PERBANDINGAN KECEPATAN AKSES DATA MENGGUNAKAN "*DISKBASED TABLE*" DENGAN "*IN-MEMORY OPTIMIZED TABLE*" PADA DATABASE RELASIONAL STUDI KASUS : HEKATON ENGINE

Mardhiya Hayaty[1)]

[1)] *Teknik Informatika STMIK AMIKOM Yogyakarta*
Jl Ring road Utara, Condongcatur, Sleman, Yogyakarta 55281
Email : mardhiya_hayati@amikom.ac.id[1)]

Abstrak

Ledakan data menyebabkan terjadinya peningkatan volume data atau big data. Ukuran data yang besar menyebabkan akses terhadap data menjadi sangat lambat. DataBase Management System (DBMS) adalah sebuah software yang digunakan untuk mengelola database. DBMS konvensional menggunakan media disk sebagai media penyimpanan datanya, hal ini menyebabkan penurunan performansi DBMS karena harus melakukan akses langsung ke disk dan membawa data yang akan diolah ke memory,maka munculah konsep in-memory database yang telah dikembangkan oleh beberapa vendor DBMS, data akan disimpan di dalam memory dan akan diproses.
Tujuan penelitian ini adalah memberikan gambaran secara detil tentang penerapan penggunaan in-memory terhadap kecepatan akses data dibandingkan dengan akses data secara langsung ke I/O disk, dari serangkaian pengamatan yang akan dilakukan diharapkan penelitian ini dapat menghasilkan data waktu (dalam skala waktu menit,detik dan atau milidetik) yang diperlukan untuk mengeksekusi sebuah perintah akses data ke database.
Pengamatan yang dilakukan pada Hekaton engine dengan spesifikasi hardware yang telah ditentukan, penggunaan memory optimized table mempunyai kecepatan akses data yang signifikan dibandingkan diskbase table pada katagori akses data "insert" yaitu 190 sampai dengan 200x lebih cepat. Katagori akses data "select, update,delete" penggunaan memory optimized table mempunyai kecepatan akses data yang signifikan pada tipe durability:schema_only dengan jenis hasil query single row tetapi tidak untuk jenis hasil query multiple row. Pemilihan tipe durability:schema_only disarankan untuk sistem yang tingkat kegagalan sangat minimal atau jarang terjadi.

Kata kunci: *In-Memory, Memory Optmized Table, Data Durability*

1. Pendahuluan

Kebutuhan akan pentingnya data dan informasi dihampir semua bidang kehidupan menyebabkan laju pertumbuhan data menjadi sangat cepat, sehingga terjadinya ledakan data (*data explosion*). Ledakan data menyebabkan terjadinya peningkatan volume data atau *big data*. *Big data* adalah fenomena yang terjadi secara alami akibat perkembangan data yang sangat besar. Ukuran data yang besar menyebabkan akses terhadap data menjadi sangat lambat sehingga berpengaruh terhadap penyajian sebuah informasi.
DataBase Management System (DBMS) adalah sebuah *software* yang digunakan untuk mengelola database. DBMS konvensional menggunakan media disk sebagai media penyimpanan datanya, dan - menggunakan memory hanya sebagai *buffer* atau tempat penampungan sementara untuk data-data yang diolah, hal ini menyebabkan penurunan performansi DBMS karena harus melakukan I/O disk untuk membawa data yang akan diolah ke *memory*. Sistem berbasis disk tersebut tidak bisa lagi menawarkan respon yang tepat waktu karena akses yang tinggi ke hard disk [1].
Relational database management system telah dirancang pada akhir tahun 1970, dirancang dengan asumsi *memory* yang terbatas dan mahal serta dengan *database* yang lebih besar dari pada *memory* utama sehingga data berada pada *disk*. *Hardware* saat ini mempunyai *memory sizes* dan *volume disk* yang beribu kali lebih besar dan *processor* yang beribu kali lebih cepat, sehingga asumsi yang dikatakan sebelumnya tidak benar, selain itu penggunaan akses *disk* tidak meningkatkan kecepatan dan tetap menjadi bagian paling lambat di system[6].
Berdasarkan penjelasan tersebut, maka munculah konsep *in-memory database* yang telah dikembangkan oleh beberapa *vendor DBMS*, data akan disimpan di dalam memory dan akan proses, cara kerja seperti ini diharapkan dapat meminimalkan lalu lintas permintaan data dari *disk* ke *memory* sehingga akses data menjadi lebih cepat.

