PERANCANGAN DAN PEMBUATAN PERANGKAT LUNAK SISTEM INVENTORI TERDISTRIBUSI DENGAN MENERAPKAN OBJECT RELATIONAL DATABASE MANAGEMENT SYSTEM PADA ORACLE9i

TUGAS AKHIR

Disusun Oleh :

ANITA SETYANINGTYAS
NRP. 5198 100 039

JURUSAN TEKNIK INFORMATIKA
FAKULTAS TEKNOLOGI INFORMASI
INSTITUT TEKNOLOGI SEPULUH NOPEMBER
SURABAYA
2003

PERANCANGAN DAN PEMBUATAN PERANGKAT LUNAK SISTEM INVENTORI TERDISTRIBUSI DENGAN MENERAPKAN OBJECT RELATIONAL DATABASE MANAGEMENT SYSTEM PADA ORACLE9i

TUGAS AKHIR

Diajukan Guna Memenuhi Sebagian Persyaratan
Untuk Memperoleh Gelar Sarjana Komputer
Pada
Jurusan Teknik Informatika
Fakultas Teknologi Informasi
Institut Teknologi Sepuluh Nopember
Surabaya

Mengetahui / Menyetujui,

Dosen Pembimbing I	Dosen Pembimbing II
(Yudhi Purwananto, S.Kom, M.Kom)	(Ir. Suhadi Lili)
NIP. 132172210	NIP. 132048148

SURABAYA
JUNI, 2003

ABSTRAK

Dalam penanganan data dibutuhkan sebuah Database Management System (DBMS) untuk mengatur dan mengontrol akses ke basis data tersebut. Dari DBMS kemudian dikembangkan Relational Database Management System (RDBMS) agar antara entitas yang satu dengan entitas yang lain dalam basis data tersebut dapat berelasi untuk mengurangi redundancy data dan menjaga integrity constraint. Kesuksesan RDBMS masih dinilai belum mampu untuk menangani aplikasi yang kompleks dan belum mendekati real-world entities akibat dari proses normalisasi. Sedangkan pada pemodelan obyek, sistem dideskripsikan berdasarkan obyek-obyek, yaitu abstraksi pemrograman yang memiliki identity, behavior dan state. Dalam pemrograman berdasar obyek, digunakan tipe data atau user-defined types untuk memodelkan struktur dan behavior dari data pada aplikasi.

Object Relational Database System (ORDBMS) akan menggabungkan antara fitur yang ada pada model obyek dan fitur yang ada pada model relasional, yaitu menerapkan konsep pada pemrograman berdasar obyek sebagai perluasan dari model relasional, sehingga pada ORDBMS ini mempunyai integrity constraint yang tetap terjaga dan mendukung untuk data yang kompleks. Selain itu, penerapan model obyek yang reusable dapat mempermudah pembuatan aplikasi basis data.

Dalam tugas akhir ini dibuat suatu aplikasi untuk sistem inventori terdistribusi dengan menerapkan konsep ORDBMS untuk basis datanya dan konsep FIFO untuk pencatatan harga barang. Desain sistem inventori ini berdasarkan enterprise architecture, yaitu arsitektur yang terbagi dalam 3 layer (layer aplikasi, layer bisnis dan layer data akses). Sistem inventori ini juga didesain menjadi aplikasi terdistribusi, yaitu masing-masing layer dapat diletakkan pada komputer yang beda yang memanfaatkan teknologi dari .NET yaitu .NET Remoting.

Aplikasi yang dibangun mengimplementasikan teknologi – teknologi tersebut. Sistem ini mengimplementasikan mekanisme FIFO dengan meletakkan operasi / metodenya pada layer bisnis. Ada beberapa kendala yang menyebabkan metode FIFO tidak dapat diletakkan pada "TYPE" di ORDBMS Oracle 9.2.0.1. Adanya keterbatasan Microsoft ADO.NET dalam mengakomodir struktur Oracle ORDBMS juga menyebabkan kekuatan masing-masing kurang bisa termanfaatkan secara penuh.

Kata Kunci : ORDBMS, FIFO, Sistem inventori terdistribusi, Enterprise Architecture, Microsoft .NET

KATA PENGANTAR

Segala puji dan syukur semata ditujukan ke hadirat ALLAH SWT yang telah memberikan rahmat dan hidayah-Nya sehingga memungkinkan penulis untuk menyelesaikan Tugas Akhir yang berjudul "**Perancangan dan Pembuatan Perangkat Lunak Sistem Inventori Terdistribusi dengan Menerapkan Object Relational Database Management System pada Oracle9i**".

Mata Kuliah Tugas Akhir yang memiliki beban sebesar 4 satuan kredit disusun dan diajukan sebagai salah satu syarat untuk menyelesaikan program strata satu (S-1) pada jurusan Teknik Informatika di Institut Teknologi Sepuluh Nopember Surabaya.

Dalam penyusunan Tugas Akhir ini, Penulis berusaha untuk menerapkan ilmu yang telah didapat selama menjalani perkuliahan dengan tidak terlepas dari petunjuk, bimbingan, bantuan, dan dukungan berbagai pihak.

Dengan tidak lupa akan kodratnya sebagai manusia, Penulis menyadari bahwa dalam karya Tugas Akhir ini masih mengandung kekurangan di sana-sini sehingga dengan segala kerendahan hati Penulis masih dan insya Allah akan tetap terus masih mengharapkan saran serta kritik yang membangun dari rekan-rekan pembaca.

Surabaya, Juni 2003

Penulis

UCAPAN TERIMA KASIH

Dengan tak bosannya mengucapkan syukur Alhamdulillah kepada **Allah SWT**, yang telah memberi terlalu banyak dari yang layak penulis terima. Di kesempatan ini, Penulis hendak menyampaikan rasa penghormatan yang setingginya serta rasa terima kasih kepada pihak-pihak yang telah memberi bantuan baik itu berupa moril maupun material dan langsung maupun tidak langsung kepada:

1. Mami dan Bapak atas segala cinta, kasih sayang, dukungan semangat, bimbingan dan kepercayaan. It's dedicated to you
2. Bapak Yudhi Purwananto, S.Kom, M.Kom dan Bapak Ir. Suhadi Lili selaku dosen pembimbing TA atas segala bimbingan dan arahannya.
3. Bapak Khakim Ghozali selaku dosen wali penulis.
4. Mas Iwan dan Mbak Naning atas dukungan dan nasehat-nasehatnya.
5. Mbak Ika dan Mas Salmon atas nasehat-nasehatnya, tumpangan tidur di Jkt, dukungan dan crita-critanya juga pinjeman motornya. ☺
6. Mbak Andri dan Mas Anto' atas dukungan dan dorongan smangatnya.
7. Ponakan-ponakanku : Aston, Sheila, Mira, Annisa atas keceriaannya
8. Mas Budi atas cinta, sayang, dukungan smangat dan kepercayaannya. Makasih buuuaaaangett. ☺
9. Pak SHL selaku pimpro SIS atas kesempatan untuk penulis mengerjakan di IAO dan fasilitas-fasilitasnya, juga pinjeman Velocis dan pen drive-nya

10. Nana atas persahabatanya, saran, masukan-masukannya dan udah nemenin melekan di malam sidang. Thx 4 being best friend, love U non. ☺
11. Yayas atas persahabatanya, saran, masukan-masukannya dan udah nemenin melekan di malam sidang, juga printernya. Thx 4 being best friend, love U non. ☺ dan gak lupa thx buat FJA. ☺
12. Keluarga Jetis yang udah jadi rumah kedua penulis. ☺
13. Holly'96 atas ide dan solusi-solusinya yang bikin tenang penulis. ☺
14. Agung'96 atas saran, kritik, masukan-masukannya dan persahabatannya. ☺
15. Reza, Kwong Gae Siang atas bantuannya selama pengerjaan TA.
16. Aby atas bantuan, saran, kritiknya dan persahabatanya.
17. Arif atas saran-sarannya dan persahabatanya.
18. Nason, mas Fery, mas Agus thx 4 being friend of mine.
19. Wonder Women yang lain : Ifah, Citra, Premi, Dewi dan Rully atas kebersamaannya selama kuliah.
20. Temen-temen di Tim SIS : HRA, ANT, RZA, NOF, KAS, RES, DIP, CHF
21. Temen-temen di IAO : DTP, KIR, YOU, IYR, ARK, DEK, RRN, NIN, SLM, YOS, BDT, YUK, PRM, HEN, KUN, NDS, AFZ, UZI, MAN, MAJ, BEH, KHD
22. Temen-temen di C0E : Isye, Faida, Luluk, Elma, Intan, Nisa, Ririn, Ita, Dwi, Hanifah, Caca, Ario, Rizki, Kendy, Iril 'thx 4 power supply. ☺', Adi, Rosi, Joko, Wongky, Fahrul, Bhakti, Leo, Deku dan yang laennya..
23. Temen-temen Arjo : Chpee, Deni, Dimas, Ade, Budi, Didit, Yudha atas bantuan-bantuannya dan selalu rela penulis repotin.. ☺

24. Temen-temen P-5 : Ni luh 'thx buat printernya dan persahabatannya ☺', Maria, Novi, Vidi, ex P-5 : bu Efi, mbak Evi, Yanti, mbak Dini, mbak Indi dan mbak Irene.
25. Keluarga Bangun Mulyo yang udah merelakan kamarnya disewa penulis.
26. Mas Yudi TU atas kemudahan administrasi yang udah diberikan kepada penulis.
27. Temen-temen di Malang : Adit 'thx buat telp-nya ☺', Yanti, Ulya, Endro, Indradi, Dimas, Novan, Anita, Verra
28. Terakhir, buat rekan-rekan saya. Banyak diantaranya yang sudah tersebut tapi lebih banyak lagi yang belum.

Tiada untaian kata yang cukup yang dapat penulis sampaikan sebagai balas atas jasa yang penulis terima melainkan hanya harapan semoga ALLAH SWT membalas semua amal tersebut. Jazakumullah Khairan Katsiran.

DAFTAR ISI

DAFTAR GAMBAR

DAFTAR TABEL

BAB 1
PENDAHULUAN

Pada bab ini dijelaskan beberapa hal yang mendasar dari pengerjaan tugas akhir yang meliputi latar belakang, permasalahan, tujuan, batasan permasalahan, metodologi serta sistematika penulisan. Dari uraian tersebut diharapkan gambaran umum permasalahan dan pemecahan tugas akhir ini dapat dipahami.

1.1 Latar Belakang

Kehadiran basis data dalam penanganan data dibutuhkan hampir disetiap aspek kehidupan, sehingga dibutuhkan sebuah *Database Management System (*DBMS*)* untuk mengatur dan mengontrol akses ke basis data tersebut. Dari DBMS kemudian dikembangkan *Relational Database Management System (*RDBMS*)* agar antara entitas yang satu dengan entitas yang lain dalam basis data tersebut dapat berelasi untuk mengurangi *redundancy* data dan menjaga *integrity constraint*.

Namun, kesuksesan RDBMS dalam 2 dekade ini masih dinilai belum mampu untuk menangani aplikasi yang kompleks. RDBMS juga dinilai belum mendekati *real-world entities* akibat dari proses normalisasi.

Pada model obyek dideskripsikan sistem berdasar obyek-obyek, yaitu abstraksi pemrograman yang memiliki *identity, behavior dan state*. Dalam pemrograman berdasar obyek, digunakan tipe data atau *user-defined types* untuk memodelkan struktur dan *behavior* dari data pada aplikasi.

Object Relational Database System (ORDBMS) akan menggabungkan antara fitur yang ada pada model obyek dan fitur yang ada pada model relasional, yaitu menerapkan konsep pada pemrograman berdasar obyek sebagai perluasan dari model relasional, sehingga pada ORDBMS ini mempunyai *integrity constraint* yang tetap terjaga dan mendukung untuk data yang kompleks. Selain itu, penerapan model obyek yang *reusable* dapat mempermudah pembuatan aplikasi basis data.

Sistem inventori yang dibangun ini merupakan sistem inventori yang menerapkan mekanisme *First In First Out* (FIFO) dalam pencatatan harga dan diterapkan dengan menggunakan sistem yang terdistribusi.

Pemilihan Oracle9i dalam tugas akhir ini karena Oracle9i memiliki teknologi berdasar obyek sehingga mendukung untuk penerapan ORDBMS. Dan dipilihnya framework Microsoft .NET dalam membangun sistem ini karena framework ini mendukung untuk penerapan sistem yang terdistribusi.

1.2 Tujuan

Adapun tujuan dari pembuatan tugas akhir ini adalah :

- Menerapkan konsep ORDBMS pada sistem inventori yang terdistribusi.
- Menemukan pola pembagian layer (3-layer) jika memakai konsep ORDBMS.

1.3 Perumusan Masalah

Yang menjadi permasalahan dalam tugas akhir ini dalam merancang dan membuat aplikasi adalah sebagai berikut :

- Perancangan sistem inventori digunakan untuk menangani banyak gudang (terdistribusi).
- Penanganan mekanisme FIFO dalam sistem inventori terdistribusi.
- Penerapan ORDBMS berdasar pada *object oriented* perspektif.
- Pembuatan *interface* antar sistem agar sistem inventori ini dapat digunakan oleh sistem yang lain (pembelian, penjualan dan produksi).

1.4 Batasan Masalah

Batasan permasalahan dari tugas akhir ini adalah :

- Penitikberatan ada pada arsitektur dari penerapan ORDBMS pada sistem inventori yang terdistribusi.
- Sistem inventori ini dipandang sebagai sistem yang mempunyai state berupa posisi persediaan dan transisi berupa pemasukan dan pengeluaran barang.
- Mekanisme pencatatan harga yang digunakan hanya menggunakan FIFO.
- Sistem yang dibangun tidak termasuk sistem keamanan.
- Pemodelan sistem ini dilakukan dengan menggunakan *Unified Modelling Language* (UML).

1.5 Metodologi Pelaksanaan Tugas Akhir

Pembuatan tugas akhir ini terbagi menjadi beberapa tahapan sebagai berikut:

1. Definisi Kebutuhan dan Pencarian Data

   Pada tahap ini dilakukan pendefinisian kebutuhan pengguna dengan cara mengumpulkan semua permasalahan yang dihadapi oleh pengguna, kebutuhan

– kebutuhan serta keinginan pengguna terhadap sistem inventori ini. Semua informasi yang nantinya terkumpul dalam tahapan ini akan dimasukkan kedalam *Use Case Document* dan dimasukkan ke dalam file model UML yang dibuat (*.mdl).

2. Perancangan Sistem dan Aplikasi

- Pemodelan dan perancangan sistem

  Pemodelan sistem menggunakan UML. Adapun langkah-langkahnya adalah sebagai berikut :

  a. *Use Case Modelling*

     - Model use case mendeskripsikan kebutuhan-kebutuhan fungsional dari sebuah sistem dalam bentuk use case. Model ini terdiri atas fungsi-fungsi dasar yang terlibat dalam sistem
     - Dalam tahap ini dirancang pelaku yang terlibat dalam sistem atau dinamakan aktor beserta aktivitas utama yang dilakukannya.

  b. Menemukan *Key Abstraction*

     Key Abstraction adalah *class* yang ada dalam sistem. Dalam tahap ini dirancang *class* yang merupakan representasi dari fungsi-fungsi yang ada dalam sistem inventori ini. Fungsi-fungsi ini juga memuat atribut – atribut yang dibutuhkan.

     - *Determine Baseline Architecture*

       Dalam tahapan ini kita mendefiniskan layer-layer ada dalam perangkat lunak yang dibuat.

     - *Identify Design Elements*

Dari fungsi – fungsi yang ada dipecah lagi menjadi urut-urutan aktivitas yang lebih rinci. Untuk itu dibuat *Activity diagram*, *Use Case Realization*, *Sequence Diagram*, *Collaboration Diagram* serta *Class Diagram*.

- *Apply Architectural Analysis*

  Pada tahap ini elemen desain yang ada dikelompokkan menjadi paket-paket, paket ini akan menjadi subsistem-subsistem.

- Perancangan antar muka

  Perancangan antar muka dilakukan dengan memanfaatkan teknik-teknik yang memungkinkan pengguna mudah mengoperasikan aplikasi ini (*user friendly*).

3. Pembuatan Aplikasi

   Dalam tahap ini, dilakukan implementasi untuk membangun aplikasi berdasarkan rancangan yang telah dibuat pada tahap sebelumnya.

4. Uji Coba dan Evaluasi

   Aplikasi yang telah selesai ini nantinya juga akan dievaluasi dan diperbaiki demi kelayakan sistem dan keberhasilan dari sistem ini sesuai dengan tujuannya dibangun.

5. Penyusunan Buku Tugas Akhir

   Pada tahap ini disusun buku sebagai dokumentasi dari pelaksanaan tugas akhir. Dokumentasi ini dibuat juga sehingga orang lain yang ingin mengembangkan sistem informasi tersebut bisa mempelajarinya dari dokumentasi ini.

1.6 Sistematika Penulisan

Buku tugas akhir ini terdiri dari beberapa bab yang tersusun secara sistematis, yaitu :

BAB I PENDAHULUAN

Bab ini gambaran umum yang membahas latar belakang dan tujuan pembuatan perangkat lunak, seta permasalahan yang dihadapi dalam pengerjaan tugas akhir ini. Selain itu dijelaskan pula pembatasan terhadap masalah yang akan dihadapi serta metodologi yang dipakai untuk menyelesaikannya.

BAB II TEORI PENUNJANG

Bab ini membahas teori-teori dasar yang menunjang pengerjaan tugas akhir, meliputi pengertian, fitur, acuan, pembuatan serta beberapa *issue* pada ORDBMS serta model framework Microsoft .NET.

BAB III ANALISIS SISTEM

Dalam bab ini diuraikan mengenai sistem yang akan dibangun. Analisis sistem ini dijelaskan dengan menggunakan notasi UML, yaitu mulai pendefinisian *use case, activity diagram, treacebility, sequence diagram, VOPC* dan *class diagram.*

BAB IV PERANCANGAN PERANGKAT LUNAK

Dalam bab ini diuraikan desain perangkat lunak yang terdiri dari :

- Desain struktur basis data dengan menggunakan Object Relational Database Management System (ORDBMS)
- Desain untuk sistem yang terdistribusi dalam framework Microsoft .NET

BAB V PEMBUATAN APLIKASI

Bab ini membahas pembuatan perangkat lunak berdasar desain yang telah dibuat dengan menggunakan bahasa pemrograman yang telah ditetapkan.

BAB VI UJI COBA DAN ANALISIS HASIL

Berisi tentang uji coba yang telah dilakukan terhadap perangkat lunak yang telah dibuat, beserta analisis dari hasil uji coba tersebut.

BAB VII PENUTUP

Berisi kesimpulan yang di dapat dari pembuatan tugas akhir ini dan dilengkapi dengan saran untuk kemungkinan pengembangan selanjutnya.

BAB 2
TEORI PENUNJANG

Dalam bab ini dijelaskan mengenai teori dasar tentang DBMS, RDBMS, konsep dasar pemrograman dengan menggunakan *object oriented* serta pengenalan, pengertian dan konsep dasar pada ORDBMS.

Penjelasan lain dalan bab ini adalah konsep dan teknologi yang digunakan untuk membangun sistem inventori ini, yaitu Microsoft .NET, termasuk didalamnya adalah framework dari .NET yang memungkinkan untuk menerapkan sistem yang terdistribusi.

2.1 Relational Database Management System

Keberadaan DBMS dalam pengelolaan basis data dirasa memberikan keuntungan yang besar bagi pengguna. Namun, masih ada beberapa kekurangan yang terdapat dalam DBMS ini. Basis data yang berelasi ini sebagai jawaban dari kekurangan – kekurangan yang terdapat dalam basis data yang tidak berelasi, terutaman dalam masalah *redundancy* data

2.1.1 Definisi Relational Database Management Sytem

Apakah basis data yang berelasi? Sekumpulan table yang disimpan dalam produk basis data yang berelasi seperti Oracle belum menjamin kita memiliki basis data yang berelasi. Basis data yang berelasi mengimplikasikan pembuatan desain basis data dengan memperhatikan teori dasar relasional, yang didasarkan

pada konsep relasi pada matematika. Teori ini digambarkan melalui aturan normalisasi [1].

2.1.2 Relasi

Pada bagian ini akan dibahas mengenai jenis-jenis relasi yang ada pada basis data yang berelasi, yaitu relasi antara entitas.

a. Relasi 1-to- many

Yaitu relasi antara 1 entitas dengan banyak entitas. Relasi 1-to-many ini akan menyebabkan adanya *foreign key*.

b. Relasi 1-to-1

Yaitu relasi antara 1 entitas dengan 1 entitas. Entitas yang dominan akan memberikan *primary key* sebagai *foreign key* bagi entitas yang lebih lemah.

c. Relasi Many-to-Many

Relasi ini akan menyebabkan entitas tambahan yang berisi *primary key* dari masing – masing entitas.

2.1.3 Normalisasi

Ketika kita mengatakan sebuah basis data yang telah dinormalisasi, maka basis data tersebut tidak lagi se-normal aslinya. Dalam hal ini istilah normalisasi diambil dari notasi normal matematika, yang berarti orthogonal. Notasi dari normalisasi pada basis data mirip dengan ide yang ada pada notasi matematika.

Mengapa kita membuat basis data yang dinormalisasi? Normalisasi diturunkan dari konjungsi dengan teori bahasa pemrograman SEQUEL. Teori dasar dari teori relasional adalah jika basis data telah dinormalisasi, kita dapat mengembangkan

subset dari data dengan menggunakan operator pada SEQUEL. Hal ini menyebabkan normalisasi menjadi penting, karena pada basis data tanpa dinormalisasi, akan sulit mendapatkan informasi dari basis data tanpa menuliskan *coding* yang kompleks. Aturan mengenai normalisasi ini penting dalam model relasional dan masih penting dalam model obyek yang berelasi (*object-relational*).

Normalisasi merupakan proses transformasi data eksisting menjadi bentuk relasional. Tujuan dari normalisasi adalah mengeliminasi duplikasi informasi dan memudahkan pengubahan struktur tabel memperkecil pengaruh perubahan struktur basis data. Ada beberapa macam bentuk normalisasi dan yang sering digunakan adalah bentuk normalisasi pertama, kedua dan ketiga.

- Normalisasi Pertama (1NF)

  Normalisasi pertama berbentuk tabel yang bersifat flat. Tidak ada pengulangan grup, tidak memiliki atribut yang *multi value*.

- Normalisasi Kedua (2NF)

  Aturan normalisasi yang kedua muncul ketika terdapat lebih dari satu *primary key* (*multi-attribute primary key*). 2NF dibuat berdasarkan konsep *FULL FUNCTIONAL DEPENDENCY*. Definisi 2NF pada skema relasi R adalah 2NF apabila setiap nonprime atribut A (yang bukan anggota *primary key*) dalam R adalah *fully dependent* pada *primary key* dari R

- Normalisasi Ketiga (3NF)

  Aturan normaliasi yang ketiga muncul ketika terdapat *transitive dependency*. Secara praktis, kita bisa menggunakan definisi : skema relasi R adalah 3NF apabila setiap atribut nonprime dari R adalah

- Full functionality dependent pada setiap key dari R

- Nontransistive dependent pada setiap key dari R

2.2 Object Oriented Programming (OOP)

Pemodelan dengan menggunakan tipe obyek mirip dengan mekanisme *class* yang ada pada C++ dan Java. Seperti halnya *class*, obyek mempermudah pemodelan yang kompleks dan dapat mendekati kenyataan (*real world entity*). Obyek yang *reusable* dapat mempermudah dan mempercepat pembuatan aplikasi basis data. Konsep dasar dari obyek adalah adanya *state*, *behavior* dan *object identity* (OID).

- Enkapsulasi

  Konsep enkapsulasi berarti bahwa object terdiri dari data dan operasi yang digunakan untuk memanipulasinya. Pada beberapa pemrograman berdasar object, konsep enkapsulasi ini dapat dicapai dengan adanya tipe data abstrak (*abstract data types* / ADTs).

- Identitas Obyek (Object Identifier)

  Kunci dari pendefinisian sebuah obyek adalah identitas yang unik. Pada sistem yang berdasar obyek, obyek diberi sebuah identitas (OID), ketika obyek tersebut dibuat, yaitu :

  - Dibuat oleh sistem

  - Unik untuk masing-masing obyek

- Tidak bisa diubah saat *lifetime*. Sekali obyek dibuat, OID tidak bisa digunakan ulang oleh obyek yang lain, bahkan setelah obyek tersebut dihapus

- Tidak bisa dilihat oleh user (idealnya)

- Method

  Metode (*method*) pada sebuah obyek mendefinisikan *behavior* dari obyek tersebut. Metode ini bisa digunakan untuk mengubah *state* dari obyek atau melakukan query terhadap nilai dari atributnya.

- Class

  Obyek yang memiliki atribut yang sama dapat dikelompokan menjadi satu kedalam sebuah *class*. *Class* hanya mendefinisikan sekali untuk atribut dan metode daripada mendefinisikan secara terpisah untuk masing-masing obyek. Obyek yang terdapat dalam *class* dinamakan *instances* dari *class* tersebut. Sehingga masing-masing instance memiliki nilai sendiri-sendiri untuk masing-masing atributnya.

- Inheritance

  Beberapa obyek dimungkinkan memiliki atribut dan metode yang mirip. Jika terdapat kemiripan yang banyak, maka akan dimungkinkan untuk membagi atribut dan metode tersebut. Konsep ini dinamakan turunan (inheritance). Turunan memungkinkan satu *class* didefinisikan berdasar *class* yang lain yang lebih umum. *Class* ini dinamakan subclass, sementara *class* lain yang lebih umum dinamakan superclass.

- Object Oriented Database Management System

  Basis data yang berorientasi obyek memanfaatkan fitur *object oriented* dan *abstract data type*. Basis data obyek mempunyai OID sehingga obyek mudah untuk diidentifikasi. OID ini mirip dengan *primary key* yang ada pada basis data berelasi. *Object Oriented Database Management System* (OODBMS) mempunyai 2 set relasi, satu relasi adalah interelasi antar data item dan satu lagi adalah relasi abstrak (*inheritance*). Sistem akan menjadikan kedua relasi tersebut untuk menggabungkan antara data item dengan metode (*encapsulation*). Hasilnya adanya relasi langsung antara aplikasi dan basis data [7].

Kekurangan dari OODBMS ini adalah kinerja, karena tidak seperti RDBMS, pada OODBMS query sangatlah kompleks.

2.3 Object Relational Database Management System

Tujuan utama dari ORDBMS adalah menggabungkan keuntungan dari model relasional dan model obyek, yaitu menggabungkan fitur yang ada pada model obyek ke RDBMS.

2.3.1 Definisi Object Relational Database Management System

Apakah obyek yang berelasi ? Jika pertanyaan ini kita ajukan kepada selusin ahli, maka akan muncul selusin jawaban juga. Obyek yang berelasi merupakan usaha penggabungan antara paradigma relasional dengan paradigma obyek. Mengkombinasikan dua hal yang berbeda tidaklah mudah [2].

Permasalah utama dari mapping antara model obyek ke model relasional adalah bahwa obyek tidak dapat disimpan dan diambil dari relasional database secara langsung. Bahwa obyek memiliki *identity*, *state* dan *behavior*, sementara RDBMS hanya menyimpan data saja. Tujuan dari relasional model adalah untuk menormalisasi data (mengelimimasi duplikasi data dalam table), sementara tujuan dari desain yang berdasar obyek adalah memodelkan bisnis proses dengan membuat *real-world object* dengan data dan *behavior*.

Pada obyek yang berelasi, akan mengkombinasikan antara model relasi dan model obyek, sehingga pada obyek yang berelasi terdapat OID dan *primary keys*. Obyek akan direlasikan baik melalui OID maupun *referential integrity*. Pada obyek yang berelasi ini juga akan terdapat *trigger* dan *method*. Tabel bisa disusun secara *independent* maupun menurunkan atribut dan metode dari struktur induk (disebut *class*).

Tantangan terbesar adalah pemanfaatan fitur-fitur baru ini. Kita telah memiliki standar *tool* dari basis data yang berelasi ditambah semua struktur baru dari basis data obyek, yaitu dimana dan bagaimana memanfaatkan struktur baru ini.

Perbedaan antara RDBMS, OODBMS dan ORDBMS [7]

Tabel 2.1 Perbedaan RDBMS, OODBMS dan ORDBMS

Kriteria	RDBMS	OODBMS	ORDBMS
Mendukung fitur object oriented	Tidak mendukung, sulit untuk map object programming dengan basis data	Mendukung	Mendukung, tapi terbatas, terutama data type
Penggunaan	Mudah	Relatif mudah untuk	Mudah kecuali

		programmer	beberapa eksensi
Mendukung relasi yang kompleks	Tidak mendukung abstract data type	Mendukung tipe data yang kompleks dan data dengan relasi yang kompleks	Mendukung abstract data type dan relasi yang kompleks
Kinerja	Bagus	Kurang bagus	Diharapkan bagus
Kematangan produk	Matang	Relatif matang	Dalam pengembangan
Keuntungan	Penggunaan query yang sederhana dan bagus dalam kinerja	Dapat menangani semua tipe data dari aplikasi yang kompleks dan code dapat digunakan ulang (reusability)	Dapat menangani aplikasi yang besar dan kompleks
Kekurangan	Tidak dapat menangani aplikasi yang kompleks	Kinerja yang kurang bagus dan tidak dapat menangani sistem yang berskala besar	Kinerja yang kurang bagus untuk aplikasi web

2.3.2 Keuntungan Penggunaan Obyek

Secara umum, model tipe obyek mirip dengan mekanisme *class* pada C++ dan Java. Seperti halnya *class*, obyek dapat mempermudah memodelkan entitas dan obyek yang dapat digunakan kembali (*reusable*) memungkinkan pembuatan basis data aplikasi menjadi lebih cepat dan lebih efisien. Object abstraksi dan enkapsulasi dari *behavior* obyek menyebabkan aplikasi lebih mudah untuk dimengerti dan dikembangkan.

Berikut adalah beberapa keuntungan obyek yang ditawarkan melalui pendekatan model relasional.

1. Obyek dapat mengenkapsulasi operasi pada data.

   Table basis data hanya menyimpan data. Obyek bisa menyertakan operasi yang dibutuhkan dalam data tersebut. Misalkan dibutuhkan metode untuk menghitung total penjualan pada kasus penjualan. Atau obyek memungkinkan memiliki metode yang mengembalikan nama pelanggan, alamat pelanggan, nomer referensi atau bahkan sejarah pembelian pelanggan, sehingga sebuah aplikasi dapat dengan mudah memanggil metode untuk mendapatkan informasi.

