© Jurnal Nasional Teknik Elektro dan Teknologi Informasi
Karya ini berada di bawah Lisensi Creative Commons Atribusi-BerbagiSerupa 4.0 Internasional
DOI: 10.22146/jnteti.v14i2.17315

Studi Komparasi Kinerja *Object-Relational Mapping* Berdasarkan Implementasi *Data Source Architectural Pattern*

Muhammad Rezy Anshari[1], Redi Ratiandi Yacoub[1], Herry Sujaini[1], Bomo Wibowo Sanjaya[1], Eva Faja Ripanti[1]

[1] Jurusan Teknik Elektro, Fakultas Teknik, Universitas Tanjungpura, Pontianak, Kalimantan Barat 78124, Indonesia

[Diserahkan: 3 Desember 2024, Direvisi: 21 Februari 2025, Diterima: 16 April 2025]
Penulis Korespondensi: Muhammad Rezy Anshari (email: research.murean@gmail.com)

INTISARI — *Object-relational mapping* (ORM) merupakan teknik pemetaan (*mapping*) antara *in-memory objects* dan tabel pada basis data. ORM mengimplementasi *data source architectural patterns* (DSAP), di antaranya Data Mapper dan Active Record. Komparasi kinerja kedua *pattern* perlu dilakukan karena adanya indikasi perbedaan kinerja serta mengingat perannya yang signifikan terhadap proses bisnis sebuah sistem. Studi ini bertujuan melakukan komparasi dan analisis terhadap kinerja durasi eksekusi dan konsumsi memori secara kuantitatif serta fungsi-fungsi yang memengaruhinya pada ORM yang mengimplementasi Data Mapper dan Active Record. Doctrine (Data Mapper) dan Eloquent (Active Record) dijadikan sebagai objek studi. *Profiling* kinerja pada ORM dilakukan dalam bentuk *library*, tidak dibundel dalam *framework*. *Profiling* mencakup operasi *create*, *read*, *update*, *and delete* (CRUD) dan *lookup* berdasarkan metrik ukur tertentu serta dilakukan dengan variasi jumlah *database record*. Proses *profiling* diotomatisasi melalui *script* yang memanfaatkan kombinasi Xdebug dan Apache Benchmark. Analisis dilakukan dengan KCachegrind dan *Big O notation*. Analisis menghasilkan grafik kinerja dan *relative percentage difference* serta kontribusi fungsi-fungsi terhadap kinerja. Hasil menunjukkan kinerja konsumsi memori Active Record unggul atas Data Mapper. Data Mapper unggul dalam kinerja durasi eksekusi pada sebagian besar kombinasi operasi dan metrik. Kelompok fungsi *database transaction*, *object serialization*, dan *record retrieval* merupakan kontributor terbesar terhadap kedua kinerja serta tambahan kelompok fungsi *object and database synchronization* untuk Active Record. Kompleksitas fungsi-fungsi kontributor terbesar pada Data Mapper lebih tinggi dibandingkan Active Record. Studi berikutnya dapat memanfaatkan konsep otomatisasi pada proses *profiling* dan mensubstitusi Xdebug sesuai kebutuhan bahasa pemrograman yang digunakan oleh ORM.

KATA KUNCI — ORM, Data Mapper, Active Record, *Data Source Architectural Patterns*, Durasi Eksekusi, Konsumsi Memori.

I. PENDAHULUAN

Object-relational mapping (ORM) adalah sebuah teknik pemetaan (*mapping*) antara *in-memory objects* dan tabel-tabel pada basis data [1]. ORM umum digunakan dalam pengembangan aplikasi yang berparadigma *object-oriented programming* (OOP) dan berinteraksi dengan *database management system* (DBMS). ORM memungkinkan eksekusi operasi DBMS tanpa perlu menulis *structured query language* (SQL) secara langsung, tidak terikat dengan DBMS tertentu, sehingga meningkatkan efisiensi pada proses pengembangan perangkat lunak [2]. ORM turut membantu pengembang fokus dalam menulis kode program, menurunkan potensi *syntax error* pada SQL, bertindak sebagai *buffer zone* dalam hal *cache* [3], dan memberikan solusi terhadap *semantic gap* antara DBMS dan OOP [4]. ORM dapat menjadi sebab penurunan kinerja pada sebuah aplikasi [4], tetapi dapat diabaikan dengan mempertimbangkan manfaat yang dimilikinya [2], [5]. Dengan demikian, bagi sebuah ORM, kinerja merupakan hal yang sangat penting [2]. ORM mengimplementasi *data source architectural patterns* (DSAP) yang memiliki beberapa *pattern*, di antaranya adalah Data Mapper dan Active Record. Perbedaan mendasar antara keduanya terletak pada independensi antara *domain object* dan basis data. Data Mapper memiliki *layer of mappers* yang menjadi perantara data antara *domain object* dan basis data, sehingga keduanya independen satu sama lain. Sebaliknya, Active Record mengharuskan adanya ikatan yang kuat (*coupled*) antara *domain object* dan basis data agar lebih sederhana [6], [7].

Data Mapper dan Active Record terkait dengan *domain model*, yaitu salah satu dari *domain logic patterns* yang umum diimplementasikan dalam *domain layer*. *Domain layer* merupakan salah satu lapisan primer dalam arsitektur perangkat lunak yang di dalamnya mengandung inti domain atau proses bisnis dari sebuah sistem. *Domain model* dapat bersifat sederhana ataupun kompleks. *Domain model* sederhana umumnya hanya memerlukan satu *domain object* untuk tiap tabel basis data [6]. Dengan kesederhanaan tersebut, logis jika *domain layer* dan *data source layer* digabungkan dalam satu *object*, sehingga Active Record dapat digunakan [6], [7]. *Domain model* kompleks dapat terdiri atas *inheritance* dan beragam *OOP pattern*, sehingga dapat terlihat sangat berbeda dengan desain basis data. Dengan kompleksitas tersebut, pemisahan antara *domain layer* dan *data source layer* lebih tepat untuk dilakukan sehingga Data Mapper dapat digunakan [6].

Perbedaan Data Mapper dan Active Record dalam mengatur interaksi antara *domain layer* dan *data source layer* mengindikasi kemungkinan terdapat perbedaan kinerja saat mengeksekusi tugas yang sama. Adanya indikasi perbedaan kinerja dan pentingnya kinerja tersebut bagi sebuah ORM serta peran kedua *pattern* yang signifikan terhadap proses bisnis sebuah sistem menyebabkan studi komparasi terhadap Data Mapper dan Active Record dinilai perlu dilakukan.

Studi ini bertujuan melakukan komparasi dan analisis terhadap kinerja durasi eksekusi dan konsumsi memori secara kuantitatif serta fungsi-fungsi yang memengaruhi kinerja

tersebut pada ORM yang mengimplementasi Data Mapper dan Active Record. Objek studi komparasi adalah ORM yang mengimplementasi Data Mapper dan Active Record. Dipilih Doctrine sebagai perwakilan Data Mapper dan Eloquent sebagai perwakilan Active Record. Keduanya merupakan ORM populer berbasis PHP. Kepopuleran keduanya tidak terlepas dari *framework* yang menaunginya, yaitu Symfony untuk Doctrine dan Laravel untuk Eloquent. Perbandingan statistik keduanya, yang dikutip dari packagist.org saat penulisan dilakukan, adalah untuk level *framework*, Laravel mengungguli Symfony dengan 396 juta berbanding 82 juta instalasi dan 33.140 *stars* berbanding 30.034 *stars*. Sementara itu, pada level ORM terjadi sebaliknya, Doctrine mengungguli Eloquent dengan 233 juta instalasi berbanding 42 juta instalasi dan 9.993 *stars* berbanding 2.703 *stars*.

