, Association for Computing Machinery, Jun. 2003, pp. 313–324. DOI: 10.1145/872757.872796.

[14] J. Stoikov, A. Nikolova, and V. Georgiev, "*Advanced Record Linkage Techniques for Improving the Data Matching between Cultural Heritage Datasets from Different Sources," TEM Journal, Vol. 11, No. 4, pp. 1906–1914, Nov. 2022, DOI: 10.18421/TEM114-59.*

[15] E. Eessaar, "*The Usage of Declarative Integrity Constraints in the SQL Databases of Some Existing Software*," in *Software Engineering and Algorithms*, R. Silhavy, Ed., Springer International Publishing, Jul. 2021, pp. 375–390. DOI: 10.1007/978-3-030-77442-4_33.

[16] F. M. Wibowo, M. Z. Nafan, M. A. Gustalika, H. Fernando, M. Hussain, and N. A. Sahadun, "*Lightweight String Similarity Approaches for Duplicate Detection in Academic Titles*," *Journal of Informatics and Web Engineering*, Vol. 4, No. 3, pp. 416–426, Oct. 2025, DOI: 10.33093/jiwe.2025.4.3.25.

[17] A. K. Elmagarmid, P. G. Ipeirotis, and V. S. Verykios, "*Duplicate Record Detection: A Survey*," *IEEE Trans. Knowl. Data Eng.*, Vol. 19, No. 1, pp. 1–16, 2007, DOI: 10.1109/TKDE.2007.250581.

[18] I. F. Ilyas and X. Chu, "*Trends in Cleaning Relational Data: Consistency and Deduplication*," *Foundations and Trends in Databases*, Vol. 5, no. 4, pp. 281–393, Oct. 2015, DOI: 10.1561/1900000045.

[19] L. Cruz-Filipe, M. Franz, A. Hakhverdyan, M. Ludovico, I. Nunes, and P. Schneider-Kamp, "*repAIrC: A Tool for Ensuring Data Consistency by Means of Active Integrity Constraints*," *arXiv preprint*, Oct. 2015, DOI: 10.5220/0005586400170026.

[20] O. Azeroual, M. Jha, A. Nikiforova, K. Sha, M. Alsmirat, and S. Jha, "*A Record Linkage-based Data Deduplication Framework with DataCleaner Extension*," *Multimodal Technologies and Interaction*, Vol. 6, No. 4, p. 27, Apr. 2022, DOI: 10.3390/mti6040027.

[21] N. Barlaug and J. A. Gulla, "*Neural Networks for Entity Matching: A Survey*," *ACM Trans. Knowl. Discov. Data*, Vol. 15, No. 3, pp. 1–37, Apr. 2021, DOI: 10.1145/3442200.

[22] J. Fu, X. Han, X. Wan, and W. Wang, "*PAT: Pattern-Perceptive Transformer for Error Detection in Relational Databases*," *arXiv preprint*, Sep. 2025, DOI: 10.48550/arXiv.2509.25907.

[23] I. F. Ilyas and T. Rekatsinas, "*Machine Learning and Data Cleaning: Which Serves the Other?*," *Journal of Data and Information Quality*, Vol. 14, No. 3, Sep. 2022, DOI: 10.1145/3506712.

[24] M. Herschel, R. Diestelkämper, and H. Ben Lahmar, "*A Survey on Provenance: What For? What Form? What From?*," *The VLDB Journal*, Vol. 26, pp. 881–906, Oct. 2017, DOI: 10.1007/s00778-017-0486-1.

[25] E. Cahyaningsih, A. Rinjatmoko, and W. P. Sari, "Pengukuran Kualitas Data menggunakan *Framework* Total Data *Quality Management*: Studi Kasus Kementerian Hukum dan Hak Asasi Manusia Rutan Klas I Jakarta Pusat," *Jurnal Teknologi Informasi dan Ilmu Komputer*, Vol. 12, No. 1, pp. 121–132, Feb. 2025, DOI: 10.25126/jtiik.2025129178.

[26] F. K. Dankar, M. K. Ibrahim, and L. Ismail, "*A Multi-Dimensional Evaluation of Synthetic Data Generators*," *IEEE Access*, Vol. 10, pp. 1–1, Jan. 2022, DOI: 10.1109/ACCESS.2022.3144765.