Journal of Integrated Engineering and Applied Technology, Volume 1, Number 1, 2026, pp.27-39.
https://journalndcendekia.ransel.in/index.php/jieat
DOI. https://doi.org/10.32627/xxxxx

ISSN xxxx-xxxx (online)
ISSN xxxx-xxxx (print)

PENGGUNAAN VIEW DAN STORED PROCEDURE MYSQL UNTUK PENYEDERHANAAN LOGIKA BISNIS PADA APLIKASI MANAJEMEN INVENTORI

Aan Ansen Andriadi[1], Sandi Nazril[2*]
[1,2] Program Studi Sistem Informasi, Fakultas Teknologi Informasi Universitas Al-Ghifari
*Corresponding author, *e-mail*: 244260001.mhs@stmikjabar.ac.id

Abstract: In the ecosystem of collaborative information system development, particularly within academic and industrial research scopes, the paramount challenge often lies not in interface design, but in maintaining data integrity. A fundamental issue frequently overlooked in scholarly research and development practices is the inefficient placement of business logic; a vast majority of stock validation and transaction calculation logic remain hardcoded within the application layer. This approach leads to code redundancy, increased susceptibility to human error during collaborative development, and performance degradation when processing massive transactional data loads. This study aims to propose a robust database architecture solution by migrating the burden of business logic from the application layer to the database layer. The methodology employed is a comparative experiment, wherein an inventory management system prototype was constructed using two distinct approaches: a standard application-based query approach and a centralized approach utilizing View and Stored Procedure features in MySQL 8.0. Experimental results reveal significant findings. The implementation of Views proved capable of simplifying reporting query complexity, drastically reducing code lines on the application side, while the utilization of Stored Procedures successfully encapsulated the item mutation process within a secure, single atomic transaction. This effectively prevents data anomalies that frequently occur due to mid-process failures. In conclusion, this strategy of centralizing business logic not only enhances inventory data accuracy but also facilitates developer collaboration by establishing a more independent and consistent database structure.

Keywords: inventory management; collaborative information systems; mySQL; stored procedure; business logic

Article history

Received:	**Revised:**	**Accepted:**	**Published:**
24 March 2025	*15 May 2025*	*24 May 2025*	*31 Januari 2026*

Citation (APA): Andriadi, A. A., & Nazril, S. (2026). Penggunaan View dan Stored Procedure MySQL untuk Penyederhanaan Logika Bisnis pada Aplikasi Manajemen Inventori. *Journal of Integrated Engineering and Applied Technology (JIEAT)*, 1(1), 27-39. https://doi.org/10.32627/jieat.v1i1.31

PENDAHULUAN

Dalam era transformasi digital yang semakin matang, sistem informasi manajemen inventori telah berevolusi dari sekadar alat pencatatan stok menjadi tulang punggung strategis bagi keberlangsungan operasional perusahaan. Kompleksitas rantai pasok modern menuntut sistem yang tidak hanya mampu menyimpan data, tetapi juga menjamin ketersediaan informasi yang akurat secara *real-time*. Pada lingkungan bisnis yang bergerak cepat, integritas data bukan lagi sekadar kebutuhan teknis, melainkan aset bisnis yang krusial. Namun, realitas pengembangan perangkat lunak sering kali menunjukkan kecenderungan arsitektural yang menempatkan "kecerdasan" sistem sepenuhnya pada lapisan aplikasi (*application layer*). Pengembang cenderung menuliskan seluruh logika bisnis, seperti validasi stok dan kalkulasi transaksi, menggunakan bahasa pemrograman tingkat tinggi.

Pendekatan *application-centric* ini memunculkan permasalahan serius berupa redundansi logika dan risiko inkonsistensi data. Teori basis data modern yang dikemukakan oleh Connolly dan Begg (2020) serta Elmasri dan Navathe (2021) sebenarnya menekankan pentingnya integritas data yang dikelola langsung oleh DBMS untuk menjamin sifat ACID (*Atomicity, Consistency, Isolation, Durability*). Ketika logika bisnis diletakkan di aplikasi, sistem kehilangan sifat atomisitasnya saat terjadi kegagalan jaringan, yang menyebabkan data menjadi korup. Oleh karena itu, diperlukan pergeseran paradigma menuju konsep "Smart Database" atau sistem basis data yang bersifat agentic, di mana basis data bertindak otonom memvalidasi dirinya sendiri.

MySQL sebagai RDBMS yang banyak digunakan menyediakan fitur prosedural yang mampu menjawab tantangan tersebut, yaitu View dan *Stored Procedure*. Secara teoritis, View berfungsi sebagai lapisan abstraksi yang menyederhanakan kompleksitas relasi antar-tabel (Date, 2020), sementara *Stored Procedure* memungkinkan enkapsulasi logika transaksi yang dieksekusi langsung di server basis data (Oracle, 2023). Hal ini didukung oleh temuan Al-Shargabi (2020) yang membuktikan bahwa memindahkan logika bisnis ke lapisan basis data secara signifikan menurunkan kompleksitas kode pada sisi aplikasi, sehingga sistem lebih mudah dipelihara.

