PERBANDINGAN KINERJA BASIS DATA RELASIONALDENGAN BASIS DATA BERORIENTASI-OBJEK STUDI KASUS: APLIKASI JPETSTORE

Muntasir Rahman dan Petrus Mursanto

Fakultas Ilmu Komputer, Universitas Indonesia, Indonesia
santo@cs.ui.ac.id

Abstrak

Basis Data Berorientasi Objek (BDBO) menggunakan model berorientasi objek untuk penyimpanan data. Selama ini penggunaan BDBO tidak populer disebabkan oleh beberapa standar yang berbeda dalam pemodelan dan perancangan skema data, serta kinerja yang dianggap kurang baik. Padahal dengan pengembangan aplikasi berorientasi objek seyogyanya penggunaan BDBO dapat menurunkan kerumitan dan meningkatkan kualitas kode aplikasi. Penelitian ini bertujuan mengkaji standar penerapan model objek data dan metode perancangan skema data pada BDBO melalui pengukuran kinerja dan kualitas kode dari aplikasi. Penelitian ini mengkaji penerapan model data ODMG 3.0 dan notasi UML pada aplikasi JPetStore dengan menggunakan transformasi Muller untuk perancangan skema data. Aplikasi JPetStore versi MySQL (BDR) dibandingkan kinerjanya dengan versi DB4O (BDBO). Hasil kajian adalah beberapa tambahan pada model ODMG 3.0 dan tambahan notasi UML untuk pemodelan data pada BDBO serta penyesuaian proses transformasi Muller. Kinerja aplikasi versi DB4O secara umum lebih cepat dibandingkan versi MySQL, kecuali dalam membaca data sederhana secara berurut. Kualitas kode aplikasi versi DB4O lebih baik dibandingkan versi MySQL.

Kata kunci: *Basis Data Relasional, Basis Data Berorientasi Objek, Object Database, Application Performance, UML*

1. Pendahuluan

Saat ini banyak pengembangan aplikasi menggunakan bahasa program yang berorientasi objek, seperti Java, C# dan C++. Pada bahasa pemrogram tersebut, setiap data dikelola dalam bentuk satu kesatuan struktur yang secara konseptual, hampir serupa dengan data pada situasi sebenarnya. Berdasarkan statistik, aplikasi yang dikembangkan dengan bahasa pemrograman berorientasi objek, umumnya menggunakan Basis Data Relasional (BDR) untuk menyimpan data dalam jangka panjang.

Pada sebuah BDR, data disimpan dalam sel-sel dari tabel. Sel-sel tersebut terbentuk dari kolom dan baris. Setiap sel menyimpan tipe-tipe data tertentu yang didukung oleh BDR yang bersangkutan. Proses penyimpanan dan pengambilan dilakukan oleh aplikasi melalui proses konversi dari data berorientasi objek (selanjutnya disebut objek data) menjadi baris dengan sel-sel dari kolom yang sesuai pada sebuah tabel.

Sejak pertengahan 1980-an berkembang teknologi bernama Basis Data Berorientasi Objek [1]. Basis Data Berorientasi Objek (BDBO) adalah sistem basis data yang menggunakan model data dalam bentuk yang digunakan bahasa pemrograman berorientasi objek [2]. Aplikasi yang menggunakan BDBO menyimpan data dan mengambil data dalam bentuk asli sesuai dengan format yang digunakan oleh aplikasi. Karena sebuah BDBO menyimpan dan mengambil data dalam format yang digunakan aplikasi, BDBO tidak perlu melakukan pemetaan objek data menjadi sel-sel dalam tabel. Kesulitan dalam pemetaan struktur data berorientasi objek menjadi sel-sel tabel pada BDR dikenal dengan istilah *Impedance Mismatch*.

Penggunaan BDBO cukup beragam, mulai dari *embedded system* sampai sistem informasi manajemen. Walaupun demikian, penggunaan BDBO tidak sebanyak penggunaan basis data relasional. Proyek aplikasi yang berorientasi objek seperti Java atau C#, jauh lebih banyak menggunakan Basis Data Relasional dibandingkan menggunakan (BDBO). Tabel 1 menunjukkan perbandingan penggunaan Java dan C# dalam proyek perangkat lunak *Open Source*, dirangkum dari situs SourceForge.net:

Tabel 1. Penggunaan Basis Data

Sistem Basis Data	Java	C#
DB4O (BDBO)	11	3
MySQL (BDR)	2439	541
SQL Server (BDR)	279	754
Postgres (BDR)	821	113
Oracle (BDR)	570	107

Menurut Leavitt dalam [3], sedikitnya penggunaan BDBO disebabkan belum adanya standarisasi dalam teknologi BDBO, dan dominasi vendor BDR dalam pasar basis data. Standar yang dimaksud adalah standar konsep-konsep yang digunakan untuk pemodelan, standar perancangan skema, standar mekanisme akses data, dan standar lain yang mencakup fitur minimal dari sebuah BDBO.

Perbedaan standar ini nampak jelas jika dibandingkan dengan standar yang ada pada basis data relasional. Basis data relasional saat ini menggunakan konsep tabel, kolom, baris. Metodologi perancangan skema pada BDR juga sudah mapan, yang di mana prosesnya terdiri dari 3 tahap, yaitu pemodelan konseptual, pemodelan logikal dan pemodelan fisik. Hampir setiap BDR mendukung konsep tabel dan metodologi perancangan 3 tahap tadi.