2. Pemakaian obyek lebih efisiens

   Pemanfaatan tipe obyek memberikan efisiensi yang besar, yaitu :

   - Tipe obyek dan metode-metodenya disimpan dalam data pada basis data, sehingga tersedia untuk semua aplikasi yang menggunakannya. *Developer* mendapatkan keuntungan dari aplikasi yang sudah dibangun sebelumnya tanpa perlu membuat ulang struktur yang sama pada setiap aplikasi.
   - Kita dapat mengambil dan memanipulasi beberapa obyek yang berleasi seperti layaknya sebuah unit. Permintaan untuk mengambil sebuah obyek dari server akan mengambil obyek-obyek lainnya yang berelasi berdasar *object reference*

3. Obyek dapat merepresentasikan *Part-Whole Relationship*

   Dalam sistem relasional, terdapat kesulitan jika kita ingin merepresentasikan *part-whole relationship*. Namun pada tipe obyek, kita memiliki kosakata yang banyak untuk merepresentasikan *part-whole*

relationship. Sebuah object dapat menjadikan object lain sebagai atribut, dan atribut object dapat memiliki object atribut.

2.3.3 Oracle Object

Tipe obyek Oracle (*Oracle object type*) adalah tipe yang didefinisikan sendiri oleh pengguna (*user define types*) yang memungkinkan untuk memodelkan entitas yang kompleks mendekati kenyataan (*real world entity*). Suatu tipe obyek yang baru dapat dibuat dari tipe-tipe dalam basis data, tipe obyek lainnya yang telah dibuat sebelumnya, referensi obyek (*object references*) dan tipe koleksi (*collection types*). Metadata untuk tipe yang didefinisikan sendiri oleh pengguna dapat disimpan dalam skema yang tersedia untuk SQL, PL/SQL, Java dan antarmuka yang terpublikasi (*published interface*) lainnya.

Tipe obyek yang berkaitan dengan fitur *object oriented* seperti *array* dan *nested table* menyediakan cara pada level yang lebih tinggi untuk mengakses data dalam basis data. Dalam layer obyek, data masih disimpan dalam kolom dan table, tetapi dimungkinkan kita bekerja dengan data dalam istilah yang nyata, yang menyebabkan data mempunyai arti (*meaningfull*). Dengan istilah kolom dan table yang mempunyai arti tersebut pada saat melakukan query, proses pemilihan data dapat dilakukan secara sederhana. Obyek membantu kita melihat hutan sesederhana kita melihat pohon-pohon.

Pada obyek yang berelasi, kita masih dapat bekerja dengan tipe data yang berelasi (*relational data types*) dan menyimpan data dalam tabel yang berelasi, tetapi sekarang kita memiliki pilihan untuk memanfaatkan keuntungan dari fitur

object oriented. Kita bisa memanfaatkan fitur object oriented dalam data yang berelasi atau kita dapat bekerja dengan melakukan pendekatan pada model berorientasi obyek. Sebagai contoh kita dapat mendefinisikan beberapa tipe data obyek dan menyimpannya dalam kolom pada tabel yang berelasi. Kita juga dapat membuat *object view* pada data yang berelasi yang sudah ada untuk mengakses dan merepresentasikan data berdasar obyek atau kita dapat menyimpan data obyek dalam tabel obyek, dimana masing-masing baris adalah sebuah obyek.

Oracle mengimplementasikan tipe obyek dalam model relasi. Antarmuka tipe obyek tetap mendukung fungsi-fungsi standar model relasi seperti *query* (SELECT...FROM...WHERE), *commit* yang cepat, *backup* dan *recovery*, *scalable connectivity, row-level locking, read consistency, partioned tables.*

2.3.3.1 Elemen Dasar dari Oracle Object

Object–relational memperkenalkan beberapa konsep baru, yang akan dijelaskan berikut ini [5]:

2.3.3.1.1 Tipe Obyek

Tipe obyek merupakan bentuk dari tipe data. Tipe obyek dapat digunakan seperti tipe data lainnya misalnya, NUMBER atau VARCHAR2. Tipe obyek memiliki beberapa perbedaan dengan tipe data yang biasa digunakan pada relational basis data, antara lain :

- Tipe obyek tidak disediakan oleh basis data, sehingga pengguna harus mendefinisikan sendiri Tipe obyek yang hendak digunakan.
- Tipe obyek terdiri dari 2 bagian, yaitu atribut dan method.

Atribut merupakan data dari obyek yang bersangkutan. Misalnya obyek Employee mempunyai atribut id, name dan address. Atribut memiliki tipe data yang telah dideklarasikan sebelumnya, yang juga dapat digunakan pada obyek yang lain.

Metode merupakan fungsi ataupun prosedur, yang digunakan agar aplikasi dapat menyediakan atribut yang berguna pada tipe obyek. Method mendefinisikan tingkah laku dari tipe obyek dan mendefinisikan apa saja yang bisa dilakukan pada tipe obyek tersebut.

- Tipe obyek lebih spesifik dibandingkan tipe data yang telah disediakan oleh basis data. Hal ini menyebabkan tipe obyek dapat memodelkan struktur yang mendekati kenyataan.

Dapat dikatakan bahwa tipe obyek merupakan *template* dari sebuah struktur dan tabel obyek dibangun berdasar obyek tersebut.

2.3.3.1.2 Tipe Turunan (Inheritance)

Tipe turunan menambah keuntungan dari pemanfaatan dari obyek dengan mengijinkan kita untuk membuat tipe obyek (*subtype*) dengan menurunkannya dari tipe obyek yang telah dibuat sebelumnya (*supertype*). *Subtype* tidak hanya menurukan semua fitur yang ada pada induknya (*supertype*) melainkan juga dapat mengembangkannya. Misalnya *subtype* dari pelanggan adalah pelanggan perusahaan, dapat menambahkan atribut atau metode baru yang belum didefinisikan pada induknya.

2.3.3.1.3 Obyek

Ketika variable dari tipe obyek dibuat, maka *instances* dari tipe tersebut akan terbuat juga dan hasilnya adalah sebuah obyek. Obyek memiliki semua atribut dan metode yang telah didefinisikan pada tipe obyek. Pada obyek sebuah nilai dapat diberikan pada atribut dan metode dapat dipanggil.

2.3.3.1.4 Method

Method merupakan fungsi ataupun prosedur yang dideklarasikan pada tipe obyek untuk mengimplementasikan *behavior* dari obyek tersebut. Kegunaan meotde yang dasar adalah untuk mengakses data pada obyek. *Method* dapat mendefinisikan operasi dari sebuah aplikasi, sehingga aplikasi tersebut tidak perlu membuat operasi tersebut. Untuk menjalankan operasi tersebut, aplikasi tinggal memanggil *method* yang dikehendaki dari obyek yang dibutuhkan.

Misalkan kita telah mendefinisikan *method get_sum()* yang akan mengembalikan nilai total penjualan pada 1 transaksi. Berikut adalah contoh script pemanggilan *method* untuk *purchase order* (po) dan mengembalikan nilai total penjualan pada variable *sum_line_item*.

```
Sum_line_item = po.get_sum();
```

Berbeda dengan fungsi atau prosedur pada PL/SQL, tanda kurung tetap dibutuhkan pada setiap pemanggilan *method*, meskipun metode tersebut tidak memiliki argument atau parameter.

Metode bisa ditulis dalam PL/SQL maupun bahasa pemrograman yang lain. Metode yang ditulis dalam PL/SQL dan Java akan disimpan dalam basis data.

Ada 2 macam metode yang bisa dideklarasikan pada pendefinisian metode yaitu :

1. Member

   Member method adalah salah satu cara yang digunakan aplikasi untuk mengakses data dalam *object instance*. *Member method* gunakan untuk mendefinisikan semua operasi pada suatu tipe obyek.

   Member method mempunyai *built-in parameter* yang disebut SELF yang mewakili *object instance* dimana metodenya sedang dieksekusi. SELF tidak perlu dideklarasikan lagi, meskipun pendeklarasian secara eksplisit juga diijinkan. SELF akan secara otomatis dipasingkan pada metode. Pada fungsi, jika SELF tidak dideklarasikan, maka secara *default* akan bermode IN, sedangkan pada prosedur, jika SELF tidak dideklarasikan, akan bermode IN OUT.

   Pemanggilan *member method* dengan menggunakan notasi "dot", yaitu *object_variable.method()*. Notasi tersebut akan menspesifikasikan obyek dari metode terlebih dahulu, kemudian metode yang akan dipanggil. Paramater-paramater didefinisikan dalam tanda kurung jika dibutuhkan.

2. Statis

   Static method akan mempengaruhi suatu tipe obyek bukan *instance* dari tipe obyek tersebut. *Static method* digunakan jika diperlukan operasi yang global terhadap suatu tipe obyek, bukan spesifik pada suatu instance dari tipe obyek tersebut. *Static method* tidak memiliki parameter SELF. Untuk dapat

menggunakan *static method* maka digunakan notasi "dot" : *Type_name.method()*

Dan terdapat juga tipe metode yang ketiga, yaitu metode konstruktor. Sistem akan mendefinisikan tiap pendefinisian tipe obyek sebagai metode konstruktor. Untuk membuat suatu instance dari suatu tipe objek dapat cukup dengan memanggil konstruktornya.

2.3.3.1.5 Tabel Obyek (Object Table)

Tabel obyek adalah tabel yang masing-masing barisnya merepresentasikan sebuah obyek. Sebagai contoh, statemen berikut mendefinisikan tipe obyek *person* dan mendefinisikan tabel obyek dari obyek person.

```
CREATE TYPE person AS OBJECT
(
    Name            VARCHAR2(30),
    Phone           VARCHAR2(20)
);
CREATE TABLE person_table OF PERSON;
```

Kita dapat melihat tabel ini dengan 2 cara :

- Sebagai kolom tunggal, dimana masing-masing baris adalah obyek *person* (pendekatan model *object oriented*).
- Sebagai multi kolom tabel, dimana masing-masing atribut dari obyek *person*, yang bernama *Name* dan *Phone*, muncul sebagai kolom (pendekatan model relasional).

Kita bisa melakukan operasi sebagai berikut :

```
INSERT INTO person_table VALUES ("John Smith","1-800-555-1212" );
SELECT VALUE(p) FROM person_table p WHERE p.name = "John Smith";
```

Pernyataan pertama menambahkan obyek *person* pada *person_table*, yang memperlakukan tabel obyek sebagai multi kolom tabel. Sedangkan pernyataan kedua memperlakukan tabel obyek sebagai kolom tunggal, pemanfaatan fungsi VALUE akan mengembalikan baris sebagai *object instances*.

2.3.3.1.6 Tipe Data REF

REF adalah *logical pointer* yang mengacu pada sebuah baris pada obyek. REF ini merupakan tipe data Oracle. REF menyediakan mekanisme yang mudah untuk navigasi antar obyek. Oracle akan melakukan proses join bila dibutuhkan atau menghindari proses join.

2.3.3.1.7 Koleksi (Collection)

Untuk memodelkan relasi *one to many*, Oracle menyediakan 2 tipe data koleksi, yaitu *array* dan *nested table*. Tipe data koleksi dapat digunakan dimana saja, selayaknya tipe data yang lain, dimungkinkan sebuah obyek memiliki atribut dengan tipe data koleksi. Misalkan tipe obyek Penerimaan_Barang memiliki attribute yang bertipe *nested table* untuk menangani detail dari Penerimaan_Barang tersebut.

- Varray

  Varray merupakan koleksi yang terurutkan. Masing – masing elemen memiliki indeks. Indeks ini dapat digunakan untuk mengakses elemen tertentu dalam koleksi. Ketika kita mendefinisikan suatu kolekso yang berupa *varray*, maka kita harus mendefinisikan jumlah maksimum elemen dalam koleksi tersebut, meskipun jumlah elemen tersebut nantinya dapat diubah. *Varray* disimpan sebagai *opaque object* (seperti halnya *raw* dan BLOB).

- Nested Table

  Nested table dapat memiliki beberapa elemen, jumlah maksimum dari element tersebut tidak perlu didefinisikan pada saat pendeklarasian *nested table*. Kita dapat melakukan proses select, insert, update dan delete layaknya tabel biasa. Elemen dari *nested table* sebenanya disimpan secara terpisah yang mempunyai kolom yang mengidentifikasikan induknya atau obyek dimana *nested table* itu berada.

Jika kita akan menyimpan sejumlah data yang telah dapat dipastikan jumlahnya, atau melakukan proses loop pada elemen yang terurut atau mengambil/memanipulasi nilai dari koleksi, maka tipe koleksi yang digunakan adalah *varray*. Jika proses query (operasi insert. update dan delete) diperlukan dalam koleksi, maka digunakan *nested table*.

2.3.3.1.8 Tipe Evolusi (*Evolution*)

Dengan menggunakan statemen ALTER TYPE, maka sebuah tipe obyek dapat dilakukan beberapa perubahan [6]:

- Menambah dan mengurangi atribut
- Menambah dan mengurangi metode
- Mengubah atribut yang bertipe *numeric*, untuk menambah atau mengurangi panjang, presisi dan skala-nya.
- Mengubah panjang dari atribut yang bertipe karakter

2.3.3.2 Mendefinisikan Obyek dan Tipe Koleksi

Untuk mendefinisikan tipe obyek dan tipe koleksi, digunakan statement CREATE TYPE. Berikut adalah pembuatan tipe obyek *person*, *lineItem*, *lineitem_table* dan *purchase_order*. *LineItem_table* merupakan tipe koleksi, yaitu *nested table*. Masing – masing baris dalam *nested table* merupakan obyek yang tertipe *lineItem*.

```
CREATE TYPE person AS OBJECT
(
    name VARCHAR2(30),
    phone VARCHAR2(20)
);

CREATE TYPE lineitem AS OBJECT
(
    item_name VARCHAR2(30),
    quantity NUMBER,
    unit_price NUMBER(12,2)
);

CREATE TYPE lineitem_table AS TABLE OF lineitem;

CREATE TYPE purchase_order AS OBJECT
(
    id NUMBER,
    contact person,
    lineitems lineitem_table,
    MEMBER FUNCTION  get_value RETURN NUMBER
);
```

Untuk mendefinisikan isi dari method *get_value()*, digunakan statement

```
CREATE OR REPLACE TYPE BODY.
```

Pendefinisian tipe obyek tidak mengalokasikan penyimpanan. Sekali kita mendefinisikan suatu tipe obyek maka, tipe obyek tersebut dapat digunakan dalam

SQL statement seperti halnya tipe data yang lain, seperti kita menggunakan atau memanfaatkan tipe data VARCHAR2 atau NUMBER.

2.3.3.3 Penyimpanan

Oracle secara otomatis akan memetakan struktur yang kompleks dari tipe obyek menjadi struktur tabel yang sederhana.

2.3.3.3.1 Leaf – Level Attributes

Tipe obyek mirip dengan struktur pohon (*tree structure*), dimana cabang-cabang merepresentasikan atribut. Setiap cabang akan berakhir dengan atribut yang memiliki *built-in attribute* (seperti VARCHAR2, NUMBER atau REF) atau tipe koleksi (seperti VARRAY atau NESTED TABLE). Masing – masing dari *leaf-level attribute* pada tipe obyek akan disimpan di kolom pada tabel. *Leaf level attribute* yang bukan merupakan koleksi disebut *leaf level scalar attribute* dan tipe obyek.

Pada tabel obyek, Oracle menyimpan data dari masing-masing *leaf level scalar attribute* atau atribut REF pada kolom yang terpisah. Masing-masing *varray* disimpan dalam kolom, kecuali jika varray yang bersangkutan terlalu besar. Oracle menyimpan *leaf-level attribute* dari *nested table* pada tabel yang terpisah yang berhubungan dengan table obyek. Tabel tersebut harus dideklarasikan pada saat pendeklarasian tabel obyek.

Ketika atribut dari obyek pada tabel obyek diakses atau diubah, Oracle akan melakukan operasi yang sama pada kolom pada tabel tersebut. Pengaksesan nilai

dari obyek akan menduplikasi obyek dengan memanggil kontruktor dari tipe tersebut, dengan menggunakan kolom dari tabel obyek sebagai argumen.

Oracle menyimpan OID yang dibuat oleh sistem dalam kolom yang tersembunyi. Oracle memanfaatkan OID untuk membuat REF dari obyek yang bersangkutan.

2.3.3.3.2 Kolom yangTersembunyi pada Tabel yang Memiliki Kolom yang Bertipe Object

Ketika sebuah tabel yang didefinisikan dengan kolom yang mempunyai tipe obyek, Oracle menambahkan kolom yang tersembunyi pada tabel untuk *leaf level attribute* pada tipe obyek. Masing –masing kolom pada tipe obyek akan berhubungan atau berkorespondensi dengan kolom yang tersembunyi yang menyimpan informasi NULL dari kolom yang bertipe object.

2.3.3.3.3 REFs

Ketika oracle membuat REF dari baris yang merupakan obyek, REF tersebut tersusun atas OID, metadata dari table obyek dan bisa digabung dengan ROWID. Ukuran dari REF pada kolom yang bertipe REF tergantung pada properti penyimpanan yang berhubungan dengan kolom. Misalnya, kolom dideklarasikan dengan REF WITH ROWID, maka oracle akan menyimpan ROWID pada kolom REF. Jika kolom mendeklarasikan REF dengan klausa SCOPE, maka kolom tersebut akan lebih kecil dengan menspesifikan metadata dari tabel obyek dan ROWID-nya. REF dengan klausa SCOPE menyimpan 16 byte.

Jika OID berdasar pada *primary key*, Oracle akan membuat satu atau lebih kolom yang akan menyimpan nilai dari *primary key* tergantung dari jumlah *primary key*.

2.3.3.3.4 Internal Layout dari Nested Table

Baris dari *nested table* disimpan dalam tabel yang terpisah. Masing-masing kolom dari *nested table* mempunyai satu relasi pada tabel penyimpanan, bukan satu tabel penyimpanan untuk masing-masing baris. Tabel penyimpanan akan menyimpan semua elemen dari *nested table*. Tabel penyimpanan memiliki kolom NESTED_TABLE_ID yang tersembunyi, yang merupakan hasil *generate* dari system dan Oracle akan memapingkan kembali kolom dari *nested table* pada baris yang sesuai.

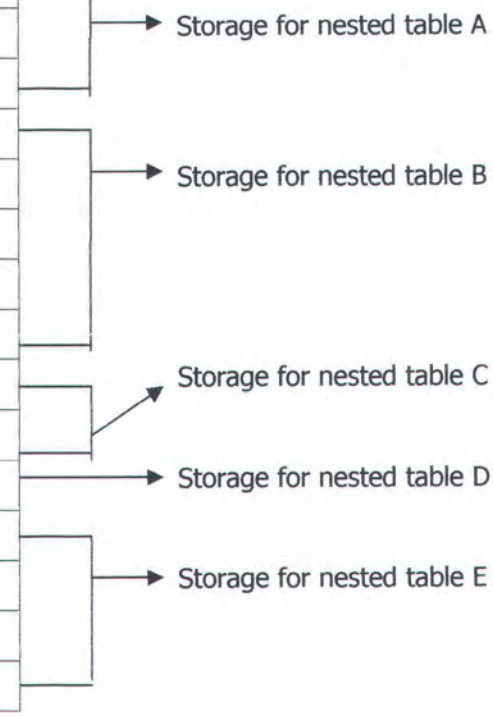
Proses query untuk mengambil elemen dari nested table dapat dipercepat dengan membuat table penyimpanan berindeks (*index-organized*), yaitu dengan menyertakan klausa ORGANIZATION INDEX didalam klausa STORE AS. Gambar 2.2 menunjukkan penyimpanan *nested table* dalam Oracle.

Sebuah *nested table* bisa terdiri atas obyek atau nilai skalar :

- Jika elemen terdiri dari obyek, maka tabel penyimpanan mirip dengan tabel obyek. Namun, baris dari *nested table* tidak memiliki OID, maka REF dari obyek pada *nested table* tidak bisa dibuat.
- Jika elemen terdiri dari nilai skalar, tabel penyimpanan terdiri *single* kolom yang disebut COLUMN_VALUE yang terdiri dari niali skalar.

DATA1	DATA2	DATA3	DATA4	NT_TABLE
...	...	...	...	A
...	...	...	...	B
...	...	...	...	C
...	...	...	...	D
...	...	...	...	E

NESTED_TABLE_ID	Value
A	A11
A	A12
A	A13
B	B21
B	B22
B	B23
B	B24
B	B25
C	C31
C	C32
D	D41
E	E51
E	E52
E	E53
E	E54



Gambar 2.2 Penyimpanan nested table dalam Oracle

2.3.3.3.5 Internal Layout untuk VARRAYs

Semua element dari varray disimpan dalam single kolom. Tergantung dari ukuran varray, dimungkinkan disimpan secara inline atau dalam BLOB.

2.3.3.4 Identitas Obyek (OID)

Setiap baris obyek pada tabel obyek memiliki sebuah *logical object identifier* (OID). Secara *default*, Oracle memberikan masing-masing baris obyek dengan OID yang unik yang dibuat oleh system, sepanjang 16 byte. Oracle tidak menyediakan dokumentasi untuk mengakses internal struktur dari OID, struktur bisa berubah kapan saja.

Kolom OID pada tabel obyek adalah tersembunyi. Pada proses pengambilan dan navigasi obyek, kita dapat mengabaikan OID dan dengan melalui referensi obyek. OID pada sebuah baris obyek, secara unik akan mengidentifikasikan baris obyek tersebut pada tabel obyek. Secara implisit, Oracle akan membuat dan mengatur indeks dari kolom OID pada tabel obyek. Pada proses distribusi dan replikasi, identitas yang dibuat oleh system secara unik akan menyebabkan Oracle mengidentifikasikan obyek dengan tidak ambigu.

2.3.3.4.1 Identitas Obyek Berdasar Primary Key

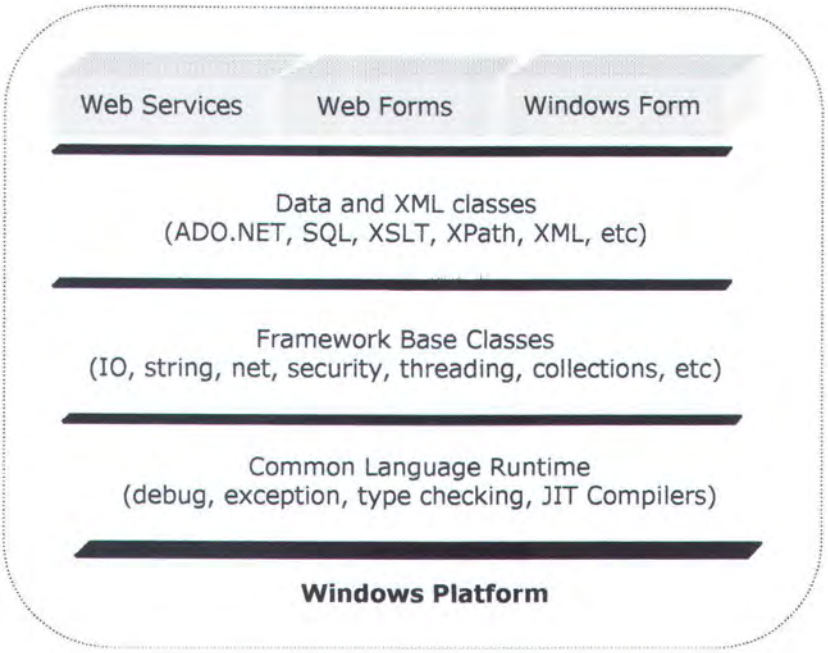
Jika *local indentifier* yang unik bisa diasumsikan sebagai *global identifier* yang unik (dengan kata lain, tabel tidak terdistribusi dan tereplikasi), maka bisa digunakan *primary key* dari baris obyek sebagai OID. Dengan menjadikan *primary key* sebagai OID akan menghemat 16 byte dari penyimpanan untuk masing-masing obyek yang dibutuhkan oleh *identifier* yang dibuat oleh system.

Identifier berdasar *primary key* juga dapat mempercepat dan mempermudah proses *load* data pada tabel obyek. Pada OID yang dibuat oleh system, masih harus dimaping ulang dengan menggunakan beberapa kunci, terutama jika *reference* juga disimpan.

2.4 Framework Microsoft .NET

Framework .NET adalah sebuah *platform* komputing baru yang dirancang untuk menyederhanakan pengembangan aplikasi yang terdistribusi. Bahasa yang didukung oleh framework .NET adalah Visual Basic .NET, C++ .NET, ASP .NET dam C#. .NET juga dapat digunakan oleh bahasa pemrograman dari pihak ketiga seperti Eiffel, COBOL dan Perl [4].

Pada gambar 2.3, terlihat framework .NET berada di atas sistem operasi, yang berupa sistem operasi buatan Microsoft, yaitu Windows 2000 (Server dan profesional), Windows NT 4.0 (Server dan workstation), Windows Millenium Edition, Windows 98 dan Windows XP Profesional.



Gambar 2.3 Arsitektur Framework .NET

Framework .NET terdiri dari sejumlah komponen sebagai berikut :

1. Common Language Runtime

   Common Language Runtime (CLR) adalah komponen terpenting dalam framwork .NET. CLR menyediakan lingkungan eksekusi kode yang melakukan manajemen kode. Manajemen kode dapat berbentuk manajemen memori, manajemen *thread*, manajemen *security*, verifikasi kode dan kompilasi dan layanan sistem lainnya. CLR dapat digunakan oleh program .NET apapun, tanpa merujuk bahasa pemrograman tertentu.

   Bahasa pemrograman .NET tidak melakukan kompilasi menjadi kode yang dieksekusi, melinkan kompilasi menjadi intermediate code yang disebut *Microsoft Intermediate Language* (MSIL). Kode MSIL kemudian dikirim ke CLR yang akan mengubah kode menjadi bahasa mesin yang dijalankan di mesin host.

2. Framework Base Class

   Framework Base Class adalah *class library* yang dapat digunakan oleh setiap bahasa yang didukung oleh .NET. *Base classes* ini mengandung sekumpulan *library* standar yang dibutuhkan seperti *collection*, input/output, tipe data dan class numerik. Sebagai tambahan, disediakan pula *class* untuk pengaksesan layanan-layanan sistem operasi, seperti grafik, jaringan, *thread* dan akses data. Juga sekumpulan *class* untuk pembuatan aplikasi berskala enterprise, seperti transaksi, event dan messaging.

3. *Class* untuk Data dan XML

   Mendukung manajemen data dan manipulasi XML. *Class* data mendukung manajemen data *persistent*, termasuk didalamnya *class* SQL dan ADO.NET. Framework .NET juga menyediakan sejumlah *class* untuk memanipulasi data XML, melakukan pencarian serta penterjemahan XML.

4. Web Service

   Web Service adalah entitas terprogram yang dapat diakses oleh banyak sistem terpisah melalui penggunaan standar internet, seperti XML dan HTTP. Sebuah web service dapat digunakan secara internal oleh sebuah aplikasi atau secara eksternal diekspos melalui internet untuk digunakan oleh banyak aplikasi. Dengan kata lain web service merupakan cara mengakses fungsi-fungsi yang ada di server secara remote.

5. Web Form

   Menyediakan sejumlah *class* untuk pembuatan GUI (Graphical User Interface) untuk aplikasi web.

6. Windows Forms

   Menyediakan sejumlah *class* untuk pembuatan GUI untuk aplikasi windows.

2.4.1 Pembagian Layer Dalam Aplikasi Terdistribusi

Prinsip utama dari aplikasi terdistribusi adalah pembagian aplikasi secara logical dalam 3 lapisan (layer) utama [8], yaitu:

2.4.1.1 Layer Aplikasi

Layer aplikasi merupakan layer yang akan langsung berhubungan dengan antarmuka. Layer aplikasi ini meliputi *rich client interface* dan *thin client interface*. *Rich client interface* merupakan antarmuka yang dibuat dengan menggunakan Microsoft Win32 API atau Windows Form, sedangkan *thin client interface* merupakan aplikasi yang dibangun dalam web browser.

2.4.1.2 Layer Bisnis

Layer bisnis merupakan *middle interface* yang merupakan tempat terjadinya pemrosesan data sesuai dengan aturan bisnis seperti perhitungan biaya dan server side validation. Selain pada layer bisnis, aturan bisnis juga bisa diterapkan dalam basis data dengan menggunakan *stored procedure* dan *view*.

2.4.1.3 Layer Data Akses

Layer data akses merupakan layer yang menyediakan fungsi-fungsi yang mengakses basis data secara langsung bagi layer aplikasi dan layer bisnis. Fungsi-fungsi tersebut misalnya insert, update, delete dan select serta fungsi-fungsi lain yang akan mengakses dalam basis data.

2.4.2 Membangun Aplikasi Terdistribusi

Untuk membangun aplikasi terdistribusi beberapa hal perlu menjadi pertimbangan adalah :

2.4.2.1 Pemodelan Aplikasi dan Data

Untuk membangun aplikasi yang ditujukan untuk level enterprise memerlukan perencanaan secara konseptual, logical dan physical. *Developer* harus mengerti proses bisnis yang sedang berjalan, struktur data yang sudah ada dan akhirnya mengusulkan sebuah solusi.

Pemodelan aplikasi dan data adalah sebuah proses identifikasi, dokumentasi dan implementasi dari data dan proses yang dibutuhkan oleh aplikasi. Hal ini melibatkan data mdel dan proses yang sudah ada untuk melihat apakah dapat kembali digunakan dan juga melibatkan proses pembuatan dan model dan proses yang baru untuk disesuaikan dengan kebutuhan.

Hal-hal yang penting dalam proses pemodelan ini adalah :

- Identifikasi data dan proses yang berhubungan
- Pendefinisian data (misalnya tentang tipe data, ukuran dan nilai asal)
- Pendefinisan proses-proses operasional
- Pemilihan teknologi penyimpanan data

Salah satu untuk memodelkan aplikasi adalah dengan menggunakan *Universal Modelling Language* (UML). UML adalah sebuah sistem notasi tentang bagaimana cara mengkonsep, mengotomatisasi proses dan mengatur interaksi dengan user. Dengan menggunakan UML, kita dapat menggunakan notasi standar untuk mendefinisikan berbagai macam aktifitas dari user dan aplikasi dengan menggunakan [3]:

- Use Case Diagram

  Use case digram menggambarkan kebutuhan fungsionalitas sistem. Dari penggambaran use case diagram dapat diketahui sistem yang diinginkan oleh pelanggan. Use case diagram ini digambarkan dengan notasi, sementara penjelasan detail dari use case ini terdapat dalam *use case spesification.*

- Activity diagram

  Diagram aktivitas ini menggambarakan alur dari use case yang ada.