Referensi [8]–[10] telah melakukan komparasi terhadap beberapa ORM yang mengimplementasi Data Mapper maupun Active Record. *Profiling* kinerja pada studi-studi tersebut dilakukan dalam level *framework*. Dengan kata lain, kinerja ORM diukur dalam kondisi setiap ORM dibundel dalam *framework* yang berbeda. Studi yang akan dilaksanakan ini melakukan *profiling* kinerja ORM dalam bentuk *library*, sehingga ORM tidak dibundel dalam *framework* sebagaimana yang dilakukan pada studi-studi sebelumnya. Pemilihan ORM dalam bentuk *library* dijadikan sebagai variabel kontrol. Dengan demikian, perbedaan perlakuan terhadap setiap ORM selama proses komparasi dapat diminimalkan. Hal tersebut dilakukan mengingat setiap *framework* memiliki alur tersendiri dalam mengeksekusi operasi yang sama. Studi ini juga berbeda dari penelitian sebelumnya [11]. Meskipun keduanya melakukan *profiling* ORM dalam bentuk *library*, studi ini menambahkan komparasi dengan ORM lain, yaitu Eloquent, yang mengimplementasi DSAP berbeda dengan Doctrine. Studi ini menggunakan metrik pengukuran [12], [13] dan menggunakan variasi jumlah *database record* [2], [9], [11], [14] dalam proses *profiling*. Selain itu, studi ini menambahkan analisis terhadap fungsi-fungsi yang memengaruhi kinerja yang diukur. Kinerja yang diukur dalam studi ini adalah durasi eksekusi dan konsumsi memori. Kedua kinerja tersebut telah banyak digunakan dalam studi komparasi. Pemilihan durasi eksekusi sebagai kinerja yang diukur telah dilakukan sebelumnya [2], [8], [10], [11], [14]–[18], demikian juga untuk konsumsi memori [2], [8], [10], [16]–[18].

Dalam proses *profiling*, mirip dengan studi-studi sebelumnya, Doctrine dan Eloquent diimplementasikan ke dalam sebuah aplikasi. *Profiling* kemudian dilakukan pada aplikasi tersebut. Perbedaan dengan studi-studi sebelumnya terletak pada proses *profiling*. Proses *profiling* dalam studi ini dilakukan secara otomatis melalui sebuah *script* yang memanfaatkan kombinasi Xdebug dan Apache Benchmark. Pemanfaatan Xdebug untuk keperluan *profiling* telah dilakukan sebelumnya [19]–[21]. Penggunaan Apache Benchmark dalam studi komparasi juga telah dilakukan [15]–[17], [20]. Hasil *profiling* berupa rekaman *call history* antara fungsi-fungsi terkait suatu proses yang kemudian dianalisis dengan bantuan Kcachegrind. Beberapa penelitian sebelumnya juga telah melakukan analisis hasil *profiling* menggunakan Kcachegrind [20], [22], [23]. Fungsi-fungsi yang memiliki kontribusi dominan akan dianalisis lebih lanjut menggunakan *Big O notation* untuk mengetahui kompleksitas dan kinerja algoritma yang digunakan. *Big O notation* telah dibahas pada [24]–[27].

Kontribusi studi ini adalah mengisi kesenjangan metodologis pada studi-studi komparasi sebelumnya yang belum membahas mengenai kontribusi fungsi-fungsi dari ORM terhadap kinerja yang terukur. Hal tersebut dimungkinkan dengan memanfaatkan kombinasi Apache Benchmark, Xdebug, Kcachegrind, dan *Big O notation*. Dengan demikian, selain didapatkan komparasi kinerja secara kuantitatif, diperoleh juga persentase dan kompleksitas fungsi-fungsi kontributor yang paling berpengaruh atas kinerja tersebut. Hasil analisis kontribusi fungsi-fungsi terkait tersebut dapat dimanfaatkan lebih jauh sebagai acuan untuk keperluan optimasi ORM dalam upaya peningkatan kinerja.

II. METODOLOGI

Proses studi komparasi dimulai dengan penentuan metrik pengukuran, pendesainan basis data untuk memenuhi kebutuhan metrik, persiapan lingkungan studi, *profiling*, dan ditutup dengan analisis.

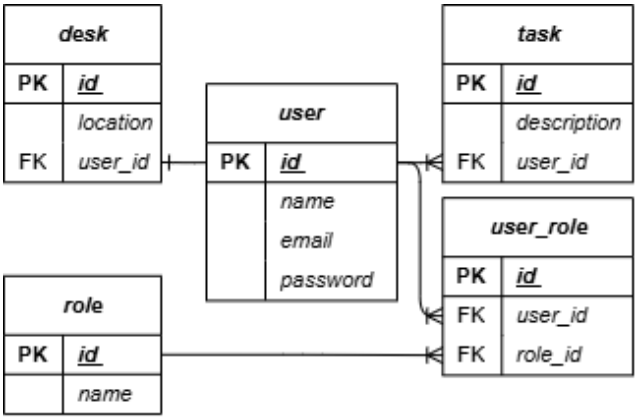
A. METRIK PENGUKURAN

Metrik pengukuran didasarkan pada serangkaian metrik yang dibahas pada studi sebelumnya [12], [28]. Metrik-metrik tersebut terkait kinerja saat menangani operasi *create*, *read*, *update*, *delete*, *and lookup* (CRUDL). Operasi *lookup* terkait dalam pengambilan data tunggal berdasarkan *id*. Metrik pengukuran pertama adalah *relationship*, yang meliputi kinerja CRUDL terkait relasi antartabel, yaitu *one-to-one*, *one-to-many*, dan *many-to-many*. Metrik kedua adalah *polymorphic query* (PQ), yang meliputi kinerja CRUDL terkait tabel-tabel yang terhubung secara *inheritance*. Metrik terakhir adalah *additional null value* (ANV), yang meliputi kinerja CRUDL terkait adanya sebagian atribut tabel yang bernilai *null*. Operasi *change propagation* dan *change isolation* dikecualikan dalam studi ini karena Doctrine dan Eloquent tidak memiliki abstraksi terhadap kedua hal tersebut.

B. DESAIN BASIS DATA

Metrik pengukuran yang telah ditentukan digunakan sebagai panduan dalam proses desain basis data. Desain untuk metrik *relationship* berkonteks manajemen karyawan. Setiap karyawan (*user*) memiliki satu meja (*desk*) pribadi, banyak tugas (*task*) pribadi, dan banyak peran (*role*) yang mungkin dapat sama atau berbeda dengan karyawan lain. Dengan demikian, relasi antara entitas *user* dan *desk* adalah *one-to-one*, antara *user* dan *task* adalah *one-to-many*, dan antara *user* dan *role* adalah *many-to-many*. Tiap entitas dapat menjadi satu tabel, mengingat adanya relasi *many-to-many*, sehingga diperlukan tambahan satu tabel pivot untuk *user* dan *role*, yaitu *user_role*. Dengan demikian, desain basis data untuk metrik *relationship* terdiri atas lima tabel, yaitu *user*, *desk*, *task*, *role*, dan *user_role*. Tabel dan atribut-atributnya pada desain basis data untuk metrik *relationship* ditunjukkan pada Gambar 1. Desain basis data tersebut digunakan untuk menguji kinerja kedua ORM dalam menangani beragam relasi antartabel.

Desain basis data metrik PQ dan ANV dibuat berdasarkan konteks manajemen informasi karyawan kontrak dan tetap. Karyawan tetap memiliki nomor induk karyawan (NIK), sedangkan karyawan kontrak memiliki durasi kontrak (*contract_duration*). Berdasarkan konteks tersebut, terdapat tiga entitas terlibat, yaitu *employee*, *contract*, dan *permanent*. Entitas *employee* yang mewakili informasi inti karyawan bertindak sebagai *parent* dari *contract* dan *permanent* yang masing-masing mewakili karyawan tetap dan kontrak. Entitas *employee* memiliki atribut *name* dan *address*. Entitas *contract* memiliki atribut *contract_duration* dan entitas *permanent* memiliki atribut nik. Metrik ANV dan PQ masing-masing akan



Gambar 1. Desain basis data metrik *relationship*.

menggunakan *mapping strategy* yang berbeda terhadap entitas-entitas tersebut.

Desain basis data metrik PQ menggunakan *mapping strategy tabel per class* (TPC) [6], sehingga diperlukan tiga tabel, yaitu *employee*, *contract*, dan *permanent*, sebagaimana terlihat pada Gambar 2. Atribut *id* pada tabel *contract* dan *permanent* merupakan *primary key* (PK) sekaligus *foreign key* (FK) dari tabel *employee*. Desain basis data menggunakan TPC digunakan untuk menguji kinerja kedua ORM dalam menangani relasi *inheritance* pada tabel-tabel terkait.