Kelemahan BDBO tentang tidak adanya standar, kurangnya kinerja dan ketidak-mampuan menangani skala besar, tidak memiliki landasan yang kuat [4]. Ada standar pemodelan seperti yang ditetapkan oleh ODMG 3.0 dari *Object Management Group* (OMG). Walaupun demikian ada konsep-konsep lain seperti yang diajukan oleh Kim [1]. Produsen produk BDBO sendiri memiliki pendapat yang berbeda mengenai konsep-konsep yang didukung, sehingga fitur-fitur yang didukung oleh sebuah produk BDBO berbeda antara satu produk dengan produk lainnya.

Dalam perancangan skema data, Blaha menyatakan perancangan skema data pada BDBO perlu memperhatikan integritas data, seperti pada Basis Data Relasional (BDR) [5]. Contoh dari integritas data misalnya adalah nilai unik sebuah variabel, integritas referensi (*Referential Integrity*) sebuah objek ke objek lain. Untuk itu diperlukan perencanaan tambahan mengenai skema data.

Pemodelan data dapat menggunakan berbagai alat pemodelan, seperti *Object-Role Modeling* (ORM), *Language for Pattern Uniform Specification* (Lepus3), *Integrated DEFinition for Object-Oriented Design* (IDEF4) dan *Unified Modeling Language* (UML). UML dipilih karena memiliki mekanisme formal untuk ekstensi notasi dalam memodelkan konsep tambahan [6]. ORM tidak digunakan karena ORM lebih digunakan untuk memodelkan konsep [7], sedangkan pemodelan data lebih berorientasi kepada pemodelan fisik (*Physical Modeling*). IDEF4 cukup bagus untuk memodelkan disain menjadi kode sumber tetapi tidak digunakan karena belum ada mekanisme formal untuk ekstensi notasinya. Keunggulan Lepus3 adalah notasi yang ringkas dan bagus untuk memodelkan *design pattern*. Namun Lepus3 tidak dipilih karena belum memiliki mekanisme formal untuk ekstensi notasi.

Konsep-konsep pemodelan yang dibutuhkan untuk memelihara integritas data belum memiliki notasi UML yang formal. Konsep-konsep untuk pemodelan integritas data dan penerapannya pada notasi UML perlu diteliti lebih lanjut.

Metode perancangan skema data BDBO juga bervariasi. Pendekatan Dennis, Wixom dan Tegarden menyatakan bahwa skema data yang akan digunakan dapat langsung menggunakan struktur data dari aplikasi [8]. Muller berpendapat skema data yang akan digunakan pada BDBO perlu melalui proses transformasi. Namun hingga saat ini belum ada metode perancangan skema data yang menjadi acuan umum seperti metode perancangan skema data BDR.

Dennis, Wixom, Tegarden lebih lanjut mengulas bahwa data yang akan disimpan pada BDBO harus berbeda dengan data yang digunakan oleh aplikasi [8]. Perbedaan antara data yang digunakan oleh aplikasi dengan data yang digunakan BDBO perlu dikelola sebuah lapisan tambahan pada aplikasi. Menurut Muller, data yang akan disimpan pada BDBO sama dengan dengan data yang digunakan aplikasi, tetapi data tersebut perlu melalui serangkaian proses transformasi [9]. Dari pendekatan yang dikemukakan Muller dan Dennis, sebuah aplikasi yang menggunakan BDBO, memerlukan perubahan pada data yang digunakan. Untuk itu perlu diteliti lebih lanjut, apakah ada perbedaan antara struktur data yang digunakan aplikasi dengan struktur data yang akan disimpan pada BDBO.

Menurut Devarakonda, kekurangan lain dari sebuah BDBO adalah kinerja yang lambat dan ketidakmampuan dalam skalabilitas [10]. Namun Boyko, Galparin dan Stepanov tidak sependapat dengan ini. Lebih lanjut Boyko et al. mengemukakan beberapa data yang mendukung bahwa kinerja BDBO lebih baik dari BDR. Untuk itu perlu diteliti lebih lanjut perbandingan kinerja aplikasi yang menggunakan BDR dengan aplikasi yang menggunakan BDBO [4].

Aplikasi yang menggunakan BDR memerlukan satu lapisan tambahan untuk memetakan antara data yang digunakan aplikasi dengan data yang disimpan pada basis data. Proses pemetaan ini menyebabkan penambahan kode aplikasi. Aplikasi yang menggunakan BDBO tidak memerlukan pemetaan, karena data yang digunakan aplikasi mirip atau sama dengan data yang disimpan pada basis data. Pengurangan kerumitan (*complexity*) adalah salah satu cara peningkatan kualitas [11]. Penambahan kerumitan pada aplikasi yang menggunakan BDR dapat menurunkan kualitas kode dari aplikasi tersebut. Untuk itu perlu diteliti lebih lanjut apakah ada perbedaan kualitas kode antara aplikasi yang menggunakan BDR dengan aplikasi yang menggunakan BDBO.

1.1. Motivasi

Berdasarkan paparan pada bagian latar belakang, disimpulkan masalah-masalah yang memotivasi penelitian ini.

1. Apakah konsep-konsep yang diperlukan untuk perancangan skema Basis Data Berorientasi Objek (BDBO)? Apakah konsep-konsep tersebut sudah dapat dimodelkan dengan notasi *Unified Markup Language* (UML) yang ada?
2. Bagaimana cara melakukan perancangan skema Basis Data Berorientasi Objek (BDBO)?
3. Bagaimana perbandingan struktur data aplikasi yang menggunakan Basis Data Berorientasi Objek (BDBO) dengan yang menggunakan Basis Data Relasional (BDR)?
4. Bagaimana perbandingan kinerja aplikasi yang menggunakan Basis Data Berorientasi Objek (BDBO) dengan yang menggunakan Basis Data Relasional (BDR)?