- Sequence Diagram

  Sequence diagram menggambarkan interaksi antar obyek berdasar urutan waktu

- Collaboration Diagram

  Collaboration diagram menunjukkan interaksi diagram yang menunjukkan urutan *message* dengan mengimplementasikan operasi.

- Deployement Diagram

  Deployememt diagram menunjukkan hubungan antara device yang digunakan dalam sistem.

Dengan menggunakan UML, kita dapat memvisualkan model yang seudah ada dan mengusulkan proses bisnis, arsitektur aplikasi, strutur data dan interaksi *user.* Diagram UML sangat memudahkan dalam memberi pengertian tentang cara kerja dan bagaiman cara interaksi dari *user.*

2.4.2.2 Bahasa Pemrograman yang Digunakan

Dalam pemilihan bahasa pemrograman yang akan digunakan dalam membangun aplikasi akan sangat tergantung pada pengalaman yang sudah dimiliki dan scope aplikasi yang akan dibangun. Apabila batasan aplikasi kecil maka biasanya aplikasi tersebut dibuat dengan menggunakan satu bahasa pemrograman, sedangkan apabila aplikasi yang akan dibuat batasannya besar biasanya dibangun dengan menggunakan beberapa bahasa pemrograman.

Dalam proses pengembangan aplikasi Visual Studio .NET memeberikan banyak alternatif bahasa pemrograman, diantaranya :

1. Visual Basic .NET
2. Visual C# .NET
3. Visual C++ .NET
4. Managed Extention for C++
5. Transact-SQL
6. Beberapa bahasa script diantaranya : VBScript, Jscript, Jscript .NET, Windows Scripting Host (WHS)

Dari alternatif bahasa pemrograman yang ada tersebut perlu dicari bahasa pemrograman yang sesuai dengan kebutuhan berdasarkan kelebihan dan kekurangan masing-masing serta kemampuan sumber daya developer.

2.4.3 Teknologi Pengaksesan Data

Dalam teknologi pengaksesan data untuk membangun aplikasi dengan menggunakan Microsoft Visual Studio .NET tersedia 3 alternatif teknologi, yaitu :

2.4.3.1 ADO .NET

ADO .NET secara spesifik dibuat untuk aplikasi yang bersifat message-based application meskipun masih menyediakan fungsi-fungsi untuk digunakan oleh arsitektur yang lain. ADO .NET memaksimalkan data sharing dengan mengurangi jumlah koneksi yang aktif (*active connection*) pada basis data, sehingga mengurangi jumlah *user* yang memanfaatkan sumber (*resource*) yang ada pada server basis data.

ADO .NET menyediakan banyak cara untuk melakukan pengaksesan data. Apabila aplikasi yang berbasis web atau XML web service membutuhkan akses pada banyak sumber data, membutuhkan proses saling berhubungan dengan aplikasi lain baik secara lokal maupun secara *remote* atau memanfaatkan data yang ada dicache maka dataset dapat menjadi alternatif yang bagus. Alternatif lain adalah ADO .NET juga menyediakan *data command* dan *data reader* untuk berkomunikasi secara langsung dengan *data source*. Operasi secara langsung dengan *data command* dan *data reader* ini meliputi proses menjalankan query dan *stored procedure*, membuat basis data obyek dan menjalankan proses menghapus dan mengubah secara langsung dengan menggunakan DDL command.

Dataset merupakan struktur data yang bersifat relasional yang dapat ditulisi dan dibaca atau serialisasi dengan menggunakan XML. Komponen-komponen dapat saling berbagi data dengan menggunakan *XML schema* untuk mendefinisikan struktur relasional dari dataset. Karakteristik utama dari dataset adalah data yang ada dalam dataset dapat diakses dan dimanipulasi dengan 2 cara:

1. Sebagai tabel dalam basis data relasional

   Dataset dapat berisikan satu atau lebih tabel ataupun tabel koleksi. Aspek penting dari dataset adalah dapat menjaga hubungan dari tabel-tabel yang dimiliki seakan-akan berada dalam memori dari penyimpanan data berelasi.

2. Sebagai struktur XML

   Data yang berada dalam dataset dapat juga diakses sebagai data XML. Dataset menyediakan juga metode-metode untuk membaca dan menulis data sebagai XML ataupun membaca dan menulis dari dataset sebagai skema XML.

2.4.3.2 ADO

Sebagaimana ADO .NET, ADO juga dimanfaatkan dalam berbagai proses pembangunan aplikasi, seperti membangun *front-end database client* dan membangun *middle tier business object*. Proses penggunaan ADO ini menggunakan koneksi secara terus menerus (*live connection*) pada data yang ada pada basis data berelasi maupun *data source* lain.

ADO juga menyediakan fasilitas yang tidak ada pada ADO.NET seperti *scrollable* dan *server side cursor*. Namun demikian, karena *server side cursor* akan memakan *resource* yang ada dalam basis data server maka akan memberikan dampak negatif cukup besar pada masalah kinerja dan *scalability* pada aplikasi yang akan dibangun. Untuk mentransfer ADO recordset melalui firewall maka kita perlu mengeset firewall agar memungkinkan proses *COM Mashaling request* sehingga dapat semakin memperumit pengaturan sistem sekuriti. Cara lain untuk

mentransfer ADO recordset ini adalah dengan membentuknya menjadi format XML dan ditransfer sebagai text.

2.4.3.3 OLE DB

OLE DB merupakan teknologi yang mendasari ADO dan ADO .NET. OLE DB merupakan *open standard* untuk mengakses berbagai macam *data source*, baik yang merupakan sumber data relasional atau yang merupakan bukan relasional seperti ISAM/VSAM dari mainframe, hierarchical database, email, file system, file text dan data geografis.

2.4.4 Proses Komunikasi dalam Aplikasi Terdistribusi

Untuk membuat aplikasi yang terdistribusi maka layer-layer yang telah dijelaskan sebelumnya dapat didistribusikan menjadi layer terpisah pada mesin lain. Ada beberapa pilihan pada .NET untuk melakukan komunikasi antar komponen antara lain :

2.4.4.1 ASP .NET Web Service

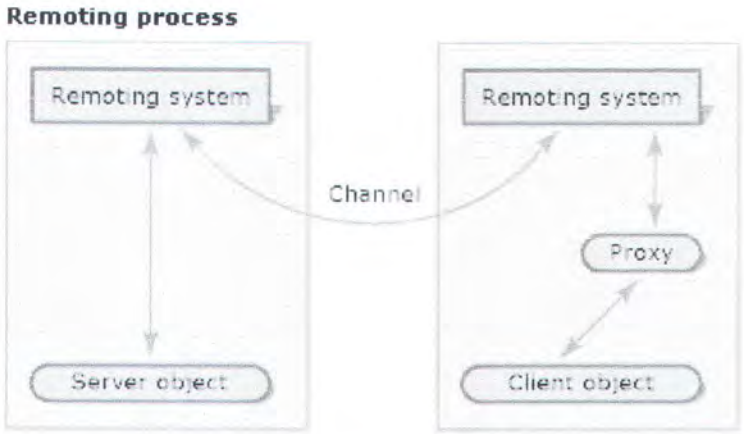
ASP .NET Web Service merupakan aplikasi yang menyediakan kemampuan untuk bertukar data alam lingkungan yang *scalable* dan *stateless* dengan menggunakan standar protokol seperti HTTP, XML, XSD, SOAP dan WSDL. Web Service memungkinkan untuk membangun aplikasi yang modular di dalam satu perusahaan atau di perusahaan yang berbeda dalam lingkungan implementasi, *platform* dan perangkat keras yang berbeda.

Beberapa karakteristik ASP .NET Web service adalah sebagai berikut :

- Komunikasi antar layer menggunakan XML Web service
- Membutuhkan IIS sebagai *host* aplikasi
- Menyediakan fungsi-fungsi yang dapat diakses oleh *client* melalui SOAP/XML
- Dapat memanfaatkan *security system* yang ada dalam IIS
- Tidak semua obyek dapat dilewatkan melalui web service
- Tidak dapat melakukan proses debugging

2.4.4.2 .NET Remoting

.NET Remoting memungkinkan aplikasi yang berbeda untuk berkomunikasi satu sama lain tanpa mempedulikan apakah aplikasi tersebut berada dalam 1 mesin yang sama atau berada dalam komputer yang berbeda, baik dalam satu *local area network* atau bukan. Berikut adalah proses dari remoting :



Gambar 2.4 Proses remoting

Remoting sistem akan membuat proxy obyek yang merepresentasikan *class* dan mengembalikan reference ke proxy tersebut kepada obyek client. Ketika

client memanggil suatu method, infrastruktur remoting akan memenuhi panggilan tersebut, dengan mengecek tipe dari informasi dan mengirim ke server melalui channel. Channel akan membawa pesan pada server. Client remoting sistem akan mengembalikan hasil dari panggilan obyek client melalui proxy.

Beberapa karakteristik yang ada dalam .NET remoting adalah sebagai berikut :

- Komunikasi dilakukan dengan melakukan passing obyek antar layer baik *by value* maupun *by reference*.
- Dapat melewatkan obyek tanpa kehilangan atribut apapun. XML web service tidak mempunyai kemampuan untuk memelihara detail obyek ketika dipasingkan
- Mempunyai mekanisme pengaturan lifetime object, client activation, serialization format dan lain-lain
- Tidak melakukan proses debugging.

BAB 3
ANALISIS KEBUTUHAN SISTEM

Bab ini membahas kebutuhan sistem akan sistem inventori yang dibuat untuk tugas akhir ini. Pendefinisian kebutuhan sistem ini digambarkan dalam notasi UML, yaitu berupa *use case diagram*, *activity diagram*, *sequence diagram*, *view of participating class* (VOPC) dan *class diagram*,.

3.1 Analisis Kebutuhan Sistem

Analisis kebutuhan mendefinisikan ruang lingkup sistem yang dibuat dalam tugas akhir ini. Sistem yang dibuat ini merupakan sistem dari rumah pemotongan ayam (RPA), sehingga sistem inventori yang dibangunpun berbeda dari sistem inventori lainnya. Dalam bab ini dijelaskan mengenai kebutuhan sistem sesuai dengan operasi bisnis dari rumah pemotongan ayam Pembahasan dimulai dari mendefinisikan *use case diagram.* Dari masing – masing use case kemudian didefinisikan *activity diagram* dan *sequence diagramnya.*

Oleh karena itu pembahasan re-analisis kebutuhan ini terdiri dari :

1. Use case Diagram

   Menjelaskan use case yang akan diimplementasi dalam sistem ini.

2. *Activity diagram*

   Menjelaskan mengenai urutan aktivitas bisnis dari sistem di Rumah Pemotongan Ayam.

3. Traceabilities diagram

   Diagram ini menunjukkan implementasi masin-masing use case yang diwujudkan dengan use case realization.

4. Sequence Diagram

   Menjelaskan interaksi antar obyek berdasar urutan waktu.

5. VOPC

   Menjelaskan *class* yang terlibat dalam satu use case.

6. Class Diagram

   Menjelaskan relasi antar *class* yang merepresentasikan struktur dari sistem.

Secara singkat, sistem yang dibuat dalam tugas akhir ini adalah sistem inventori yang mengatur proses pemasukan dan pengeluaran barang. Dari proses pemasukan dan pengeluaran barang ini akan diperoleh informasi posisi persediaan (stok) untuk masing – masing barang, untuk masing – masing gudang. Untuk memperoleh posisi persediaan stok ini setiap transaksi penerimaan dan pengeluaran barang harus diposting terlebih dahulu dan untuk membatalkan posting ini dilakukan proses unposting, namun proses unposting baru dapat dilakukan jika memenuhi beberapa kondisi yang akan dijelaskan pada Bab 4.

Untuk memudahkan penjelasan sistem dari RPA ini, maka sistem inventori ini akan dibagi menjadi beberapa bagian, yaitu :

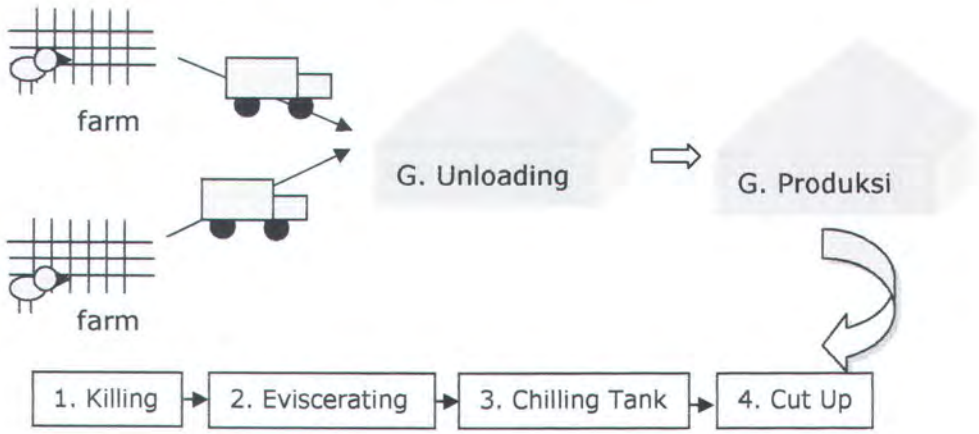
3.1.1 Penerimaan Barang

Modul ini berisi semua use case untuk proses penerimaan barang. Ada 2 aktor yang terlibat dalam proses penerimaan barang ini yang nantinya berinteraksi

langsung dengan sistem. Aktor tersebut adalah petugas gudang dan petugas gudang unloading. Perbedaan utama dari kedua aktor ini adalah jenis barang yang diterima dalam proses penerimaan barang.

1. Gudang Unloading

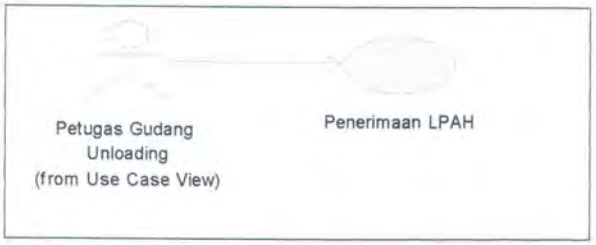
Gudang unloading ini hanya menerima ayam hidup. Ayam hidup yang diterima oleh gudang unloading ini berasal dari *farm*, yang kemudian akan diolah (mulai proses penyembelihan sampai dipotong-potong) oleh gudang produksi. Dalam gudang produksi ini ayam hidup akan melalui beberapa proses sebelum menjadi potongan ayam, proses-proses itu adalah killing, Eviscerating, chilli tank, cut up dan rendering. Gambar 3.1 adalah proses yang dilalui ayam mulai dari *farm* sampai menjadi potongan ayam.



Gambar 3.1 Proses yang dilalui ayam hidup mulai dari farm

Istilah – istilah proses yang dilalui ayam ini akan dijelaskan pada Lampiran A. Use case untuk aktor petugas gudang unloading pada modul

penerimaan barang ini hanya menerima ayam hidup. Gambar 3.2 adalah notasi UML untuk use case aktor petugas gudang unloading.

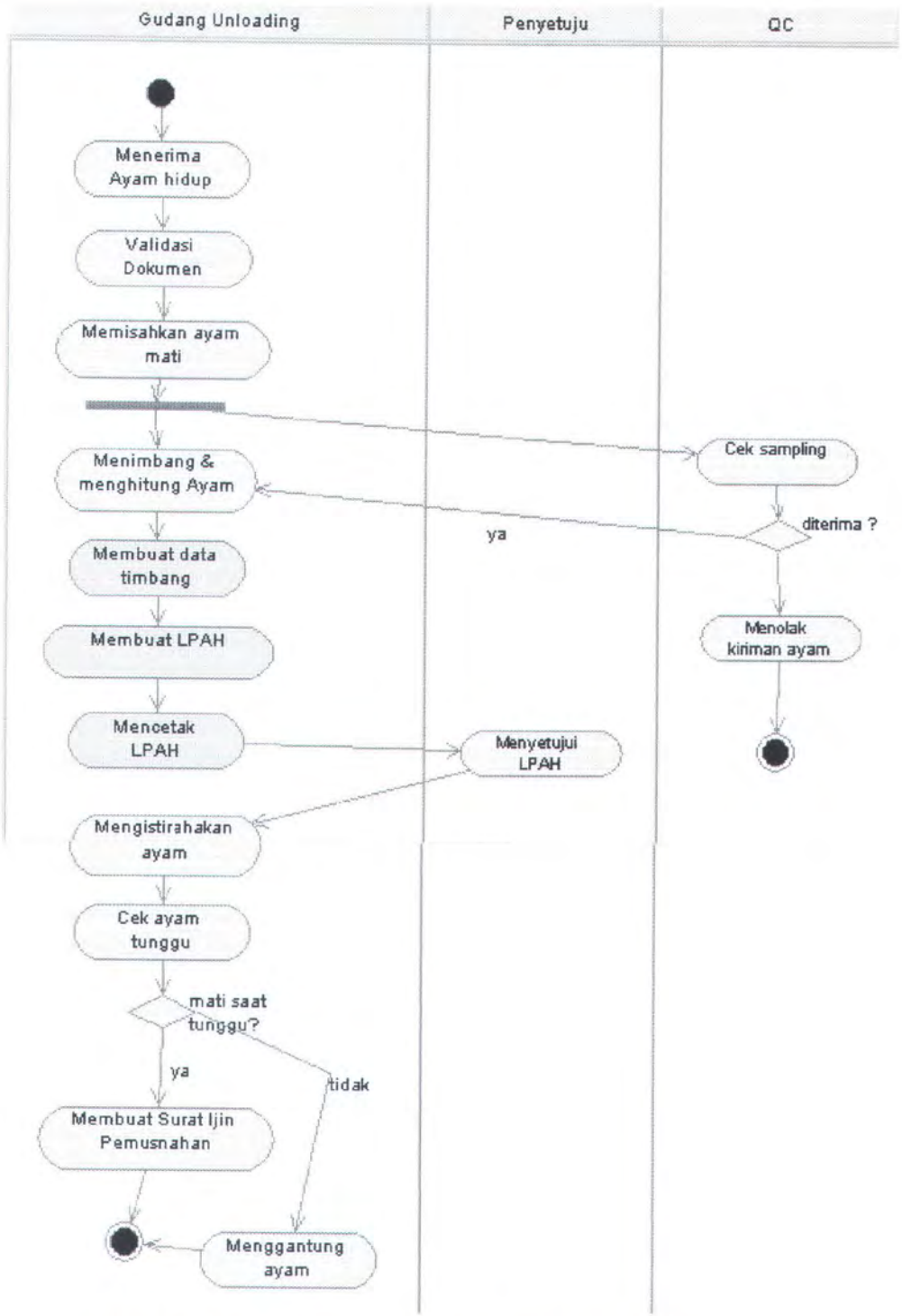


Gambar 3.2 Use case aktor petugas gudang unloading

Tujuan dari use case ini adalah mencatat seluruh transaksi penerimaan ayam hidup, termasuk didalamnya data timbang untuk masing-masing jenis ayam yang diterima pada 1 transaksi penerimaan, serta penghitungan jumlah ayam hidup yang diterima berdasar data ayam yang dikirim supplier dan data ayam yang mati di perjalanan ketika pengiriman.

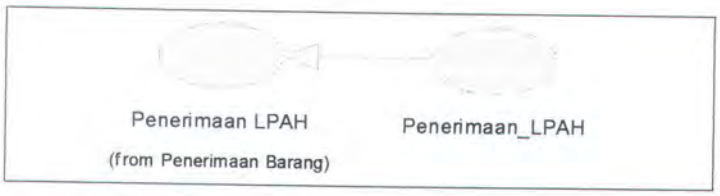
Dari use case ini kemudian dibuat *activity diagram* untuk menjelaskan alur aktivitas dari petugas gudang unloading ini. *Activity diagram* tersebut dapat dilihat pada gambar 3.3. Setelah menerima menerima ayam hidup, memvalidasi dokumen dan memisahkan ayam yang mati (ada kemungkinan ayam mati diperjanan), kemudian petugas menghitung dan menimbang ayam. Setelah ayam hidup dinyatakan diterima, data transaksi penerimaan ayam hidup ini diinputkan ke sistem dengan memasukan data timbang dan jumlah ayam yang diterima (yang diinputkan adalah jumlah ayam yang dikirim oleh fam, yang mati diperjalanan dan menurut perhitungan rumah pemotongan ayam ini). Jika LPAH ini disetujui oleh penyetuju LPAH, maka petugas mengistirahatkan ayam dan mengecek apakah ada ayam yang mati selama

proses menunggu, jika ada maka petugas membuatkan surat ijin pemusnahan ayam mati.



Gambar 3.3 Activity diagram untuk use case penerimaan LPAH

Dari use case diagram kemudian dibuat *use case realization*, untuk selanjutnya dibuat *sequence diagram*. Gambar 3.4 menjelaskan use case realization dari penerimaan LPAH.



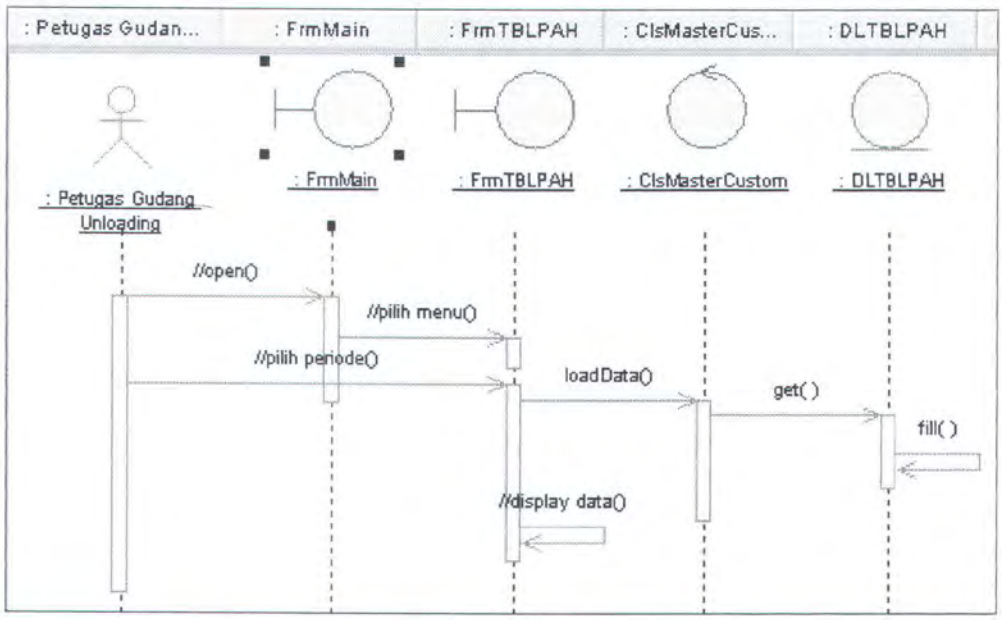
Gambar 3.4 Use case realization dari use case penerimaan LPAH

Dalam *sequence diagram* ini ada beberapa *flow*, yaitu :

- View

Flow view ini menjelaskan urutan interaksi obyek untuk melihat transaksi penerimaan ayam hidup. Gambar 3.5 menjelaskan urutan interaksi obyek untuk melihat transaksi penerimaan ayam hidup.

Pada sequence diagram tersebut, proses yang terjadi adalah actor petugas gudang membuka FrmMain, dan memilih menu untuk transaksi penerimaan ayam hidup (FrmTBLPAH). Setelah user memilih periode maka *class* FrmTBLPAH akan memanggil prosedur loadData pada ClsMasterCustom dan ClsMasterCustom akan memanggil prosedur Get() pada DLTBLPAH yang berfungsi mengambil data pada basis data. DLTBLPAH akan mengisikan data ke dataset dengan prosedur Fill(). Dan FrmTBLPAH akan menampilkan data.

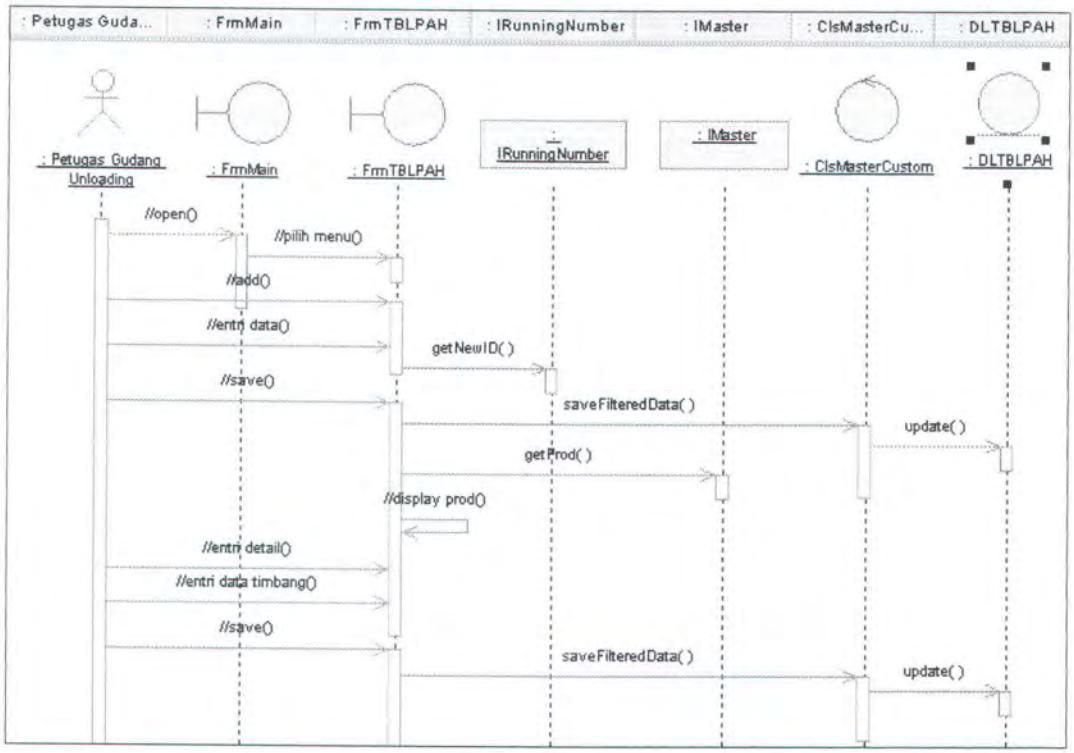


Gambar 3.5 Sequence diagram basic flow untuk use case penerimaan LPAH

- Add

Flow add pada *sequence diagram* ini ditujukan untuk mengetahui urutan interaksi *class* jika petugas gudang unloading menambah transaksi penerimaan LPAH. Gambar 3.6 menjelaskan urutan interaksi obyek untuk menambah transaksi penerimaan ayam hidup.

Pada *sequence diagram* tersebut proses yang terjadi adalah user membuka form transaksi penerimaan ayam hidup dan mengentrikan data. FrmTBLPAH akan memanggil fungsi GetNewID() pada sub sistem RunningNumber untuk mendapatkan ID transaksi. Untuk proses menyimpan data, FrmTBLPAH akan memanggil prosedur saveFilterdData() pada ClsMasterCustom dan ClsMasterCustom akan memanggil prosedur update(). Untuk proses mengisi data detail, user mengisi data produk dan data timbang dari masing-masing produk,



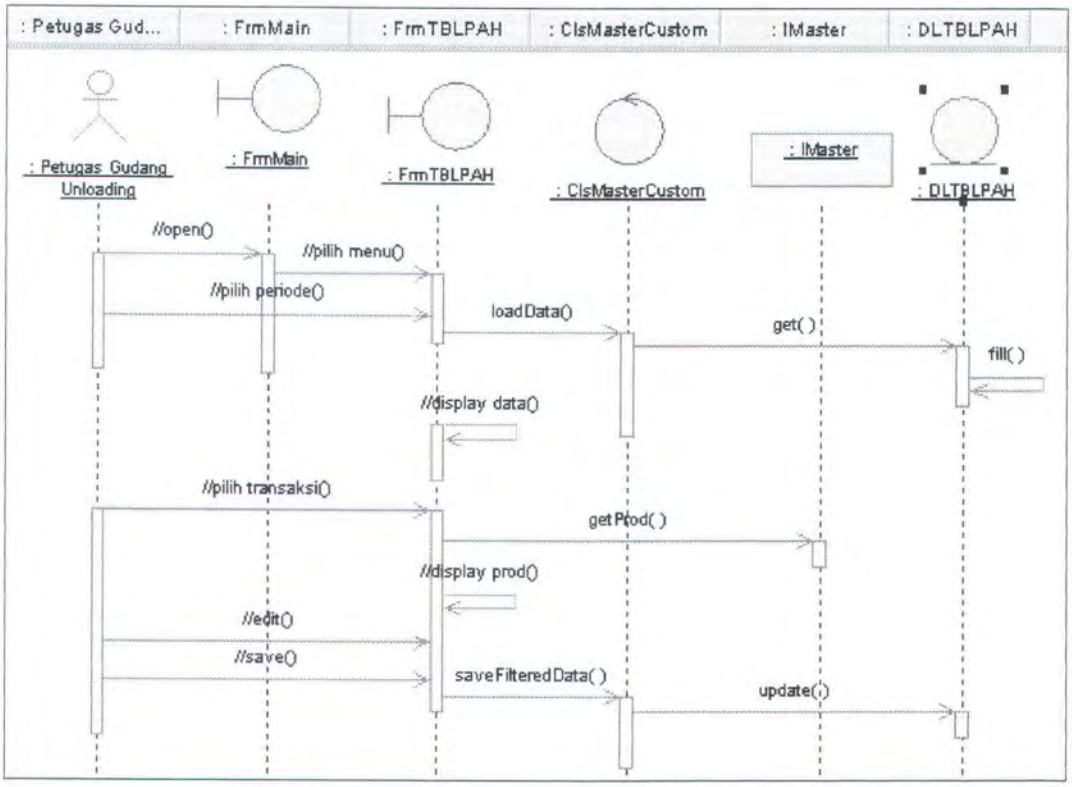
Gambar 3.6 Sequence diagram add untuk use case penerimaan LPAH

FrmTBLPAH akan memanggil prosedur saveFilteredData() pada ClsMasterCustom dan ClsMasterCustom memanggil prosedur update().

- Edit

Flow edit pada *sequence diagram* ini ditujukan untuk mengetahui urutan interaksi *class* jika petugas gudang unloading mengubah transaksi penerimaan LPAH. Gambar 3.7 menjelaskan urutan interaksi obyek untuk mengubah transaksi penerimaan ayam hidup.