Desain basis data metrik ANV menggunakan *mapping strategy single table* (ST) [6], sehingga semua atribut dari ketiga entitas digabungkan dalam tabel *employee*, sebagaimana terlihat pada Gambar 3. Pada satu *record* yang sama, salah satu dari atribut *contract_duration* atau nik dapat dipastikan akan bernilai *null* karena tidak mungkin seorang karyawan berstatus kontrak dan tetap dalam waktu bersamaan. Desain basis data menggunakan ST digunakan untuk menguji kinerja kedua ORM terhadap keberadaan sebagian atribut dengan nilai *null*.

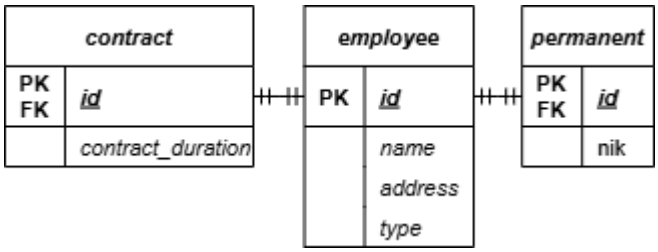
C. LINGKUNGAN STUDI

1) PERANGKAT DAN INFRASTRUKTUR

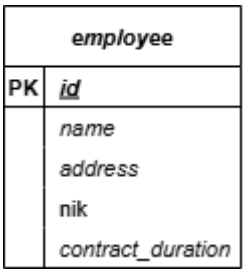
Gambar 4 memperlihatkan perangkat dan infrastruktur yang digunakan dalam studi ini. Perangkat yang digunakan berupa laptop VivoBook ASUS X409FJ_A409FJ dengan spesifikasi CPU Intel i7-8565U (8) @ 4,6Ghz dan RAM 12 GB. Infrastruktur lingkungan studi menggunakan sistem operasi Ubuntu (v20.04), PHP (v8.2), MySQL (v8), Apache Server (v2.4.41), Apache Benchmark (v2.3), Xdebug (v3.3.2), Composer (v2.5.5), Git (v2.25.1), dan Kcachegrind (v0.8). Tiap komponen infrastruktur terpisah satu sama lain, kecuali Apache, MySQL, dan PHP, yang tergabung dalam aplikasi XAMPP. Petunjuk instalasi untuk tiap komponen infrastruktur dapat diakses melalui tautan terkait, sedangkan Ubuntu dapat diakses melalui tautan https://ubuntu.com/tutorials/install-ubuntu-desktop. Sementara itu, Apache (Server dan Benchmark), MySQL, dan PHP dapat diakses melalui tautan https://www.apachefriends.org. Tautan untuk Xdebug dapat diakses melalui https://xdebug.org/docs/install; tautan untuk Composer dapat diakses melalui https://getcomposer.org /download; tautan instalasi untuk Git dapat diakses pada https://git-scm.com/book/en/v2/Getting-Started-Installing-Git; sedangkan tautan untuk Kcachegrind dapat diakses melalui https://kcachegrind.github.io/html/Download.html.

2) KOMPONEN OBJEK PENELITIAN

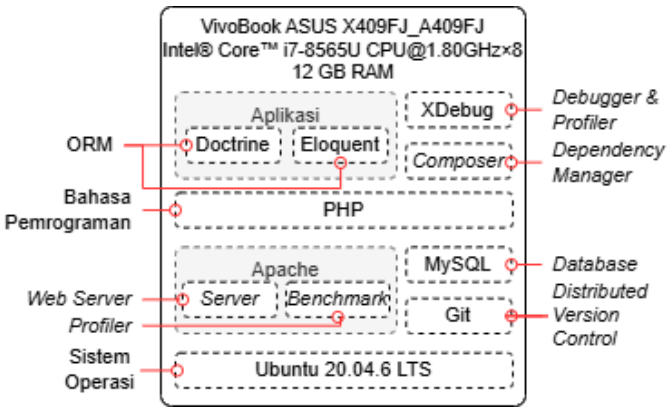
Profiler dan Profiling Object (PO) merupakan komponen objek studi yang berbentuk aplikasi berbasis bahasa



Gambar 2. Desain basis data metrik PQ.



Gambar 3. Desain *database* metrik ANV.



Gambar 4. Perangkat dan infrastruktur.

pemrograman PHP. Profiler merupakan *executable script* yang secara otomatis melakukan *profiling* pada PO. PO berisikan implementasi dari ORM berdasarkan metrik ukur. PO dikembangkan dengan arsitektur *representational state transfer* (REST). REST dipilih karena selaras dengan cara kerja Profiler yang memerlukan akses terhadap *endpoint* terkait metrik pengukuran. Instalasi Profiler dan PO dapat dilakukan dengan melakukan *repository cloning* dan mengeksekusi perintah instalasi Composer. Detail petunjuk instalasi, kode sumber, serta struktur Profiler dan PO dapat ditemukan pada repositori publik di situs Github, https://github.com/ devmurean/orm-profiling.

Secara garis besar, struktur direktori Profiler dan PO memiliki tiga direktori dasar, yaitu App, Profiler, dan sql_dump serta tiga direktori yang dihasilkan secara otomatis pada proses *profiling*, yaitu doctrine_metadata, inputs, dan result. Direktori App merupakan direktori untuk PO, yang terdiri atas dua subdirektori dengan nama Doctrine dan Eloquent. Masing-masing subdirektori memiliki tiga dokumen dengan nama sesuai metrik pengukuran, yaitu dokumen Relationship.php, PolymorphicQuery.php, dan AdditionalNullValue.php. Ketiga dokumen tersebut berisikan implementasi operasi CRUDL untuk tiap metrik pengukuran. Direktori Profiler memuat aplikasi Profiler; direktori sql_dump memuat dokumen SQL yang digunakan sebagai *data seeder* untuk setiap metrik; direktori doctrine_metadata berisikan *cache* dari *metadata* pada Doctrine; direktori inputs berisikan

daftar masukan yang digunakan oleh Profiler saat melakukan *profiling* pada operasi yang membutuhkan masukan, seperti *create* dan *update*; dan direktori result berisikan hasil *profiling* yang dapat dianalisis menggunakan Kcachegrind.

PO memiliki lima *endpoint* dinamis yang terdefinisi dalam dokumen routes.php untuk tiap operasi, dengan format /{orm}/{metrik}/{operasi}/{*id*}. Kata yang diapit oleh kurung kurawal merupakan bagian dinamis dari *endpoint* yang dapat diisi sesuai kebutuhan. Bagian {orm}, {metrik}, dan {operasi} digunakan untuk menentukan ORM dan metrik pengukuran yang akan di-*profile*, sedangkan {*id*} digunakan untuk mengakses *record* terkait. Bagian {*id*} hanya digunakan oleh operasi *update*, *delete*, dan *lookup*. Penulisan *value* untuk {orm} dan {metrik} menggunakan format *kebab-case* dan *lowercase* tanpa disingkat (contoh: *additional-null-value* untuk metrik ANV).

Algoritma implementasi CRUDL untuk tiap metrik cukup sederhana. Operasi *create* membuat *record* baru pada tabel terkait beradasarkan masukan dan melakukan *return* dalam format JavaScript Object Notation (JSON). Operasi *update* diawali dengan pencarian *record* terkait berdasarkan *id*, kemudian mencatat perubahannya pada tabel-tabel terkait berdasarkan masukan, kemudian melakukan *return* berupa *record* yang telah diperbarui dalam format JSON. Operasi *delete* diawali dengan pencarian *record* terkait berdasarkan *id*, kemudian menghapus eksistensinya pada tabel-tabel terkait. Operasi *create*, *delete*, dan *update* dibalut dalam *database transaction*. Operasi *read* mengumpulkan seluruh *record* (tanpa relasi antartabel), kemudian melakukan serialisasi dalam *object collection* dan melakukan *return* dalam format JSON. Operasi *lookup* diawali dengan pencarian *record* terkait berdasarkan *id*, kemudian melakukan serialisasi beserta relasi-relasi antartabel terkait dalam bentuk *object* dan melakukan *return* terhadap *object* tersebut dalam format JSON.