5. Bagaimana perbandingan kualitas kode pada aplikasi yang menggunakan BDBO dengan aplikasi yang menggunakan BDR? Kualitas kode aplikasi diukur dalam konteks bahasa pemrograman berorientasi objek, misalnya pewarisan, *coupling*, polimorfisme. Kode yang dimaksud adalah kode aplikasi secara keseluruhan termasuk komponen perangkat lunak pendukung (*library*) yang digunakan.

1.2. Tujuan dan Manfaat

Penelitian ini bertujuan untuk mempelajari standar penerapan pemodelan dan perancangan skema data BDBO. Penelitian ini juga bertujuan mempelajari kinerja, dan kualitas kode dari aplikasi sebuah aplikasi yang sama tetapi memiliki dua buah versi basis data, yaitu versi BDR dan versi BDBO.

Penelitian ini juga mempelajari metodologi perancangan skema data untuk BDBO, dan konsep-konsep penting yang diperlukan untuk perancangan skema data tersebut, termasuk notasi *Unified Markup Language* (UML) yang digunakan.

Manfaat yang akan diperoleh dari penelitian ini adalah pengetahuan mengenai basis data yang lebih baik untuk pengembangan aplikasi, perbaikan metode perancangan skema data pada BDBO, perbaikan alat bantu perancangan struktur data pada BDBO. Hasil penelitian dapat membantu memperluas penggunaan BDBO, jika penggunaan BDBO terbukti memiliki manfaat yang lebih baik dibandingkan BDR. Jika hasil penelitian ini menunjukkan penggunaan BDBO tidak lebih baik dibandingkan dengan BDR, akan ada informasi tambahan mengenai pada kondisi apa, penerapan BDBO perlu dihindari.

1.3. Ruang Lingkup

Masalah yang akan dibahas pada makalah ini dibatasi pada analisis dan disain dari struktur data, metode perancangan struktur data, perbandingan kinerja aplikasi dan kualitas perangkat lunak. Aspek-aspek metodologi pengembangan perangkat lunak secara umum, prasyarat perangkat lunak, di luar lingkup makalah.

Istilah struktur data yang digunakan dalam laporan ini adalah hubungan komposisi antara satu objek dari suatu *class* data dengan objek-objek lain yang menjadi penyusunnya.

Kinerja yang akan dibahas juga dibatasi hanya pada waktu yang dibutuhkan untuk menghasilkan output yang diharapkan (*time to completion*). Metrik kinerja yang lain seperti ketersediaan (*availability*), waktu respon (*response time*), latensi, kapasitas dan lain-lain, di luar ruang lingkup makalah.

Masalah juga akan dibatasi pada notasi UML diagram *class*. Diagram *Use Case* dan diagram *Sequence* UML hanya digunakan untuk memodelkan beberapa aspek dari aplikasi yang menjadi studi kasus. Alat pemodelan lainnya seperti *Entity Relation Diagram* tidak dibahas.

2. Riset Terdahulu

Model basis data berorientasi objek yang umum dijadikan rujukan adalah Model Kim dan ODMG. Keduanya memiliki beberapa persamaan dan perbedaan yang dapat diringkas pada tabel berikut:

Tabel 2. Perbandingan Model Kim dan ODMG

No	Konsep	Kim	ODMG 3.0
1	*Class*, Objek, Atribut (*Attributes*), Metode (*Method*)	√	√
2	Pewarisan (*Inheritance*)	√	√
3	Nilai Unik, *Primary Key*, *Composite Key*	√	√
4	Tipe Data *Interface*	—	√
5	Tipe Data Objek	√	√
6	Tipe Data Literal	—	√
7	Kumpulan Objek dan Literal	—	√
8	*Multiplicity*	—	√
9	Versi (*Versioning*)	√	—
10	Arah Rujukan dari Relasi (*Unidirectional & Bi-directional Relationship*)	√	√
11	Daur Hidup Objek (*Object Lifecycle*)	√	√
12	Komposisi Objek Inklusif-Eksklusif	√	—
13	Relasi yang melibatkan lebih dari 2 objek (*n-ary relation*)	—	√
14	Identitas Objek	√	√
15	Indeks (*Index*)	—	—

Kedua teori memiliki kesamaan pada konsep-konsep sebagai berikut:

1. *Class*, Objek, Atribut, Metode
2. Pewarisan
3. Nilai unik, *Primary Key*, *Composite Key*
4. Arah Rujukan dari Relasi
5. Daur Hidup Objek
6. Identitas Objek (*Object Identifier*)

Perbedaan antara kedua teori pada:

1. *Interface*
2. Versi
3. Konsep *multiplicity*
4. Konsep Komposisi Inklusif-Eksklusif
5. Tipe Data Literal
6. Kumpulan Objek dan Literal
7. Relasi yang melibatkan lebih dari satu objek

3. Metodologi Perancangan Skema Data

Metodologi perancangan skema data yang dibahas adalah metode yang diusulkan Allan Dennis, dan metode yang diusulkan Muller.

3.1. Alan Dennis

Dennis, Wixom dan Tegardern berpendapat perancangan basis data adalah bagian dari perancangan sistem [8]. Objek yang dihasilkan dari perancangan

sistem dipetakan secara langsung ke BDBO. Untuk mengurangi kaitan (*decoupling*) dan meningkatkan kegunaan ulang (*reusability*) ada beberapa prinsip yang perlu dianut:

1. Perlu dibuat 2 lapisan, yaitu *Problem Domain Layer* (PD) dan *Data Access & Manipulation Layer* (DAM).
2. Untuk masing-masing *class* pada PD Layer yang akan disimpan di BDBO, perlu dibuatkan masing-masing satu *class* khusus, yang mewakilinya di BDBO. *Class-class* tersebut disebut *Object-oriented Database Class* (OODBMS *Class*). *Class* tersebut mengetahui memiliki akses kepada *class* di PD Layer, dan bukan sebaliknya.
3. Jika BDBO tidak memiliki fitur yang mendukung konsistensi data, perlu dibuatkan *class* yang memelihara konsistensi data, yang diberi nama *Data Access & Manipulation* (DAM) *Class*.