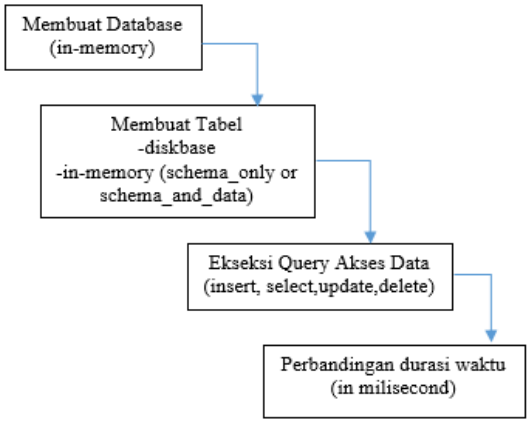
Berdasarkan latar belakang tersebut terdapat beberapa masalah yang menjadi rumusan masalah dalam penelitian ini, yaitu:

1. Apakah penggunaan *in-memory* dapat meningkatkan kecepatan akses data dibandingkan akses data langsung ke *I/O disk* ?
2. Katagori akses data yang manakah yang akan berpengaruh, apakah "insert, update, delete, select" ?

Tujuan penelitian ini adalah memberikan gambaran secara detil tentang penerapan penggunaan *in-memory* terhadap kecepatan akses data dibandingkan dengan akses data secara langsung ke I/O disk, dari serangkaian pengamatan yang akan dilakukan diharapkan penelitian ini dapat menghasilkan data waktu (dalam skala waktu menit,detik dan atau milidetik) yang diperlukan untuk mengeksekusi sebuah perintah permintaan data ke *database*. Hasil-hasil tersebut di analisa apakah terdapat pengaruh yang signifikan penggunaan *in-memory* terhadap performansi DBMS dibandingkan dengan akses data secara langsung ke *disk*.

Penelitian ini akan melakukan beberapa serangkaian eksperimen atau ujicoba dibeberapa katagori akses data ke database dengan menggunakan "*diskbase*" dan "*in-memory*", ujicoba dilakukan pada server DBMS yang sama dan dengan spesifikasi hardware yang sama. Hasil ujicoba adalah durasi/waktu eksekusi dari katagori akses data menggunakan "*diskbase*" dengan katagori akses data "*in-memory"*

Langkah-langkah dalam melakukan ujicoba dapat dilihat pada gambar 1 di bawah ini :



Gambar 1. *Langkah-langkah penelitian*

Penelitian bertema *in-memory database* sudah pernah diteliti oleh peneliti-peneliti sebelumnya diantaranya penelitian berjudul *"A study for Performance Comparison of Different In-Memory Databases"* [5]. Pada penelitian ini dilakukan perbandingan kinerja penggunaan *in-memory database* dibeberapa server DBMS yaitu TimesTen, Altibase, solidDb dan SQLite dengan perbandingan kinerja *in-memory* pada sistem operasi *windows* dan *LINUX*, penellitian tersebut menyimpulkan bahwa database Altibase yang berada pada sistem operasi LINUX menunjukan performansi yang lebih baik dibandingkan database lain.

Penelitian bertema *in-memory database* yang lain adalah *In-Memory Big Data Management and Processing: A Survey* [1]. Penelitian ini berfokus kepada prinsip-prinsip desain teknologi *in-memory* yang dimiliki database, seperti *data management system for relational data,data management system for graph data, data management system for streams*, sehingga pada penelitian ini dihasilkan teknik-teknik untuk pengoptimalkan design *in-memory database*.

Berdasarkan uraian penelitian sebelumnya, maka penulis tertarik untuk meneliti apakah penggunaan *in-memory* database mempunyai pengaruh yang signifikan terhadap kecepatan akses data dibandingkan dengan berbasis *"diskbase"*.

DBMS adalah sebuah sistem yang memungkinkan pengguna untuk mendefinisikan, membuat, memelihara, dan mengkontrol akses ke database, seperti dijelaskan oleh Thomas M Connaly " *DBMS is a software system that enables user to define, create, maintain and control access to the database"* [2].

Structured Query Language adalah sebuah bahasa yang digunakan untuk mengakses data dalam basis data relasional. Bahasa ini merupakan bahasa standar yang digunakan dalam manajemen basis data relasional.

