

Jurnal JTIK (Jurnal Teknologi Informasi dan Komunikasi)

DOI: <https://doi.org/10.35870/jtik.v9i4.3878>

Pengujian Performa UUID Sebagai Primary Key Pada Database PostgreSQL Dan MongoDB

Muhamad Kamal Shamlan ^{1*}, Sri Winarso Martyas Edi ²

^{1*,2} Teknik Informatika, Fakultas Teknologi Informasi, Universitas Kristen Satya Wacana.

article info

Article history:

Received 4 March 2025
Received in revised form
20 April 2025
Accepted 1 May 2025
Available online October
2025.

Keywords:

UUID; PostgreSQL;
MongoDB.

Kata Kunci:

UUID; PostgreSQL;
MongoDB.

abstract

The choice of primary key affects the performance and scalability of a database. This study examines the use of UUID as a primary key compared to auto-increment in PostgreSQL and ObjectId in MongoDB. The testing is conducted by importing data from a CSV file and processing it in parallel to measure efficiency in the insert process. The Composite Performance Index (CPI) method is used, with a weight of 75% for speed and 25% for storage size, considering that latency has a greater impact on system performance. The results show that auto-increment outperforms UUID in speed and storage efficiency, especially in MongoDB. In PostgreSQL, the difference in speed is not significant. This study provides insight for developers in choosing a primary key based on performance needs, taking into account latency and slight storage efficiency.

abstrak

Pemilihan primary key berpengaruh pada kinerja dan skalabilitas database. Penelitian ini menguji UUID sebagai primary key dibandingkan dengan auto-increment di PostgreSQL dan ObjectId di MongoDB. Pengujian dilakukan dengan mengimpor data dari file CSV dan mengolahnya secara paralel untuk mengukur efisiensi dalam proses insert. Metode Composite Performance Index (CPI) digunakan dengan bobot 75% untuk kecepatan dan 25% untuk ukuran penyimpanan, mempertimbangkan bahwa latensi lebih berdampak pada performa sistem. Hasil menunjukkan bahwa auto-increment lebih unggul dalam kecepatan dan efisiensi penyimpanan dibandingkan UUID, terutama di MongoDB. Pada PostgreSQL, perbedaan kecepatan tidak signifikan. Studi ini memberikan wawasan bagi developer dalam memilih primary key berdasarkan kebutuhan performa dengan mempertimbangkan latensi dan sedikit efisiensi penyimpanan.

Corresponding Author. Email: 672021262@student.uksw.edu ^{1}.

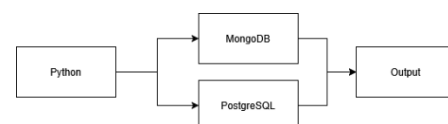
1. Pendahuluan

Perkembangan teknologi informasi semakin pesat, mencakup berbagai bidang perangkat lunak seperti aplikasi berbasis *web*, aplikasi *mobile*, hingga pengembangan *artificial intelligence* (Zhou *et al.*, 2022). Perkembangan ini menuntut perancang perangkat lunak untuk terus berinovasi guna memastikan efisiensi dalam pengelolaan sistem. Dengan implementasi yang tepat, sistem yang dibangun dapat mendukung kinerja yang optimal. UUID (*Universally Unique Identifier*) merupakan metode yang digunakan untuk menghasilkan pengenal unik yang dapat diterapkan di berbagai sistem dan aplikasi (Triebl *et al.*, 2018; National Cancer Institute, 2020). UUID sering digunakan sebagai *primary key* dalam basis data karena kemampuannya untuk menjamin keunikan secara global. Namun, terdapat beberapa tantangan terkait penerapan UUID, khususnya dalam hal kinerja dan efisiensi penyimpanan. UUID terdiri dari 128 bit yang dihasilkan melalui algoritma tertentu yang menjamin keunikannya secara global (Mishra *et al.*, 2022).

Keunikan ini sangat penting pada sistem terdistribusi, di mana beberapa node dapat menghasilkan kunci secara bersamaan tanpa risiko benturan. Meskipun UUID menawarkan fleksibilitas dan keunikan, ukuran UUID yang lebih besar dibandingkan dengan integer *auto-increment* menimbulkan tantangan tersendiri. Ukuran yang lebih besar ini dapat memengaruhi kinerja operasi *insert* serta penggunaan *index* dalam basis data, mengingat kebutuhan ruang penyimpanan yang lebih besar dan proses pemrosesan yang lebih intensif. Dengan pesatnya pertumbuhan data dan kebutuhan akan pengolahan yang cepat dan efisien, pemahaman yang lebih mendalam tentang dampak penggunaan UUID sebagai *primary key* dalam operasi *insert* menjadi penting. Pengujian kinerja ini dapat memberikan wawasan yang berguna bagi pengembang dan arsitek sistem dalam memilih metode pengelolaan kunci yang paling sesuai dengan kebutuhan aplikasi (Patil & Patil, 2024). Untuk teknologi yang digunakan dalam penelitian ini, *PostgreSQL* merupakan sistem basis data relasional *open-source* yang dikembangkan dan didukung oleh komunitas pengembang di seluruh dunia (Rajabov Azizbek Ravshanovich, 2024). Di sisi lain, *MongoDB* adalah sistem basis data *NoSQL* yang