3.2. Muller

Untuk menyimpan objek pada BDBO, diperlukan penyesuaian tertentu. Menurut Muller penyesuaian tersebut melalui sebuah proses transformasi, yang diuraikan sebagai berikut [9]:

1. Pemberian tanda «persistent» pada *class-class* yang mewakili data yang akan disimpan di BDBO.
2. *Interface* yang direalisasikan oleh *class* bertanda «persistent» juga diberi tanda tersebut.
3. Tipe data *classifier* diubah menjadi tipe enum atau *typedef*.
4. Tipe data atribut dari sebuah *class* yang bertanda «persistent» menjadi tipe data yang akan disimpan pada BDBO dengan memperhatikan proses transformasi dan batasannya.
5. Gunakan nilai *null* untuk tipe data literal, jika diminta dan didukung oleh bahasa pemrograman. Jika nilai *null* untuk tipe data literal tidak didukung, diabaikan.
6. Jika atribut sebuah *class* mengharus adanya proses pengisian nilai awal (*initializer*), perlu disesuaikan konstruktor *class* tersebut untuk menjamin pengisian nilai awal.
7. Untuk *subclass*, *superclass* perlu disertakan dalam deklarasi *class.*
8. Untuk *class* yang mewakili relasi asosiasi, dibuatkan *class* berisi data asosiasi.
9. Untuk pengenal unik objek berikan mekanisme untuk menjamin pemeriksaan keunikan.
10. Tambahkan metode (*method*) yang sesuai untuk menjamin bahwa batasan sebuah atribut dipenuhi dan pastikan sistem bisa memanggil metode tersebut saat batasan tersebut diminta.
11. Berdasarkan *multiplicity* dari sebuah relasi, buat *class* yang bisa merealisasikan hubungan tersebut, baik berupa objek atau kumpulan objek. Buat arah rujukan 2 arah dari hubungan tersebut.
12. Buat hubungan antara *class* yang mewakili relasi asosiasi dengan *class* yang berpartisipasi dalam relasi tersebut.
13. Buat kode atau gunakan fitur BDBO untuk menjamin konsistensi data dari hubungan komposisi atau agregasi.
14. Untuk relasi yang menghubungkan lebih dari 2 *class*, buat sebuah *class*, dan gunakan tipe data yang sesuai untuk atributnya dan tanda *multiplicity* yang sesuai.

4. Metoda Pengukuran Kualitas Software

Pengukuran adalah salah satu upaya untuk menentukan kondisi sesuatu dibandingkan dengan suatu standar. Menurut Kan [11], pengukuran perangkat lunak dapat dibagi menjadi 3 kategori: produk, proses dan proyek. Contoh dari pengukuran produk adalah ukuran, kerumitan, disain, kinerja, kualitas.

Salah satu konsep pengukuran perangkat lunak berorientasi objek adalah *Metrics of Object-Oriented Design* (MOOD) versi 2 yang dikembangkan oleh Abreu. Pada MOOD versi 2, kualitas perangkat lunak diukur berdasarkan aspek-aspek sebagai berikut [12]:

1. *Attribute Inheritance Factor* (AIF)
2. *Operations Inheritance Factor* (OIF)
3. *Internal Inheritance Factor* (IIF)
4. *Attribute Hiding Factor* (AHF)
5. *Operation Hiding Factor* (OHF)
6. *Attributes Hiding Effectiveness Factor* (AHEF)
7. *Operation Hiding Effectiveness Factor* (OHEF)
8. *Behavioral Polymorphism Factor* (BPF)
9. *Parametric Polymorphism Factor* (PPF)
10. *Class Coupling Factor* (CCF)
11. *Internal Coupling Factor* (ICF)

Arti dan penafsiran dari faktor tersebut dapat dilihat lebih detail dalam [13]. Faktor-faktor tersebut kemudian diberi pembobotan melalui proses *Analytic Hierarchy Process (AHP)*. Setelah ditemukan bobot dari masing-masing faktor, kemudian dapat dihitung skor dari masing-masing faktor untuk mendapatkan satu nilai yang mewakili kualitas. Bobot dari masing-masing faktor menurut Rahman adalah sebagai berikut:

Tabel 3. Bobot Faktor JMOOD *Calculator*

Komponen	Bobot
OHF	0,0658
AHF	0,2766
OHEF	0,0658
OIF	0,0781
AIF	0,0223
IIF	0,0635
BPF	0,0490
PPF	0,0490
CCF	0,0266
ICF	0,0266

Setiap aplikasi dihitung terlebih dahulu nilainya untuk masing-masing faktor. Kemudian nilai dari setiap faktor dikalikan dengan bobotnya. Hasilnya adalah sebuah angka yang dapat digunakan untuk perbandingan dengan aplikasi yang lain.