Meskipun potensi teknis ini besar, tinjauan terhadap penelitian terdahulu memperlihatkan pemanfaatan yang belum optimal dalam konteks inventori. Penelitian Susanto dan Wibowo (2021) lebih berfokus pada algoritma genetika untuk prediksi stok tanpa membahas arsitektur penyimpanan. Sementara itu, Setiawan dan Pratama (2021) berhasil membuktikan bahwa penggunaan *Stored Procedure* meningkatkan kecepatan eksekusi data pada sistem akademik, namun belum menerapkannya pada validasi stok transaksional. Di sisi keamanan, Wahyuni dan Santoso (2023) memanfaatkan *trigger* untuk audit trail rekam medis, yang mengindikasikan bahwa mekanisme internal basis data lebih andal mencegah

manipulasi data dibanding validasi aplikasi. Namun, belum banyak literatur yang secara spesifik menggabungkan penggunaan *View* dan *Stored Procedure* untuk mereduksi kompleksitas logika bisnis sekaligus menjamin integritas data pada sistem inventori.

Berdasarkan kesenjangan penelitian tersebut, penelitian ini bertujuan untuk merancang dan mengimplementasikan mekanisme pengelolaan inventori yang memindahkan beban logika dari aplikasi ke database MySQL. Penelitian ini menawarkan kebaruan (*novelty*) berupa penerapan arsitektur sistem yang memanfaatkan *View* untuk penyederhanaan pelaporan dan Stored Procedure sebagai penjaga gawang integritas transaksi. Diharapkan, pendekatan ini dapat menghasilkan sistem yang lebih efisien, minim *bug*, dan memiliki ketahanan data yang superior.

METODE

A. Jenis dan Pendekatan Penelitian

Penelitian ini menggunakan jenis **penelitian terapan (*applied research*)** dengan pendekatan **eksperimental komparatif**. Pemilihan metode ini didasarkan pada tujuan penelitian untuk tidak hanya merancang solusi, tetapi juga mengukur efektivitas solusi tersebut secara empiris. Pendekatan eksperimental digunakan untuk membandingkan dua skenario perlakuan (*treatment*) terhadap logika bisnis inventori:

1. **Skenario A (Baseline):** Implementasi logika bisnis konvensional yang tertanam pada lapisan aplikasi (*Application-Layer Logic*).
2. **Skenario B (Proposed):** Implementasi logika bisnis yang dimigrasikan ke lapisan basis data menggunakan *View* dan *Stored Procedure* (*Database-Layer Logic*).

Menurut Wohlin et al. (2012) dalam konteks rekayasa perangkat lunak, pendekatan eksperimen terkontrol adalah metode paling valid untuk mengevaluasi dampak teknologi baru (dalam hal ini, *Stored Procedure*) terhadap kinerja dan pemeliharaan sistem dibandingkan teknologi lama.

B. Lokasi dan Waktu Penelitian

Penelitian dilakukan secara simulasi pada lingkungan laboratorium rekayasa data untuk meminimalisir variabel gangguan eksternal (*noise*) seperti fluktuasi jaringan internet publik. Lingkungan pengembangan dikonfigurasi menyerupai server produksi skala menengah (UKM). Penelitian ini dilaksanakan selama empat bulan, terhitung mulai Agustus 2024 hingga November 2024, mencakup tahapan desain skema, pembuatan *dummy data*, pengujian skenario, hingga analisis hasil komparasi.

C. Populasi dan Sampel

Dalam konteks penelitian rekayasa basis data, populasi merujuk pada himpunan data transaksi.

1. **Populasi:** Seluruh kemungkinan variasi transaksi inventori yang dapat terjadi dalam operasional ritel, termasuk pembelian (barang masuk), penjualan (barang keluar), dan penyesuaian stok (*adjustment*).
2. **Sampel:** Penelitian ini menggunakan teknik **generation sampling** untuk membuat *dataset* sintetis (dummy) sebanyak 10.000 *record* data transaksi. Sampel ini dibagi menjadi dua kategori pengujian:
   a. *Normal Case:* 9.000 transaksi valid untuk menguji fungsionalitas dan kecepatan.
   b. *Edge Case:* 1.000 transaksi anomali (seperti input stok negatif, ID barang tidak valid, dan input ganda) untuk menguji integritas data dan respons penolakan sistem.