SQL mulai dikenalkan di sebuah seminar oleh E.F Codd pada sebuah laboratorium IBM di San Jose tahun 1970. Pada tahun 1974 D. Chamberlin juga dari laboratorium IBM mendefinisikan sebuah bahasa yang disebut dengan Stuctured Query Language atau dieja dengan "See-Quel" [2].

Bahasa SQL yang digunakan untuk memanipulasi data yang ada di *database*, terdiri dari sintak *Insert*, *Update*, *Delete*, *Select*. Perintah *Insert* digunakan untuk memasukkan data ke dalam database, perintah *update* digunakan untuk memodifikasi atau merubah data, perintah *delete* digunakan untuk menghapus data, perintah *select* digunakan untuk menampilkan data yang telah tersimpan didatabase.

In-memory database adalah sebuah teknologi baru yang digunakan untuk mempercepat akses data ke database yang selama ini digunakan oleh DBMS konvensional, data disimpan pada media penyimpanan cakram yang lebih lambat daripada memori RAM yang jauh lebih cepat. Dengan menggunakan media penyimpanan cakram (disk)tersebut, terdapat banyak overhead yang terjadi saat mengakses record yang tersimpan di dalam database. Namun, saat disimpan di dalam memori, record tersebut jauh lebih sederhana strukturnya, selain tentunya jauh lebih cepat kinerjanya.

In-memory database diperkenalkan sejak 1980 adalah database yang dirancang untuk menyimpan

semua data dalam memori utama. *In-memory* database memberikan kinerja yang signifikan dengan mengizinkan waktu respon permintaan yang lebih cepat dibandingkan akses langsung ke *I/O disk*. Hal ini juga ditandai peningkatan jumlah perusahaan yang menawarkan *in-memory* teknologi, seperti Oracle Exadata X31 dan SAP HANA2 [5].

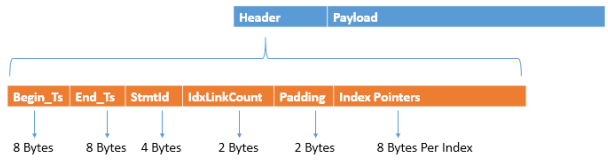
Perbedaan yang paling mendasar dalam penggunaan *"in-memory optimized table"* adalah seluruh data dan *index* tersimpan di dalam memory, proses penggunaan data akan selalu tersedia di dalam memory, sedangkan penggunaan *"disk-based table"*, mesin selalu membaca halaman data yang diperlukan dari "*disk*".

Perbedaan lainnya adalah struktur penyimpanannya. *"Disk-based tables"* menyusun baris data kedalam 8K unit yang disebut dengan "*data pages*"

Data pages adalah unit penyimpanan pada disk dan memori, mesin akan membaca dan menulis data dari *data pages* yang relevan. Sebuah *data pages* hanya berisi data dari sebuah tabel atau indeks. Proses penggunaan akan memodifikasi baris di berbagai *data pages* yang diperlukan, dan selama proses tersebut engine mencatat log ke disk. Operasi teesebut sering menyebabkan banyak akses fisik ke "I / O" secara acak.

Memory-optimized table, tidak terdapat "data pages" tetapi hanya ada baris data(*data row*) yang ditulis berurutan ke memori, dengan setiap baris mengandung indeks "pointer" ke baris berikutnya, sehingga setiap membuat table pada *memory-optimized table* harus membuat setidaknya satu indeks dapat digunakan untuk menghubungkan semua baris data [3].

Data rows pada *in-memory* setiap barisnya terdiri dari *row header* dan *payload* yang berisi atribut baris (data aktual). Struktur baris pada in-memory dapat dilihat pada gambar 2 berikut ini :



Gambar 2. *Struktur Baris In-Memory*

Row Header akan mencatat waktu transaksi pada field "Begin-Ts" dan "End-Ts", setiap perintah transaksi mempunyai Id dan tercatat pada field "StmtId", "index pointer" menunjukkan hubungan yang terjadi diantara masing-masing baris pada sebuah tabel.

PayLoad Area adalah data itu sendiri, mempunyai indek kolom beserta kolom-kolom yang lain pada sebuah baris.