4.1. Analisis Fitur

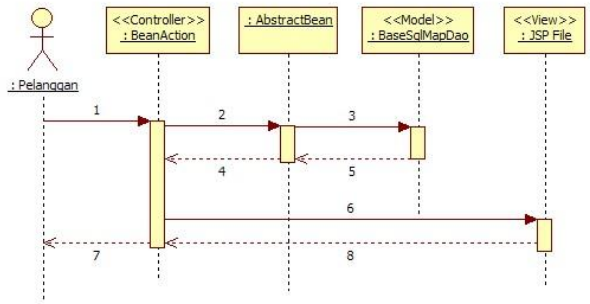
Aplikasi JPetStore menggunakan framework Struts. Struts memiliki pola disain *Model-View-Controller* (MVC). Bagian Model adalah bagian yang bertanggung jawab terhadap pengolahan data aplikasi, termasuk penyimpanan dan pembacaan data dari basis data. *Class* yang termasuk pada bagian ini adalah c*lass* BaseSqlMapDao.

Bagian *Controller* adalah bagian yang melakukan pemeriksaan input yang masuk, dan melakukan pengendalian aliran kerja aplikasi. C*lass* yang menjadi bagian *Controller* adalah *class* BeanAction dan AbstractBean.

Sedangkan bagian View adalah bagian bertanggung jawab menghasilkan presentasi tampilan. *Class* yang termasuk bagian ini adalah *class-class* yang dihasilkan dari file-file Java Server Pages (JSP).

Hubungan antara *class-class* tersebut digambarkan dalam diagram Sequence pada Gambar 1.

Bagian *Controller* memiliki file konfigurasi, yang bernama struts-config.xml. Analisis fitur dari aplikasi JPetStore diawali dengan menganalisis file ini. Bagian yang dianalis adalah bagian yang terkait dengan pemetaan, URL mana yang ditangani *class* AbstractBean. Hasilnya diringkas dalam Tabel 4.

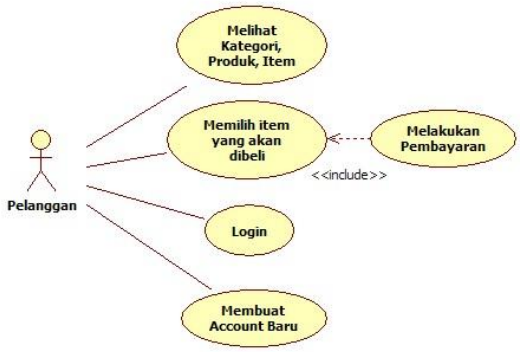


Gambar 1. Diagram Sequence aplikasi JpetStore

Berdasarkan *controller*, kemudian dikelompokkan struktur data yang terkait, dan diidentifikasikan *use case*-nya. Hubungan antara *controller*, struktur data dan *use case* digambarkan secara ringkas dalam Tabel 5.

Dari tabel tersebut, kemudian dilakukan analisis, kaitan antara satu *use case* dengan *use case* lain. Hasilnya kemudian digambarkan secara ringkas dalam Gambar 2.

Pengunjung menjelajahi kategori, produk dan item. Saat melihat data item, pengunjung dapat memasukkan item yang dipilihnya ke dalam keranjang belanja (*shopping cart*). Setelah memilih beberapa item yang akan dibeli ke dalam keranjang belanja, pengunjung harus melakukan pembayaran. Jika pengunjung sudah pernah terdaftar, pengunjung akan diminta melakukan login terlebih dahulu. Sedangkan jika belum terdaftar, pengunjung diminta mendaftar dengan cara mengisi data pribadi.



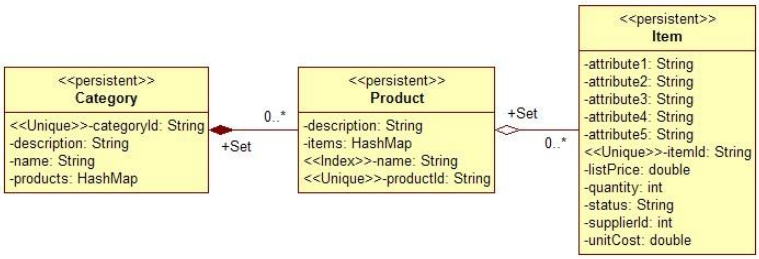
Gambar 2. Diagram *Use Case* aplikasi JpetStore

Berdasarkan analisis dari diagram, dan tabel di atas, pengelompokan struktur data dilakukan berdasarkan *controller*. Struktur data dikelompokkan dalam 3 kelompok sebagai berikut:

1. Catalog
2. Account
3. Catalog

Tabel 4. Hubungan antara URL, Struktur Data dan *Controller*

No	URL	Struktur Data	*Controller*
1	/searchProducts	Product	CatalogBean
2	/switchSearchListPage	Product	
3	/viewCategory	Category	
4	/switchProductListPage	Product	
5	/viewProduct	Product	
6	/switchItemListPage	Item	
7	/viewItem	Item	
8	/addItemToCart	CartItem	CartBean
9	/removeItemFromCart	CartItem, Cart	
10	/updateCartQuantities	CartItem	
11	/checkout	CartItem, Cart	
12	/viewCart	CartItem, Cart	
13	/switchCartPage	CartItem	
14	/newAccountForm	Account	AccountBean
15	/newAccount	Account	
16	/editAccountForm	Account	
17	/editAccount	Account	
18	/signonForm	Account	
19	/signon	Account	
20	/signoff	Account	
21	/newOrderForm	Order	OrderBean
22	/newOrder	Order, LineItem	
23	/listOrders	Order, LineItem	
24	/switchOrderPage	Order, LineItem	
25	/viewOrder	Order, LineItem	



Gambar 3. Struktur data Catalog setelah transformasi dengan notasi tambahan

4.2. Transformasi Skema Data dan Analisa Notasi UML

Class-class yang termasuk dalam kategori Catalog, diubah dengan proses transformasi Muller.