menyimpan data dalam format *key-value* (C. D. Andharini, 2022). *MongoDB* memungkinkan pengembang menyimpan data dalam bentuk objek mirip *JSON* yang sesuai dengan struktur data aplikasi (Makris *et al.*, 2021; Silalahi & Wahyudi, 2018). *ObjectId* adalah *primary key* bawaan dari *MongoDB*, yang bersifat hampir tidak terbatas karena mengacu pada waktu *UNIX EPOCH* sejak 1 Januari 1970, dengan ukuran 12-byte (KEIGHOBAD & BUGRA, 2021). Jika dibandingkan dengan database *NoSQL* lain seperti *Cassandra*, *MongoDB* unggul dalam skala vertikal, sehingga *Cassandra* lebih sesuai untuk beban kerja yang sangat berat (Cui & Chen, 2021). Sebagai pembandingan antara UUID dan *ObjectId*, ada juga metode *auto-increment* yang paling sering digunakan, di mana nilai *identifier* bertambah setiap kali data baru dimasukkan (Salunke & Ouda, 2024). Beberapa penelitian, seperti yang dilakukan oleh Juri Pebrianto, menunjukkan bahwa penggunaan UUID sebagai *primary key* dapat memengaruhi kinerja operasi *insert*, terutama karena ukuran yang lebih besar dan distribusi acaknya (Pebrianto, 2022). Penelitian lainnya, oleh Herryts Timisela, menjelaskan bahwa *MongoDB*, dengan format *BSON* (Binary *JSON*), lebih efisien dalam kecepatan dan pencarian data (HERRYTS, 2024). Penelitian yang dilakukan oleh Faizal Anugrah Bhaswara dan rekan-rekannya menunjukkan bahwa *MongoDB*, sebagai database *NoSQL*, memiliki keunggulan dalam operasi *CRUD* (*Create, Read, Update, Delete*) dibandingkan dengan basis data *SQL* (Bhaswara *et al.*, 2017). Penelitian ini bertujuan untuk membandingkan kinerja UUID sebagai *primary key* dalam operasi pembuatan data di *PostgreSQL* dan *MongoDB*. Diharapkan, penelitian ini dapat memberikan panduan bagi pengembang dan administrator basis data dalam memilih konfigurasi yang tepat sesuai dengan kebutuhan sistem mereka. Selain itu, penelitian ini diharapkan dapat menambah pemahaman tentang dampak penggunaan UUID dalam skenario nyata, baik dalam lingkungan basis data relasional maupun non-relasional.

2. Metodologi Penelitian



Gambar 1. Alur pengujian

Penelitian dilakukan dalam beberapa tahapan, Pertama data yang berupa file csv akan diterima oleh service dari python. Python akan mengolah data dan memasukan datanya kedalam database secara paralel dan mengulanginya sebanyak 100, 1.000, 10.000, 100.000, dan 1.000.000 kali. Python juga akan mencatat response time data yang masuk kedalam database. Dengan begitu, kita dapat melihat hasil komparasinya performanya.

Metodologi yang digunakan untuk mengetahui efektifitas primary key dapat menggunakan metode Composite Performance Index atau CPI (Rendy yudistira, 2019). CPI adalah metode kuantitatif untuk mengukur performa index. Metode ini cocok untuk mengevaluasi kinerja suatu sistem, karena membandingkan hasil melalui data. Variabel dan bobot yang digunakan pada penelitian ini ada 2, yakni kecepatan dan ukuran. Dengan nilai kecepatan 75% dan nilai ukuran 25% (Rumandan, 2022). Dengan skala 0 sampai dengan 1, 0 artinya negatif dan 1 positif. Untuk rumus yang akan digunakan adalah sebagai berikut:

$$X_{average} = \frac{X^x}{Y}$$

Dimana X adalah nilai rata-rata, X adalah nilai minimum, X adalah nilai maximum. Dengan rumus ini kita dapat menentukan normalisasi dari X. Nilai yang sudah ditemukan adakan digunakan untuk menghitung CPI (Satria *et al.*, 2022).

$$CPI = \sum_{j=1}^m w_i \times X_{average}$$

Rumus diatas adalah rumus untuk menghitung CPI, w adalah nilai bobot yang digunakan sebagai tolak ukur sedangkan Xaverage adalah hasil dari rata rata yang sudah dihitung sebelumnya. Sedangkan i adalah variabel bobotnya, karena terdapat 2 variabel bobot (kecepatan dan ukuran) maka akan dilakukan dua kali, menjadi (Prastowo *et al.*, 2021):

$$CPI = (w_{kecepatan} + X_{average, kecepatan}) + (w_{ukuran} + X_{average, ukuran})$$

3. Hasil dan Pembahasan

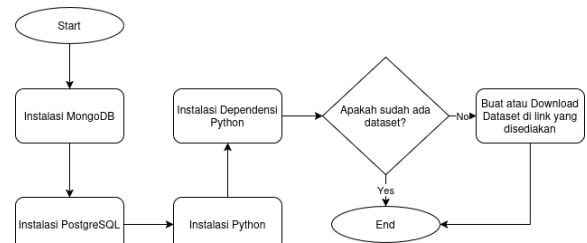
Hasil

Konfigurasi Perangkat Lunak

Perangkat keras yang dibutuhkan untuk penelitian ini adalah laptop atau komputer dengan spesifikasi yang memenuhi standar minimum. Oleh karena itu dibutuhkan requirement device untuk menjalankan program secara lancar dan optimal yakni dengan spesifikasi di tabel 1:

Tabel 1. Daftar perangkat keras

Perangkat	Minimum
<i>Processor</i>	Intel Core i5 8250U
<i>RAM</i>	8gb
<i>Cache Heirarchy</i>	L1d: 128 Kib
	L1i: 128 Kib
	L2: 1 Mib
	L3: 6 Mib
<i>Cpu Clockspeed</i>	1.60 Ghz
<i>Storage</i>	Space Kosong 5gb



Gambar 2. Alur Konfigurasi Perangkat Lunak

Perangkat lunak yang digunakan setidaknya memiliki versi yang sama agar tidak terjadi error ataupun ketidak fungsional dependensi. Versi yang berbeda juga bisa jadi mengganggu syntax yang akan digunakan karena dalam beberapa kasus dependensi di versi terbaru merubah nama fungsi yang tersedia (Nokeri, 2022). Untuk perangkat lunak yang dibutuhkan adalah sebagai berikut yang ada di tabel 2:

Tabel 2. Daftar perangkat lunak

Perangkat Lunak	Versi
Python	3.12.3
MongoDB	2.3.1
PostgreSQL	17.2

Selain perangkat lunak, dibutuhkan juga beberapa dependensi untuk python yang terdapat di tabel 3:

Tabel 3. Daftar dependensi python

Perangkat Lunak	Versi
csv	Bawaan 3.12.3
Time	Bawaan 3.12.3
uuid	Bawaan 3.12.3
Json	Bawaan 3.12.3
Concurrent	Bawaan 3.12.3
Pymongo	4.8.0
Psycopg2	2.9.10
bson	Bawaan 3.12.3

Jika belum memiliki dependensi tersebut, bisa melakukan instalasi dependensi python bisa menggunakan pip install nama_dependensi versi. Kemudian buat dataset sebanyak 5 file sebagai bahan pengujian berupa file csv. File memiliki jumlah data yang berbeda beda, data terdiri dari 100, 1.000, 10.000, 100.000, dan 1.000.000 baris data. atau bisa unduh dataset melalui website <https://www.datablist.com/learn/csv/download-sample-csv-files>. Setelah semua terinstall dan sudah memiliki dataset untuk bahan pengujian, maka pengujian dapat dilaksanakan dengan membuka kode editor untuk mulai menulis kode dan buat sebuah file baru dengan ekstensi .py di suatu direktori.