Pada *sequence diagram* tersebut, setelah data tertampil (untuk menampilkan data lihat bagian View dari use case ini) user memilih transaksi yang akan diedit datanya, kemudian melakukan pengubahan data. Untuk proses penyimpanan data sama dengan proses penyimpana data pada proses Add, yaitu FrmTBLPAH memanggil prosedur



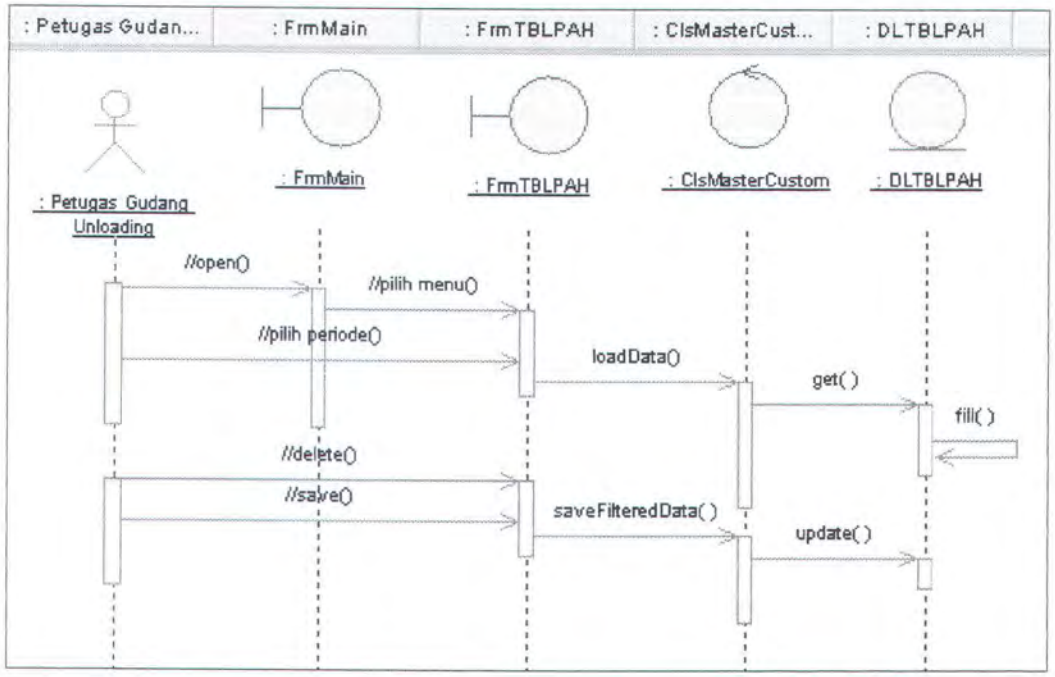
Gambar 3.7 Sequence diagram edit untuk use case penerimaan LPAH

saveFilteredData() pada ClsMasterCustom dan ClsMasterCustom akan memanggil prosedur update() pada DLTBLPAH.

- Delete

Flow delete pada *sequence diagram* ini ditujukan untuk mengetahui urutan interaksi *class* jika petugas gudang unloading menghapus transaksi penerimaan LPAH. Gambar 3.8 menjelaskan urutan interaksi obyek untuk menghapus transaksi penerimaan ayam hidup.

Pada *sequence diagram* tersebut setelah data tertampil pada FrmTBLPAH (lihat bagian view dari use case ini), user memilih transaksi yang akan dihapus dan menyimpannya setelah proses penghapusan



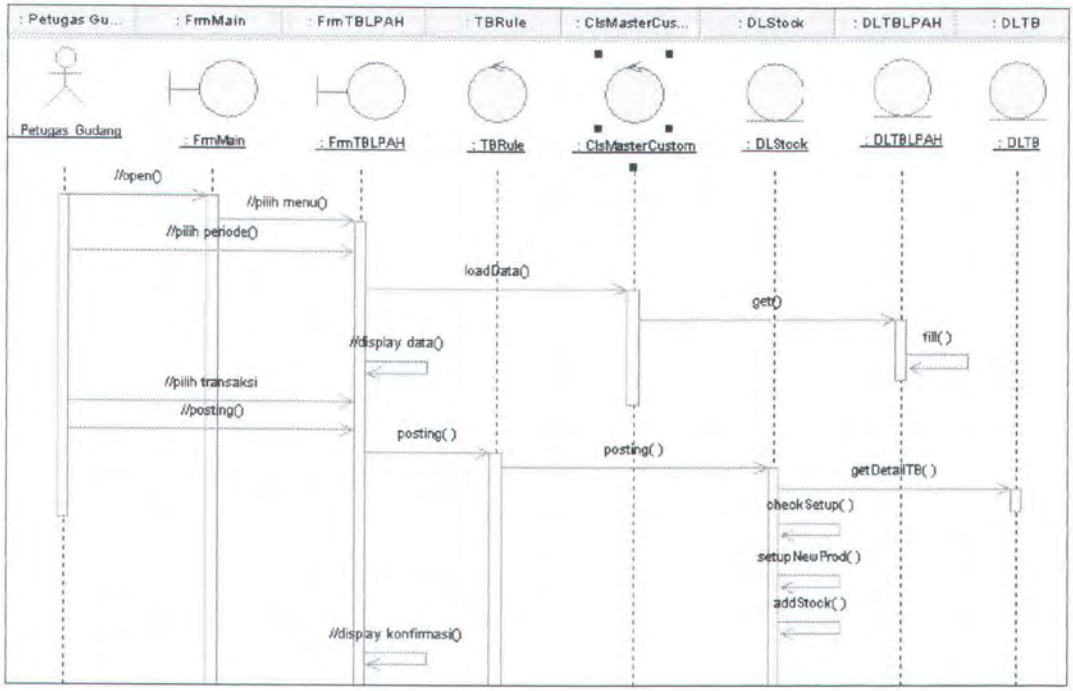
Gambar 3.8 Sequence diagram delete untuk use case penerimaan LPAH

dilakukan (untuk proses penyimpanan sama dengan proses penyimpanan pada bagian add atau edit dari use case ini)

- Posting

Flow posting pada *sequence diagram* ini ditujukan untuk mengetahui urutan interaksi *class* jika petugas gudang unloading memposting transaksi penerimaan LPAH. Gambar 3.9 menjelaskan urutan interaksi obyek untuk memposting transaksi penerimaan ayam hidup.

Dari *sequence diagram* tersebut, setelah data tertampil pada FrmTBLPAH (lihat bagian view) dan memilih transaksi yang akan diposting, maka FrmTBLPAH memanggil prosedur posting() pada TBRule dan TBRule akan memanggil prosedur posting() pada DLStock. DLStock akan mengambil semua detail dari transaksi dengan memanggil fungsi GetDetailTB() pada DLTB. Setelah mengambil detail transaksi

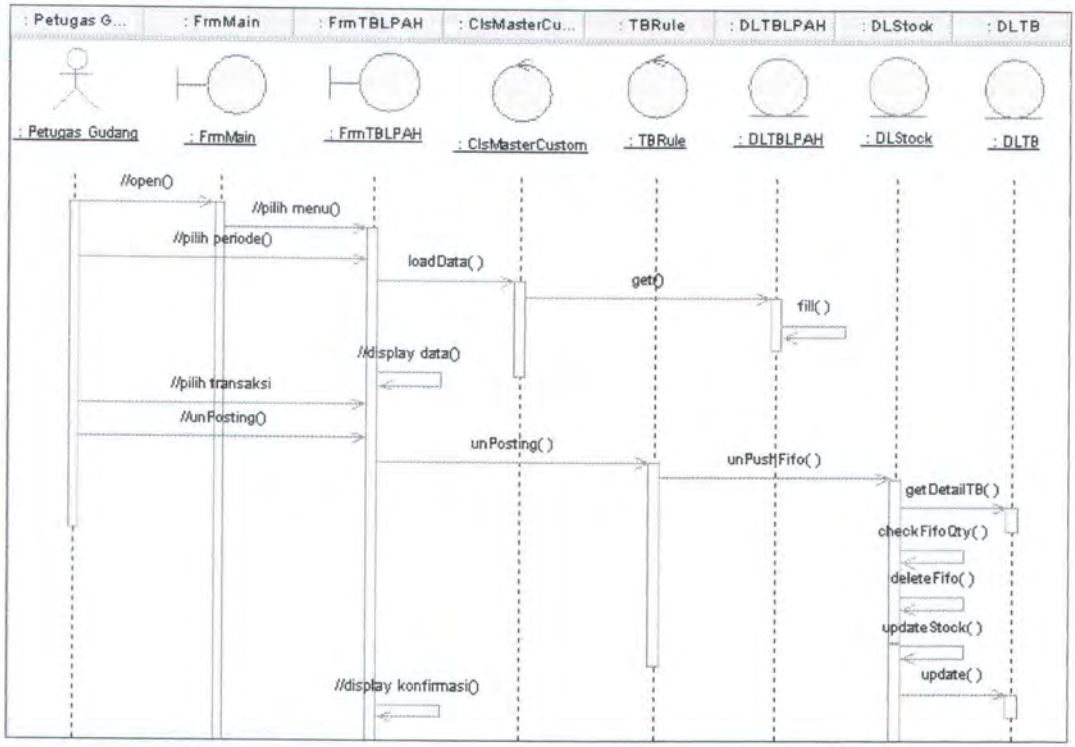


Gambar 3.9 Sequence diagram posting untuk use case penerimaan LPAH

maka DLStock akan memanggil fungsi CheckSetup() untuk mengecek apakah barang sudah di-setup, kemudian memanggil prosedur SetupNewProduk() jika barang belum di setup, kemudian memanggil prosedur addStock() untuk menambah stok. FrmTBLPAH akan memberikan konfirmasi apabila proses posting sudah dilakukan atau gagal dilakukan.

- UnPosting

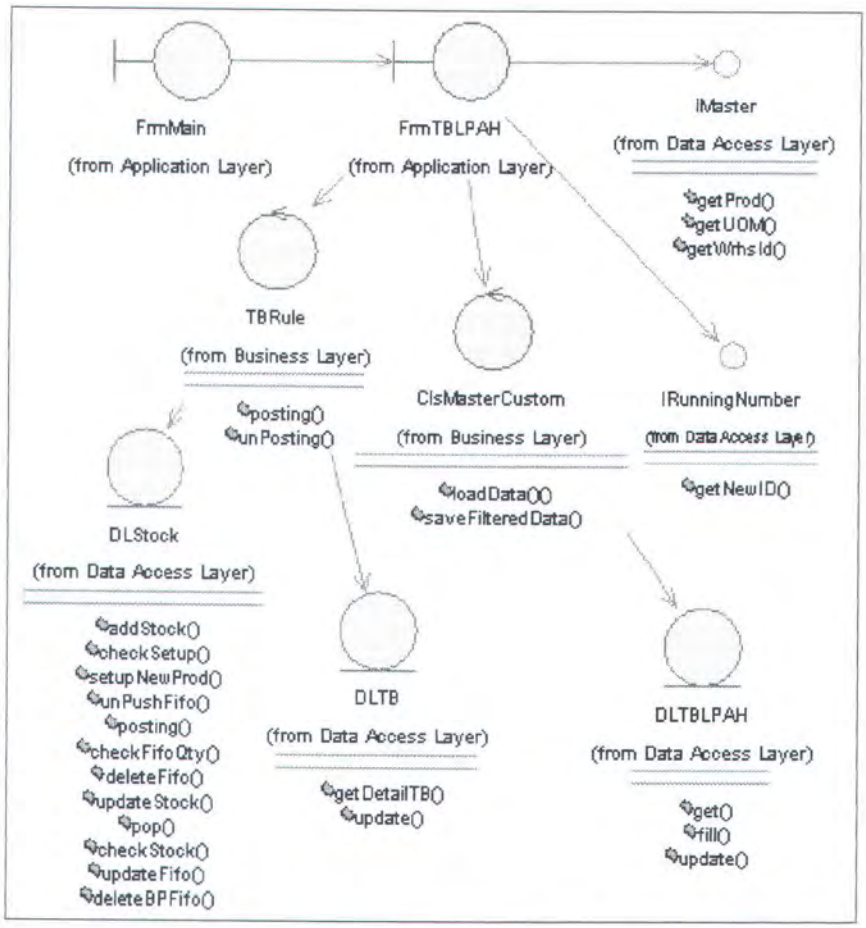
Flow edit pada *sequence diagram* ini ditujukan untuk mengetahui urutan interaksi *class* jika petugas gudang unloading menggagalkan posting transaksi penerimaan LPAH. Gambar 3.10 menjelaskan urutan interaksi obyek untuk menggagalkan proses posting transaksi penerimaan ayam hidup.



Gambar 3.10 Sequence diagram unPosting untuk use case penerimaan LPAH

Pada *sequence digram* tersebut, setelah user memilih transaksi yang akan di unPosting maka FrmTBLPAH akan memanggil prosedur unPosting() pada TBRule, TBRule akan memanggil prosedur UnPushFifo() pada DLStock, DLStock akan mengambil detail dari transaksi dengan memanggil GetDetailTB(). Secara berurut-turut DLStock akan menjalankan prosedur CheckFifoQty(), deleteFifo(), updateStock() dan memanggil prosedur update() pada DLTB. FrmTBLPAH akan menampilkan konfirmasi jika proses unPosting sudah selesai dilakukan.

Berdasar *sequence diagram* di atas, diperoleh VOPC dari use case penerimaan ayam hidup. VOPC dari use case penerimaan LPAH dapat dilihat pada gambar 3.11.



Gambar 3.11 VOPC use case penerimaan LPAH

Class yang terlibat dalam penerimaan ayam hidup adalah :

a. FrmMain : *class* ini merupakan layer aplikasi yang menjadi menu utama dari sistem inventori ini.

b. FrmTBLPAH : *class* ini merupakan layer aplikasi yang menampilkan form transaksi penerimaan ayam hidup.

c. ClsMasterCustom : *class* ini merupakan bisnis layer yang menjembatani antara layer aplikasi dan layer data akses

d. TBRule : *class* ini merupakan bisnis layer yang mengatur alur untuk menyimpan transaksi berdasar FIFO

e. DLStock : *class* ini merupakan data akses layer untuk mengambil dan menyimpan data stok dan FIFO

f. DLTB : *class* ini merupakan data akses layer untuk mengambil detail transaksi penerimaan.

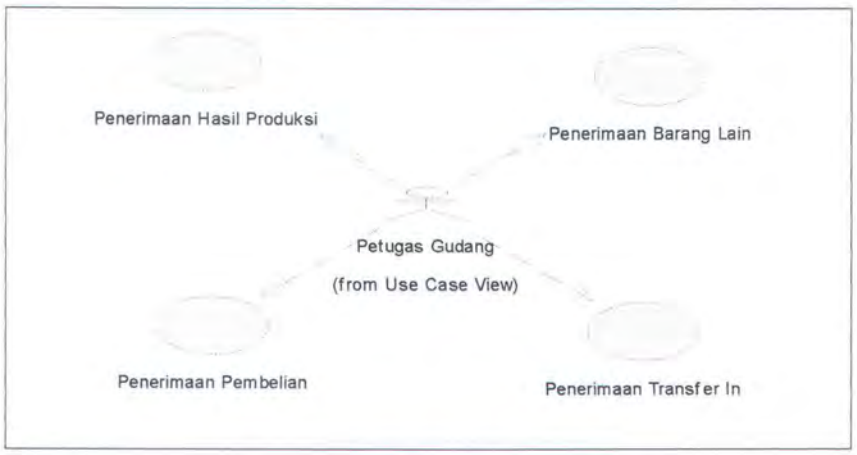
g. DLTBLPAH : *class* ini merupakan data akses untuk mengambil dan menyimpan data transaksi.

h. IMaster : *class* ini merupakan interface dengan subsistem Master

i. IRunningNumber : *class* ini merupakan interface dengan subsiste, RunningNumber.

2. Gudang Non Unloading

Yang termasuk gudang non unloading ini adalah semua gudang selain gudang unloading (untuk selanjutnya disebut gudang saja). Jenis barang yang diterima oleh gudang ini adalah semua barang, selain ayam hidup. Gambar 3.12 adalah use case untuk aktor petugas gudang non unloading (untuk selanjutnya disebut petugas gudang).



Gambar 3.12 Use case untuk petugas gudang

1. Menerima Pembelian

   Tujuan dari use case ini adalah mengelola seluruh transaksi penerimaan pembelian barang. Mengelola yang dimaksut disini mencakup melihat, menambah, mengubah, menghapus, memposting dan menggagalkan proses posting transaksi.

2. Menerima Hasil Produksi

   Tujuan dari use case ini adalah mengelola seluruh transaksi hasil produksi (dari gudang produksi). Mengelola yang dimaksut disini mencakup melihat, menambah, mengubah, menghapus, memposting dan menggagalkan proses posting transaksi.

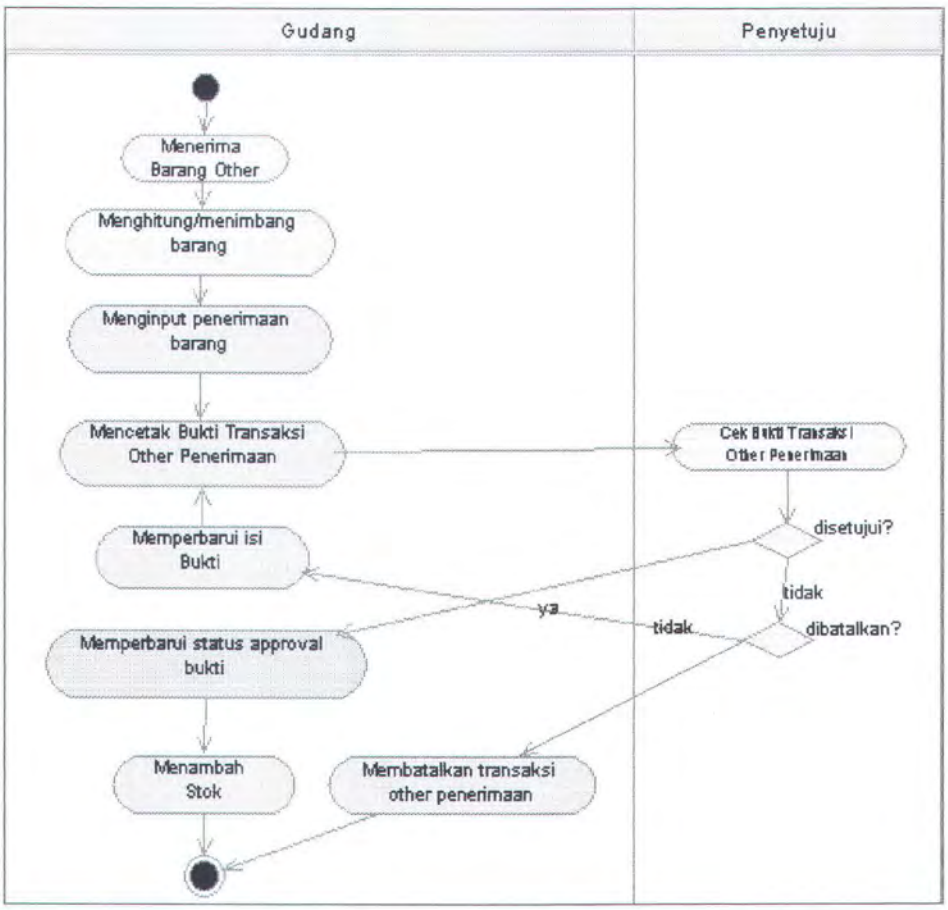
3. Menerima Barang Other (Penerimaan lain - lain)

   Tujuan dari use case ini adalah mengelola informasi transaksi penerimaan lain – lain. Mengelola yang dimaksut disini mencakup melihat, menambah, mengubah, menghapus, memposting dan menggagalkan proses posting transaksi.

4. Menerima Transfer In

   Tujuan dari use case ini adalah untuk mengelola transaksi penerimaan transfer barang dari gudang lain. Mengelola yang dimaksut disini mencakup melihat, menambah, mengubah, menghapus, memposting dan menggagalkan proses posting transaksi.

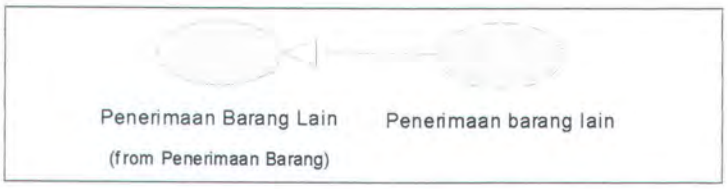
Berikut akan dijelaskan *activity diagram*, *sequence diagram* dan VOPC untuk use case penerimaan lain-lain. Gambar 3.13 menjelaskan aktivitas yang dilakukan oleh petugas gudang. Setelah petugas gudang menerima barang



Gambar 3.13 Activity diagram untuk petugas gudang

kemudian menimbang dan menghitung barang dan menambahkan transaksi penerimaan lain-lain ke sistem. Setelah mencetak bukti transaksi penerimaan lain dan menyerahkan pada penyetuju, jika transaksi yang dimasukkan disetujui, maka petugas dapat mengubah status transaksi dan menambah stok. Aktivitas ini dilakukan dalam sistem.

Dari *activity diagram* kemudian dibuat *use case realization* dan *sequence diagram*. Gambar 3.14 menjelaskan *use case realization* dari use case penerimaan lain-lain.

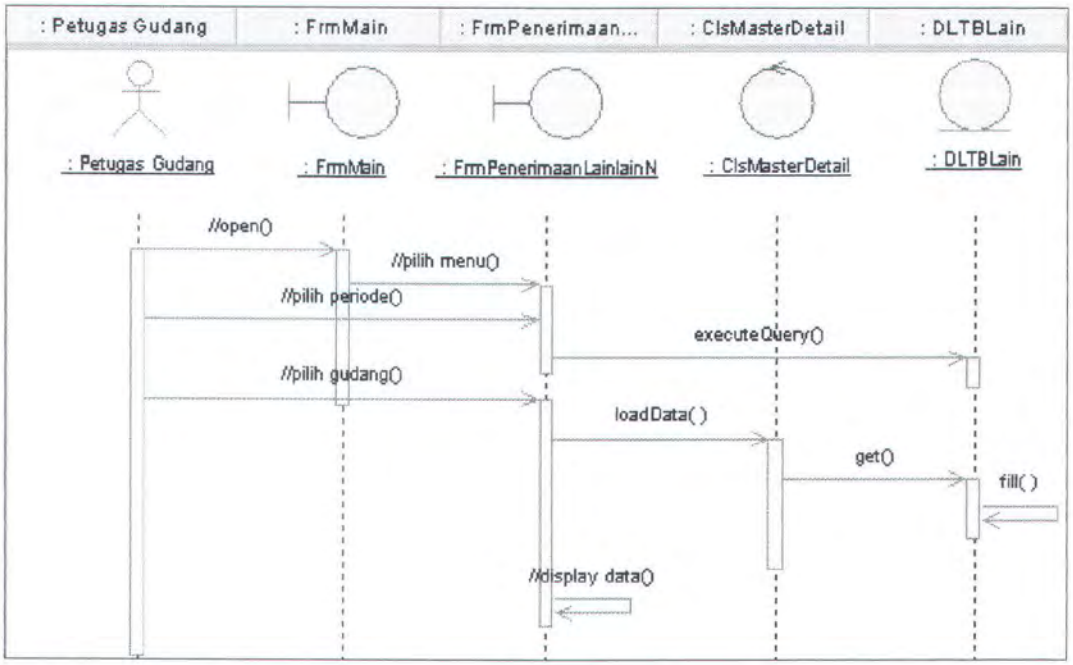


Gambar 3.14 Use case realization untuk use case penerimaan lain-lain

Sequence diagram untuk use case penerimaan lain-lain ini juga dibagi menjadi beberapa flow, yaitu :

- View

Basic flow ini menjelaskan urutan interaksi obyek untuk melihat transaksi penerimaan lain-lain. Gambar 3.15 menjelaskan urutan interaksi obyek untuk melihat transaksi penerimaan lain-lain.



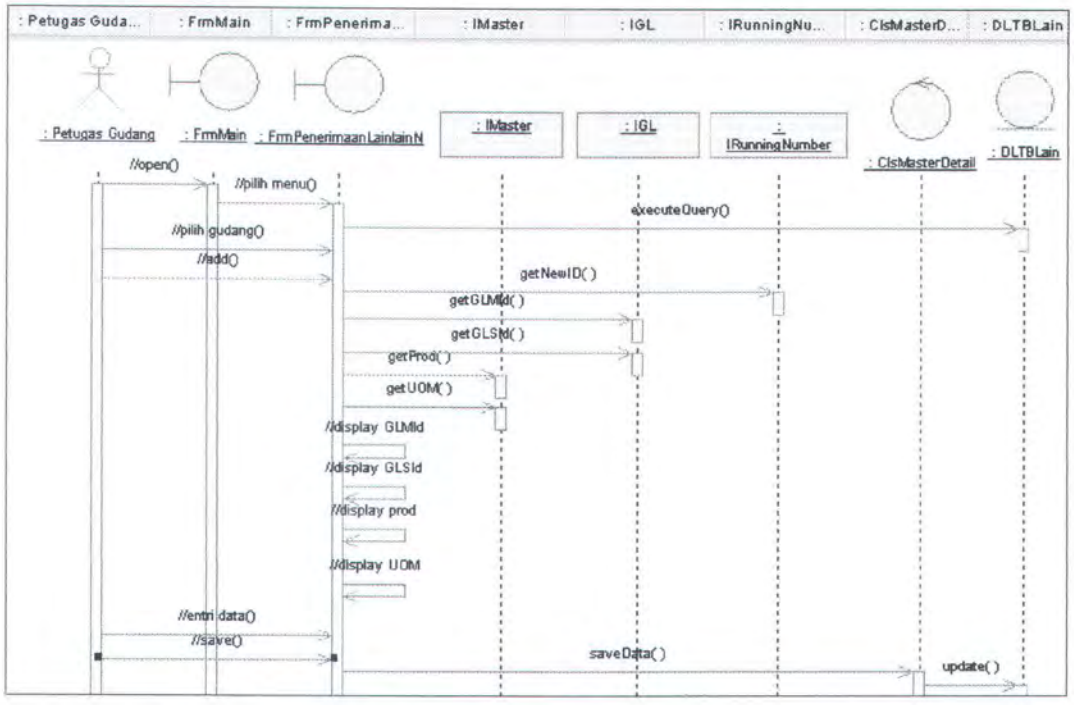
Gambar 3.15 Sequence diagram basic flow untuk use case penerimaan lain-lain

Pada *sequence diagram* tersebut, proses yang terjadi adalah aktor petugas gudang membuka FrmMain, dan memilih menu untuk transaksi penerimaan lain-lain (FrmPenerimaanLainLainN). Setelah user memilih

periode maka *class* FrmPenerimaanLainLainN akan memanggil prosedur loadData pada ClsMasterDetail dan ClsMasterDetail akan memanggil prosedur Get() pada DLTBLain yang berfungsi mengambil data pada basis data. DLTBLain akan mengisikan data ke dataset dengan prosedur Fill(). Dan FrmPenerimaanLainLainN akan menampilkan data.

- Add

Flow add pada *sequence diagram* ini ditujukan untuk mengetahui urutan interaksi *class* jika petugas gudang menambah transaksi penerimaan lain-lain. Gambar 3.16 menjelaskan urutan interaksi obyek untuk menambah transaksi penerimaan lain-lain.



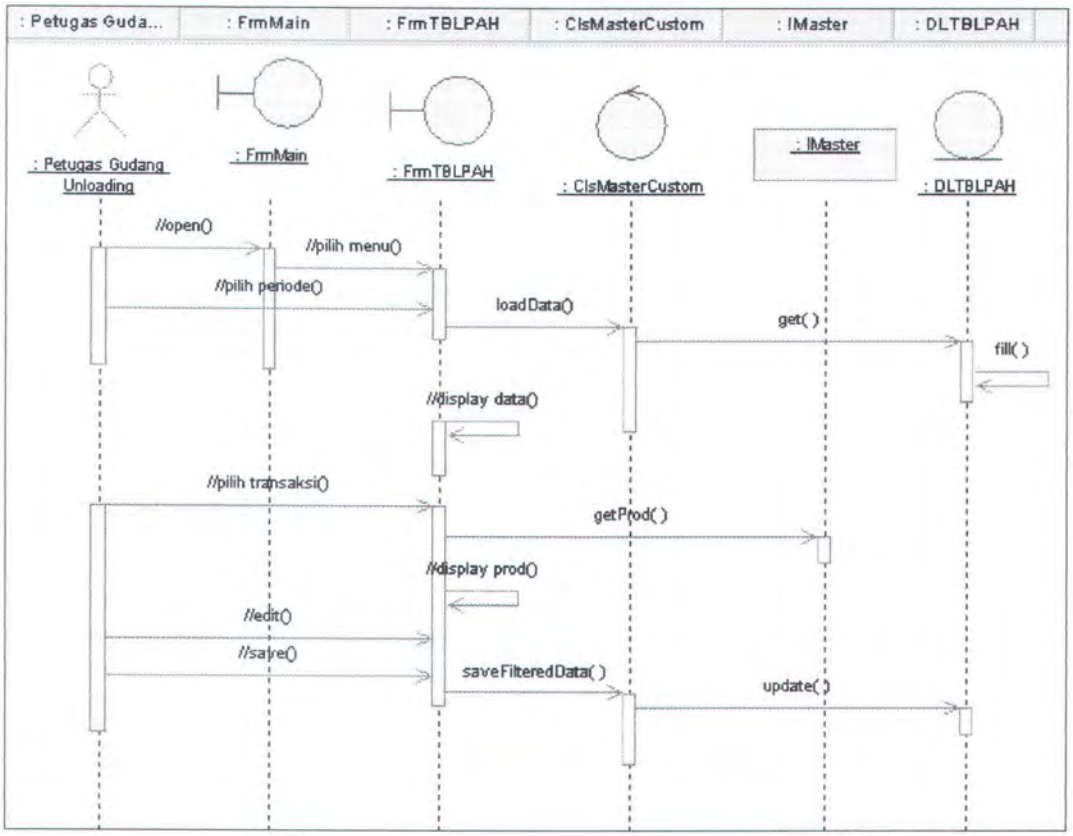
Gambar 3.16 Sequence diagram add untuk use case penerimaan lain-lain

Pada *sequence diagram* tersebut proses yang terjadi adalah user membuka form transaksi penerimaan lain-lain dan mengentrikan data. FrmPenerimaanLainLainN akan memanggil fungsi GetNewID() pada

subsistem RunningNumber untuk mendapatkan ID transaksi. FrmPenerimaanLainLainN akan mengambil GLMID, GLSID, Produck dan UOM dengan berturut-turut memanggil fungsi GetGLMID(), GetGLSID(), GetProduct() dan GetUOM() Untuk proses menyimpan data, FrmPenerimaanLainLainN akan memanggil prosedur saveData() pada ClsMasterDetail dan ClsMasterDetail akan memanggil prosedur update().