Tabel 5. Hubungan antara *Controller*, struktur data, dan *Use Case*

No	*Controller*	Struktur Data	*Use Case*
1	CatalogBean	Category Product Item	Melihat Kategori, Produk dan Item
2	CartBean	Cart CartItem	Memilih item yang akan dibeli
3	AccountBean	Account Order	Login Membuat account baru
4	OrderBean	Order LineItem	Melakukan pembayaran

Class Category, Product dan Item diberi tanda «persistent». Tanda tersebut menunjukkan bahwa *class* tersebut akan disimpan dalam basis data. *Class* item memiliki rujukan langsung kepada objek dari *class* Product, oleh karena itu atribut productId dari *class* Item dihilangkan.

Class Product mengetahui keberadaan *class* Item melalui *class* kumpulan HashMap yang berisi objek dari *class* Item. Walaupun begitu notasi UML dapat menimbulkan penafsiran *class* Item mengetahui keberadaan *class* Product, sedangkan *class* Product tidak mengetahui keberadaan *class* Item.

Sebelumnya *class* Product tidak memiliki rujukan kepada *class* Category, kecuali melalui atributnya yaitu categoryId. Pola perancangan *class* seperti ini, mungkin akibat pertimbangan penggunaan BDR. Sehingga atribut categoryId dapat digunakan untuk mencari data mengenai Category.a

Class Category mengetahui keberadaan *class* Product melalui *class* kumpulan HashMap. Sebelumnya *class* Category tidak mengetahui keberadaan *class* Product. Walaupun demikian, notasi UML formal dapat menimbulkan penafsiran bahwa *class* Category tidak mengetahui *class* Product.

Dengan demikian didapat bahwa notasi UML diagram *class* yang standar tidak memodelkan beberapa aspek dari struktur data. Kemudian dilakukan proses analisis dan penambahan notasi UML untuk memodelkan aspek-aspek tertentu. Hasilnya dapat dilihat pada Gambar 3.

Atribut categoryId pada *class* Category diberi tanda «Unique» agar objek lain dari *class* yang sama tidak memiliki atribut categoryId yang sama. Demikian juga dengan atribut productId dari *class* Product, dan atribut itemId dari *class* Item. Dengan demikian pada BDBO perlu satu fitur untuk memastikan agar jika ada penambahan objek baru dari *class* Category, Product atau Item, objek tersebut tidak memiliki atribut yang ditandai dengan nilai yang sama.

Atribut name dari *class* Product diberi tanda «Index» agar atribut tersebut dibuat indeks-nya oleh BDBO. Pemberian tanda «Index» penting, untuk mempercepat pencarian data *class* Product. Fitur *search* dari aplikasi JPetStore melakukan pencarian berdasarkan atribut 'name' dari *class* Product.

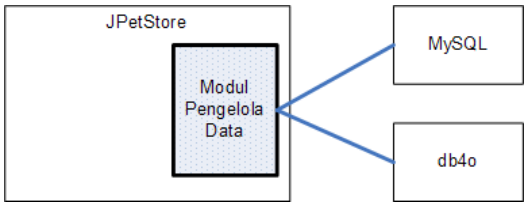
Hubungan antara *class* Category dan *class* Product diubah menjadi komposisi dengan rujukan relasi 2 arah. Dengan demikian dari notasi dapat ditafsirkan bahwa *class* Category mengetahui keberadaan *class* Product. Dan banyak objek dari *class* Product menjadi bagian komposisi dari *class* Category. Daur hidup objek dari Product lebih singkat atau sama dengan daur hidup objek dari *class* Category yang menaunginya.

Demikian juga hubungan antara *class* Product dengan *class* Item diubah menjadi hubungan agregasi dua arah. *Class* Product mengetahui keberadaan *class* Item. Dan beberapa objek dari *class* Item menjadi anggota agregasi *class* Product. Namun daur hidup objek dari *class* Item bebas dari daur hidup objek *class* Product yang menaunginya.

5. Perbandingan Kinerja BDR dan BDBO

Dari arsitektur aplikasi, bagian *View* dan *Controller* tidak mengalami perubahan yang signifikan. Sedangkan bagian Model, mengalami perubahan cukup signifikan. Bagian Model terdiri dari *class-class* yang disebut sebagai Modul Pengelola Data (MPD).

Penelitian ini akan mengubah Modul Pengelola Data, sehingga ada dua versi Modul Pengelola Data (MPD), yaitu versi MySQL (BDR) dan versi DB4O (BDBO). Hubungan antara modul pengelola data dengan basis data digambarkan dalam Gambar 4.



Gambar 4. Modul Pengelola Data dan Basis Data

Pengujian dilakukan dengan cara mengirimkan data dalam jumlah dari 1.000 sampai 10.000, ke modul pengelola data. Selanjutnya respon modul pengelola data diukur. Durasi secara keseluruhan kemudian dibagi dengan jumlah objek, sehingga didapatkan durasi rata-rata.

```
tester.init();
tester.startWriteTest(1000);
for (int i = 1000; i < 11000; i
+= 1000) {
    tester.startWriteTest(i);
}
tester.startReadTest(1000);
for (int i = 1000; i < 11000; i
+= 1000) {
    tester.startReadTest(i);
}
tester.startSeekTest(1000);
for (int i = 1000; i < 11000; i
+= 1000) {
    tester.startSeekTest(i);
}
tester.cleanUp();
```

Objek data dari *class* Category dipilih mewakili objek data sederhana. Sedangkan objek data dari *class* Order, yang berisi objek dari *class* LineItem, dan merujuk ke *class* Item, mewakili objek data kompleks.

Secara umum kinerja BDBO lebih baik dari pada kinerja BDR. Perbandingan kinerja basis data dalam 3 aktivitas diringkas dalam Tabel 6.

