Journal of Science and Social Research ISSN 2615 – 4307 (Print)
August 2026, IX (4): 5941 – 5949 ISSN 2615 – 3262 (Online)
Available online at http://jurnal.goretanpena.com/index.php/JSSR

ANALISIS KOMPARATIF KINERJA *ASYNC-SINGLETON* DAN *ASYNC-POOLING* PADA MANAJEMEN KONEKSI DATABASE MYSQL BERBASIS PYTHON

Abadi Nugroho[1], Arfittariah[2], Ahdam Ashari[3] ,Nurul Hikma Awalia[4]
Sekolah Tinggi Teknologi Bontang, Kalimantan Timur
e-mail: [1]abadi@stitek.ac.id, [2]fittarhia@gmail.com

Abstract: *Database connection management plays an important role in determining the performance of Python-based applications, particularly under high-concurrency workloads. This study compares the performance of two MySQL database connection management strategies, namely Async-Singleton and Async-Pooling, in an asynchronous programming environment. An experimental approach was employed by implementing both strategies using the aiomysql library. Performance evaluation was carried out under four concurrency levels of 10, 100, 500, and 1000 requests. Each scenario was repeated ten times to obtain consistent results. The evaluated parameters included execution time, CPU usage, and memory usage. The collected data were analyzed using descriptive statistics, including the mean, standard deviation, minimum, maximum, and percentage of performance improvement. The results show that Async-Pooling consistently achieved lower execution times than Async-Singleton across all testing scenarios. Performance improvements reached 11.66%, 36.54%, 65.90%, and 62.57% at concurrency levels of 10, 100, 500, and 1000 requests, respectively. Although Async-Pooling required higher CPU usage, the additional computational cost was accompanied by a substantial reduction in execution time. The increase in memory usage was also relatively small. These findings indicate that Async-Pooling is a more effective connection management strategy for Python applications operating under high-concurrency workloads, offering significant performance improvements without imposing substantial memory overhead.*

Keyword: *aiomysql, asynchronous programming, connection pooling, MySQL, Python.*

Abstrak: Manajemen koneksi database berperan penting dalam menentukan performa aplikasi berbasis Python terutama ketika aplikasi harus melayani banyak permintaan secara bersamaan. Penelitian ini bertujuan membandingkan kinerja *Async-Singleton* dan *Async-Pooling* sebagai strategi manajemen koneksi pada database MySQL dalam lingkungan *asynchronous* programming. Penelitian menggunakan metode eksperimen dengan mengimplementasikan kedua strategi menggunakan pustaka *aiomysql*. Pengujian dilakukan pada empat tingkat konkurensi yaitu 10, 100, 500, dan 1000 request. Setiap skenario dijalankan sebanyak sepuluh kali untuk memperoleh hasil yang lebih konsisten. Parameter yang diamati meliputi waktu eksekusi, penggunaan CPU, dan penggunaan memori. Data dianalisis menggunakan statistik deskriptif yang terdiri atas rata-rata (*mean*), standar deviasi (*standard deviation*), nilai minimum, nilai maksimum, serta persentase peningkatan performa. Hasil pengujian memperlihatkan bahwa *Async-Pooling* selalu menghasilkan waktu eksekusi yang lebih rendah dibandingkan *Async-Singleton*. Peningkatan performa yang diperoleh mencapai 11,66%, 36,54%, 65,90%, dan 62,57% pada tingkat konkurensi 10, 100, 500, dan 1000 *request*. Penggunaan CPU pada *Async-Pooling* memang lebih tinggi. Namun peningkatan tersebut diikuti penurunan waktu eksekusi yang signifikan. Penggunaan memori juga hanya bertambah dalam jumlah yang relatif kecil. Hasil penelitian menunjukkan bahwa *Async-Pooling* lebih efektif diterapkan pada aplikasi Python yang melayani beban kerja dengan tingkat konkurensi tinggi karena mampu meningkatkan performa tanpa menimbulkan peningkatan penggunaan memori yang berarti.

Journal of Science and Social Research ISSN 2615 – 4307 (Print)
August 2026, IX (4): 5941 – 5949 ISSN 2615 – 3262 (Online)
Available online at http://jurnal.goretanpena.com/index.php/JSSR

Kata kunci: aiomysql, asynchronous programming, connection pooling, MySQL, Python.