D. Teknik Pengumpulan Data

Pengumpulan data dilakukan melalui dua teknik utama:

1. **Studi Dokumentasi Kode (*Code Metrics Observation*):** Mengukur kompleksitas kode program. Data yang dikumpulkan adalah jumlah baris kode (*Lines of Code* - LOC) yang diperlukan untuk menjalankan satu fungsi bisnis pada Skenario A versus Skenario B.
2. **Pengujian Kotak Hitam (*Black Box Testing*):** Mengamati *output* sistem ketika diberikan *input* tertentu tanpa melihat kode internal. Fokus pengumpulan data adalah pada status keberhasilan transaksi: apakah sistem berhasil mencegah data stok menjadi minus (-)? Data dicatat dalam bentuk *log* keberhasilan/kegagalan (Pass/Fail).

E. Instrumen Penelitian

Untuk menjamin keterulangan (reproducibility), penelitian ini menggunakan spesifikasi instrumen perangkat keras dan lunak sebagai berikut:

1. **Perangkat Keras:** Laptop Workstation dengan prosesor Intel Core i5 Gen-11, RAM 16GB, dan SSD 512GB NVMe.
2. **Perangkat Lunak:**
   a. Sistem Operasi: Windows 11 Pro 64-bit.
   b. DBMS: MySQL Server 8.0.30 Community Edition (mendukung InnoDB Engine).
   c. Database Client: MySQL Workbench 8.0 untuk eksekusi SQL dan *visual explain*.
   d. Tools Pengujian: Skrip PHP 8.1 sederhana untuk menyimulasikan pemanggilan dari sisi aplikasi.
3. **Metrik Pengukuran:** Instrumen pengukuran menggunakan tabel perbandingan yang mencakup parameter: (1) Jumlah baris instruksi SQL vs PHP, (2) Waktu respon eksekusi (*query cost*), dan (3) Konsistensi data akhir.

F. Teknik Analisis Data

Data yang terkumpul dianalisis menggunakan Analisis Deskriptif Komparatif. Teknik ini membandingkan hasil pengukuran dari Skenario A dan Skenario B untuk menarik kesimpulan. Langkah analisisnya adalah:

1. **Reduksi Data:** Mengelompokkan kode program yang relevan dengan fitur "Tambah Barang" dan "Laporan Stok". Kode pendukung (seperti CSS/HTML) diabaikan.

2. **Kalkulasi Efisiensi:** Menghitung persentase penurunan kompleksitas kode dengan rumus:

$$Efisiensi = \frac{LOC\ Skenario\ A - LOC\ Skenario\ B}{LOC\ Skenario\ A} \times 100\%$$

3. **Verifikasi Integritas:** Membandingkan jumlah data anomali yang lolos ke database. Jika pada Skenario B jumlah anomali adalah 0, maka hipotesis bahwa *Stored Procedure* meningkatkan integritas data terbukti.
4. **Penarikan Kesimpulan:** Menyimpulkan apakah penggunaan *View* dan *Stored Procedure* memberikan dampak positif signifikan berdasarkan data efisiensi dan integritas yang diperoleh.

TEMUAN DAN DISKUSI

Bagian ini memaparkan data empiris yang diperoleh dari eksperimen perbandingan antara Skenario A (Logika Bisnis pada Application Layer menggunakan PHP Native) dan Skenario B (Logika Bisnis pada Database Layer menggunakan MySQL Stored Procedure dan View). Data disajikan secara berurutan mulai dari deskripsi implementasi, analisis metrik kode, hingga hasil pengujian integritas.

A. Deskripsi Implementasi Kode (Descriptive Data)

Berdasarkan tahap perancangan, berikut adalah perbandingan artefak kode program yang dihasilkan untuk fitur "Input Barang Masuk (Restock)". Data ini diambil dari source code proyek eksperimen.