- Edit

Flow edit pada *sequence diagram* ini ditujukan untuk mengetahui urutan interaksi *class* jika petugas gudang mengubah transaksi penerimaan lain-lain. Gambar 3.17 menjelaskan urutan interaksi obyek untuk menambah transaksi penerimaan lain-lain.



Gambar 3.17 Sequence diagram edit untuk use case penerimaan lain-lain

Pada *sequence diagram* tersebut, setelah data ditampilkan (lihat bagian View dari use case ini) user memilih transaksi yang akan diedit datanya, kemudian melakukan pengubahan data. Untuk proses penyimpanan data sama dengan proses penyimpana data pada proses Add, yaitu FrmPenerimaanLainLainN memanggil prosedur saveData() pada ClsMasterDetail dan ClsMasterDetail akan memanggil prosedur update() pada DLTBLain.

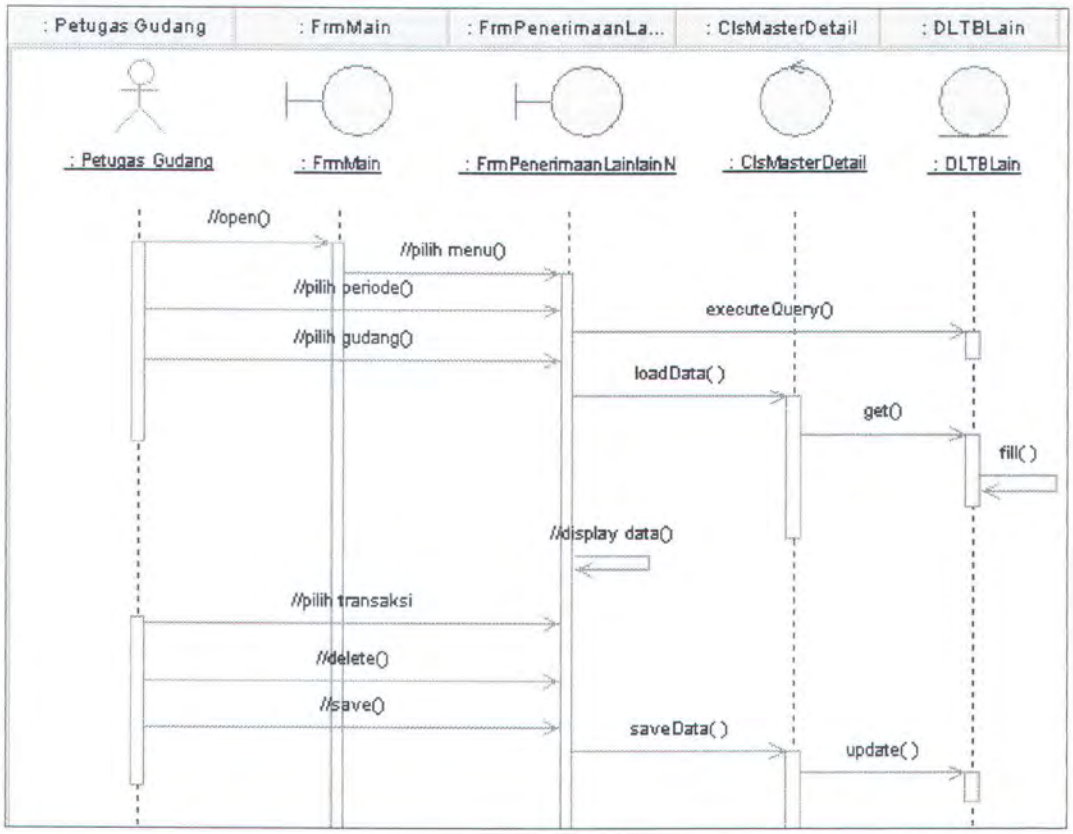
- Delete

Flow delete pada *sequence diagram* ini ditujukan untuk mengetahui urutan interaksi *class* jika petugas gudang menghapus transaksi penerimaan lain-lain. Gambar 3.18 menjelaskan urutan interaksi obyek untuk menghapus transaksi penerimaan lain-lain.

Pada *sequence diagram* tersebut setelah data tertampil pada FrmPenerimaanLainLainN (lihat bagian view dari use case ini), user memilih transaksi yang akan dihapus dan menyimpannya setelah proses penghapusan dilakukan (untuk proses penyimpanan sama dengan proses penyimpanan pada bagian add atau edit dari use case ini)

- Posting

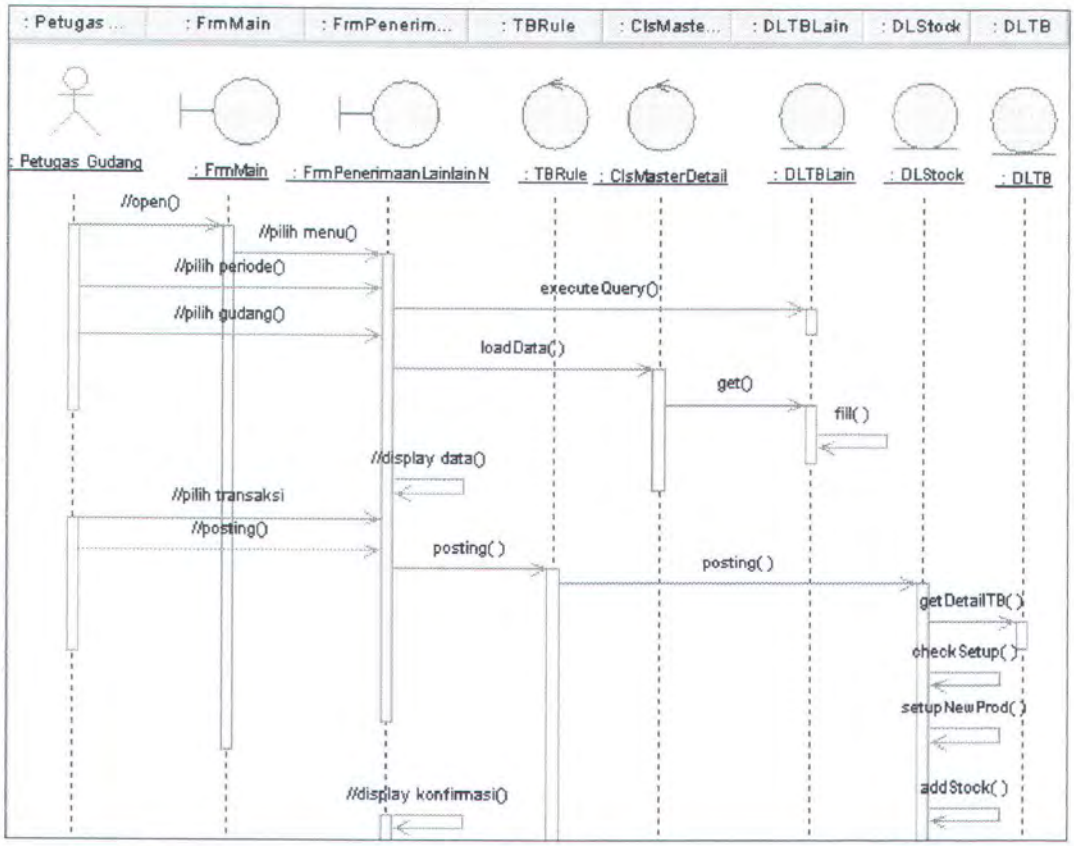
Flow edit pada *sequence diagram* ini ditujukan untuk mengetahui urutan interaksi *class* jika petugas gudang memposting transaksi penerimaan lain-lain. Gambar 3.19 menjelaskan urutan interaksi obyek untuk memposting transaksi penerimaan lain-lain.



Gambar 3.18 equence diagram delete untuk use case penerimaan lain-lain

Dari *sequence diagram* tersebut, setelah data tertampil pada FrmPenerimaanLainLainN (lihat bagian view) dan memilih transaksi yang akan diposting, maka FrmPenerimaanLainLainN memanggil prosedur posting() pada TBRule dan TBRule akan memanggil prosedur posting() pada DLStock. DLStock akan mengambil semua detail dari transaksi dengan memanggil fungsi GetDetailTB() pada DLTB. Setelah mengambil detail transaksi maka DLStock akan memanggil fungsi CheckSetup() untuk mengecek apakah barang sudah di-setup, kemudian memanggil prosedur SetupNewProduk() jika barang belum di setup, kemudian memanggil prosedur addStock() untuk menambah stok.

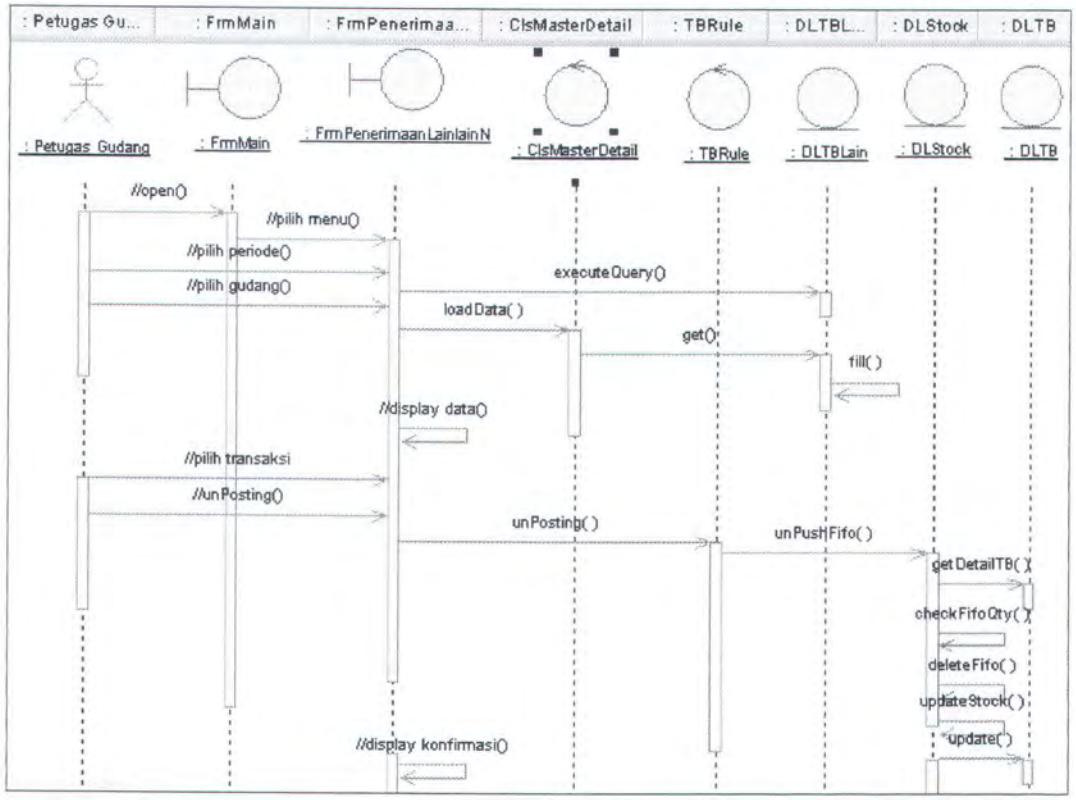
FrmPenerimaanLainLainN akan memberikan konfirmasi apabila proses posting sudah dilakukan atau gagal dilakukan.



Gambar 3.19 Sequence diagram posting untuk use case penerimaan LPAH

- UnPosting

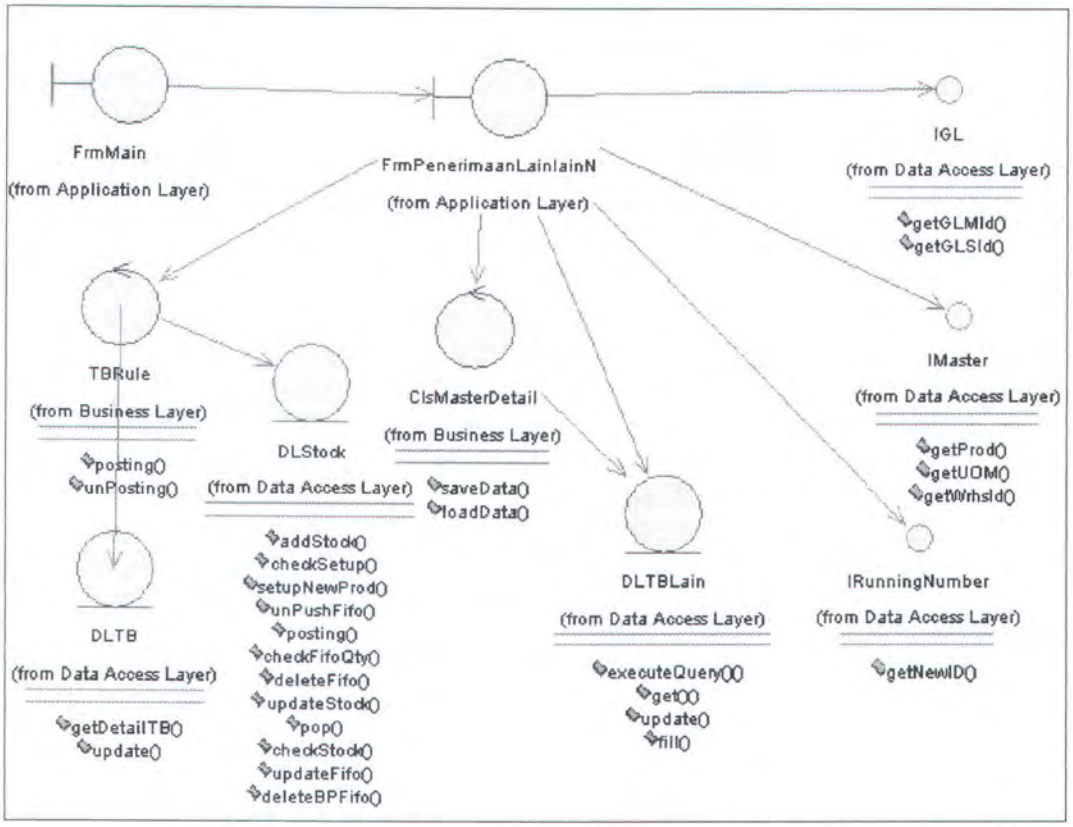
Flow edit pada sequence diagram ini ditujukan untuk mengetahui urutan interaksi *class* jika petugas gudang menggagalkan posting transaksi penerimaan lain-lain. Gambar 3.20 menjelaskan urutan interaksi obyek untuk menggagalkan proses posting transaksi penerimaan lain-lain.



Gambar 3.20 Sequence diagram unPosting untuk use case penerimaan lain-lain

Pada sequence digram tersebut, setelah user memilih transaksi yang akan di unPosting maka FrmPenerimaanLainLainN akan memanggil prosedur unPosting() pada TBRule, TBRule akan memanggil prosedur UnPushFifo() pada DLStock, DLStock akan mengambil detail dari transaksi dengan memanggil GetDetailTB(). Secara berurut-turut DLStock akan menjalankan prosedur CheckFifoQty(), deleteFifo(), updateStock() dan memanggil prosedur update() pada DLTB. FrmPenerimaanLainLainN akan menampilkan konfirmasi jika proses unPosting sudah selesai dilakukan.

Berdasar sequence diagram di atas, diperoleh VOPC dari use case penerimaan lain-lain. VOPC dari use case penerimaan lain-lain dapat dilihat pada gambar 3.21.



Gambar 3.21 VOPC untuk use case penerimaan lain-lain

Class yang terlibat dalam penerimaan lain-lain adalah :

a. FrmMain : *class* ini merupakan layer aplikasi yang menjadi menu utama dari sistem inventori ini.

b. FrmPenerimaanLainLainN : *class* ini merupakan layer aplikasi yang menampilkan form transaksi penerimaan lain-lain.

c. ClsMasterDetail : *class* ini merupakan bisnis layer yang menjembatani antara layer aplikasi dan layer data akses

d. TBRule : *class* ini merupakan bisnis layer yang mengatur alur untuk menyimpan transaksi berdasar FIFO

e. DLStock : *class* ini merupakan data akses layer untuk mengambil dan menyimpan data stok dan FIFO

f. DLTB : *class* ini merupakan data akses layer untuk mengambil detail transaksi penerimaan.

g. DLTBLain : *class* ini merupakan data akses untuk mengambil dan menyimpan data transaksi.

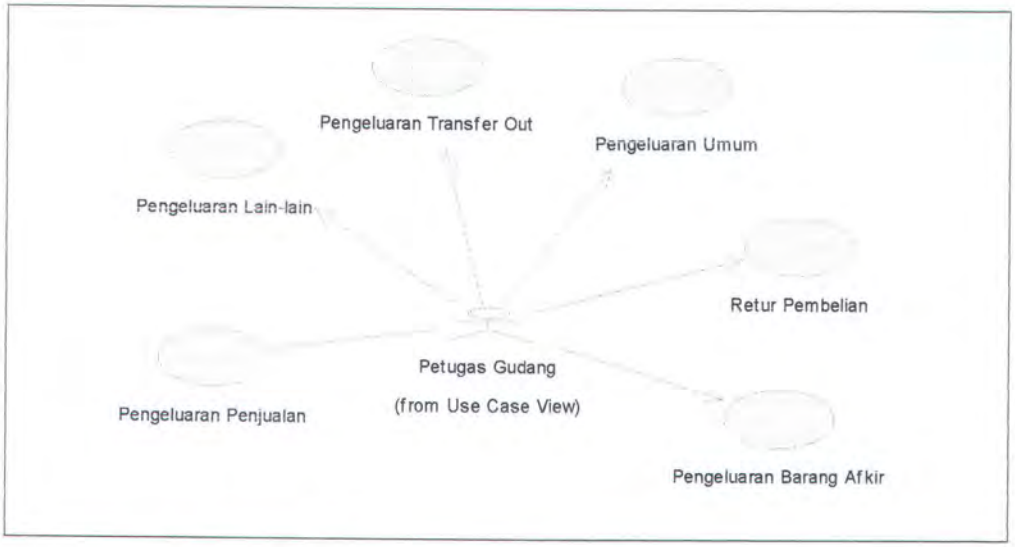
h. IMaster : *class* ini merupakan interface dengan subsistem Master

i. IGL : *class* ini merupakan interface dari subsistem GL

j. IRunningNumber : *class* ini merupakan interface dengan subsiste, RunningNumber.

3.1.2 Pengeluaran Barang

Bagian ini akan menjelaskan proses untuk pengeluaran barang. Terdapat 2 aktor yang berhubungan langsung dengan proses pengeluaran barang ini, yaitu aktor petugas gudang unloading dan aktor petugas gudang. Untuk petugas gudang mengeluarkan semua jenis barang kecuali ayam mati, yang hanya dilakukan oleh aktor petugas unloading. Berikut penjelasan masing – masing use case untuk aktor petugas gudang. Notasi UML dapat dilihat pada gambar 3.22.



Gambar 3.22 Use case diagram pengeluaran barang

1. Pengeluaran Penjualan

   Tujuan dari use case ini adalah mengelola transaksi pengeluaran penjualan. Mengelola yang dimaksut disini mencakup melihat, menambah, mengubah, menghapus, memposting dan menggagalkan proses posting transaksi.

2. Pengeluaran Barang Other (Pengeluaran lain – lain)

   Tujuan dari use case ini adalah mengelola transaksi pengeluaran lain – lain. Mengelola yang dimaksut disini mencakup melihat, menambah, mengubah, menghapus, memposting dan menggagalkan proses posting transaksi.

3. Pengeluaran Produk Afkir

   Tujuan dari use case ini adalah mengelola yang berhubungan dengan proses pemusnahan stock afkir di gudang. Mengelola yang dimaksut disini mencakup melihat, menambah, mengubah, menghapus, memposting dan menggagalkan proses posting transaksi.

4. Pengeluaran Transfer Out

   Tujuan dari use case ini adalah untuk mengelola transaksi pengeluaran transfer barang ke gudang lain. Mengelola yang dimaksut disini mencakup melihat, menambah, mengubah, menghapus, memposting dan menggagalkan proses posting transaksi.

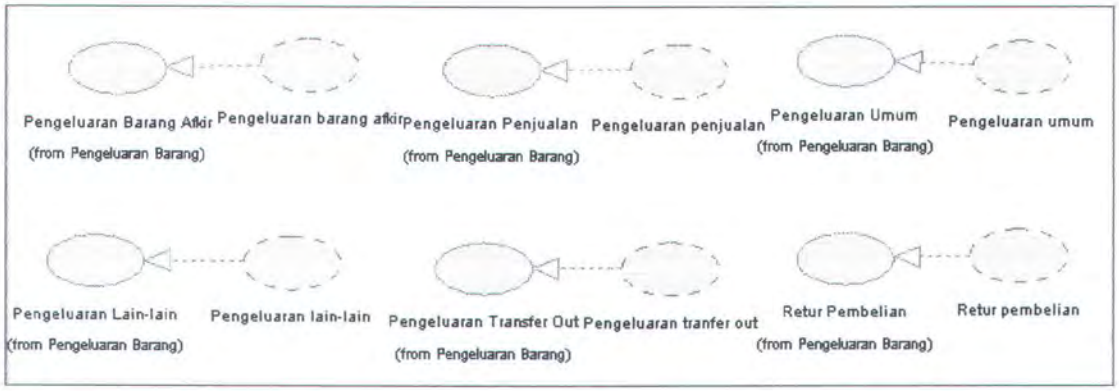
5. Pengeluaran Umum

   Tujuan dari use case ini adalah mengelola transaksi pengeluaran umum. Mengelola yang dimaksut disini mencakup melihat, menambah, mengubah, menghapus, memposting dan menggagalkan proses posting transaksi.

6. Retur Pembelian

   Tujuan dari use case ini adalah mengelola transaksi pengeluaran umum. Mengelola yang dimaksut disini mencakup melihat, menambah, mengubah, menghapus, memposting dan menggagalkan proses posting transaksi.

Dari use case diagram kemudian dibuat use case realization. Gambar 3.23 menjelaskan realization untuk semua use case dalam modul pengeluaran barang.



Gambar 3.23 Use case realization untuk modul pengeluaran barang

Berikut ini akan dijelaskan *activity diagram*, sequence diagram dan VOPC untuk use case pengeluaran lain-lain. Use case-use case yang lain memiliki kesamaan dengan sequence diagramnya, yang berbeda hanya *class* yang terlibat, yaitu boundary (FrmKeluarLain2) dan entity (DLBPLain), yang namanya disesuaikan dengan nama masing-masing use case.

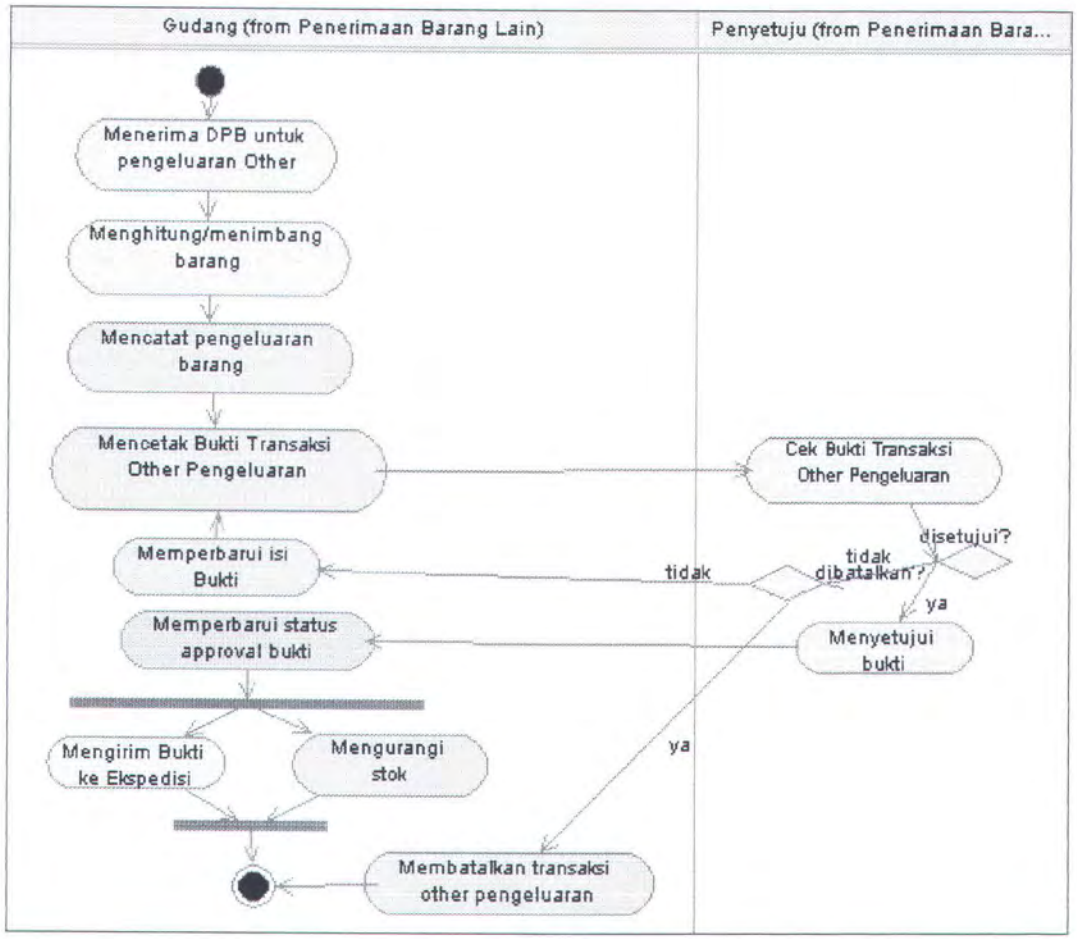
Gambar 3.24 menjelaskan aktivitas yang dilakukan oleh petugas gudang untuk proses pengeluaran barang lain - lain. Setelah petugas gudang DPB (Daftar Permintaan Barang) kemudian petugas menghitung barang yang akan dikeluarkan dan membuat transaksi pengeluaran lain-lain ke sistem. Setelah mencetak bukti transaksi pengeluaran lain dan menyerahkan pada penyetuju, jika transaksi yang dimasukkan disetujui, maka petugas dapat mengubah status transaksi dan mengurangi stock. Aktivitas ini dilakukan dalam sistem.

Dari *activity diagram* kemudian dibuat use case realization dan sequence diagram. Gambar 3.25 menjelaskan use case realization dari use case pengeluaran lain-lain.

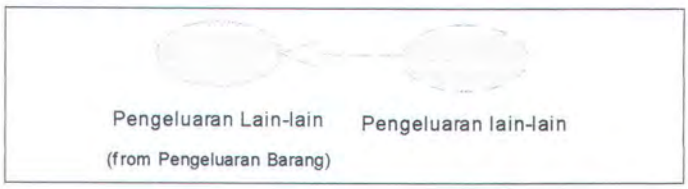
Sequence diagram untuk use case penerimaan lain-lain ini juga dibagi menjadi beberapa flow, yaitu :

- View

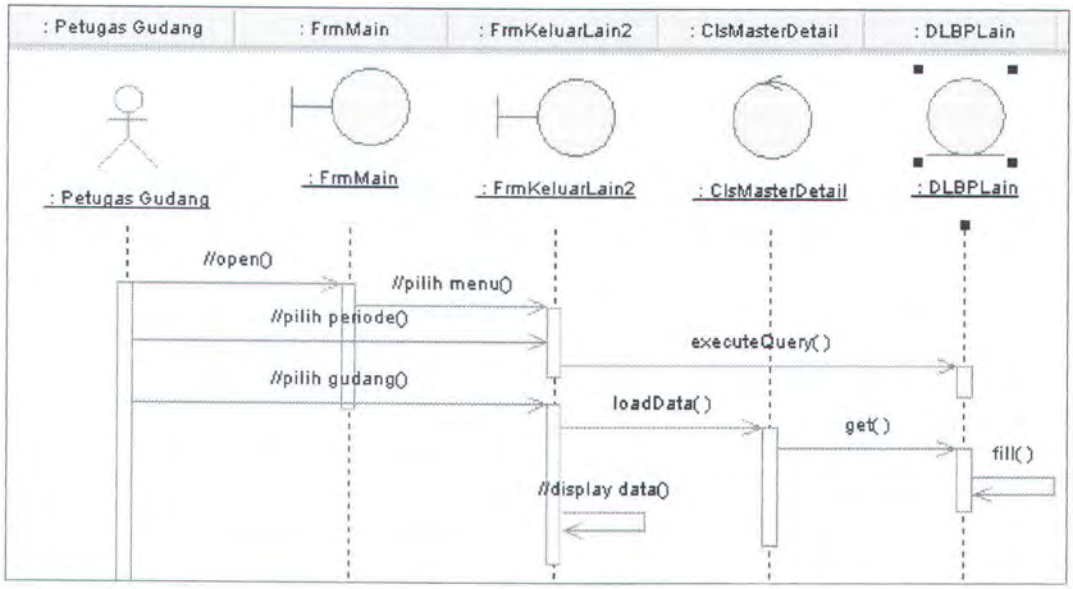
  Flow view ini menjelaskan urutan interaksi obyek untuk melihat transaksi pengeluaran lain-lain. Gambar 3.26 menjelaskan urutan interaksi obyek untuk melihat transaksi pengeluaran lain-lain.