PENDAHULUAN

Aplikasi berbasis web telah menjadi bagian penting dalam berbagai layanan digital mulai dari pemerintahan, pendidikan, kesehatan hingga industri. Peningkatan jumlah pengguna dan volume transaksi mendorong kebutuhan terhadap aplikasi yang mampu mempertahankan waktu respons tetap rendah meskipun melayani banyak permintaan secara bersamaan. Pada kondisi tersebut komunikasi antara aplikasi dan basis data menjadi salah satu faktor yang menentukan performa sistem. Pengelolaan koneksi yang kurang efisien dapat memperpanjang waktu akses data, meningkatkan penggunaan sumber daya, serta mengurangi kemampuan aplikasi dalam menangani beban kerja yang tinggi (Eyada et al., 2020; Reichardt et al., 2021).

MySQL merupakan salah satu *Relational Database Management System* (RDBMS) yang banyak digunakan dalam pengembangan aplikasi karena bersifat *open source*, mudah diintegrasikan dengan berbagai bahasa pemrograman, dan memiliki performa yang baik untuk menangani transaksi data. Sejumlah penelitian menunjukkan bahwa peningkatan performa MySQL dapat dilakukan melalui optimasi kueri, penggunaan *stored procedure*, maupun pemilihan mekanisme akses data yang sesuai dengan karakteristik aplikasi. Waktu eksekusi *stored procedure* meningkat seiring bertambahnya jumlah data yang diproses (Warman & Wildani, 2021). *Stored procedure* memiliki performa yang lebih stabil dibandingkan SELECT dan VIEW pada beberapa skenario pengujian (Aulia et al., 2023). Selain itu peningkatan tingkat konkurensi berpengaruh terhadap waktu respons MySQL sehingga strategi pengelolaan akses basis data menjadi faktor yang perlu diperhatikan (Reza & Supatmi, 2023).

Kebutuhan terhadap aplikasi yang mampu melayani banyak permintaan secara bersamaan mendorong semakin luasnya penerapan *asynchronous programming*. Berbeda dengan pendekatan sinkron yang memproses setiap tugas secara berurutan, pemrograman asinkron memungkinkan beberapa operasi I/O-*bound* berjalan secara bersamaan melalui mekanisme *event loop*. Pendekatan ini dapat memanfaatkan waktu tunggu selama proses komunikasi dengan basis data untuk menjalankan tugas lain sehingga penggunaan sumber daya menjadi lebih efisien (Mykola, 2024; Sodian et al., 2022). Pendekatan asinkron mampu memberikan efisiensi yang lebih baik dibandingkan multithreading pada beberapa skenario pemrosesan data (Toktorbaev et al., 2025). Temuan serupa juga menunjukkan bahwa pemrosesan kueri secara asinkron mampu meningkatkan performa aplikasi ketika jumlah permintaan meningkat secara signifikan (Makarov et al., 2025).

Keberhasilan implementasi *asynchronous programming* sangat dipengaruhi oleh strategi pengelolaan koneksi database. Salah satu pendekatan yang banyak digunakan adalah pola Singleton yaitu penggunaan satu objek koneksi yang dibagikan kepada seluruh proses aplikasi. Pendekatan ini sederhana dan memiliki kebutuhan memori yang relatif rendah karena hanya mempertahankan satu koneksi aktif. Penerapan pola Singleton mampu meningkatkan efisiensi penggunaan memori dibandingkan pembentukan koneksi baru secara berulang (Azizi et al., 2025). Pendekatan lain yang banyak diterapkan adalah *Connection Pooling* yaitu penyediaan sejumlah koneksi yang dapat digunakan kembali tanpa harus membuat koneksi baru setiap kali transaksi dilakukan. Strategi ini terbukti mampu meningkatkan *throughput*,

mengurangi biaya pembentukan koneksi, serta menjaga stabilitas aplikasi pada lingkungan dengan tingkat konkurensi tinggi (Boronnikov et al., 2024; Gupta, 2025; Sobri et al., 2022).