D. PROFILING

Proses *profiling* dilakukan dengan beberapa variasi jumlah *record*, yaitu 100, 1.000, 10.000, dan 100.000. Proses tersebut dilakukan secara otomatis dengan mengeksekusi *script* (dengan izin administrator atau *sudo*) melalui perintah `php profiler`. Perintah tersebut dieksekusi pada *command line interface* (CLI) yang kemudian memicu Profiler untuk beroperasi. Profiler mengeksekusi Apache Benchmark untuk mengakses setiap *endpoint* pada PO, sehingga memicu Xdebug untuk memulai proses *profiling* yang menghasilkan *callgrind-compatible file.* Setiap *file* tersebut akan tersimpan dalam direktori bernama *result* (jika tidak dilakukan perubahan pada konfigurasi PO). Penamaan *file* dilakukan dengan format tertentu. Format penamaan tersebut mengandung nama ORM, metrik, operasi, jumlah *record*, dan *timestamp*. Sebagai contoh, untuk keluaran Doctrine, metrik *PQ*, operasi *create*, saat jumlah *record* 100, namanya adalah cachegrind.out._doctrine_polymorphic-query_create _record=100.1234.gz.

E. ANALISIS

Analisis dilakukan dalam tiga tahap, diawali dengan komparasi hasil *profiling* yang dipaparkan berdasarkan grafik komparasi kinerja dan grafik *relative percentage difference* (RPD) [29]. Grafik RPD memperlihatkan signifikansi perbedaan kinerja antara kedua ORM dalam satuan persen. RPD dengan tren positif menunjukkan bahwa signifikansi perbedaan kinerja kedua ORM makin besar seiring pertambahan jumlah *record*. Demikian dengan sebaliknya, ketika tren negatif, signifikansi perbedaan kinerja makin mengecil.

Tahap berikutnya adalah tabulasi fungsi-fungsi *implementor* DSAP. Fungsi-fungsi tersebut merupakan kontributor dominan terhadap kinerja kedua ORM pada setiap operasi basis data dan metrik pengukuran. Tabulasi tersebut dilakukan berdasarkan analisis menggunakan Kcachegrind. Titik awal analisis tersebut dimulai dari fungsi yang mengimplementasi operasi CRUDL, yaitu fungsi-fungsi dari setiap *class* terkait metrik pengukuran, yaitu *relationship*, *AdditionalNullValue*, dan *PolymorphicQuery*. Kriteria penentuan dominansi tersebut didasari prinsip Pareto [30], sehingga hanya 20% fungsi-fungsi yang berkontribusi terhadap 80% kinerja yang akan dianggap sebagai dominan. Tahapan terakhir dari analisis adalah melakukan penilaian menggunakan *Big O notation* untuk mengetahui kompleksitas dan kinerja algoritma dari fungsi-fungsi yang berkontribusi terbesar pada operasi di setiap metrik pengukuran.

III. HASIL DAN DISKUSI

A. KOMPARASI KINERJA DURASI EKSEKUSI

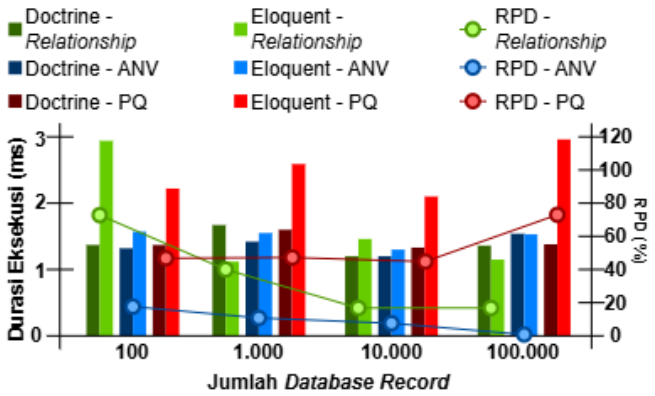
Kinerja operasi *create* kedua ORM secara bergantian mengungguli satu sama lainnya pada metrik *relationship* (Gambar 5) dengan tren RPD negatif seiring pertambahan jumlah *record*. Didapati bahwa durasi kinerja Eloquent pada jumlah *record* 100 lebih tinggi (> 200%) dibandingkan pada jumlah *record* lainnya. Melalui visualisasi Kcachegrind, penyebab kenaikan durasi tersebut adalah kenaikan durasi yang merata pada fungsi-fungsi terkait. Doctrine memiliki keunggulan kinerja pada metrik ANV dan PQ. Sejalan dengan metrik *relationship*, RPD metrik ANV memiliki tren negatif, sedangkan untuk metrik PQ memiliki tren positif.

Kinerja operasi *read* Doctrine unggul pada metrik *relationship*, seperti terlihat pada Gambar 6, dengan tren RPD negatif. Kinerja Eloquent unggul pada metrik ANV dan PQ dengan tren RPD positif untuk kedua metrik tersebut. Durasi eksekusi kedua ORM pada operasi *read* berbanding lurus dengan jumlah pertambahan *record*.

Pada metrik *relationship* operasi *update* dan *delete*, keunggulan kinerja dimiliki oleh Eloquent (Gambar 7 dan Gambar 8). RPD untuk kedua operasi, *update* dan *delete*, memiliki tren positif. Berdasarkan grafik, didapati adanya lonjakan durasi eksekusi untuk Doctrine antara saat *record* berjumlah 10.000 dan 100.000. Lonjakan sebesar 86% terjadi pada operasi *update* dan sebesar 130% pada operasi *delete*. Selaras dengan durasi eksekusi, lonjakan pada RPD juga terjadi dengan kondisi yang sama. RPD untuk operasi *update* mengalami lonjakan sebesar 57,6% dan untuk operasi *delete* sebesar 106,2%. Lonjakan-lonjakan tersebut disebabkan oleh hal yang sama, yaitu peningkatan durasi eksekusi fungsi *execute* dari *class PDOStatement* (internal PHP) sebesar ±1.000% pada jumlah *record* 100.000 dibandingkan dengan jumlah *record* 10.000.

Untuk metrik ANV dan PQ, pada operasi *update* (Gambar 7) dan *delete* (Gambar 8), kinerja Doctrine mengungguli Eloquent. Tren positif terjadi pada RPD untuk kedua metrik pada operasi *update* dan *delete*.

Pada operasi *lookup*, kinerja Doctrine mengungguli Eloquent (Gambar 9) pada metrik *relationship* dan PQ. RPD operasi *lookup* untuk metrik *relationship* memiliki tren negatif , sedangkan untuk metrik PQ memiliki tren positif. Pada metrik *relationship* terjadi lonjakan pada durasi eksekusi antara saat



Gambar 5. Durasi eksekusi dan RPD pada operasi *create*.

Gambar 6. Durasi eksekusi dan RPD pada operasi *read*.

record berjumlah 10.000 dan 100.000 untuk Doctrine (686%) dan Eloquent (571%). Penyebab lonjakan tersebut sama dengan sebab terjadinya lonjakan pada operasi *update* dan *delete* untuk metrik yang sama. Meskipun mengalami lonjakan pada durasi eksekusi dengan sebab yang sama, tidak didapati terjadinya lonjakan pada RPD. Untuk metrik ANV, kinerja Eloquent unggul atas Doctrine dengan tren RPD negatif.

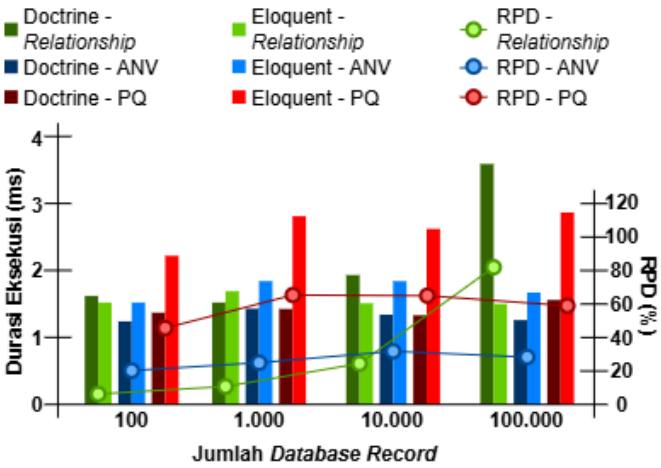
B. KOMPARASI KINERJA KONSUMSI MEMORI

Kinerja konsumsi memori Doctrine dan Eloquent cenderung konstan, dengan pengecualian pada operasi *read*, yang berbanding lurus dengan pertambahan jumlah *record* (Gambar 10). Eloquent secara mutlak mengungguli kinerja Doctrine di setiap operasi dan metrik. Konsumsi memori kedua ORM untuk semua metrik meningkat selaras dengan jumlah *record*. RPD untuk metrik *relationship* memiliki tren positif, sedangkan untuk metrik ANV dan PQ memiliki tren negatif.