Kode program 1. Melakukan import

```
import csv
import time
import uuid
import json
import concurrent.futures
from pymongo import MongoClient
import psycopg2
import bson
```

Selanjutnya, buka file tersebut dan lakukan impor terhadap dependensi *csv*, *time*, *uuid*, *json*, *concurrent.futures*, *psycopg2*, *bson*, dan *MongoClient* dari dependensi *pymongo*. Modul *csv* digunakan untuk membaca dan menulis file CSV dalam rangka pengelolaan data berbasis tabel. Modul *time* berfungsi untuk menangani aspek waktu, seperti penundaan atau pencatatan durasi eksekusi. Modul *uuid* digunakan untuk menghasilkan *UUID* pada setiap versi yang dibutuhkan. Modul *json* digunakan untuk memanipulasi dan mengelola data dalam format *JSON*. Sementara itu, *concurrent.futures* memungkinkan eksekusi fungsi secara paralel, yang mendukung pengolahan data dalam waktu yang lebih

efisien. *Pymongo* adalah library yang digunakan untuk menghubungkan dan berinteraksi dengan basis data *MongoDB*, sementara *psycopg2* digunakan untuk mengelola koneksi serta operasi terhadap basis data *PostgreSQL*. Terakhir, *bson* digunakan untuk menangani data dalam format *Binary JSON* yang kompatibel dengan *MongoDB*.

Kode program 2. Mengambil file csv

```
csv_file_path = './data/data-100.csv'
```

Setelah melakukan import buat sebuah variabel untuk menampung jalur atau path yang digunakan untuk menyimpan dataset csvnya. Setelah itu buat koneksi menuju *MongoDB* dan *PostgreSQL*, berikut contoh kodenya. Namun perlu diingat username, password, host, dan nama database disesuaikan.

Kode program 3. Melakukan koneksi ke basis data

```
mongo_client = MongoClient('mongodb://localhost:27017')
mongo_db = mongo_client['test_db']
mongo_collection_uuid = mongo_db['test_collection_uuid']
mongo_collection_auto = mongo_db['test_collection_auto']
mongo_collection_object = mongo_db['test_collection_object']

pg_conn = psycopg2.connect(
    dbname='test_db',
    user='kamal',
    password='***',
    host='localhost',
    port='5432'
)
pc_cursor = pg_conn.cursor()
```

Setelah koneksi berhasil dibuat, beberapa fungsi perlu dikembangkan untuk melaksanakan pengujian. Fungsi-fungsi tersebut meliputi:

- 1) Fungsi untuk membaca file CSV yang telah diinisialisasi pada variabel, yang bertugas untuk memuat data dari file CSV ke dalam struktur data yang dapat diproses lebih lanjut.
- 2) Fungsi untuk memasukkan data ke dalam tabel atau koleksi basis data. Fungsi ini bertanggung jawab untuk melakukan penyisipan data yang telah dibaca dari file CSV ke dalam *database PostgreSQL* atau *MongoDB* sesuai dengan pengaturan yang telah dilakukan sebelumnya.
- 3) Fungsi untuk menjalankan perintah *insert* secara paralel dengan memanfaatkan dependensi *concurrent.futures*. Penggunaan eksekusi paralel bertujuan untuk meningkatkan efisiensi proses

penyisipan data dengan menjalankan beberapa tugas secara bersamaan.

Selain itu, penanda berupa perintah *print* dapat ditambahkan pada setiap fungsi untuk memonitor dan menampilkan proses yang sedang berjalan selama pengujian, sehingga memudahkan pemantauan dan pemecahan masalah jika terjadi kesalahan.