Berbagai penelitian mengenai optimasi basis data, *asynchronous programming*, dan *connection pooling* telah banyak dipublikasikan. Namun sebagian besar penelitian masih mengevaluasi aspek-aspek tersebut secara terpisah. Kajian sebelumnya lebih banyak berfokus pada optimasi kueri, evaluasi performa DBMS, atau implementasi *connection pooling* pada platform dan bahasa pemrograman tertentu. Penelitian yang secara khusus membandingkan performa *Async-Singleton* dan *Async-Pooling* pada pengelolaan koneksi MySQL menggunakan Python masih terbatas. Selain itu pengaruh penggunaan mekanisme sinkronisasi *asyncio.Lock()* terhadap performa *Async-Singleton* pada kondisi konkurensi tinggi juga belum banyak dibahas. Kondisi tersebut menunjukkan masih adanya ruang penelitian yang perlu dikaji melalui pendekatan eksperimental.

Berdasarkan kondisi tersebut maka penelitian ini akan menganalisis dan membandingkan performa antara *Async-Singleton* dan *Async-Pooling* pada manajemen koneksi database MySQL berbasis Python. Pengujian dilakukan menggunakan pustaka *aiomysql* pada empat tingkat konkurensi yaitu 10, 100, 500, dan 1000 *request*, dengan sepuluh kali pengulangan pada setiap skenario. Evaluasi difokuskan pada waktu eksekusi, penggunaan CPU, dan penggunaan memori. Kontribusi utama penelitian ini terletak pada penyajian bukti empiris mengenai perbedaan performa dua strategi manajemen koneksi database dalam lingkungan asynchronous programming, termasuk pengaruh penggunaan *asyncio.Lock()* pada implementasi *Async-Singleton*. Hasil penelitian diharapkan dapat menjadi referensi bagi pengembang dalam memilih strategi manajemen koneksi yang sesuai dengan kebutuhan aplikasi berbasis Python.

METODE

Penelitian ini menggunakan metode eksperimen (*experimental research*) untuk membandingkan kinerja dua strategi manajemen koneksi database yaitu *Async-Singleton* dan *Async-Pooling* pada aplikasi berbasis Python. Kedua strategi diuji menggunakan skenario yang sama sehingga setiap perbedaan hasil yang diperoleh berasal dari mekanisme pengelolaan koneksi yang diterapkan. Pendekatan eksperimen dipilih karena mampu memberikan evaluasi yang objektif terhadap performa sistem pada berbagai tingkat konkurensi (Reza & Supatmi, 2023).

Tahapan penelitian dimulai dengan implementasi kedua strategi menggunakan pustaka *aiomysql*. Kemudian dilakukan pengujian pada beberapa tingkat konkurensi, dilanjutkan dengan pengumpulan data performa dan analisis hasil menggunakan statistik deskriptif.

Implementasi dan Skenario Pengujian

Implementasi *Async-Singleton* menggunakan satu objek koneksi database yang dibagikan kepada seluruh *coroutine*. Untuk menjaga konsistensi akses, setiap proses disinkronkan menggunakan mekanisme *asyncio.Lock()* sehingga hanya satu *coroutine* yang dapat menggunakan koneksi pada satu waktu. Sebaliknya pada *Async-Pooling* memanfaatkan sekumpulan koneksi (*connection pool*) yang dapat digunakan kembali oleh beberapa *coroutine*.

Tabel 1 Konfigurasi Pengujian

Parameter	Konfigurasi
Strategi manajemen koneksi	*Async-Singleton*, *Async-Pooling*
Database	MySQL 8.0.30
Bahasa Pemrograman	Python 3.13.7

Library	*aiomysql, asyncio*
Tingkat konkurensi	10, 100, 500, dan 1000 *request*
Pengulangan	10 kali pada setiap skenario
Parameter evaluasi	Waktu eksekusi, penggunaan CPU, dan penggunaan memori
Metode analisis	Statistik deskriptif dan persentase peningkatan performa

Parameter Pengujian

Evaluasi dilakukan menggunakan tiga parameter, yaitu waktu eksekusi, penggunaan CPU, dan penggunaan memori. Waktu eksekusi digunakan untuk mengukur lama penyelesaian seluruh *request* pada setiap skenario.

Analisis Data

Data hasil pengujian dianalisis menggunakan statistik deskriptif untuk menggambarkan karakteristik performa masing-masing strategi. Analisis dilakukan terhadap waktu eksekusi, penggunaan CPU, dan penggunaan memori pada setiap tingkat konkurensi. Statistik yang digunakan meliputi rata-rata (*mean*), standar deviasi (*standard deviation*), nilai minimum, dan nilai maksimum. Pendekatan ini dipilih karena mampu memberikan gambaran yang jelas mengenai kecenderungan dan variasi data hasil eksperimen(Ju et al., 2024; Reza & Supatmi, 2023).