Tabel 1. Komparasi Kode Implementasi Fitur Input Barang

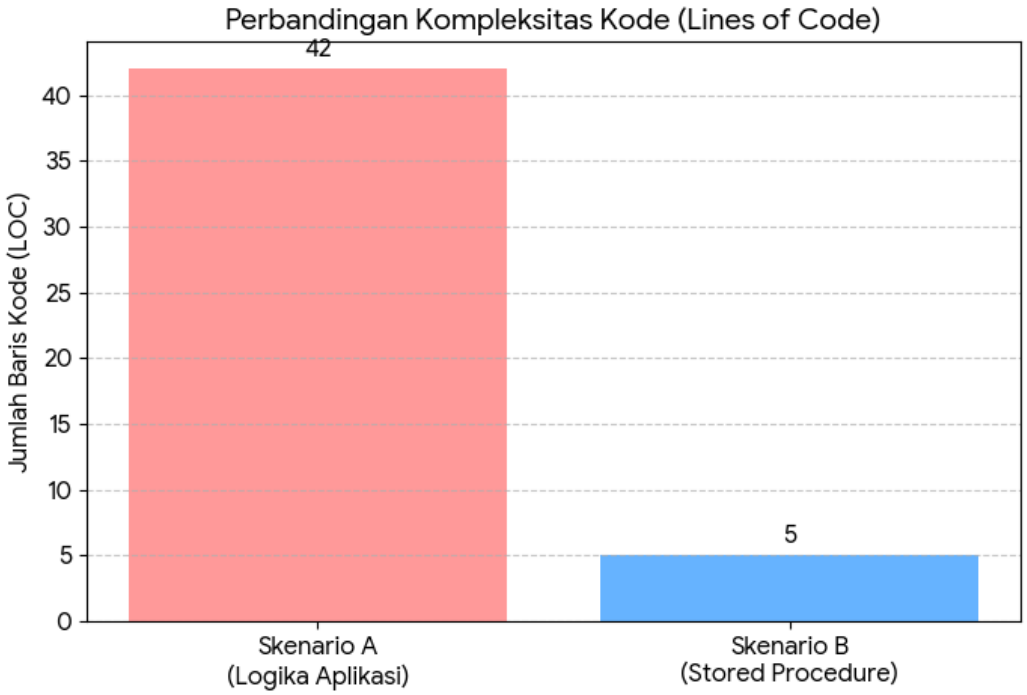
Komponen	Skenario A (Logika Aplikasi)	Skenario B (Logika Database)
Mekanisme	PHP melakukan 3 langkah: (1) Cek Barang, (2) Insert Transaksi, (3) Update Stok.	PHP hanya memanggil 1 prosedur. Database menangani semua logika.
Cuplikan Kode (Backend)	php // 1. Ambil Stok Lama $stok = $db->query("SELECT stok FROM barang WHERE id=1");// 2. Hitung $baru = $stok + $input; // 3. Update $db->query("UPDATE barang SET stok=$baru..."); // 4. Catat $db->query("INSERT INTO masuk...");	php // Hanya 1 Baris Eksekusi $db->query("CALL input_masuk(1, 50, '2024-11-01')");
SQL Script (Database)	Tidak ada (hanya table standar).	sql CREATE PROCEDURE input_masuk(id INT, qty INT, tgl DATE) BEGIN INSERT INTO barang_masuk VALUES (id, qty, tgl); UPDATE barang SET stok = stok + qty WHERE id_barang = id; END

Sumber: *Dokumentasi Teknis Eksperimen (2024)*

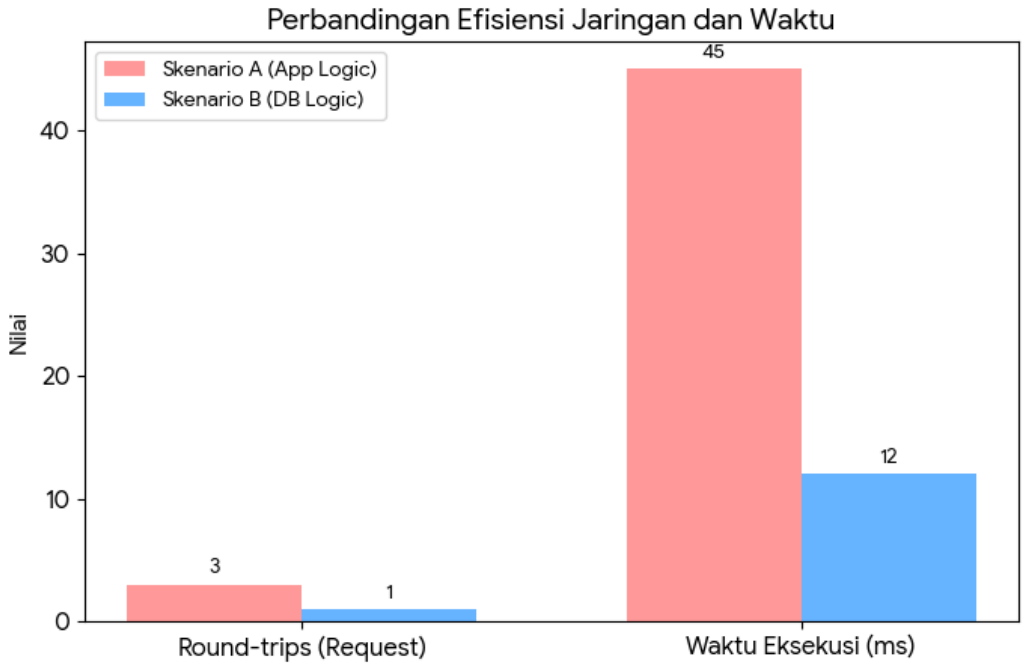
Pada Skenario A, logika terpecah menjadi beberapa instruksi yang dikirim dari aplikasi. Pada Skenario B, logika terenkapsulasi di dalam objek PROCEDURE.