Kode program 4. Pengujian basis data

```
# Fungsi untuk membaca CSV dan mengembalikan data
dalam bentuk list
def read_csv(file_path):
    with open(file_path, mode='r') as file:
        reader = csv.DictReader(file)
        return [row for row in reader]

# Fungsi untuk menyisipkan data ke MongoDB
menggunakan UUID
def insert_to_mongodb_uuid(data):
    print("MASUK UUID MONGO")
    start_time = time.time()
    documents = [{"_id":
bson.Binary.from_uuid(uuid.uuid4()), "data": row}
for row in data]
    mongo_collection_uuid.insert_many(documents)
    return time.time() - start_time

# Fungsi untuk menyisipkan data ke MongoDB dengan
auto-increment
def insert_to_mongodb_auto(data):
    print("MASUK AUTO MONGO")
    start_time = time.time()
    counter = 1
    documents = [{"_id": int(counter + i), "data":
row} for i, row in enumerate(data)]
    mongo_collection_auto.insert_many(documents)
    return time.time() - start_time
# Fungsi untuk menyisipkan data ke MongoDB dengan
auto-increment
def insert_to_mongodb_object(data):
    print("MASUK OBJECT MONGO")
    start_time = time.time()
    documents = [{"data": row} for i, row in
enumerate(data)]
    mongo_collection_object.insert_many(documents)
    return time.time() - start_time

# Fungsi untuk menyisipkan data ke PostgreSQL
menggunakan UUID
def insert_to_postgresql_uuid(data):
    print("MASUK UUID PG")
    start_time = time.time()
    pc_cursor.executemany(
        psycopg2.sql.SQL("INSERT INTO test_table_uuid
VALUES (%s, %s)").as_string(pc_cursor),
        [(str(uuid.uuid4()), json.dumps(row)) for
row in data]
    )
    pg_conn.commit()
    return time.time() - start_time

# Fungsi untuk menyisipkan data ke PostgreSQL
menggunakan auto-increment
def insert_to_postgresql_auto(data):
    print("MASUK AUTO PG")
```

```
start_time = time.time()
pc_cursor.executemany(
    psycopg2.sql.SQL("INSERT INTO test_table_auto
(data) VALUES (%s)").as_string(pc_cursor),
    [(json.dumps(row),) for row in data]
)
pg_conn.commit()
return time.time() - start_time

# Fungsi utama untuk mengukur performa
def measure_performance(data):
    with concurrent.futures.ThreadPoolExecutor() as
executor:
        future_mongo_auto =
executor.submit(insert_to_mongodb_auto, data)
        future_mongo_object =
executor.submit(insert_to_mongodb_object, data)
        future_mongo_uuid =
executor.submit(insert_to_mongodb_uuid, data)
        future_pg_uuid =
executor.submit(insert_to_postgresql_uuid, data)
        future_pg_auto =
executor.submit(insert_to_postgresql_auto, data)

        mongo_uuid_time = future_mongo_uuid.result()
        mongo_auto_time = future_mongo_auto.result()
        mongo_object_time =
future_mongo_object.result()
        pg_uuid_time = future_pg_uuid.result()
        pg_auto_time = future_pg_auto.result()

    return mongo_uuid_time, mongo_auto_time,
mongo_object_time, pg_uuid_time, pg_auto_time
```

Untuk menjalankan semua fungsi tersebut dibutuhkan fungsi yang berperan sebagai fungsi utama. Se-hingga saat file eksekusi, blok kode tersebut dieksekusi terlebih dahulu. Dalam python ada cara agar blok tertentu di eksekusi di awal saat file (Nokeri, 2022) di eksekusi. Dengan menggunakan `if __name__ == "__main__"` Maka saat file di eksekusi program melakukan pengecekan apa saja yang perlu dijalankan terlebih dahulu. Hal ini dapat kita implementasikan sebagai berikut:

Kode program 5. Eksekusi kode pengujian basis data

```
mongo_collection_auto.delete_many({})
mongo_collection_uuid.delete_many({})
mongo_collection_object.delete_many({})
pc_cursor.execute(''')
    truncate test table auto
''')

pc_cursor.execute(''')
    truncate test_table_uuid
''')

pg_conn.commit()

# Membaca data dari file CSV
data = read_csv(csv_file_path)

# Mengukur waktu penyisipan data secara paralel
mongo_uuid_time, mongo_auto_time,
mongo_object_time, pg_uuid_time, pg_auto_time =
measure_performance(data)
print()
```



```

print("-----TEST KECEPATAN-----")
print()
print(f"MongoDB      UUID      response      time:
{mongo_uuid_time:.4f} seconds")
print(f"MongoDB      Auto-Increment response time:
{mongo_auto_time:.4f} seconds")
print(f"MongoDB      ObjectId response time:
{mongo_object_time:.4f} seconds")
print(f"PostgreSQL      UUID      response      time:
{pg_uuid_time:.4f} seconds")
print(f"PostgreSQL Auto-Increment response time:
{pg_auto_time:.4f} seconds")

# mengetahui ukuran table dan collection setelah
data masuk
print()
print("-----TEST UKURAN-----")
print()
# mongo
stats = mongo_db.command("collStats",
"test_collection_uuid")
size = stats.get('size')
print(f"Ukuran collection uuid: {size} bytes atau
{(size / 1024):.2f} kb atau {(size / 1024 /
1024):.2f} mb")
del(size)
stats = mongo_db.command("collStats",
"test_collection_auto")
size = stats.get('size')
print(f"Ukuran collection auto: {size} bytes atau
{(size / 1024):.2f} kb atau {(size / 1024 /
1024):.2f} mb")
del(size)
stats = mongo_db.command("collStats",
"test_collection_object")
size = stats.get('size')
print(f"Ukuran collection objectId: {size} bytes
atau {(size / 1024):.2f} kb atau {(size / 1024 /
1024):.2f} mb")
del(size)

# postgres
pc_cursor.execute("""
SELECT
pg_total_relation_size('test_table_uuid');
""")
table_size = pc_cursor.fetchone()[0]
print(f"Ukuran tabel UUID: {table_size} bytes
atau {(table_size / 1024):.2f} kb atau {(table_size
/ 1024 / 1024):.2f} mb")
del(table_size)

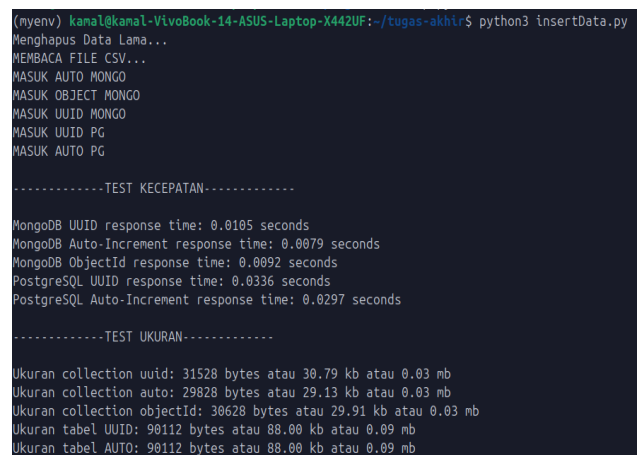
pc_cursor.execute("""
SELECT
pg_total_relation_size('test_table_auto');
""")
table_size = pc_cursor.fetchone()[0]
print(f"Ukuran tabel AUTO: {table_size} bytes
atau {(table_size / 1024):.2f} kb atau {(table_size
/ 1024 / 1024):.2f} mb")
del(table_size)

# Menutup koneksi database
mongo_client.close()
pc_cursor.close()
pg_conn.close()