Selain statistik deskriptif penelitian ini menghitung persentase peningkatan performa (*performance improvement*) untuk mengukur efektivitas *Async-Pooling* dibandingkan *Async-Singleton*. Perhitungan dilakukan berdasarkan nilai rata-rata waktu eksekusi menggunakan Persamaan (1).

$$PI(\%) = \frac{T_{Singletone} - T_{Pooling}}{T_{Singletone}} x100\% \quad (1)$$

Keterangan :
PI = *Performance Improvement*
$T_{Singletone}$= rata-rata waktu eksekusi metode Async-Singleton.
$T_{Pooling}$ = rata-rata waktu eksekusi metode Async-Pooling.

HASIL DAN PEMBAHASAN

Hasil Pengujian

Pengujian dilakukan untuk mengevaluasi performa *Async-Singleton* dan *Async-Pooling* dalam mengelola koneksi database MySQL pada lingkungan *asynchronous programming*. Kedua strategi diuji menggunakan empat tingkat konkurensi yaitu 10, 100, 500, dan 1000 *request*. Setiap skenario dijalankan sebanyak sepuluh kali pengulangan dengan konfigurasi pengujian yang sama. Parameter yang diamati meliputi waktu eksekusi, penggunaan CPU, dan penggunaan memori.

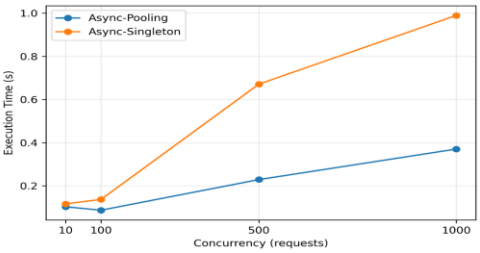
Analisis Waktu Eksekusi

Tabel 2 Statistik Deskriptif Waktu Eksekusi

Conc	Metode	Mean (s)	SD	Min	Max
10	*Async-Singleton*	0.1165	0.0496	0.0827	0.2333
10	*Async-Pooling*	0.1029	0.0248	0.0749	0.1468
100	*Async-Singleton*	0.1372	0.0268	0.1038	0.1907
100	*Async-Pooling*	0.087	0.0046	0.0802	0.0939
500	*Async-Singleton*	0.6714	0.1404	0.3698	0.8143
500	*Async-Pooling*	0.2289	0.0168	0.2105	0.257
1000	*Async-Singleton*	0.9894	0.1235	0.7992	1.2239
1000	*Async-Pooling*	0.3704	0.0279	0.3354	0.4217

Untuk memperjelas kecenderungan perubahan waktu eksekusi pada setiaptingkat konkurensi, hasil pengujian ditampilkan dalam Gambar 1.

Journal of Science and Social Research ISSN 2615 – 4307 (Print)
August 2026, IX (4): 5941 – 5949 ISSN 2615 – 3262 (Online)
Available online at http://jurnal.goretanpena.com/index.php/JSSR



Gambar 1 Perbandingan Rata-rata Waktu Eksekusi *Async-Singleton* dan *Async-Pooling* pada Berbagai Tingkat Konkurensi

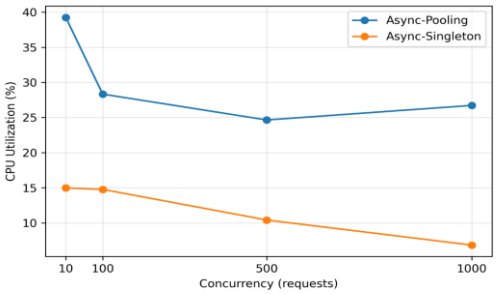
Gambar 1 memperlihatkan bahwa waktu eksekusi kedua metode meningkat seiring bertambahnya tingkat konkurensi. Akan tetapi laju peningkatan pada *Async-Pooling* jauh lebih rendah dibandingkan *Async-Singleton*.