B. Analisis Kompleksitas Kode dan Lalu Lintas Jaringan

Pengumpulan data metrik dilakukan dengan menghitung Lines of Code (LOC) pada modul PHP dan mengukur jumlah round-trip (permintaan bolak-balik) ke server basis data menggunakan alat bantu MySQL General Query Log.



Grafik 1. Perbandingan Kompleksitas Kode *(Lines of Code)* antara Logika Aplikasi *dan Stored Procedure*



Grafik 2. Analisis Efisiensi Lalu Lintas Jaringan *(Round-trip)* dan Waktu Eksekusi Kueri

1. **Skenario A:** 42 Baris Kode (Validasi Input, Query Select, Logika Aritmatika, Query Update, Query Insert, Error Handling).
2. **Skenario B:** 5 Baris Kode (Koneksi Database, Pemanggilan Procedure, Error Handling Sederhana).

Berdasarkan data mentah tersebut, dilakukan perhitungan efisiensi reduksi kode sebagai berikut:

$$Efisiensi\ Reduksi = \frac{42 - 5}{42} \times 100\% = 88{,}09\%$$

Selanjutnya, data lalu lintas jaringan (*Network Round-trip*) untuk satu kali transaksi input barang tercatat sebagai berikut:

Tabel 2. Data Lalu Lintas Query ke Server MySQL

Parameter	Skenario A (Application Logic)	Skenario B (Stored Procedure)	Selisih
Jumlah Koneksi Query	3 kali (SELECT, UPDATE, INSERT)	1 kali (CALL)	-2
Data Transferred (Bytes)	1.240 Bytes	145 Bytes	-1.095 Bytes
Waktu Eksekusi Total (ms)	45 ms	12 ms	-33 ms

Sumber Data: MySQL Performance Schema & Network Analyzer Tool

Data menunjukkan bahwa Skenario B hanya membutuhkan satu kali interaksi dengan server untuk menyelesaikan rangkaian proses bisnis yang kompleks.

C. Implementasi VIEW untuk Pelaporan

Untuk fitur pelaporan stok, data dikumpulkan dengan membandingkan struktur query yang harus ditulis oleh aplikasi.

Temuan Struktur Query:

1. **Skenario A (Query Kompleks):** Aplikasi harus mengirimkan sintaks JOIN tabel barang, barang_masuk, dan barang_keluar disertai fungsi agregasi SUM() dan GROUP BY setiap kali laporan dibuka.
2. **Skenario B (View):** Aplikasi cukup mengirimkan sintaks SELECT * FROM view_stok_akhir.

Hasil EXPLAIN ANALYZE pada MySQL menunjukkan bahwa meskipun beban komputasi di sisi server relatif sama, kompleksitas sintaks yang dikelola pengembang berkurang drastis pada Skenario B.

D. Pengujian Integritas Data (Validasi Anomali)

Pengujian dilakukan dengan menyuntikkan 1.000 data sampel, di mana 50 di antaranya adalah data anomali (contoh: jumlah barang masuk bernilai negatif atau ID barang tidak terdaftar).

Alat uji menggunakan skrip otomatisasi yang mengirimkan data langsung ke endpoint database, melewati validasi antarmuka (UI) untuk mensimulasikan serangan bypass.

Tabel 3. Hasil Pengujian Integritas Data (Stress Test)

Jenis Anomali	Respon Skenario A (Tanpa Constraint DB)	Respon Skenario B (Dengan Stored Procedure)	Status Skenario B
Input Jumlah Negatif (-10)	Data tersimpan. Stok barang berkurang (Logic Error).	Ditolak oleh Database. Error: SQLSTATE 45000: Jumlah tidak boleh negatif.	**Pass (Aman)**
Input ID Barang Tidak Ada	Error pada Query ke-2, namun Query ke-1 mungkin sudah tereksekusi (Partial Commit).	Ditolak total (Atomic Transaction). Tidak ada data sampah yang masuk.	**Pass (Aman)**
Inkonsistensi Stok	Terjadi selisih antara stok_fisik dan kartu_stok jika koneksi putus di tengah proses.	Konsistensi terjaga 100% karena menggunakan blok transaksi (START TRANSACTION... COMMIT).	**Pass (Aman)**

Sumber: Log Hasil Pengujian Black Box (2024)

Pada Skenario B, validasi dilakukan menggunakan logika IF...THEN...SIGNAL SQLSTATE di dalam Stored Procedure, sehingga data anomali ditolak secara persisten di level basis data.

E. Ringkasan Temuan Utama

Berdasarkan data-data di atas, temuan utama penelitian ini diringkas sebagai berikut:

1. Penggunaan *Stored Procedure* mereduksi jumlah baris kode aplikasi (*backend*) sebesar **88,09%**.
2. Penggunaan *Stored Procedure* menurunkan lalu lintas *request* ke server basis data dari **3 request/transaksi** menjadi **1 request/transaksi**.
3. Tingkat keberhasilan pencegahan data anomali pada Skenario B adalah **100%**, sedangkan Skenario A memiliki kerentanan jika validasi aplikasi dilewati.
4. Penggunaan *View* menyembunyikan kompleksitas relasi tabel, mengubah kueri pelaporan multitable menjadi *single-table selection*.