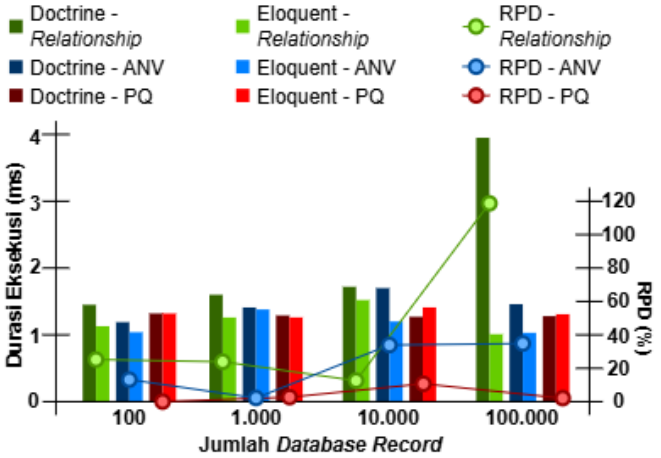
C. KONTRIBUSI FUNGSI-FUNGSI IMPLEMENTOR DSAP

1) DOCTRINE

Berdasarkan analisis Kcachegrind, didapati tujuh fungsi *implementor* DSAP berkontribusi dominan pada Doctrine yang terbagi dalam dua kelompok, yaitu *mapper* dan *domain object*. Fungsi yang termasuk dalam *mapper* adalah `App\Doctrine\EM::make` (D1), `Doctrine\ORM\EntityManager->flush` (D2), `Doctrine\ORM\EntityManager->persist` (D3), `Doctrine\ORM\EntityManager->find` (D4), dan `Doctrine\ORM\EntityRepository->findAll` (D5). Fungsi D1 merupakan *constructor* untuk *mapper*, fungsi D2 terkait proses *database transaction*, fungsi D3 terkait manajemen sinkronisasi antara *object* dan basis data, fungsi D4



Gambar 7. Durasi eksekusi dan RPD pada operasi *update*.



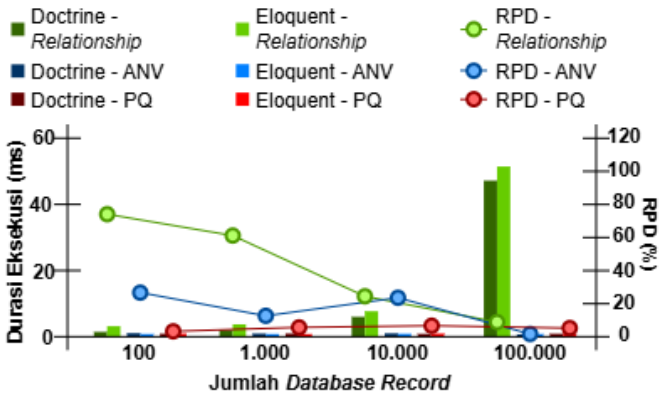
Gambar 8. Durasi eksekusi dan RPD pada operasi *delete*.

terkait *single record retrieval*, dan fungsi D5 terkait *multiple record retrieval*.

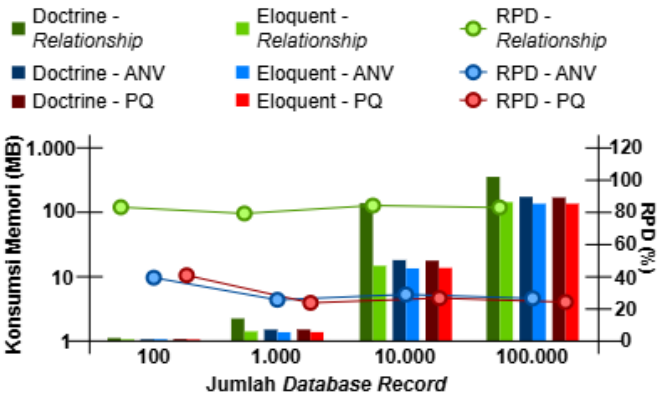
Fungsi-fungsi dalam *domain object* adalah `App\Doctrine\Helper\ModelCollection->serialize` (D6), dan `App\Doctrine\Models\User->serialize` (D7). Fungsi D6 terkait serialisasi untuk *collection* dan fungsi D7 terkait serialisasi spesifik untuk *domain object User*.

Pada metrik *relationship*, operasi *create*, didapati kontribusi pada durasi eksekusi didominasi oleh fungsi D1 dan D2, sedangkan konsumsi memori didominasi oleh fungsi D2 dan D3. Pada operasi *read*, kontribusi didominasi oleh fungsi D5, baik pada durasi eksekusi maupun konsumsi memori. Untuk operasi *update* dan *delete*, kontribusi pada durasi eksekusi didominasi oleh fungsi D2 dan D4, sedangkan untuk konsumsi memori didominasi oleh fungsi D4. Untuk operasi *lookup*, kontribusi durasi eksekusi dan konsumsi memori didominasi fungsi D4 dan D7.

Pada metrik ANV, operasi *create*, kontribusi pada durasi eksekusi didominasi oleh fungsi D1 dan D2, sedangkan kontribusi pada konsumsi memori didominasi oleh fungsi D2 dan D3. Untuk operasi *read*, kontribusi pada durasi eksekusi didominasi oleh fungsi D5, sedangkan kontribusi pada konsumsi memori didominasi oleh fungsi D5 dan D6. Untuk operasi *update*, kontribusi pada durasi eksekusi didominasi oleh fungsi D1, D2, dan D4, sedangkan untuk konsumsi memori didominasi oleh fungsi D4. Untuk operasi *delete*, kontribusi pada durasi eksekusi didominasi oleh fungsi D1, D2, dan D4, sedangkan untuk konsumsi memori didominasi oleh D4. Untuk operasi *lookup*, kontribusi pada durasi eksekusi



Gambar 9. Durasi eksekusi dan RPD pada operasi *lookup*.



Gambar 10. Konsumsi memori dan RPD operasi *read*.

didominasi oleh fungsi D1 dan D4, sedangkan untuk konsumsi memori didominasi oleh fungsi D4.

Pada metrik PQ, operasi *create*, kontribusi pada durasi eksekusi didominasi oleh fungsi D1 dan D2, sedangkan pada konsumsi memori didominasi oleh fungsi D2 dan D3. Untuk operai *read*, kontribusi pada durasi eksekusi didominasi oleh fungsi D5, sedangkan pada konsumsi memori kontribusi didominasi oleh fungsi D5 dan D6. Untuk operasi *update*, kontribusi pada durasi eksekusi didominasi oleh fungsi D1, D2, dan D4, sedangkan pada konsumsi memori didominasi oleh fungsi D4. Untuk operasi *delete* pada metrik PQ, kinerja durasi eksekusi didominasi oleh fungsi D1, D2, dan D4, sedangkan pada konsumsi memori didominasi oleh fungsi D4. Untuk operasi *lookup*, kontribusi pada durasi eksekusi didominasi oleh fungsi D1 dan D4, sedangkan dominasi kinerja konsumsi memori dimiliki oleh fungsi D4.

Berdasarkan penelurusan kontribusi fungsi-fungsi terkait implementasi Data Mapper, ditemukan empat fungsi dari Doctrine yang memiliki kontribusi terbesar pada metrik dan operasi tertentu, yaitu fungsi D2, D4, D5, dan D7. Tabel I memperlihatkan rentang persentase dan Tabel II menyajikan posisi kontribusi fungsi-fungsi tersebut terhadap operasi dan metrik.