Analisis Penggunaan CPU

Tabel 3 Statistik Deskriptif Penggunaan CPU

Conc	Metode	Mean (%)	SD	Min	Max
10	*Async-Singleton*	13.66	5.81	5.1	23.2
10	*Async-Pooling*	34.84	14.52	15	54.2
100	*Async-Singleton*	8.88	5.74	1	17.9
100	*Async-Pooling*	37.47	7.34	28	50.9
500	*Async-Singleton*	12.63	5.95	4.9	21.4
500	*Async-Pooling*	38.89	8.13	25.1	49.3
1000	*Async-Singleton*	13.38	5.82	4.8	22.9
1000	*Async-Pooling*	40.61	7.95	29.4	52.7

Berdasarkan Tabel 3, penggunaan CPU pada *Async-Pooling* cenderung lebih tinggi dibandingkan *Async-Singleton* di seluruh skenario pengujian.



Gambar 2 Perbandingan rata-rata penggunaan CPU *Async-Singleton* dan *Async-Pooling* pada berbagai tingkat konkurensi.

Gambar 2 memperlihatkan bahwa peningkatan penggunaan CPU pada *Async-Pooling* tidak diikuti oleh peningkatan waktu eksekusi. Sebaliknya metode ini tetap menghasilkan waktu *respons* yang lebih baik dibandingkan *Async-Singleton*. Hasil tersebut menunjukkan bahwa sumber daya prosesor dimanfaatkan secara lebih efektif untuk meningkatkan kemampuan sistem dalam memproses permintaan secara bersamaan.

Analisis Penggunaan Memori

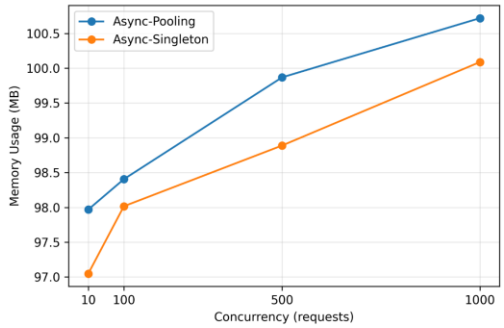
Statistik penggunaan memori pada kedua metode disajikan pada Tabel 4.

Tabel 4 Statistik Deskriptif Penggunaan Memori

Conc	Metode	Mean (MB)	SD	Min	Max
10	*Async-Singleton*	97.13	0.2664	96.6992	97.5391
10	*Async-Pooling*	98	0.2782	97.5859	98.5078
100	*Async-Singleton*	98.13	0.2768	97.6406	98.5508
100	*Async-Pooling*	98.61	0.3731	98.0078	99.1484
500	*Async-Singleton*	99.01	0.3752	98.4805	99.6133
500	*Async-Pooling*	99.92	0.3776	99.1758	100.6016
1000	*Async-Singleton*	100.08	0.3988	99.3359	100.7227

1000	*Async-Pooling*	100.73	0.2902	100.3164	101.1172

Data pada Tabel 4 menunjukkan bahwa penggunaan memori pada kedua metode relatif stabil pada seluruh tingkat konkurensi. *Async-Pooling* menggunakan memori sedikit lebih besar dibandingkan *Async-Singleton* karena mempertahankan beberapa koneksi aktif di dalam *connection pool*. Meskipun demikian selisih penggunaan memori tersebut relatif kecil apabila dibandingkan dengan peningkatan performa yang dihasilkan.



Gambar 3 Perbandingan rata-rata penggunaan memori *Async-Singleton* dan *Async-Pooling* pada berbagai tingkat konkurensi.

Berdasarkan Gambar 3 peningkatan penggunaan memori pada *Async-Pooling* cenderung konsisten pada setiap tingkat konkurensi. Kondisi tersebut menunjukkan bahwa biaya tambahan akibat penggunaan beberapa koneksi aktif masih berada pada tingkat yang wajar.

Pembahasan

Perbandingan Performa Waktu Eksekusi

Perbedaan performa antara kedua metode terutama dipengaruhi oleh mekanisme pengelolaan koneksi database. Pada *Async-Singleton*, seluruh *coroutine* mengakses satu koneksi yang sama sehingga setiap proses harus menunggu giliran memperoleh akses melalui mekanisme *asyncio.Lock()*. Kondisi tersebut menyebabkan antrean semakin panjang ketika jumlah permintaan meningkat. Sebaliknya *Async-Pooling* menyediakan beberapa koneksi yang dapat digunakan kembali sehingga beberapa transaksi dapat berlangsung secara bersamaan tanpa harus menunggu pembentukan koneksi baru.