```

Pada kode program ini, langkah pertama adalah melakukan *truncate* pada semua koleksi dan tabel yang akan digunakan untuk pengujian. Tujuan dari langkah

ini adalah untuk memastikan bahwa koleksi dan tabel dalam keadaan kosong, tanpa data, sebelum proses pengujian dilakukan. Setelah itu, program akan membaca file CSV melalui fungsi yang telah dibuat sebelumnya dan memasukkan data ke dalam sebuah variabel. Variabel ini kemudian digunakan sebagai parameter dalam fungsi yang menjalankan semua proses *insert* secara paralel. Fungsi ini akan menghasilkan lima keluaran yang ditangkap dalam lima variabel baru, yang selanjutnya digunakan dalam sintaks *print()* untuk menampilkan hasil eksekusi yang telah tercatat. Setelah pengujian kecepatan selesai, langkah selanjutnya adalah mengukur ukuran dari koleksi dan tabel. Untuk MongoDB, perintah *collStats* dapat digunakan untuk memperoleh statistik koleksi, termasuk ukuran data, yang dapat langsung dieksekusi oleh program. Sedangkan untuk PostgreSQL, fungsi bawaan *pg_total_relation_size* digunakan untuk mengukur ukuran tabel. Fungsi ini dijalankan melalui query *SELECT* dengan nama tabel sebagai parameter. Hasil yang diperoleh akan disimpan dalam variabel. Karena hasil yang diberikan dalam satuan byte, program kemudian perlu mengonversinya ke ukuran kilobyte dan megabyte dengan membagi hasil tersebut dengan 1024. Setelah semua langkah selesai, program akan menutup koneksi ke MongoDB dan PostgreSQL. Setelah itu, koneksi basis data akan ditutup untuk memastikan bahwa semua sumber daya dilepaskan dengan benar. Program kemudian dapat dijalankan dengan mengetikkan perintah di terminal: `python3 namaFile.py`. Jika tidak terjadi error, maka hasilnya akan ditampilkan sesuai dengan yang diharapkan.



```

(myenv) kamal@kamal-VivoBook-14-ASUS-Laptop-X442UF:~/tugas-akhir$ python3 insertData.py
Menghapus Data Lama...
MEMBACA FILE CSV...
MASUK AUTO MONGO
MASUK OBJECT MONGO
MASUK UUID MONGO
MASUK UUID PG
MASUK AUTO PG

-----TEST KECEPATAN-----

MongoDB UUID response time: 0.0105 seconds
MongoDB Auto-Increment response time: 0.0079 seconds
MongoDB ObjectId response time: 0.0092 seconds
PostgreSQL UUID response time: 0.0336 seconds
PostgreSQL Auto-Increment response time: 0.0297 seconds

-----TEST UKURAN-----

Ukuran collection uuid: 31528 bytes atau 30.79 kb atau 0.03 mb
Ukuran collection auto: 29828 bytes atau 29.13 kb atau 0.03 mb
Ukuran collection objectId: 30628 bytes atau 29.91 kb atau 0.03 mb
Ukuran tabel UUID: 90112 bytes atau 88.00 kb atau 0.09 mb
Ukuran tabel AUTO: 90112 bytes atau 88.00 kb atau 0.09 mb

```

Gambar 3. Hasil keluaran program

Hasil Rancangan

Untuk melihat hasil rancangan kita perlu melakukan eksekusi terlebih dahulu. Eksekusi akan dilakukan sebanyak 5 kali setiap csv. Sehingga nantinya csv dengan 100 data akan di eksekusi sebanyak 5 kali, begitu juga csv dengan 1.000 data, dan seterusnya. Hasil dari eksekusi akan menampilkan waktu dari proses eksekusi dan ukuran dari collection dan table. Berikut hasil pengujian kecepatan dalam detik:

1) Hasil pengujian 100 data

Hasil pengujian ukuran dalam satuan byte sebesar 31528 menggunakan UUID MongoDB, 29828 menggunakan auto increment MongoDB, 30628 menggunakan ObjectId MongoDB, 30628 menggunakan UUID PostgreSQL, dan 30628 menggunakan auto increment PostgreSQL. Sedangkan, kecepatan dalam proses eksekusi seperti yang ada di tabel 4 dan 5:

Tabel 4. Hasil pengujian 100 data MongoDB

Percobaan ke-n	UUID	Auto Increment	ObjectId
1	0.0110	0.0143	0.0147
2	0.0076	0.0022	0.0053
3	0.0100	0.0046	0.0145
4	0.0174	0.0090	0.0188
5	0.0116	0.0086	0.0111

Tabel 5. Hasil pengujian 100 data PostgreSQL

Percobaan ke-n	UUID	Auto Increment
1	0.0418	0.0351
2	0.0313	0.0326
3	0.0387	0.0330
4	0.0386	0.0336
5	0.0374	0.0291

2) Hasil pengujian 1.000 data

Hasil pengujian ukuran dalam satuan byte sebesar 315321 menggunakan UUID MongoDB, 298321 menggunakan auto increment MongoDB, 306321 menggunakan ObjectId MongoDB, 434176 menggunakan UUID PostgreSQL, dan 409600 menggunakan auto increment PostgreSQL. Sedangkan, kecepatan dalam proses eksekusi seperti yang ada di tabel 6 dan 7.

Tabel 6. Hasil Pengujian 1.000 data MongoDB

Percobaan ke-n	UUID	Auto Increment	ObjectId
1	0.0933	0.0388	0.0899
2	0.0788	0.0241	0.0813
3	0.0844	0.0457	0.0681
4	0.0891	0.0363	0.0656
5	0.0967	0.0615	0.0808

Tabel 7. Hasil Pengujian 1.000 data PostgreSQL

Percobaan ke-n	UUID	Auto Increment
1	0.0933	0.0388
2	0.0788	0.0241
3	0.0844	0.0457
4	0.0891	0.0363
5	0.0967	0.0615

3) Hasil pengujian 10.000 data

Hasil pengujian ukuran dalam satuan byte sebesar 3164395 menggunakan UUID MongoDB, 2994395 menggunakan auto increment MongoDB, 3074395 menggunakan ObjectId MongoDB, 3874816 menggunakan UUID PostgreSQL, dan 3629056 menggunakan auto increment PostgreSQL. Sedangkan, kecepatan dalam proses eksekusi seperti yang ada di tabel 8 dan 9.