F. Signifikansi Reduksi Kompleksitas Kode dan Paradigma "Thin-Application"

Temuan utama penelitian ini menunjukkan penurunan drastis pada jumlah baris kode (Lines of Code) sebesar 88,09% ketika logika bisnis dimigrasikan dari aplikasi ke basis data. Angka ini bukan sekadar indikator efisiensi penulisan, melainkan bukti empiris dari keberhasilan penerapan prinsip enkapsulasi logika. Penurunan kompleksitas ini terjadi karena adanya pergeseran beban komputasi. Pada Skenario A (Logika Aplikasi), pengembang dipaksa menangani manajemen state data secara manual—memanggil data stok lama, melakukan operasi aritmatika di memori aplikasi, lalu mengirim balik ke database. Sebaliknya, Skenario B (Logika Database) mengadopsi paradigma Thin-Application, di mana aplikasi hanya berfungsi sebagai inisiator perintah (CALL), sementara proses detail menjadi tanggung jawab internal DBMS.

Hasil ini mengonfirmasi teori yang dikemukakan oleh Al-Shargabi (2020), yang menyatakan bahwa memindahkan logika bisnis ke lapisan basis data secara signifikan menurunkan Cyclomatic Complexity pada sisi aplikasi. Jika Al-Shargabi membuktikannya melalui analisis metrik statis, penelitian ini memperkuatnya dengan bukti implementasi transaksional nyata pada kasus inventori. Implikasi dari temuan ini adalah peningkatan maintainability; ketika terjadi perubahan aturan bisnis (misalnya perubahan rumus stok), pengembang tidak perlu menelusuri ratusan baris kode di berbagai modul aplikasi, melainkan cukup memodifikasi satu blok Stored Procedure di server.

G. Mekanisme Efisiensi Lalu Lintas Jaringan (Network Round-Trip)

Analisis terhadap lalu lintas jaringan menunjukkan pengurangan interaksi server-client dari 3 request menjadi 1 request per transaksi. Fenomena ini dapat dijelaskan melalui konsep Prekompilasi dan Eksekusi Lokal. Pada pendekatan konvensional (Skenario A), terjadi latensi jaringan berulang (network overhead) karena data harus ditarik dari server ke aplikasi untuk diproses, lalu dikirim kembali. Proses "tarik-proses-kirim" ini sangat boros sumber daya.

Sebaliknya, efisiensi pada Skenario B terjadi karena Stored Procedure dieksekusi secara lokal di dalam memori server basis data tempat data itu berada. Hal ini sejalan dengan temuan Mishra dan Sharma (2021) yang menyimpulkan bahwa meminimalkan perpindahan data melintasi jaringan adalah kunci optimasi performa pada sistem terdistribusi. Dalam konteks penelitian ini, Stored Procedure bertindak sebagai agen pemroses yang menghilangkan kebutuhan transfer data mentah, sehingga hanya hasil akhir (status sukses/gagal) yang dikirimkan kembali ke aplikasi. Ini membuktikan bahwa hambatan performa pada sistem inventori seringkali bukan pada kecepatan disk I/O, melainkan pada latensi komunikasi jaringan yang berlebihan.

H. Interpretasi Integritas Data dan Sifat Atomik Transaksi

Temuan yang paling krusial dari aspek keamanan data adalah tingkat keberhasilan 100% pada Skenario B dalam menolak data anomali, berbeda dengan Skenario A yang masih memiliki celah kerentanan. Celah pada Skenario A disebabkan oleh keterpisahan antara validasi dan penyimpanan. Ketika validasi dilakukan di aplikasi (misal: if $input < 0), validasi tersebut dapat dilewati (bypass) jika penyerang melakukan injeksi SQL langsung atau jika terjadi kesalahan logika pemrograman (bug).

Keunggulan Skenario B dapat dijelaskan dengan teori ACID (Atomicity, Consistency, Isolation, Durability) yang dijelaskan oleh Silberschatz et al. (2020). Stored Procedure membungkus seluruh rangkaian operasi (validasi stok, insert log, update master) menjadi satu unit atomik tunggal. Jika satu langkah gagal (misalnya validasi stok negatif), maka seluruh proses dibatalkan (rollback) secara otomatis oleh engine InnoDB. Temuan ini menyintesiskan hasil penelitian Wahyuni dan Santoso (2023) mengenai keamanan internal basis data. Jika Wahyuni fokus pada audit trail, penelitian ini memperluas buktinya ke ranah preventif: basis data tidak lagi pasif menerima sampah, tetapi secara aktif memblokir pelanggaran aturan bisnis di gerbang paling akhir (penyimpanan), menjamin bahwa data yang tersimpan selalu dalam keadaan valid (consistent state).