Temuan ini sejalan dengan Ju et al. (2024) yang melaporkan bahwa penerapan *connection pooling* pada lingkungan *asynchronous programming* mampu meningkatkan *throughput* dan menurunkan waktu respons aplikasi. Gupta (2025) juga menyatakan bahwa pengelolaan koneksi yang efisien merupakan salah satu faktor utama dalam meningkatkan performa aplikasi yang mengakses basis data secara intensif. Hasil penelitian ini memperkuat temuan tersebut melalui pengujian langsung pada implementasi Python–MySQL menggunakan pustaka aiomysql.

Pengaruh terhadap Penggunaan CPU

Peningkatan penggunaan CPU pada *Async-Pooling* menunjukkan bahwa lebih banyak *coroutine* dapat dieksekusi secara bersamaan. Meskipun utilisasi prosesor lebih tinggi namun peningkatan tersebut memberikan dampak positif terhadap waktu eksekusi karena proses tidak lagi didominasi oleh kondisi menunggu akses ke database. Dengan kata lain sumber daya komputasi digunakan untuk menyelesaikan pekerjaan yang lebih banyak dalam waktu yang sama.

Hasil tersebut mendukung penelitian Makarov et al. (2025) dan Toktorbaev et al. (2025) yang menunjukkan bahwa pendekatan asinkron mampu meningkatkan efisiensi pemanfaatan sumber daya sistem pada aplikasi yang didominasi operasi I/O-*bound*. Dengan demikian peningkatan penggunaan CPU pada penelitian ini dapat dipandang sebagai konsekuensi yang wajar dari meningkatnya tingkat paralelisme.

Pengaruh terhadap Penggunaan Memori

Penggunaan memori pada *Async-Pooling* sedikit lebih tinggi dibandingkan

Async-Singleton karena beberapa koneksi dipertahankan tetap aktif selama proses pengujian. Tambahan penggunaan memori tersebut diperlukan agar aplikasi tidak harus membangun koneksi baru setiap kali menerima permintaan. Strategi ini menghasilkan proses akses database yang lebih cepat dengan konsekuensi adanya alokasi memori tambahan.

Temuan tersebut sejalan dengan Sobri et al. (2022) dan Boronnikov et al. (2024) yang menjelaskan bahwa *connection pooling* meningkatkan efisiensi layanan melalui mekanisme penggunaan ulang koneksi. Walaupun memerlukan memori yang sedikit lebih besar, pendekatan tersebut mampu menjaga stabilitas performa ketika aplikasi melayani banyak permintaan secara bersamaan.

Implikasi Penelitian

Hasil penelitian menunjukkan bahwa pemilihan strategi manajemen koneksi memberikan pengaruh langsung terhadap performa aplikasi berbasis Python yang menggunakan MySQL. *Async-Singleton* masih sesuai untuk aplikasi dengan tingkat konkurensi rendah karena implementasinya sederhana dan kebutuhan sumber dayanya relatif kecil. Sebaliknya, *Async-Pooling* lebih tepat diterapkan pada aplikasi yang melayani beban kerja tinggi karena mampu mempertahankan waktu eksekusi tetap rendah tanpa meningkatkan penggunaan memori secara signifikan.

Temuan ini memberikan kontribusi empiris terhadap penelitian mengenai optimasi koneksi database pada lingkungan *asynchronous programming*. Selain menunjukkan perbedaan performa antara dua strategi manajemen koneksi, penelitian ini juga memperlihatkan bagaimana mekanisme *asyncio.Lock()* memengaruhi waktu tunggu akses database ketika jumlah permintaan meningkat. Hasil tersebut dapat menjadi referensi praktis bagi pengembang dalam menentukan strategi manajemen koneksi yang sesuai dengan karakteristik beban kerja aplikasi.

Persentase Peningkatan Performa

Untuk memberikan gambaran mengenai efektivitas *Async-Pooling* dibandingkan *Async-Singleton*, dilakukan perhitungan persentase peningkatan performa berdasarkan rata-rata waktu eksekusi pada setiap tingkat konkurensi. Hasil perhitungan disajikan pada Tabel 5.