Tabel 8. Hasil pengujian 10.000 data MongoDB

Percobaan ke-n	UUID	Auto Increment	ObjectId
1	0.7113	0.2401	0.4065
2	0.7427	0.2583	0.4133
3	0.7834	0.2207	0.4581
4	0.7590	0.2579	0.4820
5	0.7253	0.2177	0.3873

Tabel 9. Hasil pengujian 10.000 data PostgreSQL

Percobaan ke-n	UUID	Auto Increment
1	4.1589	4.0986
2	4.1767	4.0175
3	4.2243	4.1257
4	4.3305	4.1668
5	4.4483	4.3292

4) Hasil pengujian 100.000 data

Hasil pengujian ukuran dalam satuan byte sebesar 31750447 menggunakan UUID MongoDB,

30050447 menggunakan auto increment MongoDB, 30850447 menggunakan ObjectId MongoDB, 39067648 menggunakan UUID PostgreSQL, dan 35913728 menggunakan auto increment PostgreSQL. Sedangkan, kecepatan dalam proses eksekusi seperti yang ada di tabel 10 dan 11.

Tabel 10. Hasil pengujian 100.000 data MongoDB

Percobaan ke-n	UUID	Auto Increment	ObjectId
1	5.3017	1.7442	3.3189
2	5.3639	1.5575	3.1106
3	5.0406	1.3625	3.0423
4	5.6912	1.7837	3.2588
5	4.9520	1.6281	3.1744

Tabel 11. Hasil pengujian 100.000 data PostgreSQL

Percobaan ke-n	UUID	Auto Increment
1	23.4172	23.2064
2	23.7946	23.7946
3	23.5211	23.3267
4	23.5338	23.3531
5	22.2473	22.3254

5) Hasil pengujian 1.000.000 data

Hasil pengujian ukuran dalam satuan byte sebesar 318482027 menggunakan UUID MongoDB, 301482027 menggunakan auto increment MongoDB, 309482027 menggunakan ObjectId MongoDB, 388136960 menggunakan UUID PostgreSQL, dan 360366080 menggunakan auto increment PostgreSQL. Sedangkan, kecepatan dalam proses eksekusi seperti yang ada di tabel 12 dan 13.

Tabel 12. Hasil pengujian 1.000.000 data MongoDB

Percobaan ke-n	UUID	Auto Increment	ObjectId
1	57.4907	16.3120	37.7385
2	69.3985	18.8558	34.7981
3	66.5925	19.5193	36.3762
4	64.3131	18.7899	33.8675
5	67.8080	19.9754	36.8057

Tabel 13. Hasil pengujian 1.000.000 data

PostgreSQL		
Percobaan ke-n	UUID	Auto Increment
1	230.4203	230.8444
2	292.3990	290.2642
3	259.4126	258.6288
4	282.8291	281.2218
5	306.2742	303.5570

Perhitungan

Data yang sudah berhasil didapatkan dari pengujian sekarang diolah menjadi skor composite performance index atau CPI. Skor CPI dapat dihitung menggunakan rumus yang sudah dijelaskan pada bab 3. Data yang berhasil di dapatkan sekarang di ambil titik minimal dan titik maksimalnya untuk mengetahui titik normal dari data tersebut. Dikarenakan bobot 75% pada kecepatan dan 25% pada ukuran maka perhitungan dilakukan 2 kali sebagai berikut (Rendy yudistira, 2019).

$$CPI = \left(\frac{kecepatan_{min}}{kecepatan} \cdot 0.75 \right) + \left(\frac{ukuran_{min}}{ukuran} \cdot 0.25 \right)$$

Langkah untuk menghitungnya dengan mengambil nilai paling kecil (minimum) dari ukuran dan kecepatan. Normalisasi nilai kecepatan dan ukuran untuk mendapatkan rata rata dari kecepatan dan ukuran dari setiap metode dan jumlah data uji. Hitung CPI dengan bobot 75% untuk kecepatan dan 25% untuk ukuran (Rumandan, 2022). Dari hasil normalisasi rata rata setiap pengujian didapatkan hasil:

Tabel 14. Hasil normalisasi MongoDB

Metode	Kecepatan	Ukuran
UUID	0.01152	31528
AUTO INCREMENT	0.00774	29828
OBJECTID	0.01288	30628

Tabel 15. Hasil normalisasi PostgreSQL

Metode	Kecepatan	Ukuran
UUID	0.0418	90112
AUTO INCREMENT	0.0351	90112

Tabel 16. Hasil normalisasi MongoDB

Metode	Kecepatan	Ukuran
UUID	0.01152	31528
AUTO INCREMENT	0.00774	29828
OBJECTID	0.01288	30628

Tabel 17. Hasil normalisasi PostgreSQL

Metode	Kecepatan	Ukuran
UUID	0.4505	434176
AUTO INCREMENT	0.4193	409600

Tabel 18. Hasil normalisasi MongoDB

Metode	Kecepatan	Ukuran
UUID	0.74434	3164395
AUTO INCREMENT	0.23894	2994395
OBJECTID	0.42944	3074395

Tabel 19. Hasil normalisasi PostgreSQL

Metode	Kecepatan	Ukuran
UUID	4.1589	3874816
AUTO INCREMENT	4.0986	3629056

Tabel 20. Hasil normalisasi MongoDB

Metode	Kecepatan	Ukuran
UUID	5.26988	31750447
AUTO INCREMENT	1.6152	30050447
OBJECTID	3.181	30850447

Tabel 21. Hasil normalisasi PostgreSQL

Metode	Kecepatan	Ukuran
UUID	23.4172	39067648
AUTO INCREMENT	23.2064	35913728

Tabel 22. Hasil normalisasi MongoDB

Metode	Kecepatan	Ukuran
UUID	65.1206	318482027
AUTO INCREMENT	18.0905	301482027
OBJECTID	35.9172	309482027