Tabel 6. Perbandingan Kinerja Durasi Rata-rata

Query		DB4O (ms)	MySQL (ms)
Menyimpan Objek	Sederhana	0,30	10,32
	Kompleks	0,42	77,21
Membaca Objek secara berurutan	Sederhana	0,12	0,02
	Kompleks	0,15	4,54
Membaca Objek secara acak	Sederhana	0,28	0,61
	Kompleks	0,24	5,32

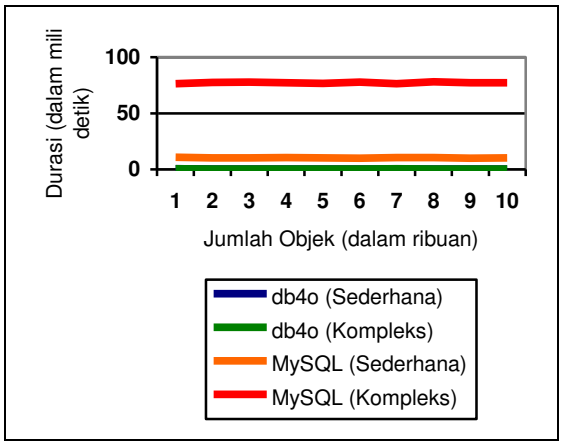
Pada aktivitas penyimpanan objek, DB4O dapat menyimpan objek dari *class* data sederhana 34 kali lebih cepat bila dibandingkan MySQL. Kecepatan DB4O dalam menyimpan data objek dari *class* dengan struktur kompleks, 183 kali lebih cepat bila dibandingkan MySQL. Demikian juga dalam aktivitas pencarian data objek sederhana secara acak, DB4O lebih cepat 2 kali dibandingkan MySQL. Dalam pencarian data objek kompleks secara acak, DB4O lebih cepat 22 kali dibandingkan MySQL. Satu-satunya pengecualian adalah dalam membaca data objek sederhana secara berurutan, MySQL lebih cepat 6 kali dibandingkan DB4O.

5.1. Penyimpanan Objek

DB4O menyimpan objek data lebih cepat dibandingkan MySQL. Fenomena tersebut digambarkan dalam Gambar 5.

Kedua basis data tidak menunjukkan peningkatan atau penurunan. Dalam menyimpan objek dari *class* kompleks, DB4O membutuhkan waktu 40% lebih lama jika dibandingkan dengan waktu yang dibutuhkan untuk menyimpan objek dari *class* sederhana.

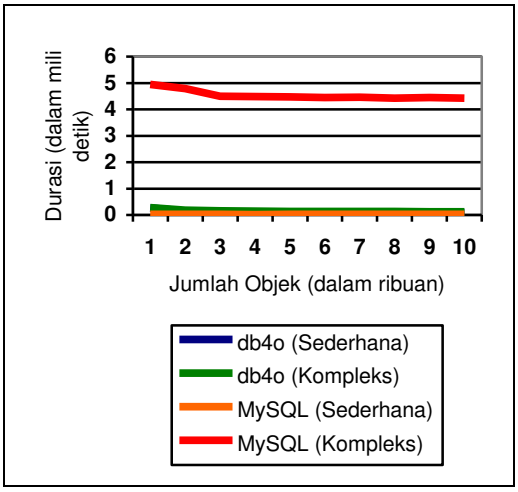
MySQL juga membutuhkan waktu 8 kali (800%) lebih lama untuk menyimpan objek dari *class* kompleks dibandingkan waktu untuk menyimpan objek dari *class* sederhana.



Gambar 5. Durasi Rata-rata untuk Menyimpan Objek

5.2. Pencarian Objek secara Berurut

Secara umum kinerja DB4O untuk membaca data objek dari *class* sederhana lebih lambat dibandingkan MySQL. Sedangkan untuk membaca data objek dari *class* kompleks, DB4O jauh lebih cepat dibandingkan MySQL.

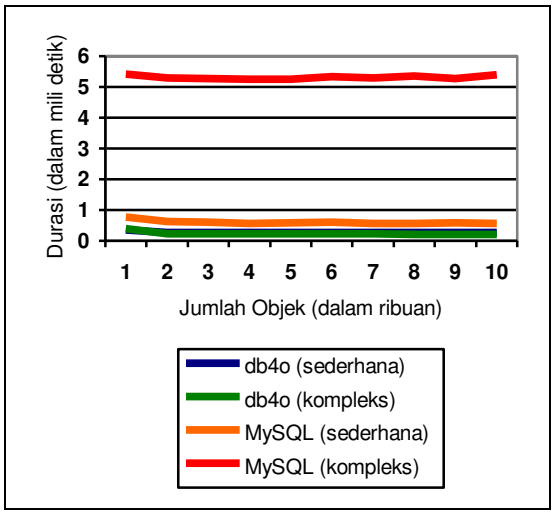


Gambar 6. Durasi Rata-rata untuk Membaca Objek Secara Berurut

Dapat disimpulkan, kinerja basis data untuk membaca data objek tergantung dari struktur data yang akan dibaca. Perbandingan kinerja masing-masing basis data, untuk membaca data objek secara berurut digambarkan dalam Gambar 6.

5.3. Pencarian Objek secara Acak

Secara umum tidak ada perubahan yang berarti antara pencarian acak 1.000 objek dengan 10.000 objek. Kedua basis data tidak menunjukkan peningkatan atau penurunan. DB4O lebih cepat dibandingkan MySQL dalam melakukan pencarian data secara acak. Fenomena tersebut dapat dilihat dalam Gambar 7.