Tabel 5 Persentase Peningkatan Performa *Async-Pooling* terhadap *Async-Singleton*

Conc	Peningkatan Performa (%)
10	11.66
100	36.54
500	65.9
1000	62.57

Secara keseluruhan, hasil ini mengindikasikan bahwa manfaat penggunaan *connection pooling* semakin nyata pada aplikasi yang beroperasi dengan tingkat konkurensi menengah hingga tinggi. Dengan memanfaatkan beberapa koneksi yang dapat digunakan kembali, proses akses database berlangsung lebih efisien sehingga waktu eksekusi dapat ditekan tanpa peningkatan penggunaan memori yang signifikan.

SIMPULAN

Penelitian ini telah berhasil membandingkan kinerja dua strategi manajemen koneksi database MySQL berbasis Python yaitu *Async-Singleton* dan *Async-Pooling* melalui pengujian pada empat tingkat konkurensi dengan parameter waktu eksekusi, penggunaan CPU, dan penggunaan memori. Hasil penelitian menunjukkan bahwa *Async-Pooling* secara konsisten menghasilkan waktu eksekusi yang lebih rendah dibandingkan *Async-Singleton* pada seluruh skenario pengujian. Peningkatan performa yang diperoleh masing-masing sebesar 11,66%, 36,54%, 65,90%, dan 62,57% pada tingkat konkurensi 10, 100, 500, dan 1000 *request*. Temuan ini menunjukkan bahwa mekanisme

connection pool mampu mengurangi waktu tunggu akses ke database sehingga lebih efektif dalam menangani peningkatan jumlah permintaan secara bersamaan.

Dari sisi penggunaan sumber daya pada *Async-Pooling* memanfaatkan CPU lebih tinggi dibandingkan *Async-Singleton* sebagai konsekuensi dari kemampuan mengeksekusi beberapa *coroutine* secara paralel. Meskipun demikian peningkatan utilisasi CPU tersebut diikuti oleh penurunan waktu eksekusi yang signifikan sehingga pemanfaatan sumber daya menjadi lebih efektif. Selain itu penggunaan memori pada *Async-Pooling* hanya sedikit lebih tinggi dibandingkan *Async-Singleton* karena adanya beberapa koneksi aktif yang dipertahankan di dalam *connection pool*. Tambahan penggunaan memori tersebut masih berada pada tingkat yang wajar apabila dibandingkan dengan peningkatan performa yang diperoleh.

Berdasarkan hasil penelitian dapat disimpulkan bahwa *Async-Singleton* masih sesuai untuk aplikasi dengan tingkat konkurensi rendah yang mengutamakan kesederhanaan implementasi dan penggunaan sumber daya yang minimal. Sebaliknya pada *Async-Pooling* lebih direkomendasikan untuk aplikasi yang harus menangani beban akses tinggi karena mampu memberikan peningkatan performa yang signifikan tanpa peningkatan penggunaan memori yang berarti. Hasil penelitian ini memberikan bukti empiris mengenai efektivitas penerapan *connection pooling* pada aplikasi Python yang menggunakan MySQL serta dapat menjadi referensi dalam pemilihan strategi manajemen koneksi untuk membangun aplikasi yang memiliki skalabilitas dan performa tinggi.

Penelitian ini masih terbatas pada penggunaan MySQL, pustaka *aiomysql* dan satu konfigurasi *connection pool* pada lingkungan pengujian tertentu. Untuk penelitian selanjutnya dapat mengevaluasi pengaruh variasi ukuran *connection pool*, karakteristik *workload*, serta implementasi pada DBMS lain seperti PostgreSQL atau MariaDB. Penelitian selanjutnya juga dapat mengkaji performa pustaka asinkron lain atau membandingkan strategi manajemen koneksi pada framework Python yang berbeda sehingga diperoleh pemahaman yang lebih komprehensif mengenai optimasi koneksi database pada aplikasi dengan tingkat konkurensi tinggi.

UCAPAN TERIMA KASIH

Penulis mengucapkan terima kasih kepada Lembaga Penelitian dan Pengabdian kepada Masyarakat (LPPM) Sekolah Tinggi Teknologi (STITEK) Bontang atas dukungan pendanaan melalui Program Dana Isian Pelaksanaan Anggaran (DIPA) Tahun Anggaran 2026 berdasarkan Kontrak Penelitian Nomor: 009/LPPM/K-LT/2026. Dukungan pendanaan tersebut telah memungkinkan terlaksananya seluruh rangkaian penelitian, mulai dari pelaksanaan eksperimen, pengumpulan dan analisis data hingga penyusunan artikel ilmiah ini. Penulis juga menyampaikan apresiasi kepada semua pihak yang telah memberikan bantuan, masukan, dan dukungan selama proses penelitian sehingga penelitian ini dapat diselesaikan dengan baik.