Tabel 23. Hasil normalisasi PostgreSQL

Metode	Kecepatan	Ukuran
UUID	230.4203	388136960
AUTO INCREMENT	230.8444	360366080

Setelah melakukan normalisasi rumus CPI dapat diimplementasikan untuk menghitung index performanya. Skor terdiri dari 0 sampai dengan 1, menggunakan tren positif. Artinya metode yang mendekati nilai 0 tidak efektif dan metode yang mendekati 1 artinya efektif. Untuk menghitung menggunakan rumus yang sudah disebutkan sebelumnya. Berikut adalah hasil-nya pada setiap jumlah data (Nurdianto *et al.*, 2024):

Tabel 24. Hasil perhitungan CPI MongoDB

Metode	CPI
UUID	0.9426
AUTO INCREMENT	1.0000
OBJECTID	0.9374

Tabel 25. Hasil perhitungan CPI PostgreSQL

Metode	CPI
UUID	0.9625
AUTO INCREMENT	1.0000

Tabel 26. Hasil perhitungan CPI MongoDB

Metode	CPI
UUID	0.9507
AUTO INCREMENT	1.0000
OBJECTID	0.9764

Tabel 27. Hasil perhitungan CPI PostgreSQL

Metode	CPI
UUID	0.8977
AUTO INCREMENT	1.0000

Tabel 28. Hasil perhitungan CPI MongoDB

Metode	CPI
UUID	0.9577
AUTO INCREMENT	1.0000
OBJECTID	0.9766

Gambar 9. Hasil YCSB UUID

Gambar 10. Hasil YCSB ObjectId

Gambar 11. Hasil YCSB ObjectId

Gambar 12. Hasil YCSB Auto Increment

Pembahasan

Berdasarkan hasil pengujian dan analisis yang dilakukan dalam penelitian ini, dapat disimpulkan bahwa pemilihan metode primary key seperti UUID memiliki dampak signifikan terhadap kinerja dan efisiensi penyimpanan, terutama dalam konteks penggunaan *NoSQL* dan *SQL* databases. Dalam pengujian dengan MongoDB, yang menggunakan format BSON dan penyimpanan berbasis *key-value* (Andharini, 2022), penggunaan UUID sebagai primary key menunjukkan performa yang lebih lambat dibandingkan dengan *auto-increment* dan *ObjectId*, terutama saat data yang diinsert berjumlah besar. Hal ini sesuai dengan temuan sebelumnya bahwa ukuran UUID yang lebih besar dan distribusi kunci yang acak berpotensi memperlambat kecepatan *insert* (Bhaswara *et al.*, 2017). Di sisi lain, pada PostgreSQL, meskipun penggunaan UUID sedikit lebih lambat dibandingkan dengan *auto-increment*, perbedaan antara keduanya tidak begitu signifikan, mengingat PostgreSQL lebih mengutamakan transaksi yang konsisten dan terstruktur (Salunke &

Ouda, 2024). Pada aspek efisiensi penyimpanan, UUID cenderung membutuhkan lebih banyak ruang dibandingkan dengan *auto-increment* dan *ObjectId*. Hal ini juga tercermin dalam hasil pengujian ukuran data pada MongoDB dan PostgreSQL, di mana UUID menghasilkan ukuran koleksi atau tabel yang lebih besar. Hal ini dapat dipahami, mengingat bahwa UUID memiliki ukuran 128-bit yang lebih besar dibandingkan dengan nilai *auto-increment* atau *ObjectId* yang lebih kecil (Makris *et al.*, 2021). Penggunaan UUID sebagai *primary key* memberikan keuntungan dari segi keunikan global, yang penting dalam sistem terdistribusi, tetapi mengorbankan ruang penyimpanan yang lebih besar dan sedikit menurunkan efisiensi (Mishra *et al.*, 2022).

Hasil perbandingan dengan metode *Composite Performance Index* (CPI) yang diterapkan pada setiap pengujian memperlihatkan bahwa *auto-increment* memberikan hasil yang lebih optimal dari segi kecepatan dan efisiensi penyimpanan. Nilai CPI untuk *auto-increment* cenderung mendekati 1, menunjukkan bahwa metode ini lebih efisien dalam kedua aspek tersebut, sementara UUID mendapatkan nilai CPI yang lebih rendah, baik pada MongoDB maupun PostgreSQL. Hal ini sejalan dengan penelitian yang menyatakan bahwa meskipun UUID menawarkan keunikan, penggunaannya harus dipertimbangkan dengan hati-hati terutama pada aplikasi yang membutuhkan pengelolaan data dalam jumlah besar (Pebrianto, 2022; Prastowo *et al.*, 2021). Penelitian ini memberikan wawasan bagi pengembang dan administrator basis data dalam memilih metode primary key yang paling sesuai dengan kebutuhan sistem mereka. Bagi aplikasi yang mengutamakan kecepatan *insert* dan efisiensi penyimpanan, *auto-increment* lebih direkomendasikan, sementara UUID lebih cocok untuk aplikasi yang membutuhkan keunikan global, meskipun dengan sedikit pengorbanan dalam hal kecepatan dan efisiensi ruang penyimpanan (Zhou *et al.*, 2022).

4. Kesimpulan dan Saran

Penelitian ini berhasil menguji performa UUID dalam dua aspek utama, yaitu kecepatan dan ukuran. UUID memiliki keunggulan dalam hal keunikan yang tidak diragukan lagi, namun ukurannya yang cukup besar

dapat menjadi masalah pada skala data yang besar. Pada sistem basis data SQL, penggunaan UUID tidak menunjukkan dampak signifikan terhadap kecepatan proses, meskipun penggunaannya mempengaruhi efisiensi penyimpanan karena ukurannya yang lebih besar. Sementara itu, pada sistem basis data NoSQL, pengujian terhadap data kecil (100 hingga 1.000 baris) menunjukkan bahwa performa UUID tidak terlalu terasa, baik dari segi kecepatan maupun ukuran, karena perbedaannya hanya beberapa kilobyte. Namun, pada data yang lebih besar (100.000 hingga 1.000.000 baris), penggunaan UUID mulai menunjukkan penurunan kinerja yang lebih nyata, baik dalam hal kecepatan maupun efisiensi penyimpanan.