Gambar 7. Durasi Rata-rata untuk Membaca Objek Secara Acak

6. Perbandingan Kualitas Kode

Pengukuran kualitas dengan menggunakan JMOOD Calculator menghasilkan nilai dari masing-masing faktor. Nilai kualitas hasil perhitungan jMOOD Calculator adalah JPetStore versi BDR adalah 0,3329 sedangkan JPetStore versi BDBO adalah 0,3516. Hasil ini menunjukkan bahwa secara keseluruhan kualitas JPetStore versi BDBO lebih baik dibandingkan JPetStore versi BDR.

7. Kesimpulan

Masalah yang didefinisikan pada bagian pendahuluan dijawab pada bagian ini. Kesimpulan-kesimpulan sebagai jawaban atas masalah adalah sebagai sebagai berikut:

1. Konsep-konsep yang dibutuhkan untuk pemodelan Basis Data Berorientasi Objek menggunakan konsep-konsep yang didefinisikan pada ODMG 3.0. Ada beberapa konsep yang belum ada tetaoi diperlukan, yaitu: indeks, versi dan atribut relasi.
2. Notasi diagram *class* UML yang ada sebagian besar dapat memodelkan konsep ODMG 3.0. Namun ada 2 kelompok notasi yang kurang, yaitu notasi penanda atribut dan notasi penanda khusus relasi. Notasi penanda atribut sangat penting untuk memelihara integritas data BDBO tetapi belum menjadi bagian dari notasi formal UML. Notasi penanda khusus relasi antar-*class*, dapat memodelkan kaitan antara satu *class* dengan *class* lainnya dengan lebih baik, tapi belum menjadi bagian formal notasi UML.
3. Perancangan skema data dapat menerapkan metode transformasi Muller, dengan beberapa penyesuaian yaitu penambahan pada tahapan tertentu. Penyesuaian tersebut diperlukan agar ada kesesuaian antara konsep-konsep pemodelan struktur data BDBO, bahasa pemrograman yang digunakan, dan fitur-fitur tertentu dari BDBO yang digunakan.
4. Kinerja aplikasi versi DB4O (BDBO) secara umum lebih cepat dibandingkan aplikasi versi MySQL (BDR).
5. Struktur data dari aplikasi JPetStore versi BDBO lebih mirip dengan konsep yang akan dimodelkan, dibandingkan dengan struktur data versi BDR. Struktur data aplikasi versi BDBO menunjukkan hubungan antara *class-class* data dengan lebih jelas.
6. Kualitas kode JPetStore versi BDBO lebih baik daripada versi BDR, diukur dengan alat JMOOD Calculator. JMOOD Calculator menggunakan konsep kualitas metrik MOOD versi 2. Kualitas kode aplikasi versi BDBO adalah 0,3516, sedangkan aplikasi versi BDR adalah 0,3329. Peningkatan kualitas kode disebabkan oleh perbaikan diberbagai aspek. Namun perbaikan kualitas kode ini salah satunya disebabkan menurunnya kerumitan (*Complexity*) kode. Penurunan kerumitan kode ini disebabkan karena hilangnya mekanisme pemetaan dari data objek menjadi data dalam bentuk tabel.

REFERENSI

[1] Kim, Won. *Introduction to Object-Oriented Databases. Cambridge*: MIT Press, 1992.

[2] Wikipedia. Object Database. Web. 16 January 2009. <http://en.wikipedia.org/wiki/Object_database>.

[3] Leavitt, Neal. "Whatever Happened to Object-oriented Database". Leavitt Communication, August 2000. Web. 25 January 2009. <http://www.leavcom.com/db_08_00.htm>.

[4] Boyko, Y., Simon Galperin, Alexander Stepanov. "Should we use Object Database Management Systems". Dept. Computer Science, Drexel University, 2007. Web. 25 January 2009. <https://www.cs.drexel.edu/~bmitchel/course/cs575/WorkshopF0708/2T5Paper.pdf>.

[5] Blaha, Michael. "Referential Integrity is Important for Databases". Modelsoft Consulting Group, 17

November 2005. Web. 12 January 2009. <http://www.odbms.org/ experts.aspx#article5>.

[6] Object Management Group. "Unified Modeling Language Specification 1.5". March, 2003. Web. 19 January 2009. <http://www.omg.org/cgibin/apps/doc?formal/03-03-01.pdf>

[7] Halpin, Terry and Anthony Bloesch. "Data Modeling in UML and ORM: a comparison". Journal of Database Management 10.4. (Dec 1999): 4-13.

[8] Dennis, A., Barbara H. Wixom, and David Tegarden. *System Analysis and Design with UML Version 2.0: an Object-Oriented Approach. 2nd ed*. Hoboken: John Wiley & Sons, Inc. 2005.

[9] Muller, Robert J. *Database Design for Smarties: Using UML for Data Modeling*. San Fransisco: Morgan Kaufman, 1999.

[10] Devarakonda, Ramakanth S. "Object Relational Database System – The Road Ahead. Association for Computing Machinery". Web. 17 January 2009.

[11] Kan, Stephen H. *Metrics and Models in Software Quality Engineering. 2nd ed*. Harlow: Addison Wesley, 2002.

[12] Abreu, Fernando B. "Using OCL to Formalize Object-Oriented Metrics Definition". Technical Report ES007/2001 version 1.0. FCT/UNL and INESC, June 2001. Web. 25 January 2009. <http://www.iro.umontreal.ca/~sahraouh/qaoose01/OCL_Metrics.pdf>.

[13] Rahman, Zaki. *Penerapan Metrics for Object-Oriented Design dalam Pengukuran Kualitas Desain Perangkat Lunak Berorientasi Objek*. Jakarta: Universitas Indonesia, 2009.