Fungsi D2, untuk durasi eksekusi, memiliki persentase kontribusi terbesar pada setiap operasi *create* pada semua metrik. Meskipun memiliki persentase kontribusi terbesar, persentase kontribusi fungsi D2 cenderung menurun seiring pertambahan jumlah *record*. Pada operasi *update* dan *delete* semua metrik, fungsi D2 juga berkontribusi dominan, meskipun bukan yang terbesar, karena operasi *create*, *update*, dan *delete* umumnya melibatkan *database transaction*. Fungsi D2 juga menjadi kontributor terbesar untuk kinerja konsumsi memori pada setiap operasi *create* semua metrik.

TABEL I
DOCTRINE: FUNGSI-FUNGSI KONTRIBUTOR TERBESAR

ORM	Kode	Dominasi Persentase Kontribusi	
		Durasi Eksekusi	*Konsumsi Memori*
Doctrine	D2	55,8% – 62,2%	63,4% – 66,2%
	D4	42,4% – 68,1%	68,9% – 88,5%
	D5	81,8% – 98,8%	69,5% – 86,8%
	D7	31,3% – 93,5%	–

TABEL II
DOCTRINE: POSISI KONTRIBUSI TERHADAP OPERASI DAN METRIK

Metrik	Durasi Eksekusi					Konsumsi Memori				
	C	*R*	*U*	*D*	*L*	*C*	*R*	*U*	*D*	*L*
Relationship	D2	D5	D4	D4	D7	D2	D5	D4	D4	D4
ANV	D2	D5	D4	D4	D4	D2	D5	D4	D4	D4
PQ	D2	D5	D4	D4	D4	D2	D5	D4	D4	D4

Fungsi D4, D5, dan D7 memiliki kontribusi dominan pada operasi yang melakukan *record retrieval* (*read*, *update*, *delete*, dan *lookup*). Fungsi D4 memiliki persentase kontribusi terbesar pada operasi yang melibatkan *single record retrieval* (*update*, *delete*, dan *lookup*) pada setiap metrik kecuali untuk metrik *relationship* pada operasi *lookup*. Pada operasi *update* dan *delete* untuk metrik *relationship*, persentase kontribusi fungsi D4 cenderung meningkat berbanding lurus dengan pertambahan jumlah *record*, sedangkan pada operasi *lookup* untuk metrik yang sama terjadi sebaliknya. Fungsi D4 juga menjadi kontributor terbesar untuk kinerja konsumsi memori pada setiap operasi *update*, *delete*, dan *lookup* semua metrik.

Fungsi D5 memiliki persentase kontribusi terbesar untuk metrik dan operasi yang melibatkan *multiple record retrieval*, yaitu operasi *read* pada semua metrik. Persentase kontribusi fungsi D5 meningkat berbanding lurus dengan pertambahan jumlah *record*. Fungsi D5 juga menjadi kontributor terbesar untuk kinerja konsumsi memori pada setiap operasi *read* semua metrik.

Fungsi D7, terkait *single record retrieval*, memiliki kontribusi terbesar untuk metrik *relationship* pada operasi *lookup* yang cenderung meningkat seiring pertambahan jumlah *record*. Fungsi D7 tidak menjadi kontributor terbesar terhadap konsumsi memori pada operasi *lookup* untuk metrik *relationship*. Posisi tersebut ditempati oleh fungsi D4.

2) ELOQUENT

Terdapat sepuluh fungsi yang berkontribusi dominan pada Eloquent, yang berdasarkan cakupan kerja, dapat dikelompokkan menjadi *database transaction manager*, *object serialization*, *object and database synchronization manager*, *record retrieval manager*, dan *relationship manager*.

Kelompok *database transaction manager* terdiri atas satu fungsi, yaitu `Illuminate\Database\Capsule\Manager::__callStatic` (E1). Kelompok *object serialization* beranggotakan satu fungsi, yaitu `Illuminate\Database\Eloquent\Model->jsonSerialize` (E2). Kelompok *object and database synchronization manager* terdiri atas empat fungsi, yaitu fungsi `Illuminate\Database\Eloquent\Model::__callStatic` (E3), fungsi `Illuminate\Database\Eloquent\Model->saveOrFail` (E4), `Illuminate\Database\Eloquent\Model->push` (E5) dan fungsi `Illuminate\Database\Eloquent\Model->delete` (E6). Secara spesifik, fungsi E3 bertindak sebagai *model constructor*. Fungsi E4 menangani sinkronisasi pada operasi *create* dan *update* serta akan melakukan *throw*

exception jika terjadi *error*. Fungsi E5 menangani sinkronisasi yang juga mencakup *relationship* terkait. Fungsi E6 khusus menangani operasi *delete*.

Kelompok *record retrieval manager* terdiri atas dua fungsi, yaitu `Illuminate\Database\Eloquent\Model::all` (E7) dan `Illuminate\Database\Eloquent\Builder->find` (E8). Fungsi E7 menangani *multiple record retrieval* sedangkan E8 menangani *single record retrieval*.

Kelompok terakhir, *relationship manager*, terdiri atas dua fungsi, yaitu fungsi `Illuminate\Database\Eloquent \Model->loadMissing` (E9) dan fungsi `Illuminate \Database\Eloquent \Model::with` (E10). Kedua fungsi tersebut digunakan untuk memuat *relationship* tertentu untuk *domain object*. Terkait dengan perbedaan keduanya, fungsi E10 dipanggil saat proses inisiasi *domain object*, sedangkan fungsi E9 hanya dapat dipanggil setelah inisiasi.

Pada metrik *relationship*, operasi *create*, kontribusi pada durasi eksekusi didominasi oleh fungsi E1, E2, dan E3. Untuk operasi *read*, kontribusi pada durasi eksekusi didominasi oleh fungsi E2 dan E7. Untuk operasi *update*, kontribusi pada durasi eksekusi didominasi oleh fungsi E1, E2, E3, dan E4. Untuk operasi *delete*, kontribusi pada durasi eksekusi didominasi oleh fungsi E1 dan E3. Untuk operasi *lookup*, kontribusi pada durasi eksekusi didominasi oleh fungsi E2 dan E8. Untuk kinerja konsumsi memori, pada operasi *create*, *update*, dan *delete* didominasi fungsi E1. Operasi *read* didominasi oleh fungsi E7, sedangkan operasi *lookup* didominasi oleh fungsi E8.

Pada metrik ANV, operasi *create*, kontribusi pada durasi eksekusi didominasi oleh fungsi E1, E2, dan E3. Untuk operasi *read*, kontribusi pada durasi eksekusi didominasi oleh fungsi E2 dan E7. Untuk operasi *update*, kontribusi pada durasi eksekusi didominasi oleh fungsi E1, E3, dan E4. Untuk operasi *delete*, kontribusi pada durasi eksekusi didominasi oleh fungsi E1, E3, dan E7. Untuk operasi *lookup*, kontribusi pada durasi eksekusi didominasi oleh fungsi E3 dan E2. Kinerja konsumsi memori pada operasi *create*, *update*, dan *delete* didominasi oleh fungsi E1. Untuk operasi *read* didominasi oleh fungsi E7 dan untuk operasi *lookup* didominasi oleh fungsi E3.

Pada metrik PQ, operasi *create*, kontribusi pada durasi eksekusi didominasi oleh fungsi E1, E2, E3, dan E9. Untuk operasi *read*, kontribusi pada durasi eksekusi didominasi oleh fungsi E2 dan E7. Untuk operasi *update*, kontribusi pada durasi eksekusi didominasi oleh fungsi E1, E2, E5 dan E8. Untuk operasi *delete*, kontribusi pada durasi eksekusi didominasi oleh fungsi E1, E3, dan E6. Untuk operasi *lookup*, kontribusi pada durasi eksekusi didominasi oleh fungsi E8 dan E10. Kinerja konsumsi memori pada operasi *create* didominasi oleh fungsi E1 dan E3. Operasi *read* didominasi oleh fungsi E7, operasi *update* didominasi oleh fungsi E1 dan E8, operasi *delete* didominasi oleh fungsi E1, dan operasi *lookup* didominasi oleh fungsi E8.

Tabel III dan Tabel IV masing-masing menyajikan rangkuman fungsi-fungsi kontributor terbesar dan posisi kontribusi tersebut terhadap metrik dan operasi tertentu. Fungsi E1, E2, E3, dan E8 menjadi kontributor terbesar untuk kinerja durasi eksekusi, sedangkan fungsi E1, E3, E7, dan E8 menjadi kontributor terbesar untuk kinerja konsumsi memori.