Peranan penggunaan "index" pada in-memory optimized table sangat penting, melalui index tersimpan pointer yang bisa menghubungkan antar baris di dalam sebuah tabel. Setiap tabel minimal harus mempunyai satu index dan maksimal 8 buah index. Terdapat 2 tipe index yatu range indexes dan hash indexes. Setiap kolom yang didefinisikan sebagai primary key maka secara default akan diberikan hash index, sedangkan kolom non-primary key bisa diberikan hash index atau range index.

Hash indexes menggunakan fungsi "hash" untuk nilai di kolom kunci indeks di setiap baris. Setiap nilai hash yang disimpan dalam sebuah *hash bucket*. *Hash bucket* pada dasarnya adalah sebuah array dari pointer [4].

Hash index berguna untuk data yang relatif unik yang bisa didapatkan dengan query berpredikat kesetaraan, tetapi jika akan mencari data berdasarkan rentang nilai atau kisaran tertentu, maka penggunaan hash index tidak tepat, akan lebih baik menggunakan *range index* [3].

Range index menghubungkan semua baris dalam tabel "leaf level" mereka. Setiap baris dalam sebuah tabel akan dapat diakses oleh pointer di "leaf level" ukuran halaman default adalah 8K tapi yang dapat bervariasi berdasarkan pada data yang menempati halaman indeks.

Penggunaan in-memory pada database memerlukan sebuah *"memory –optimized_data " file group* yang berisikan minimal satu "container" yang digunakan untuk menyimpan *"checkpoint files"* selama *database recovery*. Sintak untuk pembuatan database seperti script di bawah ini :

```
CREATE DATABASE [databasename]
ALTER DATABASE [databaseName]
ADD FILE (NAME = [InMemory_dir], FILENAME=
    '.....folder location.')
TO FILEGROUP [groupName];

ALTER DATABASE [databaseName]  ADD FILEGROUP
    [groupName]
CONTAINS MEMORY_OPTIMIZED_DATA;
```

Sintak untuk membuat "memory-optimized table" secara umum sama dengan sintak membuat *disk-based table*, tetapi ada beberapa aturan dan batasan seperti tipe data, indexes, contraints yang dapat di dukung oleh *memory optimized table*. Menentukan apakah sebuah tabel merupakan "memory-optimized table", pada deklarasi sintak pembuatan tabel menggunakan "memory_optimized=on", setelah itu diikuti oleh kolom-kolom suatu tabel beserta tipe datanya, termasuk menentukan minimal satu buah index. Sintak untuk pembuatan "memory-optimized table" seperti script di bawah ini :

```
CREATE TABLE [tablename]
( colName1 vachar(30) NOT NULL PRIMARY KEY
NONCLUSTURED HASH WITH (BUCKET
COUNT=10000),
Colname2 varchar(10),
Colname3 datetime)
WITH (MEMORY_OPTIMIZED=ON,
DURABILITY=SCHEMA_AND_DATA);
```

Struktur data *memory-optimized table* semua data disimpan dalam memori sepanjang waktu, tetapi untuk terus menjamin daya tahan (durability) data, diperlukan *log* transaksi pada *memory-optimized table* (bukan indeks) dan log transaksi disimpan pada *disk* disebut dengan *checkpoint files.File* digunakan hanya untuk proses *database recovery* jika terjadi crash pada mesin database yang menyebabkan mesin database tidak berfungsi dengan baik.
In-memory oprimized menyediakan 2 pilihan untuk *data durability* yaitu *SCHEMA_ONLY* dan *SCHEMA_AND_DATA* [3].
Jika sebuah *memory optimized table* didefinisikan *DURABILITY=SCHEMA_ONLY* maka mesin hanya mencatat perubahan *schema*(struktur tabel) bukan data, sehingga table tidak akan mempunyai data setelah proses *recovery database* selesai. Berbeda halnya jika *DURABILITY=SCHEMA_AND_DATA*, mesin akan akan mencatat perubahan *schema* dan data, sehingga jika terjadi *crash*, data akan tetap terselamatkan pada proses *recovery* selesai.

2. Pembahasan

Penelitian dilakukan menggunakan "hekaton engine" yang dimiliki oleh software DBMS SQL server 2014, dengan spesifikasi hardware processor 2.93Hz, 64bit operating system x64–based processor, memory (RAM) 16Gb. Membuat sebuah database dengan bertipe *"memory-optimized"*. Pengamatan dilakukan dengan membandingkan durasi waktu eksekusi perintah *Data Manipulation Language (insert, update, delete, select)* antara disk-based table *(non optimized table)* dengan penggunaan *in-memory (optimized table)*.