Gambar 3.24 Activity diagram untuk use case pengeluaran lain-lain



Gambar 3.25 Use case realization untuk use case pengeluaran barang lain

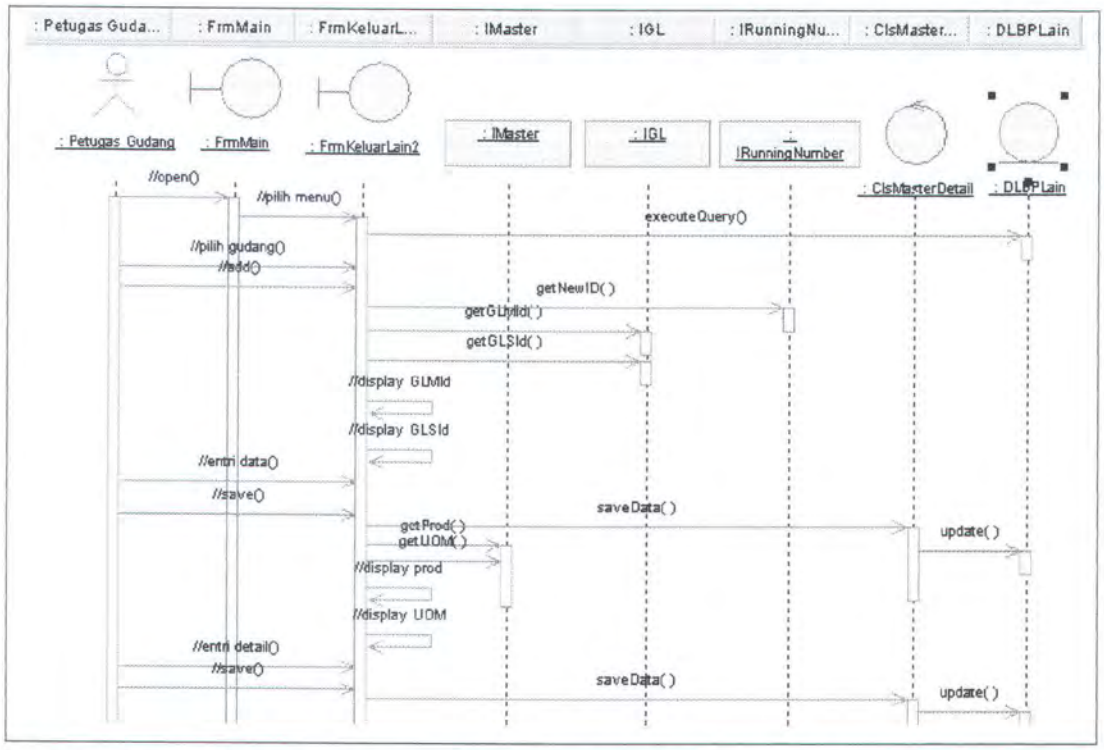


Gambar 3.26 Sequence diagram basic flow untuk use case pengeluaran lain-lain

Pada *sequence diagram* tersebut, proses yang terjadi adalah aktor petugas gudang membuka FrmMain, dan memilih menu untuk transaksi pengeluaran lain-lain (FrmKeluarLain2). Setelah user memilih periode maka *class* FrmKeluarLain2 akan memanggil prosedur loadData pada ClsMasterDetail dan ClsMasterDetail akan memanggil prosedur Get() pada DLBPLain yang berfungsi mengambil data pada basis data. DLBPLain akan mengisikan data ke dataset dengan prosedur Fill(). Dan FrmKeluarLain2 akan menampilkan data.

- Add

Flow add pada *sequence diagram* ini ditujukan untuk mengetahui urutan interaksi *class* jika petugas gudang menambah transaksi pengeluaran lain-lain. Gambar 3.27 menjelaskan urutan interaksi obyek untuk menambah transaksi pengeluaran lain-lain.

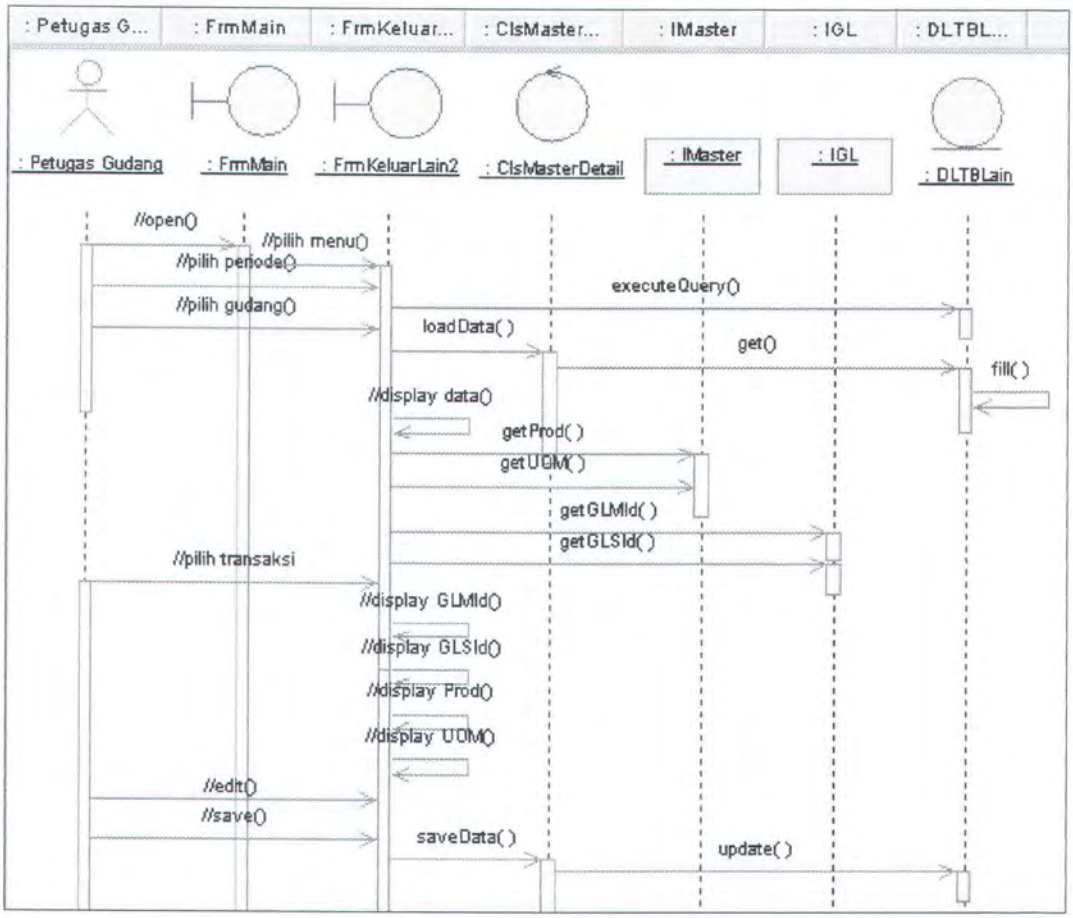


Gambar 3.27 Sequence diagram add untuk use case pengeluaran lain-lain

Pada *sequence diagram* tersebut proses yang terjadi adalah user membuka form transaksi penerimaan lain-lain dan mengentrikan data. FrmKeluarLain2 akan memanggil fungsi GetNewID() pada subsistem RunningNumber untuk mendapatkan ID transaksi. FrmKeluarLain2 akan mengambil GLMID, GLSID, Produck dan UOM dengan berturut-turut memanggil fungsi GetGLMID(), GetGLSID(), GetProduct() dan GetUOM() Untuk proses menyimpan data, FrmKeluarLain2 akan memanggil prosedur saveData() pada ClsMasterDetail dan ClsMasterDetail akan memanggil prosedur update().

- Edit

Flow edit pada *sequence diagram* ini ditujukan untuk mengetahui urutan interaksi *class* jika petugas gudang mengubah transaksi pengeluaran lain-lain.



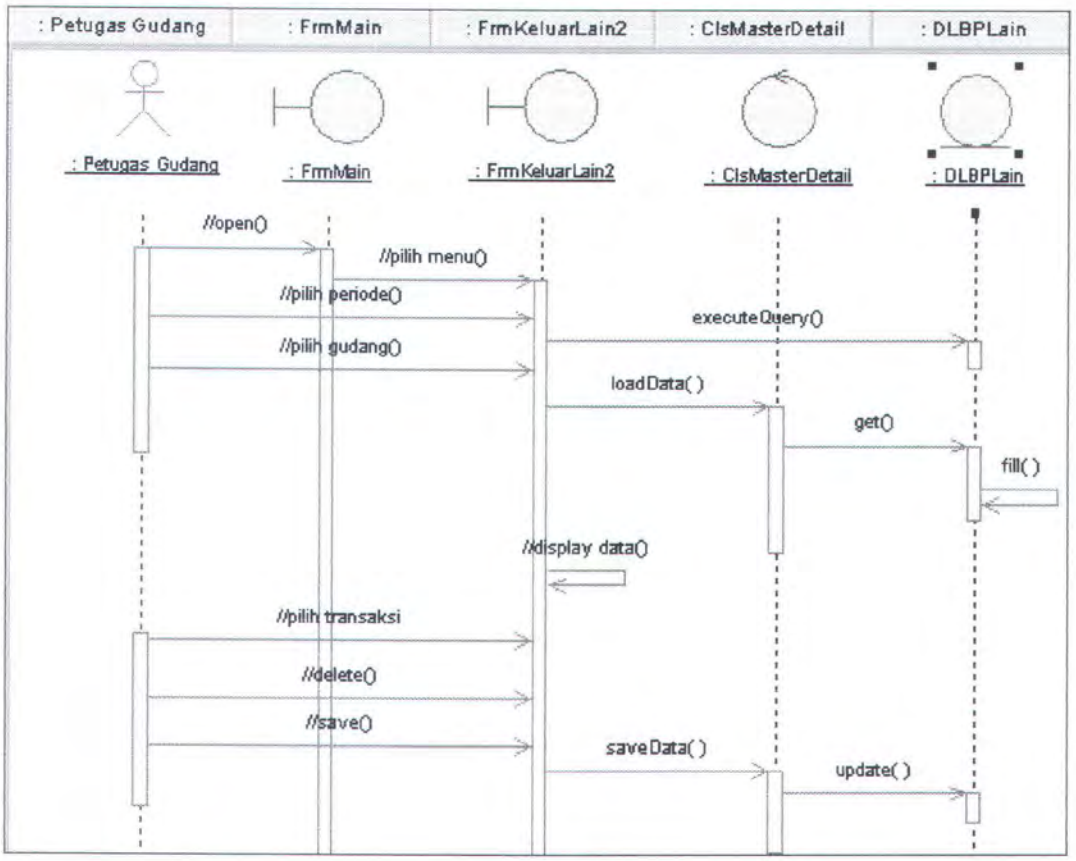
Gambar 3.28 Sequence diagram edit untuk use case pengeluaran lain-lain

Pada *sequence diagram* tersebut, setelah data ditampilkan (lihat bagian View dari use case ini) user memilih transaksi yang akan diedit datanya, kemudian melakukan pengubahan data. Untuk proses penyimpanan data sama dengan proses penyimpana data pada proses Add, yaitu FrmKeluarLain2 memanggil prosedur saveData() pada ClsMasterDetail dan ClsMasterDetail akan memanggil prosedur update() pada DLTBLain.

- Delete

Flow delete pada *sequence diagram* ini ditujukan untuk mengetahui urutan interaksi *class* jika petugas gudang menghapus transaksi pengeluaran lain-lain.

Gambar 3.29 menjelaskan urutan interaksi obyek untuk menghapus transaksi pengeluaran lain-lain.



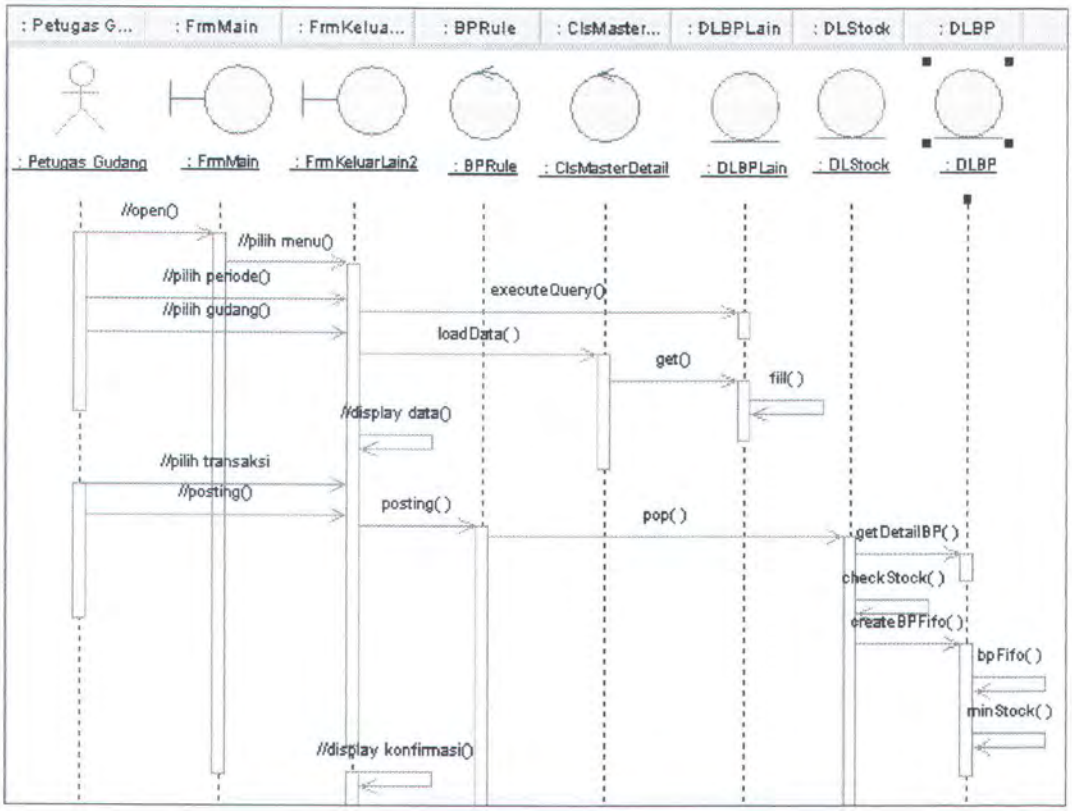
Gambar 3.29 Sequence diagram delete untuk use case pengeluaran lain-lain

Pada *sequence diagram* tersebut setelah data tertampil pada FrmKeluarLain2 (lihat bagian view dari use case ini), user memilih transaksi yang akan dihapus dan menyimpannya setelah proses penghapusan dilakukan (untuk proses penyimpanan sama dengan proses penyimpanan pada bagian add atau edit dari use case ini)

- Posting

Flow edit pada *sequence diagram* ini ditujukan untuk mengetahui urutan interaksi *class* jika petugas gudang memposting transaksi pengeluaran lain-

lain. Gambar 3.30 menjelaskan urutan interaksi obyek untuk memposting transaksi pengeluaran lain-lain.

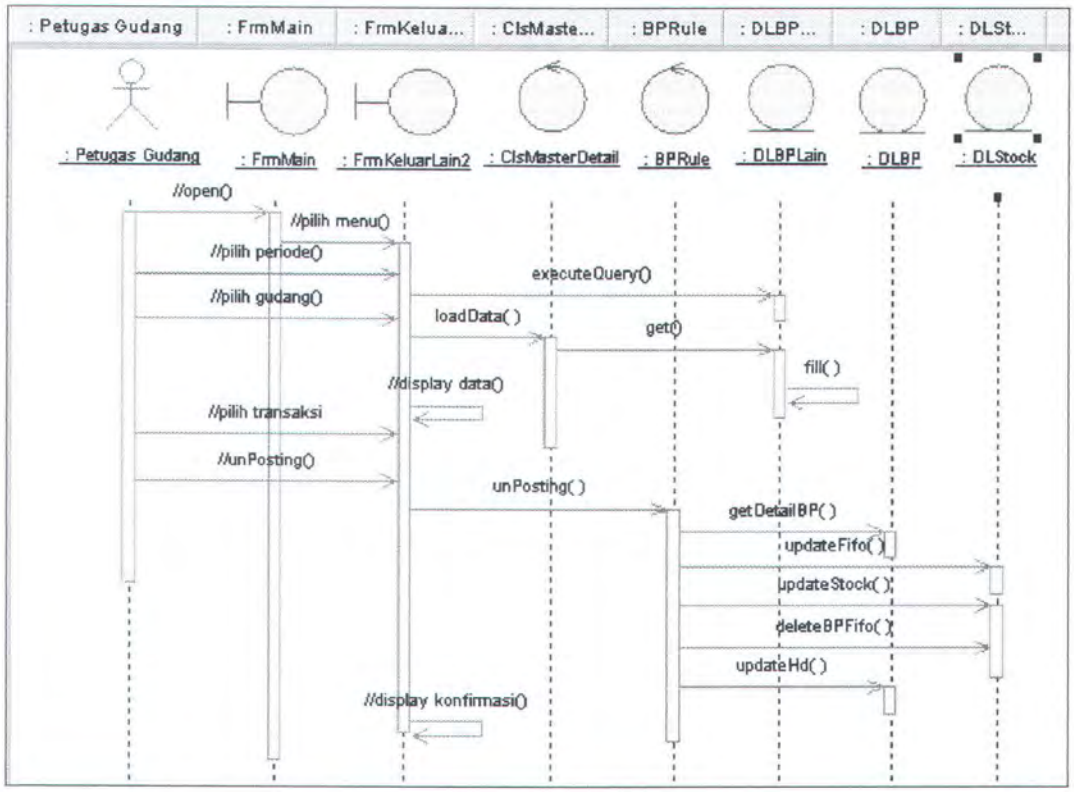


Gambar 3.30 Sequence diagram posting untuk use case pengeluaran lain-lain

Dari *sequence diagram* tersebut, setelah data tertampil pada FrmKeluarLain2 (lihat bagian view) dan memilih transaksi yang akan diposting, maka FrmKeluarLain2 memanggil prosedur posting() pada BPRule dan BPRule akan memanggil prosedur pop() pada DLStock. DLStock akan mengambil semua detail dari transaksi dengan memanggil fungsi GetDetailBP() pada DLBP. Setelah mengambil detail transaksi maka DLStock akan memanggil fungsi checkStock() untuk mengecek stok barang, kemudian memanggil prosedur createBPFifo() di DLBP. Dan DLStock berturut-turut memanggil prosedur bpFifo() dan minStock() untuk mengurungi stok..

- UnPosting

Flow edit pada *sequence diagram* ini ditujukan untuk mengetahui urutan interaksi *class* jika petugas gudang menggagalkan posting transaksi pengeluaran lain-lain. Gambar 3.31 menjelaskan urutan interaksi obyek untuk menggagalkan proses posting transaksi pengeluaran lain-lain.



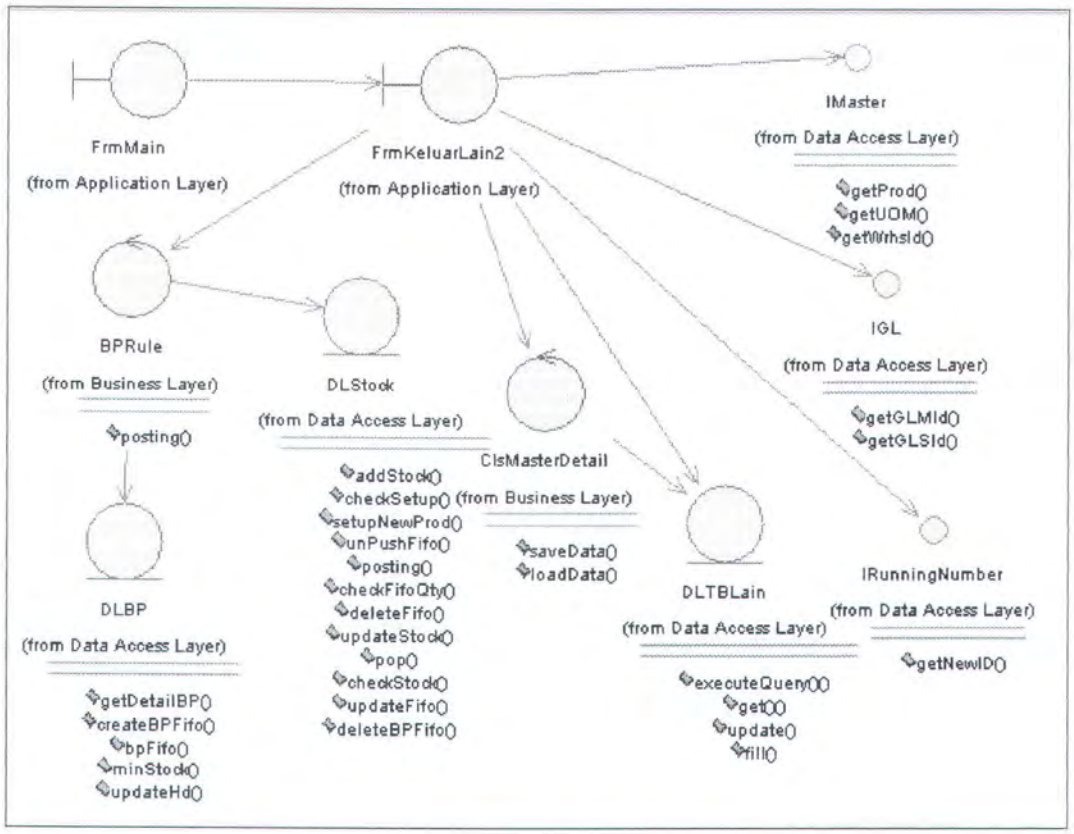
Gambar 3.31 Sequence diagram unPosting untuk use case pengeluaran lain-lain

Pada *sequence digram* tersebut, setelah user memilih transaksi yang akan di unPosting maka FrmKeluarLain2 akan memanggil prosedur unPosting() pada BPRule, BPRule akan mengambil detail dari transaksi dengan memanggil GetDetailTB() pada DLBP(). BPRule berturut-turut akan memanggil prosedur updateFifo(), updateStock(), deleteBPFifo() pada

DLStock dan memanggil updateHd() pada DLBP. FrmKeluarLain2 akan menampilkan konfirmasi jika proses unPosting sudah selesai dilakukan.

Berdasar *sequence diagram* di atas, diperoleh VOPC dari use case pengeluaran lain-lain. VOPC dari use case pengeluaran lain-lain dapat dilihat pada gambar 3.32. *Class* yang terlibat dalam pengeluaran lain-lain adalah :

a. FrmMain : *class* ini merupakan layer aplikasi yang menjadi menu utama dari sistem inventori ini.
b. FrmKeluarLain2 : *class* ini merupakan layer aplikasi yang menampilkan form transaksi pengeluaran lain-lain.
c. ClsMasterDetail : *class* ini merupakan bisnis layer yang menjembatani antara layer aplikasi dan layer data akses
d. BPRule : *class* ini merupakan bisnis layer yang mengatur alur untuk menyimpan transaksi berdasar FIFO
e. DLStock : *class* ini merupakan data akses layer untuk mengambil dan menyimpan data stok dan FIFO
f. DLBP : *class* ini merupakan data akses layer untuk mengambil detail transaksi pengeluaran.
g. DLBPLain : *class* ini merupakan data akses untuk mengambil dan menyimpan data transaksi.
h. IMaster : *class* ini merupakan interface dengan subsistem Master
i. IGL : *class* ini merupakan interface dari subsistem GL
j. IRunningNumber : *class* ini merupakan interface dengan subsistem RunningNumber.



Gambar 3.32 VOPC dari use case pengeluaran lain-lain

3.1.3 Monitoring Stok

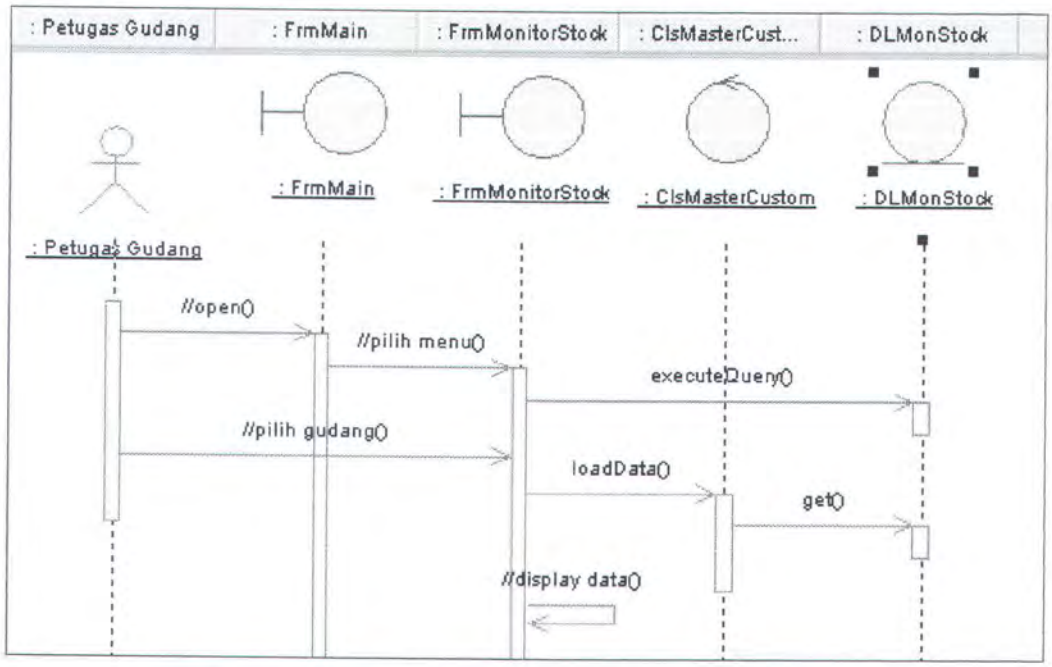
Berikut ini akan dijelaskan proses untuk memonitor stok, yaitu melihat posisi stok akhir dari barang tertentu dan untuk gudang tertentu beserta rincian detail jumlah pemasukan barang berdasar FIFO. Use case untuk proses ini dapat dilihat pada gambar 3.33.

Gambar 3.33 Use case untuk monitoring stok

Namun pada bagian ini akan dijelaskan use case monitor stok saja, untuk use case cetak kartu stok dan cetak posisi persediaan barang akan dijelaskan pada modul report pada bagian lain dari bab ini. Dari use case diagram dibuat use case realization. Gambar 3.34 menunjukkan use case realization dari use case monitor stok.

Gambar 3.34 Use case realization untuk use case monitor stok

Dari use case realization kemudian dibuat sequence diagram. Gambar 3.35 menunjukkan sequence diagram dari use case monitor stok.

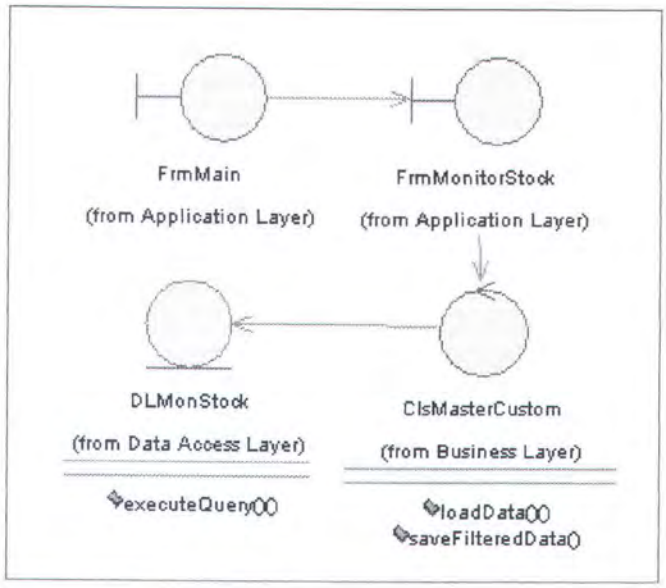


Gambar 3.35 Sequence diagram untuk use case monitor stock.

Pada *sequence diagram* tersebut, proses yang terjadi adalah aktor petugas gudang membuka FrmMain, dan memilih menu untuk monitor stok (FrmMonitorStock). Setelah user memilih gudang maka *class* FrmMonitorStock akan memanggil prosedur loadData() pada ClsMasterCustom dan ClsMasterCustom akan memanggil prosedur Get() pada DLMonStock yang berfungsi mengambil data pada basis data. DLMonStock akan mengisikan data ke dataset dengan prosedur fill(). Dan FrmMonitorStock akan menampilkan data.

Dari *sequence diagram* kemudian dibuat VOPC untuk use case ini. Gambar 3.36 menjelaskan VOPC dari use case monitor stok. *Class* yang terlibat dalam pengeluaran lain-lain adalah :

a. FrmMain : *class* ini merupakan layer aplikasi yang menjadi menu utama dari sistem inventori ini.



Gambar 3.36 VOPC dari use case monitor stok

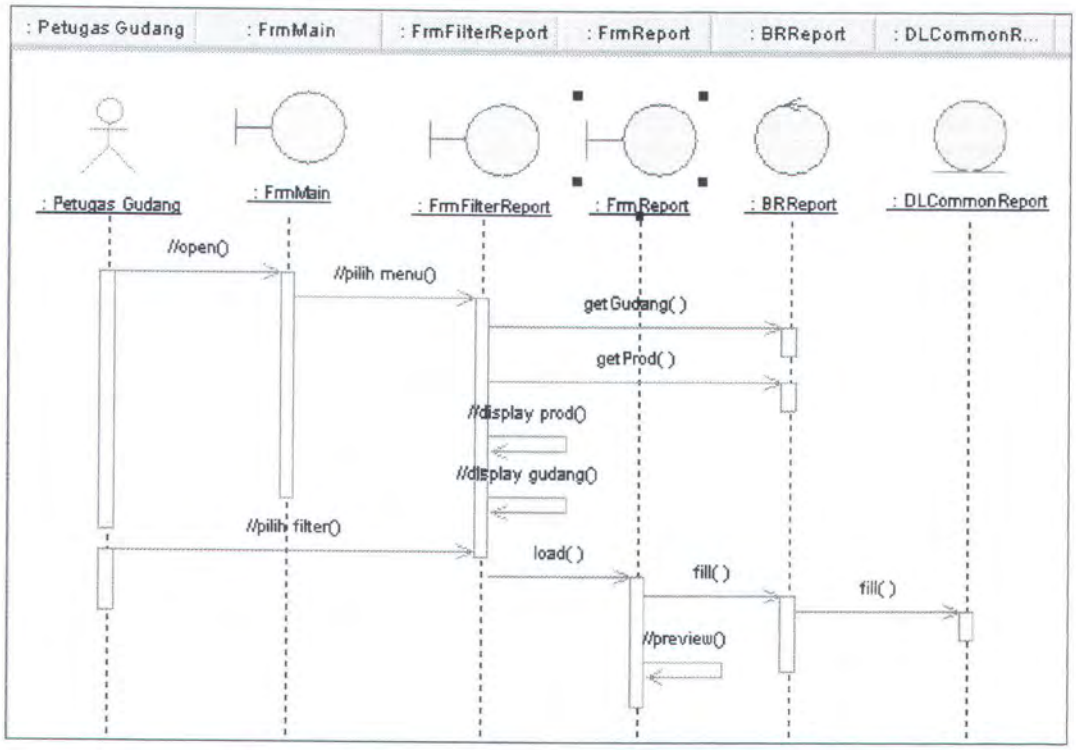
b. FrmMonitorStock : *class* ini merupakan layer aplikasi yang menampilkan form monitor stok..

c. ClsMasterCustom : *class* ini merupakan bisnis layer yang menjembatani antara layer aplikasi dan layer data akses

d. DLMonStock : *class* ini merupakan data akses untuk mengambil data

3.1.4 Laporan

Modul ini akan menjelaskan proses untuk mencetak report. Gambar 3.37 menjelaskan sequence diagram untuk mencetak repot. Pada *sequence diagram* tersebut, user membuka FrmMain dan memilih menu untuk mencetak laporan (FrmFilterReport). Setelah user memilih filter FrmFilterReport akan memanggil prosedur load() dari FrmReport. FrmReport memanggil prosedur fill() pada BRReport dan BRReport memanggil prosedur fill() pada DLCommonReport.