Terhadap durasi eksekusi, fungsi E1 memiliki kontribusi terbesar pada operasi *create*, *update*, dan *delete* untuk metrik *relationship*. Untuk metrik *relationship* pada operasi *update*, persentase E1 berbanding lurus dengan pertambahan jumlah *record.* Fungsi E1 juga berkontribusi terbesar pada operasi *update* dan *delete* untuk metrik ANV. Fungsi E1 sebagai kontributor terbesar juga terlihat pada operasi *create* dan *delete* untuk metrik PQ. Untuk konsumsi memori, fungsi E1 berkontribusi terbesar pada operasi *create*, *update*, dan *delete* untuk semua metrik.

TABEL III
ELOQUENT: FUNGSI-FUNGSI KONTRIBUTOR TERBESAR

ORM	Kode	Dominasi Persentase Kontribusi	
		Durasi Eksekusi	***Konsumsi Memori***
Eloquent	E1	28,2% – 51,5%	68,6% – 93,3%
	E2	55,8% – 62,1%	–
	E3	30,3% – 72,5%	90,5%
	E7	–	88,0% – 94,4%
	E8	25,5% – 97,7%	82,5% – 87,2%

TABEL IV
ELOQUENT: POSISI KONTRIBUSI TERHADAP OPERASI DAN METRIK

Metrik	Durasi Eksekusi					Konsumsi Memori				
	C	***R***	***U***	***D***	***L***	***C***	***R***	***U***	***D***	***L***
Relationship	E1	E2	E1	E1	E8	E1	E7	E1	E1	E8
ANV	E3	E2	E1	E1	E3	E1	E7	E1	E1	E3
PQ	E1	E2	E8	E1	E8	E1	E7	E1	E1	E8

Fungsi E3 terhadap durasi eksekusi merupakan kontributor terbesar pada operasi *create* and *lookup* untuk metrik ANV. Untuk konsumsi memori, fungsi E3 menjadi kontributor terbesar pada operasi *lookup* untuk metrik ANV karena ketiadaan relasi antartabel jika dibandingkan dengan metrik lainnya.

Pada operasi *read* semua metrik, kontribusi terbesar untuk durasi eksekusi dimiliki oleh fungsi E2, sedangkan untuk konsumsi memori dimiliki oleh fungsi E7. Maka, dapat disimpulkan untuk Active Record bahwa pada operasi *read*, proses *object serialization* merupakan kontributor terbesar untuk durasi eksekusi, sedangkan proses *record retrieval* merupakan kontributor terbesar untuk konsumsi memori. Persentase kontribusi fungsi E2 terhadap durasi eksekusi meningkat seiring pertambahan jumlah *record*. Persentase kontribusi fungsi E7 untuk konsumsi memori menurun seiring pertambahan jumlah *record*.

Fungsi E8 untuk durasi eksekusi memiliki kontribusi terbesar pada operasi *lookup* untuk metrik *relationship* sekaligus pada operasi *update* dan *lookup* untuk metrik PQ. Sementara itu, untuk konsumsi memori, fungsi E8 berkontribusi terbesar pada operasi *lookup* untuk metrik *relationship* dan PQ. Persentase kontribusi E8 terhadap durasi eksekusi meningkat seiring pertambahan jumlah *record.*

D. BIG O NOTATION

Big O notation untuk tiap fungsi dengan kontribusi terbesar, baik untuk durasi eksekusi maupun konsumsi memori, telah didapatkan. Pada Doctrine, fungsi D2 memiliki kompleksitas O($mn+o$), dengan konteks variabel m menyatakan jumlah *entity* pada proses *insertions*; variabel n menyatakan jumlah atribut terkait; dan variabel o menyatakan jumlah *triggered event*. Fungsi D4 dan D5 memiliki kompleksitas O(mn), dengan konteks variabel m menyatakan jumlah *record* yang diproses (khusus D5 merupakan jumlah *record* pada tabel terkait) dan variabel n menyatakan jumlah atribut tiap *record*. Sementara itu, Fungsi D7 memiliki kompleksitas O(n), dengan konteks variabel n menyatakan jumlah atribut pada *record* yang diproses.

Pada Eloquent, fungsi E1 dan E3 memiliki kompleksitas O(1). Fungsi E2, E7, dan E8 memiliki kompleksitas yang sama,

yaitu O(*n*). Konteks variabel untuk E2 merupakan jumlah *record* hasil *query*, untuk E7 merupakan jumlah *record* pada tabel terkait, dan E8 merupakan jumlah *record* berdasarkan *id list* yang menjadi parameter pencarian.

Berdasarkan komparasi kompleksitas, didapati fungsi-fungsi pada Doctrine lebih kompleks dengan didapatinya dua atau lebih variabel yang terlibat dibandingkan dengan Eloquent yang hanya melibatkan variabel tunggal atau bahkan konstan. Hal tersebut terjadi karena sentralisasi penanganan operasi basis data oleh Data Mapper yang ditangani oleh sebuah *manager*, yaitu EntityManager. Hal tersebut selaras dengan didapatinya fungsi D2, yang berkaitan dengan *database transaction*, memiliki kompleksitas tertinggi dibandingkan fungsi-fungsi lainnya. Sebaliknya, desentralisasi penanganan operasi basis data pada Active Record menjadi sebab didapatinya kompleksitas yang lebih rendah, yang dibuktikan dengan fungi E1, yang menangani proses serupa dengan fungsi D2, hanya memiliki kompleksitas O(1).

E. BATASAN

Studi ini hanya melakukan komparasi, sehingga tidak membahas mengenai peningkatan atau optimasi pada kinerja. Komparasi hanya dilakukan pada ORM berbasis PHP. Batasan tersebut dilakukan karena salah satu komponen Profiler, yaitu Xdebug, hanya dapat melakukan *profiling* pada aplikasi berbasis PHP. Kinerja yang diukur hanya durasi eksekusi dan konsumsi memori. Batasan terhadap kinerja yang diukur tersebut didasarkan pada batasan fitur yang tersedia pada Kcachegrind.

IV. KESIMPULAN

Active Record yang diwakili oleh Eloquent memiliki konsumsi memori yang lebih rendah dibandingkan Data Mapper yang diwakili oleh Doctrine, dengan RPD 40–90% (*relationship*) dan 20–60% (PQ dan ANV). Kinerja durasi eksekusi Data Mapper unggul atas Active Record, dengan rata-rata RPD 35,3%, tetapi dengan beberapa pengecualian pada operasi dan metrik tertentu. Pengecualian tersebut terjadi pada operasi *delete* (semua metrik), *update* (*relationship*), *read* (*relationship*), dan *lookup* (ANV), dengan rata-rata RPD untuk masing-masingnya adalah 23%, 30%, 17,4%, dan 16%. Fungsi-fungsi terkait *database transaction*, *object serialization*, dan *record retrieval* untuk kedua DSAP merupakan kontributor terbesar terhadap kedua kinerja, dengan tambahan *object and database synchronization*, khusus untuk Active Record. Rentang persentase fungsi-fungsi dengan kontribusi terbesar untuk Data Mapper sebesar 31–98%, sedangkan untuk Active Record sebesar 25–97%. Kompleksitas fungsi tertinggi adalah O(*mn*+*o*) untuk Data Mapper dan O(*n*) untuk Active Record.

Studi berikutnya dapat memanfaatkan konsep otomatisasi pada Profiler yang digunakan dalam studi ini. Komponen pada Profiler, terutama Xdebug, dapat disubstitusi dengan komponen serupa sesuai dengan bahasa pemrograman aplikasi yang akan dilakukan *profiling* terhadapnya.

KONFLIK KEPENTINGAN

Penulis menyatakan bahwa tidak terdapat konflik kepentingan.