Proses eksekusi query dilakukan secara *single connection* dan atau *single user* artinya satu koneksi untuk satu user, hasil baris/record eksekusi query terdiri dari 2 jenis yaitu *single row affected* (hasil eksekusi query menghasilkan satu baris) dan *multiple row affected* (hasil ekseskusi query menghasilkan banyak baris). *Multiple row affected* terbagi dalam beberapa *record :* 10,100,1000 dan 2000 *record* khusus untuk perintah *query update, select, delete.* Query insert akan diujicoba memasukkan data sebanyak 500,5.000,50.000,500.000 record. Pengamatan dilakukan dengan melihat durasi waktu, kecepatan dan perlambatan diantara *disk-based table* dengan *memory-optimized table* untuk masing-masing jenis *durability*.
Hasil pengamatan untuk katagori *DML-Insert* seperti tabel 1 di bawah ini:

Tabel 1. *Waktu Eksekusi pada DML-Insert*

ROW AFFECTED	WAKTU EKSEKUSI (in-milisecond)			SPEED UP/SPED DOWN	
	DISK BASED	MEMORY OPTIMIZED DURABILITY		DISKBASED VS MEMORY OPT	
		SchemaOnly	Schema_and_data	SchemaOnly	Schema_and_data
500	3047	16	3234	190	0.94
5000	30703	157	31437	196	0.98
50000	303375	1516	317188	200	0.96
500000	3215100	16641	3165492	193	1.02

Ujicoba dengan melakukan *insert* data sebanyak 500 sampai dengan 500.000 *record*. Hasil ujicoba ditunjukkan pada tabel 1 bahwa penggunaan *memory-optimized table* untuk katagori *durability :schema only* terjadi peningkatan kecepatan 190x sampai dengan 200x lebih cepat dibandingkan *diskbase table*, sedangkan penggunaan *memory-optimized table* untuk katagori *durability :schema_and_data* memiliki kecepatan yang hampir sama dengan *diskbase table*, sedangakan hasil pengamatan katagori *DML-Select* ditunjukkan seperti tabel 2 di bawah ini :

Tabel 2. Waktu Eksekusi pada DML-Select

ROW AFFECTED	WAKTU EKSEKUSI (in-milisecond)			SPEED UP/SPED DOWN	
	DISK BASED	MEMORY OPTIMIZED DURABILITY		DISKBASED VS MEMORY OPT	
		SchemaOnly	Schema_and_data	SchemaOnly	Schema_and_data
Single Row	16	0	0	16.00	16.00
Multiple Row					
10	47	47	47	1.00	1.00
100	78	47	63	1.66	1.24
1000	172	47	147	3.66	1.17
2000	188	63	203	2.98	0.93

Ujicoba dilakukan dengan menghasilkan single row (1 baris) data dan multiple row (banyak baris) data.
Hasil ujicoba bisa dilihat pada tabel 2, penggunaan memory-optimized table pada perintah *select* (*single row)* mempunyai peningkatan kecepatan 16x lebih cepat dari pada *diskbase table* untuk tipe *durability:schema_only* maupun *shema_and_data*, sedangkan *multiple row* tidak terjadi peningkatan yang signifikan antara *diskbase table* maupun untuk semua tipe *durability*.
Waktu eksekusi pada perintah *Update (single row)* terdapat peningkatan kecepatan 31x lebih cepat penggunaan *memory-optimized table* dengan tipe *durability:schema only* dibandingkan *diskbased table* dan hanya 1.94x lebih cepat untuk tipe *durability:schema_and_data*. Pada hasil *multiple row* penggunaan memory optimized table tipe *durability:schema_only* hanya 1-2x lebih cepat dan lebih lambat pada tipe durability:schema_and_data, Hasil ujicoba katagori *Update* dapat dilihat pada tabel 3 dibawah ini:

Tabel 3. *Waktu Eksekusi pada DML-Update*

ROW AFFECTED	WAKTU EKSEKUSI (in-milisecond)			SPEED UP/SPED DOWN	
	DISK BASED	MEMORY OPTIMIZED DURABILITY		DISKBASED VS MEMORY OPT	
		SchemaOnly	Schema_and_data	SchemaOnly	Schema_and_data
Single Row	31	0	16	31.00	1.94
Multiple Row					
10	32	32	63	1.00	0.51
100	62	31	78	2.00	0.79
1000	94	47	203	2.00	0.46
2000	93	47	93	1.98	1.00

Hasil ujicoba untuk perintah *delete* dapat dilihat pada tabel 4. Penggunaan *memory optimized table* pada hasil *single row* mempunyai kecepatan 32x lebih cepat dibandingkan *diskbase table*, sedangkan untuk *multiple row* mempunyai kecepatan yang bervariasi, dapat dilihat pada tabel 4 dibawah ini:

Tabel 4. *Waktu Eksekusi pada DML-Delete*

ROW AFFECTED	WAKTU EKSEKUSI (in-milisecond)			SPEED UP/SPED DOWN	
	DISK BASED	MEMORY OPTIMIZED DURABILITY		DISKBASED VS MEMORY OPT	
		SchemaOnly	Schema_and_data	SchemaOnly	Schema_and_data
Single Row	32	0	0	32.00	32.00
Multiple Row					
10	15	31	31	0.48	0.48
100	63	31	62	2.03	1.02
1000	172	47	62	3.66	2.77
2000	297	31	47	9.58	6.32

Berdasarkan dari ujicoba yang telah dilakukan, penggunaan *memory optimized table* untuk tipe *durability:schema_only* lebih cepat dibanding akses data menggunakan *diskbase table*, hal tersebut dikarenakan pada tipe *durability:schema_and_data* mesin mencatat dan menyimpan *log* kepada *disk (checkpoint file)* sebagai persiapan *recovery* jika mesin terjadi *crash* atau kegagalan proses.

3. Kesimpulan

Pengamatan yang dilakukan pada Hekaton engine dengan spesifikasi hardware yang telah ditentukan, penggunaan *memory optimized table* mempunyai kecepatan akses data yang signifikan dibandingkan *diskbase table* pada katagori akses data "insert" yaitu 190 sampai dengan 200x lebih cepat. Katagori akses data "select, update,delete" penggunaan *memory optimized table* mempunyai kecepatan akses data yang signifikan pada tipe durability:schema_only dengan jenis hasil query *single row* tetapi tidak untuk jenis hasil query *multiple row*. Pemilihan tipe durability:schema_only disarankan untuk sistem yang tingkat kegagalan sangat minimal atau jarang terjadi.
Penelitian selanjutnya diharapkan terdapat parameter berbagai ukuran memori yang diperlukan untuk menghasilkan kecepatan akses data yang maksimal berdasarkan ukuran *database* atau pertumbuhan data pada sebuah *database*.

Daftar Pustaka

[1] Hao Zhang; Gang Chen & others. "In-Memory Big Data Management and Processing: A Survey". *Knowledge and Data Engineering, IEEE Transactions* Volume: 27 no 7 (Pp: 1920 – 1948) IEEE, July 2015
[2] Thomas M. Connaly, Carolyn E, *"Database Systems"*, Pearson, 2010
[3] Kalen Delaney, *"SQL Server Internal : In-Memory OLTP Inside the SQL Server 2014 Hekaton Engine"*, Simple Talk Publishing, 2014
[4] Adam Jorgensen, Bradley Ball, Steven Wort & others. *"Professional - SQL Server 2014 Administration"*, Wiley Brand, 2014
[5] Oguducu, S.G.; Gayberi, M.; Akpimar, E, "A study for Performance Comparison of Different In-Memory Databases", In *Application of Information and Communication Technologies (AICT), 7th International Conference* . IEEE, 2013
[6] Benjamin Nevarez, *Query Tuning & Optimization*, Mc Graw Hill Education, 2015

Biodata Penulis

Mardhiya Hayaty, memperoleh gelar Sarjana Teknik (S.T), Fakultas Teknik Industri Jurusan Teknik Informatika Universitas Ahmad Dahlan Yogyakarta, lulus tahun 2003. Memperoleh gelar Magister Komputer (M.Kom) Program Pasca Sarjana Magister Teknik Informatika STMIK AMIKOM Yogyakarta, lulus tahun 2013. Saat ini menjadi Dosen di STMIK AMIKOM Yogyakarta.

ISSN : 2302-3805