Gambar 3.37 Sequence diagram untuk mencetak laporan

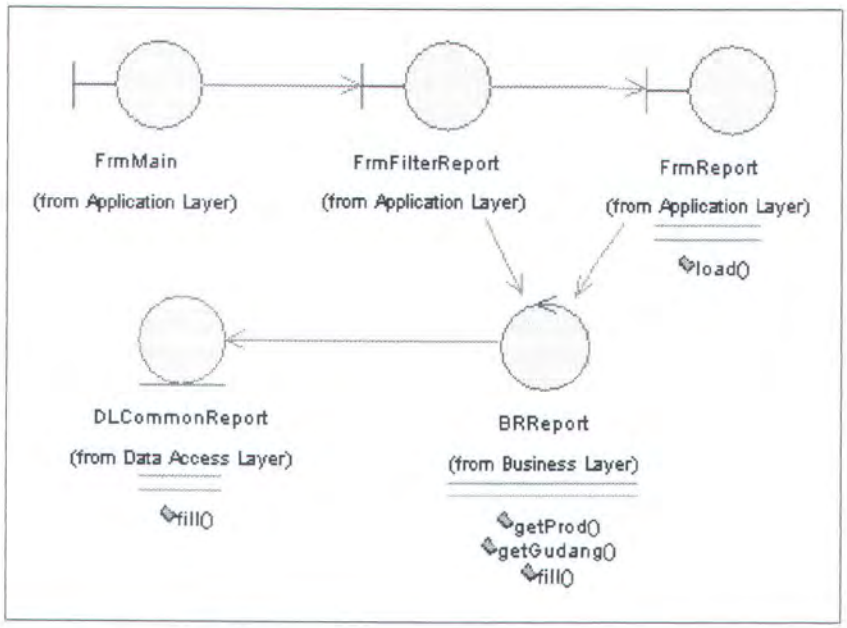
Dari sequence diagram, kemudian dibuat VOPC. Gambar 3.38 menjelaskan VOPC dari use case mencetak report. *Class* yang terlibat dalam mencetak laporan adalah :

a. FrmMain : *class* ini merupakan layer aplikasi yang menjadi menu utama dari sistem inventori ini.

b. FrmFilterReport : *class* ini merupakan layer aplikasi sebagai filter laporan yang akan dicetak

c. FrmReport : *class* ini merupakan layer aplikasi sebagai *class* pemanggil *class* pada layer bisnis

d. BRReport : *class* ini merupakan layer bisnis yang menjembatani antara layer aplikasi dan layer data akses



Gambar 3.38 VPOC untuk use case mencetak laporan

e. DLCommonReport : *class* ini merupakan layer data akses yang mengambil data pada basis data.

Use case specification dari masing – masing use case akan dijelaskan pada lampiran A.

3.2 First In First Out

Metode FIFO ini digunakan untuk proses posting transaksi penerimaan dan pengeluaran terutama dalam kaitannya pencatatan harga barang. Konsep penyediaan informasi nominal ini berdasar urutan waktu pemasukan barang. Untuk keperluan penerapan metode FIFO ini, semua informasi penerimaan barang akan disimpan berdasar urutan waktu.

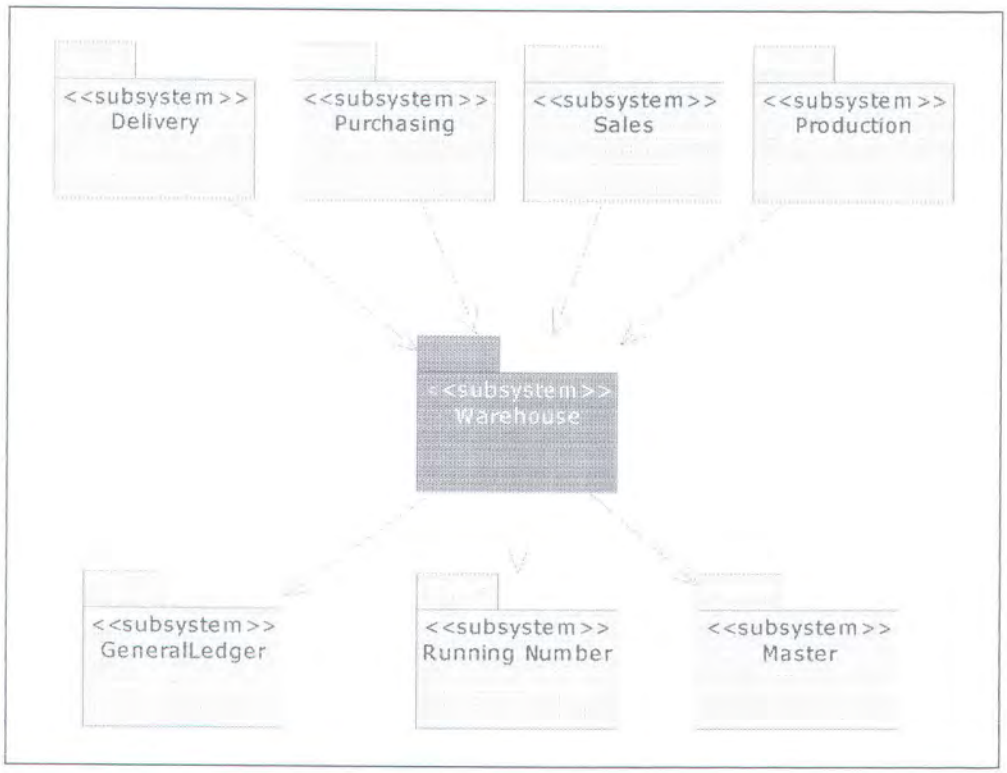
BAB 4
PERANCANGAN PERANGKAT LUNAK

Bab ini membahas perancangan perangkat lunak secara menyeluruh, meliputi perancangan basis data dengan menerapkan ORDBMS serta perancangan arsitektur untuk aplikasi yang terdistribusi. Perancangan ini merupakan penerapan dari teori yang telah dijelaskan pada bab 2.

4.1 Keterkaitan dengan Sistem Lain

Sistem inventori ini bukanlah sistem yang berdiri sendiri, melainkan merupakan bagian dari sebuah sistem yang kompleks. Sistem yang kompleks tersebut, dipecah-pecah menjadi subsistem-subsistem untuk mempermudah proses pembuatannya, salah satunya adalah sistem inventori ini, sehingga sistem ini berkaitan atau berhubungan dengan sistem yang lainnya. Dengan kata lain, sistem inventori ini membutuhkan data dari sistem yang lain dan sebaliknya, sistem yang lain juga membutuhkan data dari sistem inventori ini. Gambar 4.1 adalah gambaran keterkaitan sistem inventori ini dengan sistem lain dan sebaliknya.

Gambar 4.1 menjelaskan bahwa sistem inventori ini dapat menggunakan *interface* yang disediakan oleh subsistem – subsistem yang berada dilevel bawahnya (General Ledger, Running Number, Master) untuk mendapatkan informasi yang terdapat pada subsistem tersebut. Dan sistem inventori ini juga akan menyediakan suatu *interface* agar dapat digunakan oleh subsistem yang



Gambar 4.1 Keterkaitan sistem inventori dengan sistem lain

berada pada level di atasnya.

4.2 Arsitektur Sistem

Arsitektur untuk sistem inventori ini didasarkan pada *Enterprise Architecture*, yang dibagi menjadi 3 logical layer, yaitu :

1. Layer Aplikasi

   Layer ini merupakan layer yang menyediakan antarmuka untuk aplikasi. Validasi sisi client juga dilakukan pada layer ini. Untuk sistem inventori ini antarmuka menggunakan GUI Windows UI, yang dibangun dalam framework .NET dengan bahasa C#.

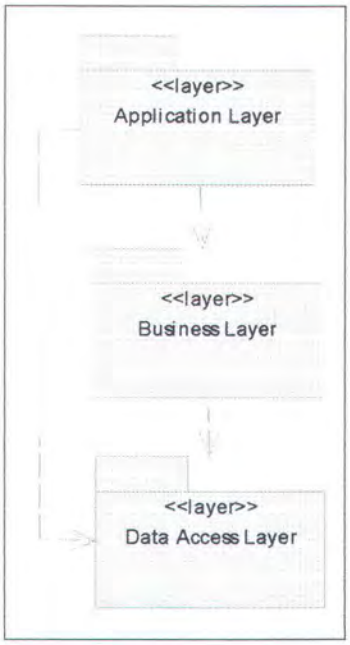
2. Layer Bisnis

Layer ini menangani segala sesuatu yang berkaitan dengan proses bisnis. Dalam sistem inventori, proses bisnis yang dijalankan pada layer ini adalah proses posting dan unposting transaksi penerimaan dan pengeluaran.

3. Layer Data Akses

Layer ini akan mengembalikan hasil query atas permintaan dari layer bisnis atau melakukan proses ke basis data(insert, update, delete).

Gambar 4.2 adalah arsitektur yang diterapkan untuk pembuatan tugas akhir ini:



Gambar 4.2 Arsitektur sistem inventori

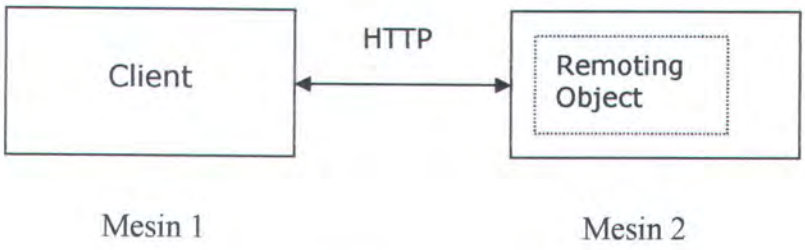
Penjelasan arsitektur :

1. Layer Aplikasi
   - Layer Aplikasi menggunakan aturan bisnis yang berada di layer bisnis untuk proses posting dan unposting transaksi penerimaan dan pengeluaran. (Contoh form yang menggunakan business layer ini adalah FrmTrmBeli.cs yang menggunakan method posting(string pID) yang berada pada layer bisnis yang digunakan untuk memposting transaksi penerimaan pembelian)
2. Layer Bisnis
   - Layer bisnis digunakan oleh layer aplikasi, seperti yang telah dijelaskan sebelumnya.
   - Layer bisnis menggunakan layer data akses untuk proses akses ke basis data (insert, update dan delete)
3. Layer Data Akses
   - Layer ini digunakan untuk akses ke basis data, dan atau yang akan mengembalikan hasil query di database untuk dikembalikan ke layer bisnis.

4.3 Arsitektur Distribusi

Disamping menerapkan *enterprise architecture*, sistem ini juga didesain untuk menerepakan konsep muti tier (distribusi). Konsep dari distribusi yang menggunakan .NET remoting ini adalah memisahkan layer dalam mesin yang

berbeda. Chanel yang digunakan untuk transport pesan dari dan ke remote obyek adalah channel HTTP.

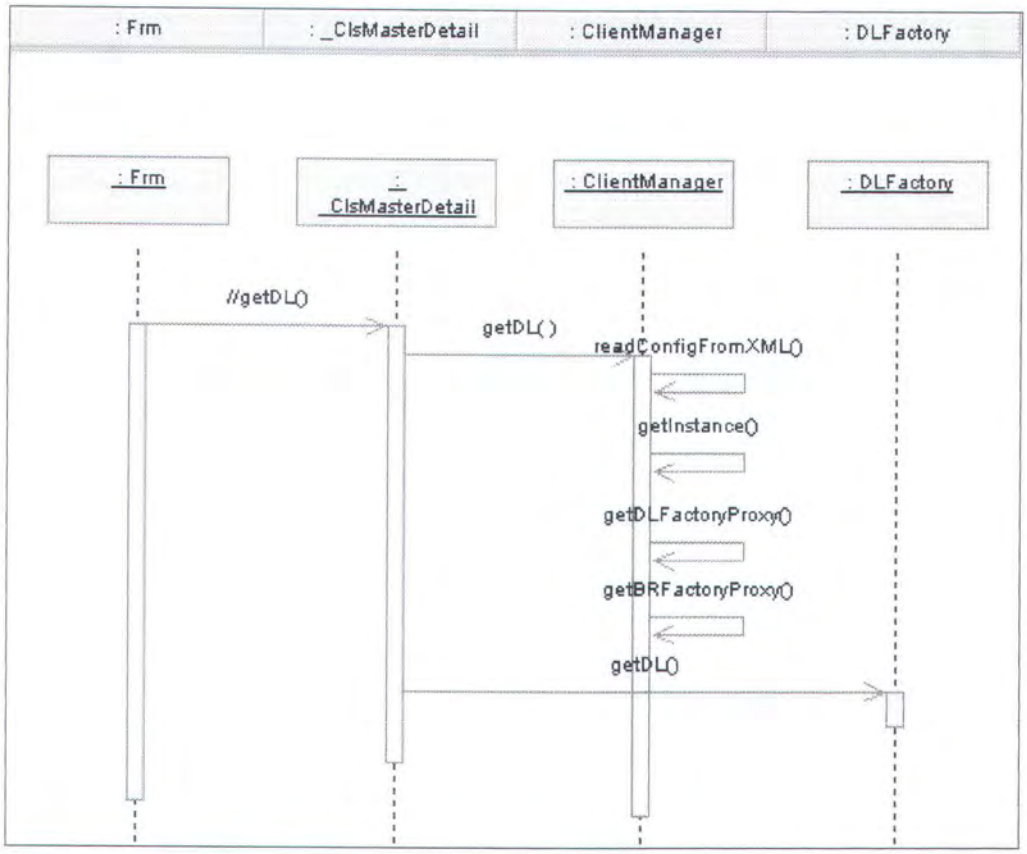


Gambar 4.3 Remoting menggunakan HTTP

Sequence diagram pada gambar 4.4 menggambarkan urutan interaksi *class* untuk proses distribusi. *Class* Frm mewakili aplikasi layer yang mengimplemenstasi interface IMasterDetail, IMasterCustom, IMasterLarge dll yang membutuhkan layer data akses. ClsMasterDetail akan memanggil prosedur GetDL() pada ClientManager. ClientManager inilah yang akan mendapatkan proxy object yang ada pada mesin yang berbeda. ClientManager akan mengembalikan instance dari DLFactory dan DLFactory inilah yang akan menciptakan obyek yang dibutuhkan Frm.

4.4 Pembuatan Basis Data dengan Menerapkan ORDBMS

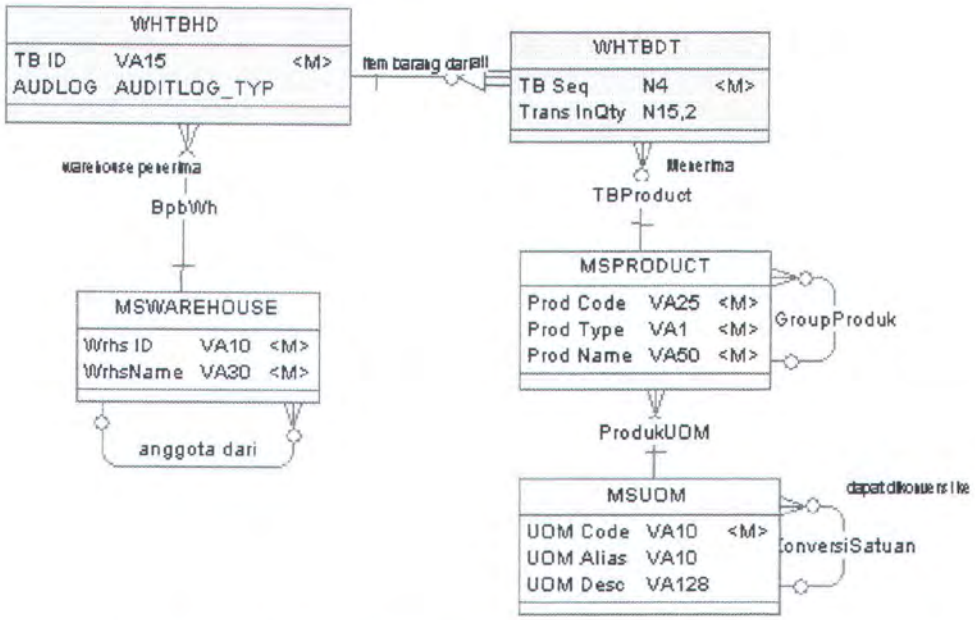
Pembuatan basis data dengan menerapkan ORDBMS berbeda dengan basis data dengan menerapkan RDBMS. Berikut beberapa perubahan pada pembuatan basis data yang menerapkan ORDBMS.



Gambar 4.4 Urutan interaksi *class* untuk proses distribusi

4.4.1 Perbedaan Query Dasar pada RDBMS dan ORDBMS

Query yang digunakan untuk basis data yang menerapkan Oracle obyek berbeda dengan RDBMS. Untuk dapat membandingkan antar query dalam RDBMS dan query pada ORDBMS akan digunakan contoh relasi antara table yang merupakan master detail seperti berikut ini :



Gambar 4.5 Contoh tabel yang berelasi pada RDBMS

Berikut ini adalah beberapa contoh query berdasar table tersebut di atas dalam RDBMS :

- Query select

  Pada RDBMS relasi antara table dihubungkan dengan foreign key. Berikut adalah syntax query pada RDBMS untuk mendapatkan nama warehouse :

```
select a.tbid, a.wrhsid, b.wrhsname
from whtbhd a, mswarehouse b
where a.wrhsid = b.wrhsid
```

  Dalam query tersebut di atas dibutuhkan kondisi untuk mendapatkan nama warehouse yang tepat.

- Query Insert untuk Header

  Untuk menambahkan data pada table WHTBHD, maka query berikut dapat dijalankan :

```
Insert into WHTBHD (tbid, wrhsid)
values ('0303.06.PL.0001', 'A002')
```

- Query Insert untuk Detail

Untuk menambahkan data pada detail WHTBHD, yaitu table WHTBDT maka query berikut dapat dijalankan

```
Insert into WHTBDT (tbid, tbseq, prodcode, transinqty)
Values ('0303.06.PL.0001', 1, 'A3240015R', 100)
```

- Query Update untuk Header

Untuk mengubah data pada header maka query berikut dapat digunakan :

```
Update WHTBHD set wrhsid = 'D001'
where a.tbid = '0303.06.PL.0001'
```

-

- Query Update untuk Detail

Untuk mengubah data pada detail maka query berikut dapat digunakan:

```
Update WHTBDT set transinqty = 250
Where TBID = '0303.06.PL.0001' and TBSeq = 1
```

- Query Delete untuk Header

Untuk menghapus data pada header maka query yang digunakan adalah :

```
Delete WHTBHD where TBID = '0303.06.PL.0001'
```

- Query Delete untuk Detail

Untuk menghapus pada detail maka query yang digunakan adalah sebagai berikut :

```
Delete WHTBDT where TBID = '0303.06.PL.0001' and TBSeq = 1
```

Masih berdasar table yang sama, dalam ORDBMS masing-masing entitas utama dijadikan tipe obyek yang kemudian dibuat table obyek sesuai dengan

kebutuhan. Perbedaan yang paling mendasar antara konsep RDBMS dan ORDBMS adalah pada relasi master-detail, di ORDBMS relasi ini diwujudkan dengan nested table, yaitu table WHTBDT dijadikan inner table dari WHTBHD. Dan relasi antara table (misal antara WHTBHD dan MSWarehouse tidak diwujudkan dengan foreign key namun direlasikan dengan object reference, sehingga pada table WHTBHD akan menyimpan OID dari MSWarehouse). Sehingga dengan adanya beberapa perbedaan ini maka query dasar yang digunakan dalam ORDBMS berbeda dengan query yang digunakan pada RDBMS. Berikut adalah contoh query dasar pada ORDBMS :

- Query Select

  Query select pada ORDBMS sedikit berbeda dari RDBMS. Perbedaan itu terletak pada query select yang melibatkan 2 tabel. Dalam RDBMS ke 2 tabel ini saling berelasi dengan foreign key sebagai penghubung, namun dalam ORDBMS tabel obyek yang 1 akan menyimpan referensi dari tabel obyek yang lain. Misal relasi antara master penerimaan (WHTBHD yang mempunyai primary key TBID) dengan gudang (MSWAREHOUSE yang mempunyai primary key WRHSID). Query select untuk mendapatkan nama gudang pada ORDBMS adalah sebagai berikut :

```
select a.tbid, a.wrhsid.wrhsid, a.wrhsid.wrhsname
from whtbhd a
```

- Query Insert untuk Header

  Query menambah data untuk header berbeda dengan query pada RDBMS, perbedaannya terletak pada pendefinisian obyek untuk detail yang merupakan

nested table dan penyimpanan tipe REF dari tabel obyek lain yang berelasi. Berikut adalah contoh query menambah data intuk master penerimaan :

```
Insert into whtbhd (tbid, trxdate, wrhsid, whtbdt)
values ('0303.06.PL.0001', to_date('23/05/2003','dd/mm/yyyy'), (select ref(w) from mswarehouse w where
w.wrhsid = 'A002'), whtbdt_ntabtyp())
```

- Query Insert untuk Detail

  Query menambah data untuk detail ini digunakan untuk menginsertkan data pada nested table. Berikut adalah contoh query menambah data untuk detail dari master dengan id = 0303.06.PL.0001.

```
Insert into table(select a.whtbdt from whtbhd where tbid = '0303.06.PL.0001')
(tbseq, prodcode, uomnonstd, transinqty)
values(1, (select ref(p) from msproduct p where p.prodcode = 'A3240015R'), (select ref(u) from msuom u
where u.uomcode = 'kg'), 100)
```

- Query Update untuk Header

  Query mengubah untuk header pada ORDBMS ini sama dengan query update pada RDBMS.

```
Update whtbhd a
set a.wrhsid = (select ref(w) from mswarehouse w where w.wrhsid = 'D001')
where a.tbid = '0303.06.PL.0001'
```

- Query Update untuk Detail

  Query mengubah data untuk detail sedikit berbeda dengan RDBMS. Berikut adalah contoh query update untuk detail yang merupakan nested table.

```
Update table(select a.whtbdt from whtbhd where tbid = '0303.06.PL.0001') b
set transInQty = 250
where b.tbseq = 1
```

- Query Delete untuk Header

  Query untuk menghapus data pada master sama halnya pada RDBMS.

```
Delete whtbhd where tbid = '0303.06.PL.0001'
```

- Query Delete untuk Detail

  Query untuk menghapus data pada detail yang merupakan nested table sedikit berbeda dengan RDBMS.

```
Delete table(select a.whtbdt from whtbhd where tbid = '0303.06.PL.0001')b
where b.tbseq = 1
```

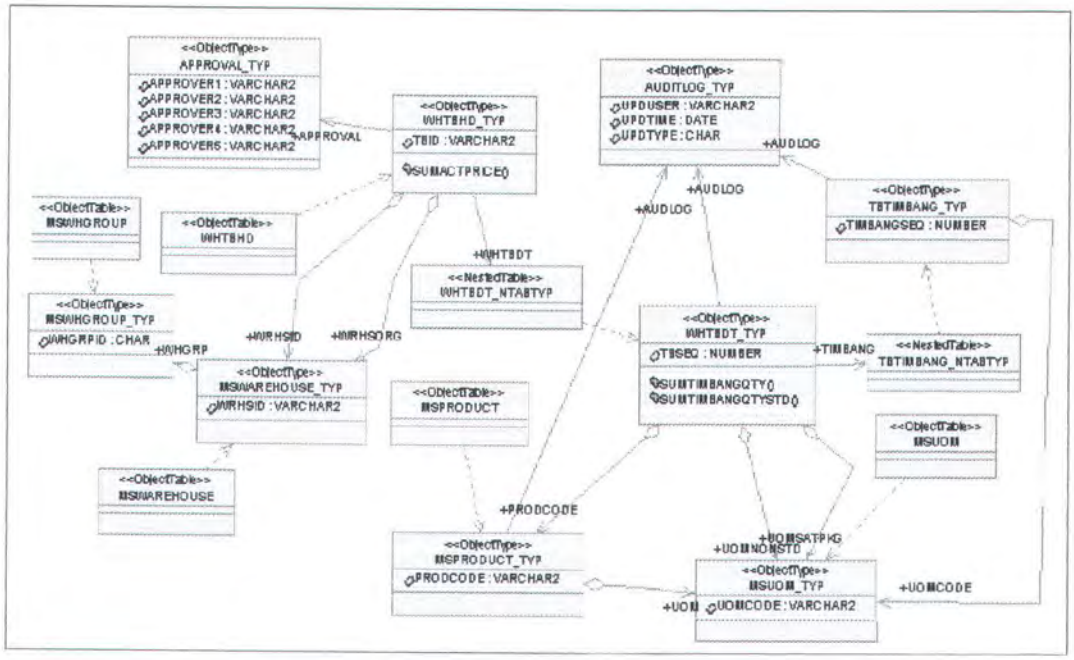
Berikut akan dijelaskan data model dari sistem yang berdasar ORDBMS beserta dengan pembuatannya ke dalam basis data (dalam hal ini database menggunakan Oracle 9i). Untuk menyederhanakan data model maka akan dipecah menjadi beberapa data model. Data model ini menggambarkan relasi antara tipe obyek, table obyek, nested table dan relasi jika 1 obyek mereferensi obyek yang lain.

1. Data model Penerimaan Barang

   Data model penerimaan barang yang digambarkan pada gambar4.3 diperoleh berdasar dari Analisis sistem akan proses penerimaan barang yang terjadi pada rumah pemotongan ayam, seperti yang telah diuraikan dalam Bab3.

2. Data Model Pengeluaran Barang

   Data model untuk pengeluaran barang ini juga peroleh berdasar Analisis sistem akan proses pengeluaran barang yang telah diuraikan pada Bab 3. Data model untuk pengeluaran barang ini dapat dilihat pada gambar 4.4.



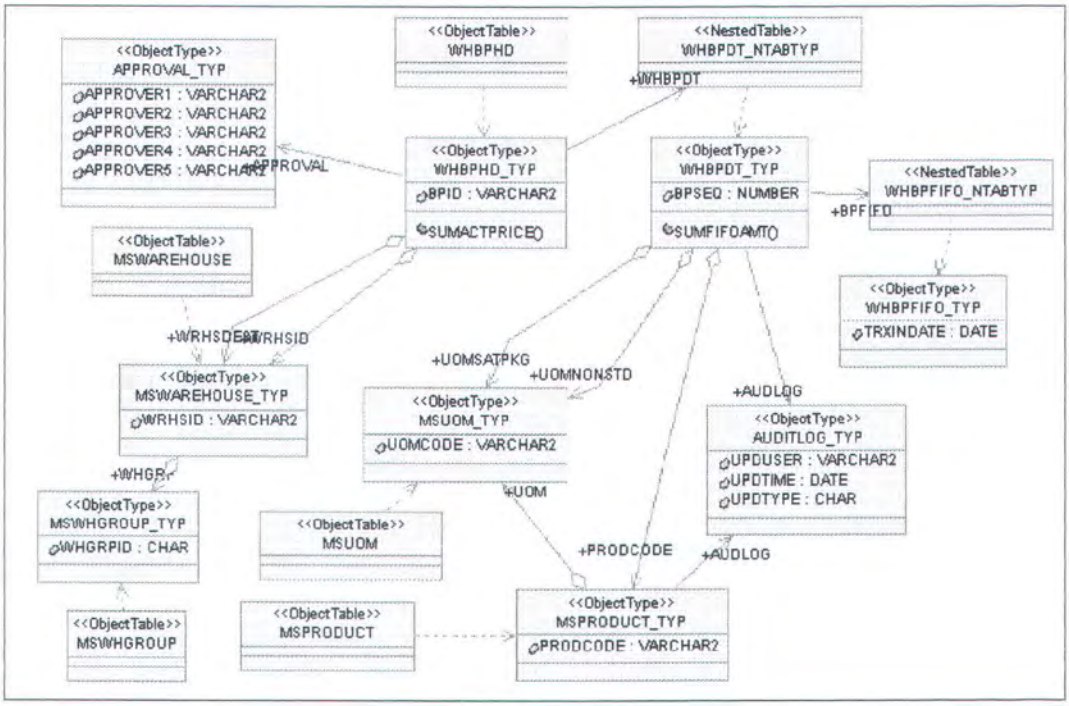
Gambar 4.6 Data model Penerimaan Barang

3. Data Model Stock

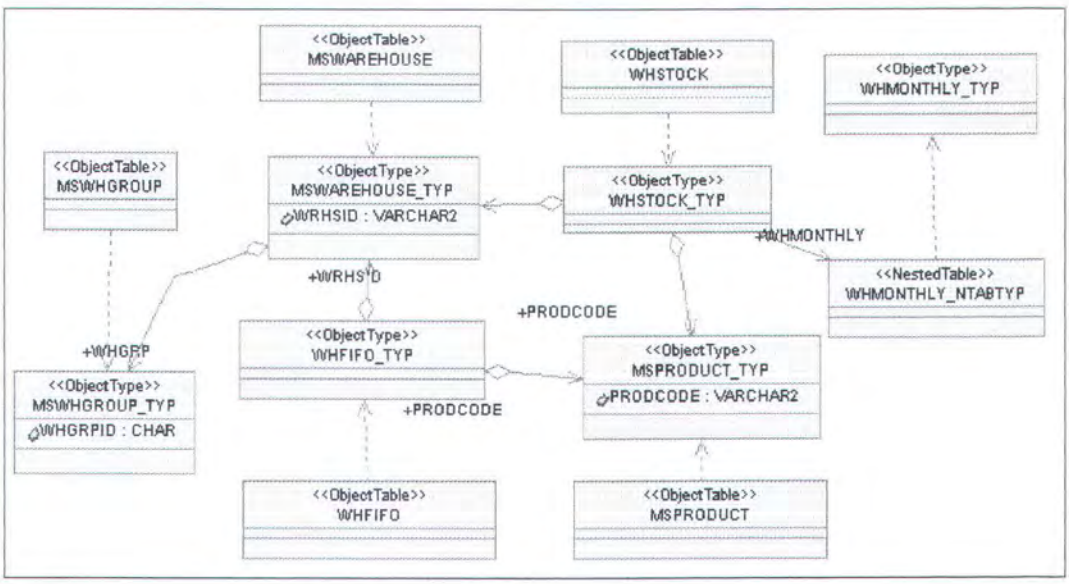
Data model untuk stock ini diperoleh berdasar Analisis sistem pada Bab 3. Data model ini nantinya digunakan untuk menyimpan informasi stok per barang untuk masing-masing gudang dan menyimpan proses pemasukan dan pengeluaran barang berdasar konsep FIFO. Data model untuk stok dapat dilihat dalam gambar 4.5

Relasi antara *class* :

1. Dependency Relationship : Relasi ini terjadi antara table obyek dan tipe obyek, hal ini dikarenakan perubahan pada tipe obyek akan mengakibatkan perubahan pada table obyek



Gambar 4.7 Data model pengeluaran barang



Gambar 4.8 Data model Stock

2. Association Relationship : Relasi ini terjadi antara obyek tipe dengan nested table-nya dan juga antara suatau tipe obyek dengan APPROVAL_TYP atau AUDLOG_TYP
3. Aggregation Relationship : Relasi ini terjadi antara suatu tipe obyek yang satu dengan tipe obyek yang lain yang merepresentasikan hubungan "part-whole"

Ada 2 pendekatan untuk dapat mengimplementasikan ORDBMS, yaitu :

1. Membuat *object table*
2. Membuat *object view* untuk mereprsentasikan virtual tabel obyek dari data yang berelasional yang sudah ada.