KONTRIBUSI PENULIS

Konseptualisasi, Muhammad Rezy Anshari, Redi Ratiandi Yacoub, dan Herry Sujaini; metodologi, Muhammad Rezy Anshari; perangkat lunak, Muhammad Rezy Anshari; validasi, Bomo Wibowo Sanjaya dan Eva Faja Ripanti; analisis formal, Muhammad Rezy Anshari; investigasi, Muhammad Rezy Anshari; sumber daya, Muhammad Rezy Anshari; kurasi data, Muhammad Rezy Anshari; penulisan—penyusunan draf asli, Muhammad Rezy Anshari, Redi Ratiandi Yacoub, dan Herry Sujaini; penulisan—peninjauan dan penyuntingan, Muhammad Rezy Anshari, Redi Ratiandi Yacoub, Herry Sujaini, Bomo Wibowo Sanjaya, dan Eva Faja Ripanti; visualisasi, Muhammad Rezy Anshari dan Bomo Wibowo Sanjaya; pengawasan, Redi Ratiandi Yacoub dan Herry Sujaini; administrasi proyek, Muhammad Rezy Anshari; akuisisi pendanaan, Muhammad Rezy Anshari.

REFERENSI

[1] V. Sivakumar, T. Balachander, Logu, dan R. Jannali, “Object relational mapping framework performance impact,” *Turk. J. Comput. Math. Educ.*, vol. 12, no. 7, hal. 2516–2519, Apr. 2021.

[2] A.E. Güvercin dan B. Avenoglu, “Performance analysis of object-relational mapping (ORM) tools in .NET 6 environment,” *Bilişim Teknol. Derg.*, vol. 15, no. 4, hal. 453–465, Okt. 2022, doi: 10.17671/gazibtd.1059516.

[3] G. Vial, “Lessons in persisting object data using object-relational mapping,” *IEEE Softw.*, vol. 36, no. 6, hal. 43–52, Nov./Des. 2019, doi: 10.1109/MS.2018.227105428.

[4] M. Gorodnichev dkk., “Exploring object-relational mapping (ORM) systems and how to effectively program a data access model,” *PalArch’s J. Archaeol. Egypt/Egyptol.*, vol. 17, no. 3, hal. 615–627, Nov. 2020, doi: 10.48080/jae.v17i3.141.

[5] A. Joshi dan S. Kukreti, “Object relational mapping in comparison to traditional data access techniques,” *Int. J. Sci. Eng. Res.*, vol. 5, no. 6, hal. 540–543, Jun. 2014.

[6] M. Fowler, *Patterns of Enterprise Application Architecture*. Boston, MA, AS: Addison-Wesley, 2003.

[7] T. Nguyen, “Elementary event storage,” Skripsi, Metropolia University of Applied Sciences, Helsinki, Finlandia, 2018.

[8] A. Niarman, Iswandi, dan A.K. Candri, “Comparative analysis of PHP frameworks for development of academic information system using load and stress testing,” *Int. J. Softw. Eng. Comput. Sci.*, vol. 3, no. 3, hal. 424–436, Des. 2023, doi: 10.35870/ijsecs.v3i3.1850.

[9] P. Garbarz dan M. Plechawska-Wójcik, “Comparative analysis of PHP frameworks on the example of Laravel and Symfony,” *J. Comput. Sci. Inst.*, vol. 22, hal. 18–25, Mar. 2022, doi: 10.35784/jcsi.2781.

[10] P.R. Chavan dan S. Pawar, “Comparison study between performance of Laravel and other PHP frameworks,” *IJRESM*, vol. 4, no. 10, hal. 27–29, Okt. 2021.

[11] M. Choina dan M. Skublewska-Paszkowska, “Performance analysis of relational databases MySQL, PostgreSQL and Oracle using Doctrine libraries,” *J. Comput. Sci. Inst.*, vol. 24, hal. 250–257, Sep. 2022, doi: 10.35784/jcsi.3000.

[12] S. Holder, J. Buchan, dan S.G. MacDonell, “Towards a metrics suite for object-relational mappings,” dalam *Model-Based Softw. Data Integr.*, R.D. Kutsche dan N. Milanovic, Eds. 2008, hal. 43–54, doi: 10.1007/978-3-540-78999-4_6.

[13] M. Lorenz dkk., “Object-relational mapping reconsidered,” dalam *Proc. 50th Hawaii Int. Conf. Syst. Sci.*, 2017, hal. 4877–4886.

[14] U. Ibrahim, J.B. Hayfron-Acquah, dan F. Twum, “Comparative analysis of CodeIgniter and Laravel in relation to object-relational mapping, load testing and stress testing,” *Int. Res. J. Eng. Technol. (IRJET)*, vol. 5, no. 2, hal. 1471–1475, Feb. 2018.

[15] J.A. Yang dan S.A. Aklani, “Performance analysis between interpreted language-based (Laravel) and compiled language-based (Gin) web frameworks,” *Comput. Based Inf. Syst. J.*, vol. 11, no. 1, hal. 12–16, Mar. 2023, doi: 10.33884/cbis.v11i1.6583.

[16] M. Laaziri, K. Benmoussa, S. Khoulji, dan M.L. Kerkeb, “A comparative study of PHP frameworks performance,” *Procedia Manuf.*, vol. 32, hal. 864–871, Apr. 2019, doi: 10.1016/j.promfg.2019.02.295.

[17] H. Abutaleb, A. Tamimi, dan T. Alrawashdeh, “Empirical study of most popular PHP framework,” dalam *2021 Int. Conf. Inf. Technol. (ICIT)*, 2021, hal. 608–611, doi: 10.1109/ICIT52682.2021.9491679.

[18] D. Zmaranda dkk., “Performance comparison of CRUD methods using NET object relational mappers: A case study,” *Int. J. Adv. Comput. Sci. Appl. (IJACSA)*, vol. 11, no. 1, hal. 55–65, Jan. 2020, doi: 10.14569/IJACSA.2020.0110107.

[19] S. Selvaraj, "Performance monitoring and debugging," dalam *Building Real-Time Marvels with Laravel*. Berkeley, CA, AS: Apress, 2024, hal. 259–283.

[20] A. Šimec, D. Lozić, dan L.T. Golubić, "Benchmarking PHP modules," *Informatologia*, vol. 50, no. 1/2, hal. 95–100, Jun. 2017.

[21] B.A. Azad, P. Laperdrix, dan N. Nikiforakis, "Less is more: Quantifying the security benefits of debloating web applications," dalam *SEC'19, Proc. 28th USENIX Conf. Secur. Symp.*, 2019, hal. 1697–1714.

[22] A. Gocht, R. Schöne, dan J. Frenzel, "Advanced Python performance monitoring with Score-P," dalam *Tools High Perform. Comput. 2018/2019*, H. Mix dkk., Eds. 2021, hal. 261–270, doi: 10.1007/978-3-030-66057-4_14.

[23] J. Buša Jr., S. Hnatič, dan O.V. Rogachevsky, "Performance analysis and optimization of MPDRoot," dalam *Proc. 9th Int. Conf. Distrib. Comput. Grid Technol. Sci. Educ. (GRID'2021)*, 2021, hal. 75–79, doi: 10.54546/MLIT.2021.22.70.001.

[24] S. Bae, *JavaScript Data Structures and Algorithms: An Introduction to Understanding and Implementing Core Data Structure and Algorithm Fundamentals.* Berkeley, CA, AS: Apress, 2019.

[25] I. Chivers dan J. Sleightholme, "An introduction to algorithms and the big O notation," dalam *Introduction to Programming with Fortran*. Cham, Swiss: Springer, 2015, hal. 359–364.

[26] A.J. Lockett, "Performance analysis," dalam *General-Purpose Optimization Through Information Maximization.* Heidelberg, Jerman: Springer, 2020, hal. 239–262.

[27] F. Shamssoolari, "The examination of analyzing data by algorithm performance," *Int. J. Comput. Sci. Mob. Comput.*, vol. 8, no. 9, hal. 167–171, Sep. 2019.

[28] Z. Xu, J. Zhu, L. Yang, dan C. Zuo, "Mining the relationship between object-relational mapping performance anti-patterns and code clones," dalam *35th Int. Conf. Softw. Eng. Knowl. Eng.*, 2023, hal. 1–6, doi: 10.18293/SEKE2023-161.

[29] R.E. Miller, *Optimization: Foundations and Applications.* New York, NY, AS: John Wiley & Sons, 2011.

[30] J. Backhaus, "The Pareto principle," *Anal. Krit.*, vol. 2, no. 2, hal. 146–171, Nov. 1980, doi: 10.1515/auk-1980-0203.