DAFTAR PUSTAKA

Aulia, C. P., Pratama, M. Y., & Dewi, H. L. (2023). Perbandingan Performa Query Select Dasar, View, dan Stored Procedure pada Database MySQL. *Prosiding Seminar Nasional Teknologi Dan Sistem Informasi*.

Azizi, B., Pratama, A. A., Dysa, G. M., & Carudin, C. (2025). Analisis Penerapan Design Pattern Singletone dalam Pengelolaan Koneksi Database Untuk Efisiensi Memori. *Jurnal Informatika Dan Teknik Elektro Terapan*, *13*(3).

Boronnikov, A. S., Tsyngalev, P. S., Ilyin,

V., & Demenkova, T. A. (2024). Evaluation of Connection Pool PgBouncer Efficiency for Optimizing Relational Database Computing Resources. *Russian Technological Journal*. https://doi.org/10.32362/2500-316X-2024-12-3-7-24

Eyada, M. M. K., Saber, W., Genidy, M. E. E., & Amer, F. (2020). Performance Evaluation of IoT Data Management Using MongoDB Versus MySQL Databases in Different Cloud Environments. *IEEE Access*. https://doi.org/10.1109/ACCESS.2020.3002164

Gupta, R. (2025). Optimizing Database Performance Through Efficient Connection Management. *World Journal of Advanced Research and Reviews*. https://doi.org/10.30574/wjarr.2025.26.1.1111

Ju, L., Yadav, A., Khan, A., Sah, A. P., & Yadav, D. (2024). Using Asynchronous Frameworks and Database Connection Pools to Enhance Web Application Performance in High-Concurrency Environments. *2024 8th International Conference on I-SMAC (IoT in Social, Mobile, Analytics and Cloud) (I-SMAC)*, 742–747. https://doi.org/10.1109/i-smac61858.2024.10714639

Milojković, K., Spalevic, P., Vasić, N., Milojković, N., & Milojković, H. (2025). SDLC-Independent Python-Based Query Performance Benchmarking Approach and Practical Optimal Database Selection Guidelines. *Sinteza 2025*. https://doi.org/10.15308/sinteza-2025-536-543

Mykola, K. (2024). Using Asynchronous Programming in Python to Improve Application Performance. *The American Journal of Engineering and Technology*. https://doi.org/10.37547/tajet/volume06issue12-06

Reichardt, M., Gundall, M., & Schotten, H. (2021). Benchmarking the Operation Times of NoSQL and MySQL Databases for Python Clients. *IECON 2021*. https://doi.org/10.1109/IECON48115.2021.9589382

Reza, D., & Supatmi, S. (2023). Concurrency Performance Analysis on MySQL Relational Database Management System in Virtual Machine Environment. *ICSPIS 2023*. https://doi.org/10.1109/ICSPIS59665.2023.10402765

Sobri, N. A. N., Abas, M. A. H., Yassin, A. M. I., Ali, M. M. S. A., Tahir, M. N., & Zabidi, A. (2022). Database Connection Pool in Microservice Architecture. *Journal of Electrical & Electronic Systems Research*. https://doi.org/10.24191/jeesr.v20i1.004

Sodian, L., Wen, J. P., Davidson, L., & Loskot, P. (2022). Concurrency and Parallelism in Speeding Up I/O and CPU-Bound Tasks in Python 3.10. *CEI 2022*. https://doi.org/10.1109/CEI57409.2022.9950068

Toktorbaev, A., Karabaev, S., & Toktomuratova, Z. (2025). Comparative Analysis of Asynchronous and Multithreaded Programming Methods in Python for Big Data Processing. *Bulletin of Science and Practice*. https://doi.org/10.33619/2414-2948/114/19

Warman, I., & Wildani, W. (2021). Analisa Kinerja Query Stored Procedure pada Database Management System (DBMS) MySQL. *Jurnal Sains Dan Teknologi*.

Zhao, J., & Li, M. (2025). Large-Scale Concurrency for MySQL Database Performance Analysis. *ICAACE 2025*. https://doi.org/10.1109/ICAACE65325.2025.11020302