Berdasarkan hasil penelitian ini, terdapat beberapa saran yang dapat menjadi pertimbangan bagi pengembang dan peneliti di masa depan. Pertama, bagi pengembang, disarankan untuk mempertimbangkan penggunaan UUID sebagai primary key pada aplikasi yang memerlukan keunikan indeks, seperti pada sistem keranjang e-commerce, di mana setiap entri dalam tabel memerlukan identifikasi unik. Kedua, untuk penelitian selanjutnya, disarankan untuk mengeksplorasi optimasi penggunaan UUID dalam konteks basis data, terutama terkait performa operasi insert. Peneliti juga dapat melakukan pengujian lebih lanjut dengan variasi jenis data dan skenario penggunaan yang berbeda, serta mempertimbangkan pembandingan lain seperti ULID, KSUID, atau MD5 untuk mendapatkan gambaran yang lebih komprehensif mengenai performa UUID dalam berbagai kondisi.

5. Daftar Pustaka

- Bhaswara, F. A., Sarno, R., & Sunaryono, D. (2017). Perbandingan Kemampuan Database NoSQL dan SQL dalam Kasus ERP Retail. *Jurnal Teknik ITS*, 6(2), A511-A514. <https://doi.org/10.12962/j23373539.v6i2.24031>.
- Cahyani, A. D., Basuki, A., & Nafisah, D. (2022). *Pengolahan Basis Data Mongo DB*. Media Nusa Creative (MNC Publishing).
- Cui, X., & Chen, W. (2021). Performance comparison test of HBase and Cassandra based on YCSB. In *2021 IEEE/ACIS 19th International Conference on Computer and Information Science (ICIS)*, 70–77. <https://doi.org/10.1109/ICIS51600.2021.9516864>.
- Keighobad, A., & Demirel, F. B. (2021). Software Platform for Design and Management of Real-time Data from Microneedle-based Wearable Sensors: A front-end (iOS) and a server-side (Node.js) implementation for an IoT system.
- Makris, A., Tserpes, K., Spiliopoulos, G., Zissis, D., & Anagnostopoulos, D. (2021). MongoDB Vs PostgreSQL: A comparative study on performance aspects. *GeoInformatica*, 25, 243-268.
- Mishra, N., Chakraborty, A., & Mukhopadhyay, D. (2022, August). *Breaking Cross-world isolation on ARM TrustZone through EM Faults, Coredumps, and UUID Confusion*.
- Nokeri, T. C. (2021). Python Web Frameworks and Apps. In *Web App Development and Real-Time Web Analytics with Python: Develop and Integrate Machine Learning Algorithms into Web Apps* (pp. 79-85). Berkeley, CA: Apress.
- Pebrianto, J. (2022). Perbandingan Kecepatan Baca Dan Tulis Data Pada Mysql Menggunakan Primary Key Auto Increment Dengan Universally Unique Identifier (UUID). *Media Jurnal Informatika*, 14(2), 86-96. <https://doi.org/10.35194/mji.v14i2.2681>.
- Prastowo, A. A., Purwadi, A., Murtono, T., Upe, A., Rusli, M., Hos, J., Moita, S., Wicaksono, G., Wan Khairuldin, W. M. K. F., & Sono, M. G. (2021). Application of decision support system using composite performance index algorithm. *Journal of Physics: Conference Series*, 1933(1), 012018. <https://doi.org/10.1088/1742-6596/1933/1/012018>.
- Ravshanovich, A. R. (2024). DATABASE STRUCTURE: POSTGRESQL DATABASE. *PSIXOLOGIYA* 1/4

- SOTSIOLOGIYA ILMIAH JURNALI*, 2(7), 50-55.
- Rumandan, R. J. (2022). Implementasi Composite Performance Index (CPI) Pada Sistem Pendukung Keputusan Pemilihan Mitra Pengiriman Barang. *KLIK: Kajian Ilmiah Informatika Dan Komputer*, 3(1), 17-25.
- Salunke, S. V., & Ouda, A. (2024). A Performance Benchmark for the PostgreSQL and MySQL Databases. *Future Internet*, 16(10), 382. <https://doi.org/10.3390/fi16100382>.
- Satria, B., Sidauruk, A., Wardhana, R., Al Akbar, A., & Ihsan, M. A. (2022). Penerapan composite performance index (CPI) sebagai metode pada sistem pendukung keputusan seleksi penerima beasiswa. *The Indonesian Journal of Computer Science*, 11(2). <https://doi.org/10.33022/ijcs.v11i2.3056>.
- Silalahi, M. (2018). Perbandingan performansi database mongodb dan mysql dalam aplikasi file multimedia berbasis web. *Computer Based Information System Journal*, 6(1), 63-63. <https://doi.org/10.33884/cbis.v6i1.574>.
- Singgih, M. L., & Gunarta, I. K. (2024, August). Building Composite Performance Index for Broadband Internet Customer Experience. In *2024 IEEE International Symposium on Consumer Technology (ISCT)* (pp. 669-675). IEEE. <https://doi.org/10.1109/ISCT62336.2024.10791161>.
- Timisela, H. (2024). *PERBANDINGAN KINERJA JENIS-JENIS BASIS DATA NOSQL DALAM MANAJEMEN DATA BIOMEDIK= PERFORMANCE COMPARISON OF NOSQL DATABASE TYPES IN BIOMEDICAL DATA MANAGEMENT* (Doctoral dissertation, Universitas Hasanuddin).
- Triebel, D., Reichert, W., Bosert, S., Feulner, M., Okach, D. O., Slimani, A., & Rambold, G. (2018). A generic workflow for effective sampling of environmental vouchers with UUID assignment and image processing. *Database*, 2018. <https://doi.org/10.1093/database/bax096>.
- Zhou, X., Chai, C., Li, G., & Sun, J. (2022). Database meets artificial intelligence: A survey. *IEEE Transactions on Knowledge and Data Engineering*, 34(3), 1096–1116. <https://doi.org/10.1109/TKDE.2020.2994641>.