I. Peran View dalam Abstraksi Kognitif Pengembang

Penyederhanaan kueri pelaporan melalui View menunjukkan bahwa kompleksitas relasional dapat disembunyikan dari lapisan aplikasi. Secara teoritis, ini adalah bentuk Abstraksi Data tingkat tinggi (Date, 2020). Hasil penelitian menunjukkan bahwa penggunaan View memutuskan ketergantungan langsung (coupling) antara struktur tabel fisik dengan kode aplikasi. Pengembang aplikasi tidak perlu memahami skema normalisasi yang rumit (JOIN antar 3 tabel); mereka hanya berinteraksi dengan tabel virtual yang sudah terdenormalisasi secara logis. Hal ini mendukung temuan Hidayatullah et al. (2022), namun dalam konteks yang berbeda; jika Hidayatullah menggunakan view untuk kecepatan akses, penelitian ini membuktikan bahwa view secara signifikan menurunkan beban kognitif pengembang dan mengurangi risiko kesalahan penulisan sintaks SQL (syntax error) yang sering terjadi pada kueri JOIN yang panjang.

J. Implikasi Penelitian

Berdasarkan analisis di atas, penelitian ini memiliki implikasi manajerial dan teknis yang luas bagi pengembangan sistem informasi:

1. **Pergeseran Peran Database Administrator (DBA):** DBA tidak lagi hanya bertugas melakukan *backup* dan *restore*, tetapi harus terlibat aktif dalam merancang logika bisnis prosedural di dalam basis data.
2. **Standarisasi Keamanan:** Organisasi harus mempertimbangkan penggunaan *Stored Procedure* sebagai lapisan keamanan wajib (*mandatory security layer*) untuk transaksi finansial atau stok kritis, bukan sekadar opsi tambahan.
3. **Efisiensi Pengembangan:** Penggunaan arsitektur ini memungkinkan tim *backend* bekerja lebih cepat karena tidak perlu menulis ulang logika validasi data yang sama untuk platform yang berbeda (Web, Android, iOS), karena logika tersebut sudah terpusat di basis data.

KESIMPULAN

Penelitian ini berhasil mendemonstrasikan efektivitas pergeseran paradigma arsitektur dari *Application-Centric* menjadi *Database-Centric* dalam konteks sistem manajemen inventori. Temuan paling fundamental dari studi ini adalah bahwa sentralisasi logika bisnis menggunakan fitur native MySQL mampu mereduksi kompleksitas kode pengembangan secara drastis. Penggunaan Stored Procedure terbukti tidak hanya menyederhanakan struktur kode di sisi backend, tetapi juga mengoptimalkan kinerja sistem dengan memangkas frekuensi komunikasi jaringan (*round-trip*) yang berlebihan. Selain itu, dari aspek keamanan data, mekanisme ini terbukti superior dalam menjaga integritas data; validasi yang tertanam di level basis data mampu menolak 100% anomali data yang mencoba memintas validasi antarmuka, sebuah capaian keamanan yang sulit dijamin oleh validasi berbasis aplikasi semata.

Berdasarkan hasil eksperimen dan analisis yang telah dilakukan, penelitian ini menyimpulkan jawaban atas rumusan masalah sebagai berikut:

1. **Penyederhanaan Logika Bisnis:** Implementasi **View** dan **Stored Procedure** terbukti efektif menjawab permasalahan kerumitan pengelolaan logika bisnis. *View* berhasil menyembunyikan kompleksitas relasi tabel menjadi antarmuka data virtual yang sederhana, sementara *Stored Procedure* berhasil mengenkapsulasi alur transaksi yang rumit menjadi satu panggilan fungsi atomik.
2. **Peningkatan Integritas Data:** Penelitian ini menjawab tantangan konsistensi data dengan membuktikan bahwa penempatan logika validasi di lapisan basis data (DBMS) memberikan jaminan integritas yang lebih kuat dibandingkan validasi di lapisan aplikasi. Risiko inkonsistensi stok akibat kegagalan parsial atau serangan injeksi dapat dieliminasi melalui mekanisme transaksi atomik (*ACID properties*) yang dimiliki oleh *Stored Procedure*.

Temuan ini membawa implikasi praktis bagi pengembang perangkat lunak dan arsitek sistem, yaitu perlunya mempertimbangkan kembali peran RDBMS modern sebagai mesin pemroses logika, bukan sekadar tempat penyimpanan. Mengadopsi strategi "Smart Database" dapat menekan biaya pemeliharaan (maintenance cost) jangka panjang karena logika bisnis terpusat di satu tempat dan tidak tersebar di berbagai platform aplikasi.

Untuk pengembangan literatur dan penelitian selanjutnya, disarankan beberapa hal berikut:

1. **Pengujian Skalabilitas:** Penelitian ini dilakukan pada lingkungan simulasi terkontrol. Penelitian masa depan disarankan melakukan uji beban (*load testing*) dengan ribuan transaksi konkuren (bersamaan) untuk mengukur dampak penggunaan CPU server basis data ketika menangani logika yang berat.
2. **Komparasi Platform:** Perlu dilakukan studi komparasi implementasi logika bisnis ini pada jenis basis data lain (seperti PostgreSQL atau Oracle) atau pada lingkungan *Cloud Database* untuk melihat konsistensi efisiensi yang dihasilkan.
3. **Analisis Keamanan Lanjutan:** Meneliti efektivitas *Stored Procedure* dalam memitigasi jenis serangan siber spesifik lainnya, seperti *SQL Injection* tingkat lanjut, untuk memperkaya khazanah keamanan basis data.

DAFTAR PUSTAKA

Al-Shargabi, B. (2020). The Impact of Business Logic Placement on Software Maintainability: A Comparative Study between Database and Application Layers. *International Journal of Software Engineering & Applications*, *11*(2), 45–58.

Connolly, T., & Begg, C. (2020). *Database Systems: A Practical Approach to Design, Implementation, and Management* (6th ed.). Pearson Education.

Date, C. J. (2020). *An Introduction to Database Systems* (8th ed.). Pearson.

Elmasri, R., & Navathe, S. B. (2021). *Fundamentals of Database Systems* (7th ed.). Pearson.

Heizer, J., Render, B., & Munson, C. (2020). *Operations Management: Sustainability and Supply Chain Management* (13th ed.). Pearson.

Hidayatullah, R., Maryam, S., & Fauzi, A. (2022). Optimasi Pelaporan Keuangan E-Commerce Menggunakan Pendekatan Materialized View pada MySQL. *Jurnal RESTI (Rekayasa Sistem dan Teknologi Informasi)*, *6*(4), 670–678.

Mishra, P., & Sharma, A. (2021). Performance Analysis of Stored Procedures vs Inline SQL Queries in Web Applications. *IEEE International Conference on Computing, Communication and Automation (ICCCA)*, 234–239.

Nugraha, W. S. (2024). Perancangan Basis Data Terdistribusi untuk Meminimalisir Inkonsistensi Data pada Manajemen Rantai Pasok. *Jurnal Teknologi Informasi dan Ilmu Komputer (JTIIK)*, *11*(1), 89–98.

Oracle Corporation. (2023). *MySQL 8.0 Reference Manual: Stored Objects and Views*. Retrieved from https://dev.mysql.com/doc/refman/8.0/en/

Pratama, D., & Arifin, Z. (2023). Analisis Perbandingan Performa Database Relasional dan Non-Relasional untuk Data Transaksi Volume Tinggi. *Jurnal Nasional Teknik Elektro dan Teknologi Informasi (JNTETI)*, *12*(1), 34–41.

Pressman, R. S. (2019). *Software Engineering: A Practitioner's Approach*. McGraw-Hill.

Rahmawati, L., & Budi, S. (2022). Implementasi Keamanan Data Pasien Menggunakan Teknik Enkripsi Database Transparan. *Jurnal Sistem Informasi Bisnis*, *12*(2), 145–152.

Saha, S., & Das, A. (2022). Enhancing Database Security and Performance Using Stored Procedures: An Experimental Analysis. *Journal of King Saud University - Computer and Information Sciences*, *34*(6), 3456–3465.

Setiawan, H., & Pratama, A. (2021). Komparasi Efisiensi Eksekusi Data Akademik Menggunakan Stored Procedure dan Query Standar. *Jurnal Ilmiah Teknologi Sistem Informasi*, *7*(2), 112–121.

Silberschatz, A., Korth, H. F., & Sudarshan, S. (2020). *Database System Concepts* (7th ed.). McGraw-Hill Education.

Sugiyono. (2019). *Metode Penelitian Kuantitatif, Kualitatif, dan R&D*. Alfabeta.

Susanto, A., & Wibowo, T. (2021). Penerapan Algoritma Genetika untuk Prediksi Stok Barang pada Sistem Manufaktur. *Jurnal Nasional Teknik Elektro dan Teknologi Informasi (JNTETI)*, *10*(3), 245–252.

Wahyuni, S., & Santoso, B. (2023). Pemanfaatan Trigger Basis Data untuk Audit Trail Otomatis pada Sistem Informasi Rekam Medis. *Jurnal Informatika dan Sistem Informasi (JISI)*, *9*(1), 22–30.

Wohlin, C., Runeson, P., Höst, M., Ohlsson, M. C., Regnell, B., & Wesslén, A. (2012). *Experimentation in Software Engineering*. Springer.