Namun dalam tugas akhir ini dipakai pendekatan pertama, yaitu membuat tabel obyek. Berdasar analisis sistem dan desain data model seperti yang telah dijelaskan pada bab 3, berikut ini akan dijelaskan implementasi pembuatan basis data dengan menerapkan ORDBMS. Dalam penerapan ORDBMS ini, entitas utama akan dijadikan tipe obyek dan referensi obyek (*object reference*) akan merepresentasikan relasi diantaranya. Dari tipe obyek ini kemudian di jadikan table obyek sesua dengan kebutuhan, sementara multi valued attribute akan diimplementasi dalam tipe koleksi (*collection type*)

4.4.2 Mendefinisikan Tipe

Membuat *user defined type* dengan menggunakan statemen CREATE TYPE. Berikut adalah pembuatan dari masing –masing tipe obyek:

1. Pembuatan MSUOM_TYP

Instance MSUOM_TYP adalah obyek yang merepresentasikan satuan dari produk (tabel obyek MSPRODUCT akan mereferensi ke tabel obyek MSUOM yang merupakan *instance* dari MSUOM_TYP). Dan script untuk membuat tipe obyek itu dalam basis data Oracle9i dapat dilihat sebagai berikut:

```
create or replace type MSUOM_TYP as OBJECT (
  UOMCODE          VARCHAR2(10),
  UOMCONVTO        VARCHAR2(10),
  UOMALIAS         VARCHAR2(10),
  UOMDESC          VARCHAR2(128),
  UOMCONVFACTOR    NUMBER(12,6)
)
```

2. Pembuatan MSWHGROUP_TYP

Tipe obyek MSWHGROUP_TYP merepresentasikan group dari gudang (pada pembuatan tipe obyek MSWAREHOUSE_TYP akan mereferensi tipe obyek ini). Dan script untuk membuat tipe obyek itu dalam basis data Oracle9i dapat dilihat sebagai berikut :

```
create or replace type MSWHGROUP_TYP as OBJECT (
  WHGRPID          CHAR(2),
  WHGRPNAME        VARCHAR2(20))
```

3. Pembuatan MSWAREHOUSE_TYP

Object type MSWAREHOUSE_TYP ini merupakan instance untuk object yang berisi data dari warehouse. Object type MSWAREHOUSE_TYP ini mempunyai atribut yang bertipe varchar dan MSWHGROUP_TYP yang merupakan user define type yang telah dibuat sebelumnya. Dan script untuk membuat tipe obyek itu dalam basis data Oracle9i dapat dilihat sebagai berikut

```
create or replace type MSWAREHOUSE_TYP as OBJECT (
  WRHSID          VARCHAR2(10),
  CITYCODE        VARCHAR2(10),
  WHGRP           REF MSWHGROUP_TYP,
  WRHSPARENT      VARCHAR2(10),
  WRHSNAME        VARCHAR2(30),
  WRHSALIAS       VARCHAR2(100),
  WRHSADDR1       VARCHAR2(80),
  WRHSADDR2       VARCHAR2(50),
  WRHSZIPCODE     VARCHAR2(8),
  WRHSPHONE       VARCHAR2(20),
  WRHSFAX         VARCHAR2(30),
  WRHSTYPE        VARCHAR2(1),
  FGSTATUS        VARCHAR2(1)
)
```

Berikut adalah daftar lengkap tipe obyek yang digunakan untuk membuat sistem inventori :

Tabel 4.1 Daftar type untuk pembuatan sistem inventori

No	Nama Object Type	Keterangan
1	APPROVAL_TYP	Merepresentasikan nama-nama penyetuju
2	AUDITLOG_TYP	Merepresentasikan nama, waktu dan tipe dari Action yang dilakukan pada object table
3	MSUOM_TYP	Merepresentasikan informasi satuan barang
4	MSWHGROUP_TYP	Merepresentasikan informasi group gudang
5	MSWAREHOUSE_TYP	Merepresentasikan informasi gudang
6	WHFIFO_TYP	Merepresentasikan informasi untuk proses FIFO
7	MSPRODUCT_TYP	Merepresentasikan informasi produk
8	TBTIMBANG_TYP	Merepresentasikan informasi data timbang object type ini akan dijadikan nested table dari Detail penerimaan
9	WHTBDT_TYP	Merepresentasikan informasi detail penerimaan Object type ini akan dijadikan nested table dari header penerimaan
10	WHTBHD_TYP	Merepresentasikan informasi header penerimaan
11	WHBPFIFO_TYP	Merepresentasikan informasi urutan FIFO untuk pengeluaran barang object type ini akan dijadikan nested table dari Detail

		pengeluaran
12	WHBPDT_TYP	Merepresentasikan informasi detail pengeluaran Object type ini akan dijadikan nested table dari header penerimaan
13	WHBPHD_TYP	Merepresentasikan informasi header pengeluaran
14	WHMONHTLY_TYP	Merepresentasikan informasi stock bulanan Object type ini akan dijadikan nested table dari header Stock
15	WHSTOCK_TYP	Merepresentasikan informasi stock
16	WHOPNAME_TYP	Merepresentasikan informasi SK Opname
17	WHOPNAMEDT_TYP	Merepresentasikan informasi detail hasil stock opname Object type ini akan dijadikan nested table dari header hasil stock opname
18	WHOPNAMEHD_TYP	Merepresentasikan informasi header hasil stock opname

Script pembuatan tipe obyek dan table obyek.ada di Lampiran D.

4.4.3 Mendefinisikan Metode

Masing – masing tipe obyek dapat memiliki metode (yang berupa fungsi atau prosedur) sesuai dengan kebutuhan. Metode ini merupakan *behavior* dari obyek yang bersangkutan. Berikut adalah pendefinisian metode yang terdapat pada object type WHBPDT_TYP.

```
create or replace type WHBPDT_TYP AS OBJECT (
  BPSEQ          NUMBER(4),
  CURRCODE       CHAR(3),
  TBID           REF WHTBHD_TYP,
  TBSEQ          NUMBER(4),
  GLMID          CHAR(10),
  GLSID          CHAR(8),
  UPERFECTQTY    NUMBER(15,2),
  UPERFECTQTY1   NUMBER(15,2),
  PRODCODE       REF MSPRODUCT_TYP,
  UOMCODE        REF MSUOM_TYP,
  AMTORG         NUMBER(17,2),
  EXCHRATE       NUMBER(11,5),
```

```
    AMT                 NUMBER(16,2),
    TRANSOUTQTY         NUMBER(15,2),
    TRANSOUTQTY1        NUMBER(15,2),
    QTYPACKAGEOUT       NUMBER(15,2),
    TRXPRICE            NUMBER(17,2),
    BPHRGSAT            NUMBER(11,2),
    BPHRGSATPRI         NUMBER(16,2),
    TRANSDESC           VARCHAR2(128),
    PRID                VARCHAR(15),
    PRSEQ               NUMBER(4),
    BPFIFO              WHBPFIFO_NTABTYP,
    AUDLOG              AuditLog_Typ,

    member function sumFifoAmt return number
)
```

Tipe obyek WHBPDT_TYP digunakan untuk membuat tabel obyek WHBPDT_NTAB (yang akan dijelaskan pada bagian pembuatan tabel obyek). WHBPDT_NTAB memiliki *nested table*, yaitu WHFIFO_NTAB, yang digunakan untuk menyimpan data jumlah barang yang dikeluarkan menurut aturan FIFO. Sehingga perlu diketahui jumlah total harga barang yang dikerluarkan yang sesuai dengan aturan FIFO. Untuk mengetahui jumlah harga barang, maka dibuat 1 fungsi yang akan mengembalikan total harga barang.

Untuk mendefinisikan metode pada suatu tipe obyek adalah dengan mendeklarasikan metode tersebut pada tipe obyek yang bersangkutan, dalam hal ini fungsi dideklarasikan dengan statement :

```
member function sumFifoAmt return number
```

Pada saat pendeklarasian tipe obyek, metode juga hanya dideklarasikan saja, untuk mendefinisikan isi dari metode tersebut dilakukan pada pendeklarasian body dari tipe obyek yang bersangkutan. Berikut adalah body dari tipe obyek WHBPDT_TYP.

```
create or replace type body Whbpdt_Typ as
 member function sumFifoAmt return number is
      i integer;
      total number := 0;
 begin
    for i in 1..self.bpFifo.count loop
      total := total + bpFifo(i).trxAmt;
    end loop;
    return total;
 end;
END;
```

Pernyataan:

```
    for i in 1..self.bpFifo.count loop
      total := total + bpFifo(i).trxAmt;
    end loop;
```

Proses *looping* akan dilakukan sebanyak jumlah obyek BPFIFO yang dimiliki oleh masing-masing detail dari pengeluaran. Dan statement

```
bpFifo(i).trxAmt
```

akan mengambil nilai pada atribut trxAmt pada masing-masing obyek BPFIFO, dan menjumlahkannya.

Selain pada tipe obyek WHBPDT_TYP, tipe-tipe obyek lain yang memiliki metode adalah :

1. WHTBDT_TYP

   Metode yang dimiliki pada tipe obyek WHTBDT_TYP adalah metode untuk menghitung data timbang dari masing-masing penerimaan (yaitu penerimaan ayam hidup). Data timbang yang akan perlu diketahui jumlahnya adalah dalam satuan standar dan non standar, sehingga diperlukan 2 macam metode untuk mendapatkan jumlah keduanya (data timbang dalam satuan

standard dan data timbang dalam satuan non standar). Metode – metode tersebut juga dideklarasikan pada saat pendeklarasian tipe obyek WHTBDT_TYP. Berikut adalah script untuk mendeklarasikan fungsi pada WHTBDT_TYP.

```
member function sumTimbangQty return number,
member function sumTimbangQtyStd return number
```

2. WHTBHD_TYP

Metode yang terdapat pada WHTBHD_TYP adalah metode yang digunakan untuk menghitung total harga untuk 1 penerimaan. Berikut adalah script untuk mendeklarasikan fungsi pada WHTBHD_TYP.

```
member function sumActPrice return number
```

3. WHBPHD_TYP

Metode yang terdapat pada WHBPHD_TYP adalah metode yang digunakan untuk menghitung total harga untuk 1 pengeluaran. Script deklarasi fungsi pada WHBPHD _TYP.

```
member function sumActPrice return number
```

4.4.4 Pembuatan Nested Table

Nested table adalah set dari elemen beberapa data yang memiliki tipe data yang sama. Tipe-tipe obyek yang akan dijadikan *nested table* harus dideklarasikan sebagai *nested table*. Berikut adalah pendeklarasian *nested table* untuk data timbang.

```
create or replace type TBTIMBANG_TYP as OBJECT (
  TIMBANGSEQ      NUMBER(4),
  UOMCODE         REF MSUOM_TYP,
  TIMBANGQTY      NUMBER(15,2),
```

```
    TIMBANGQTYSTD   NUMBER(15,2),
    AUDLOG          AuditLog_Typ
)

create type TBTIMBANG_NTABTYP as TABLE of TBTIMBANG_TYP
```

Sehingga langkah untuk membuat sebuah nested table adalah mendeklarasikan sebuah tipe obyek, yang kemudian mendeklarasikan sebuah tipe lagi yang merupakan table dari tipe obyek yang telah dideklarasikan terlebih dahulu.

Dan instance header dari nested table tersebut (WHTBDT_TYP) mempunyai satu atribute yang bertipekan TBTIMBANG_NTABTYP dan bukan lagi TBTIMBANG_TYP.

```
    TRANSRANGE      VARCHAR2(10),
    TIMBANG         TBTIMBANG_NTABTYP,
    AUDLOG          AuditLog_Typ,
```

4.4.5 Pembuatan Tabel Obyek

Setelah pendefinisian tipe obyek, maka perlu didefinisikan tabel obyek sebagai tempat penyimpanan data. Tabel obyek terdiri dari obyek-obyek yang masing-masing mempunyai atribut yang sama yang telah didefinisikan pada tipe obyek. Berikut adalah pendefinisian tabel obyek dari MSWAREHOUSE yang terdiri dari tipe obyek MSWAREHOUSE_TYP. Berikut adalah script pembuatan tabel obyek.

```
create table MSWAREHOUSE of MSWAREHOUSE_TYP (
  primary key (WRHSID),
  foreign key (WHGRP) REFERENCES MSWHGROUP,
  WRHSNAME          not null,
  constraint FK_MSWAREHO_ANGGOTA_D_MSWAREHO foreign key (WRHSPARENT)
     references MSWAREHOUSE (WRHSID),
  constraint FK_MSWAREHO_KOTA_WARE_MSCITY foreign key (CITYCODE)
```

```
        references MSCITY (CITYCODE)
    )
      object ID primary key
```

Melalui statement "of" berarti object table MSWAREHOUSE memiliki kolom seperti atribut-atribut yang telah didefinisikan pada tipe obyek MSWAREHOUSE_TYP. Dan masing-masing baris merupakan tipe obyek MSWAREHOUSE_TYP.

4.4.6 Object Table dengan Nested Table

Statemen berikut adalah untuk membuat tabel obyek WHTBHD

```
create table WHTBHD of WHTBHD_TYP (
  primary key (TBID),
  foreign key (WRHS) REFERENCES MSWAREHOUSE,
  foreign key (WRHSORG) REFERENCES MSWAREHOUSE,
  POID       REFERENCES POHD (POID),
  GLMID      REFERENCES GLM (GLMID),
  GLSID      REFERENCES GLS (GLSID),
  EXPD       REFERENCES MsExpedisi(Expdid),
  TRX        REFERENCES RnTRx(TRXID),
  REL        REFERENCES MsRelasi(Relid),
  TRXDATE          default sysdate  not null,
  TRXSTAT          default '0'  not null,
  constraint FK_WHTBHD_TRANS_IN_RNTRX foreign key (TRX)
     references RNTRX (TRXID),
  constraint FK_WHTBHD_TBRELASI_MSRELASI foreign key (REL)
     references MSRELASI (RELID),
  constraint FK_WHTBHD_TBPUNYAPO_PODT foreign key (POID, POSEQ)
     references PODT (POID, POSEQ),
  constraint FK_WHTBHD_WHTBGL_GLMAP foreign key (GLMID, GLSID)
     references GLMAP (GLMID, GLSID),
  constraint FK_WHTBHD_LPAHEXPED_MSEXPEDI foreign key (EXPD)
     references MSEXPEDISI (EXPDID)
)
  object id primary key
  NESTED TABLE WHTBDT STORE AS WHTBDT_NTAB
   ((PRIMARY KEY(NESTED_TABLE_ID, TBSEQ))
   ORGANIZATION INDEX COMPRESS
    NESTED TABLE TIMBANG STORE AS TBTIMBANG_NTAB)
```

Pada pengimplementasian konsep ORDBMS ini terdapat kendala, yaitu adanya error yang belum ditemukan solusinya oleh pihak Oracle (Lihat Lampiran E). *Error* itu adalah ***ORA-00600: [qctcfx : len], [0], [0], [], [], [], [], []*** yang terjadi pada saat pembuatan cursor yang mengakses *nested table* dan/atau query yang mengakses *nested table*. Query dan cursor itu digunakan untuk kepentingan alur bisnis dalam sebuah stored prosedur. Sehingga solusi yang gunakan adalah mengatur alur bisnis ini dalam layer bisnis. Berikut adalah contoh dari pembuatan cursor yang mengakses nested table (huruf berwarna merah) yang digunakan untuk pembuatan stored procedure untuk kepentingan alur bisnis yang menyebabkan error ORA-00600 :

```
procedure TbPosting (pErr out NUMBER, pTbID in WhTbHd.TbID%TYPE) is
    tMonth    CHAR;
    tStat     WhTbHd.TrxStat%TYPE;
    tSign     NUMBER;
    tCount    NUMBER;
    tProd     ref Msproduct_Typ;
    cursor cDt is
        select h.TbID, h.TrxDate, deref(h.Wrhs).WrhsId WrhsId, deref(d.ProdCode).ProdCode ProdCode,
               d.TransInQty, d.Amt
        from WhTbHd h, table(h.whtbdt)d
        where h.TbID = pTbID;
  begin
    select TRUNC(TrxDate, 'MONTH'), TrxStat into tMonth, tStat
     from WhTbHd
     where TbID = pTbID;
    if tStat in ('8', '9') then
      pErr := 1;  -- the Trx is already posted
      ERR.SetErr ('Wh.TbPosting: Transaksi ' || pTbID || ' sudah pernah diposting sebelumnya!');
      return;
    end if;
    for cp in cDt loop
       select count(d.lvlMonth) into tCount
       from WhStock h, table(h.whmonthly) d
       where h.wrhsid = cp.WrhsId and h.ProdCode = cp.ProdCode and d.lvlMonth = tMonth;
      if tCount != 1 then
        pErr := 2;  -- do Setup first!
        ERR.SetErr ('WH.TbPosting: Jalankan proses Setup untuk bulan ' || tMonth || ' dulu!');
```

```
            rollback;
            return;
          end if;
          insert into WhFIFO (WrhsID, ProdCode, TransDate, FifoQty, FifoPrice)
            values (cp.WrhsID, cp.ProdCode, cp.TrxDate, cp.TransInQty, (cp.Amt / cp.TransInQty));

        update WhTbHd
          set TrxStat = '8'
        where TblD = pTblD;
        pErr := 0;
      end;
```

4.5 Perancangan Antarmuka

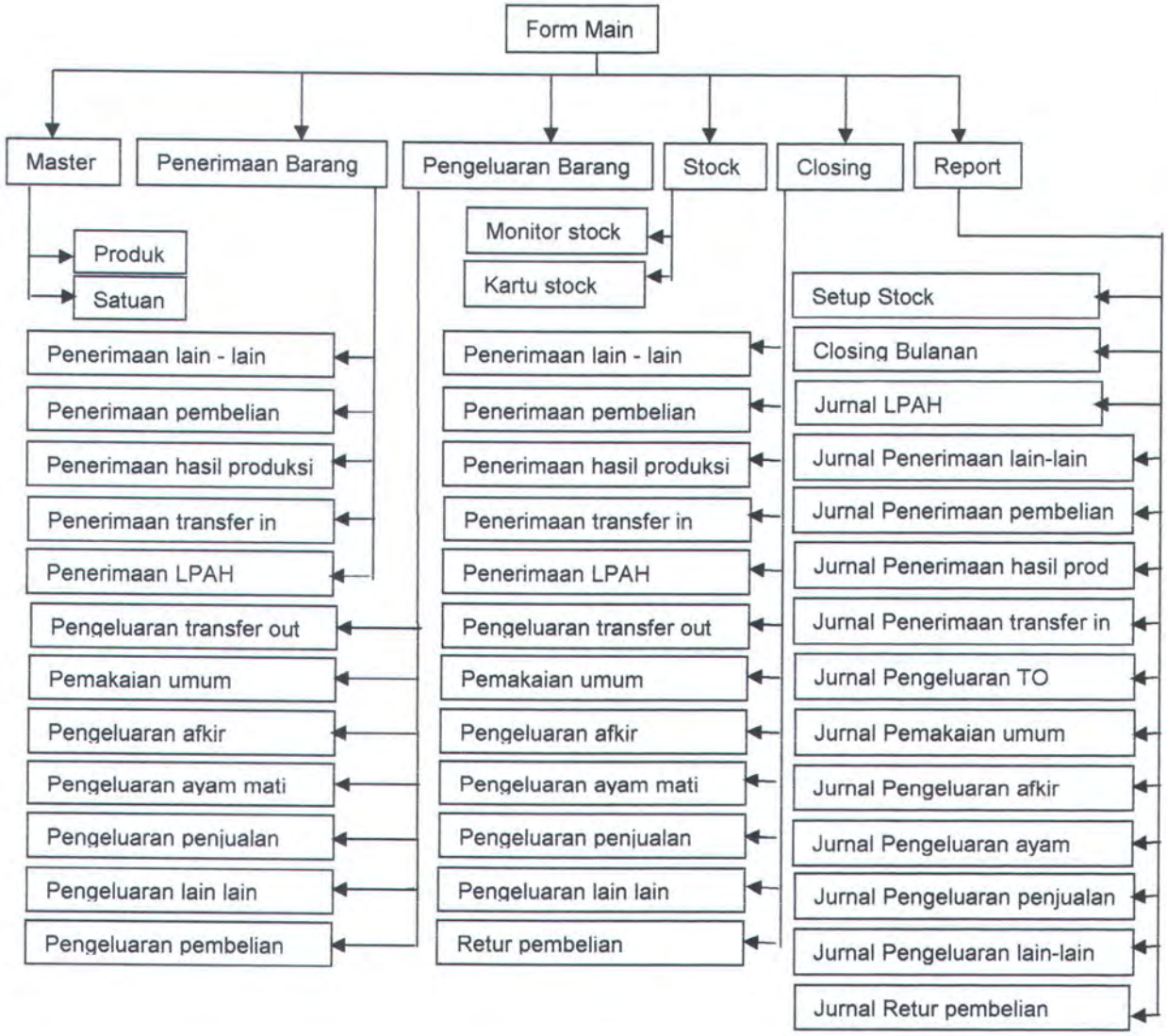
Dalam pembuatan aplikasi sistem inventori ini, diperlukan rancangan antarmuka untuk memberikan gambaran bagaimana pengaksesan data dalam sistem yang akan dibuat. Antarmuka aplikasi terbagi menjadi 6 kelompok utama, yaitu kelompok Master, Penerimaan, Pengeluaran, Stock, Closing dan Report. Gambaran menu utama sistem inventori dapat dilihat pada gambar 4.9.

4.5.1 Menu Master

Fasilitas yang ada di form master adalah menambah data, mengubah data, menghapus data, melihat data master. Yang termasuk data master disini adalah data produk dan satuan. Pada fasilitas yang tersedia dalam masing – masing sub menu master ini adalah menambah, mengubah dan menghapus data.

4.5.2 Menu Penerimaan

Fasilitas yang ada di form – form penerimaan adalah menambah data, mengubah data, menghapus data, melihat data transaksi yang ada. Pada form penerimaan ini



Gambar 4.9 Rancangan menu pada antarmuka aplikasi sistem inventori

terdapat 2 tab, yaitu tab untuk entri data master dan list detail dari tiap – tiap transaksi. Sementara tab kedua merupakan list dari transaksi pada periode dan gudang yang terpilih. Data transaksi yang ditampilkan adalah transaksi-transaksi yang mempunya status belum di closing. Untuk transaksi yang telah berstatus *approved* sebelum proses pengeditan dilakukan harus di unPosting terlebih dahulu Pada list data transaksi digunakan untuk melakukan posting dan

unPosting transaksi yang ada. Transaksi yang dapat di unPosting adalah transaksi–transaksi yang detail barangnya belum dikeluarkan.

4.5.3 Menu Pengeluaran

Antarmuka pengeluaran ini adalah sama dengan antarmuka penerimaan.

4.5.4 Menu Monitor Stock

Menu monitor stock ini digunakan untuk Setup Stock dan Monitor Stock. Untuk setup stock pada periode tertentu, terdapat constraint, yaitu setup stock pada bulan sebelumnya harus sudah dilakukan. Sementara sub menu monitor stock hanya menampilkan posisi stock barang dan FIFO untuk barang tersebut pada gudang tertentu.

4.5.5 Menu Setup & Closing

Menu ini digunakan untuk meng-close (merubah status transaksi dari approved menjadi closed). List transaksi yang ditampilkan adalah transaksi-transaksi yang mempunyai status approved dan closed. Transaksi yang telah di closed tidak dapat dilakukan perubahan data lagi.

4.5.6 Menu Report

Menu report ini digunakan untuk kepentingan pencetakan laporan.

BAB 5
PEMBUATAN APLIKASI

Setelah dilakukan pemodelan dan perancangan sistem pada bab 3 dan bab 4 di atas, selanjutnya dilakukan pembuatan aplikasi berdasarkan rancangan tersebut. Penjelasan mengenai pembuatan aplikasi ini dibagi menjadi beberapa tahap, pembuatan layer aplikasi, layer bisnis, layer data akses, penerapan konsep FIFO, laporan dan pembuatan aplikasi terdistribusi.

5.1 Pembuatan Layer Aplikasi

Pada pembuatan layer aplikasi akan dijelaskan pembuatan *interface* yang dibutuhkan pada layer aplikasi dan pembuatan *user control* untuk layer aplikasi.

5.1.1 Pembuatan Interface

Pada C#, sebuah *class* dimungkinkan untuk menurunkan hanya dari sebuah *single parent*. Namun, sebuah *class* dimungkinkan untuk mengimplementasi beberapa *interface*, sehingga *class* juga memiliki semua metode yang dideklarasikan pada *interface* yang diimplementasikan.

Pada pembuatan aplikasi sistem inventori ini, masing – masing *class* (pada layer aplikasi) akan mengimplementasi, minimal 1 interface. Adapun interface yang dibutuhkan untuk pembuatan aplikasi ini adalah : IMasterDetail, IMasterLarge, IMasterSmall, IMasterCustom, IFilter.

Masing – masing *interface* tersebut mendeklarasikan masing – masing fungsionalnya pada IMater.cs. Berikut adalah fungsi – fungsi dari masing-masing interface yang dideklarasikan pada IMaster.

Tabel 5.1 Fungsi-fungsi pada interface layer aplikasi

Nama Interface	Nama Fungsi	Return value	Kegunaan
IMasterSmall	*AllowAddNew*	bool	Cek apakah proses diijinkan menambah record
	SetupGrid	void	Mengatur grid
IMasterLarge	*SetupGrid*	void	Mengatur grid
	BindDetailControl	void	Binding kontrol untuk detail
	SetViewMode	void	Mengatur enable/disable field
	CheckInput	bool	Cek apakah proses selanjutnya dapat dijalankan
	UpdateDataSet	void	Mengupdate dataset
IMasterDetail	*SetupMasterGrid*	void	Mengatur grid master
	SetupDetailGrid	void	Mengatur grid detail
	BindDetailControl	void	Binding kontrol untuk detail
	SetViewMode	void	Mengatur enable/disable field
	CheckInput	bool	Cek apakah proses selanjutnya dapat dijalankan
	UpdateDataSet	void	Mengupdate dataset
	SetupSupportControl	void	Setup kontrol pendukung
IMasterCustom	*AllowActionDelegate*	bool	Cek apakah proses selanjutnya dapat dijalankan
	ActionDelegate	bool	Proses yang dijalankan
Ifilter	*GetFilterString*	string	Mengembalikan filter
	GetUpdateCriteria	string	Mengembalikan kondisi untuk update

Masing – masing *class* akan mengimplementasi beberapa interface yang dibutuhkan. Berikut adalah pendeklarasian *class* FrmTrmBeli yang mengimplementasi interface IMasterDetail dan IFilter.

```
public class FrmTrmBeli : System.Windows.Forms.Form, IMasterDetail, IFilter
```

Sehingga pada *class* FrmTrmBeli harus menyediakan implementasi dari semua metode secara lengkap yang telah dideklarasikan oleh IMasterDetail dan IFilter. Jika *class* yang mengimplementasi *interface* tersebut tidak mendeklarasikan semua metode dari interface yang diimplementasi maka akan

error. Berikut adalah contoh pendeklarasian metode IFilter pada *class* FrmTrmBeli.

```
# region IFilter Implementation
    public string GetFilterString ()
    {
            // deklarasi isi method disini
    }
    public string GetUpdateCriteria ()
    {
            // deklarasi isi method disini
    }
# endregion
```

Pada IMaster, IFilter telah mendeklarasikan metode *GetFilterString()* dan *GetUpdateCriteria()*.

5.1.2 Pembuatan Komponen Antarmuka

Untuk memudahkan pembuatan aplikasi, maka dibuat komponen – komponen untuk menangani masalah kontrol antarmuka. Kontrol yang dibuatkan komponen adalah kontrol-kontrol yang sering digunakan, missal kontrol untuk list of value (LOV), pengambil tanggal (datePicker), toolbar. Metode untuk masing – masing kontrol didefinisikan sendiri. Misal untuk kontrol LOV mempunyai

Berikut adalah komponen antarmuka yang dibuat untuk sistem ini :

Tabel 5.2 Komponen antarmuka yang digunakan untuk pembuatan sistem inventori

Nama komponen	Kegunaan
ctlLOV	menampilkan list of value
ctlMonthPicker	mengambil bulan-tahun
cGudang	menampilkan gudang
cPlusMin	tombol tambah dan kurang untuk detail
ctlToolBarReport	toolbar untuk report
cToolBarAll	tollbar untuk form
drdLOV	drop down LOV
ucPilih	radio button

5.2 Pembuatan Layer Bisnis

Layer bisnis difungsikan sebagai jembatan antara layer aplikasi dengan basis data (layer data akses) dan juga difungsikan untuk mengatur alur bisnis dari aplikasi yang ada. Untuk sistem inventori ini, layer bisnis digunakan untuk mengisi dataset dan untuk mengatur proses posting dan unPosting pada masing – masing transaksi, dimana dalam hal ini untuk kepentingan penerapan konsep FIFO.

5.2.1 Posting Transaksi Penerimaan

Pada setiap form transaksi penerimaan, terdapat fasilitas untuk memposting. Posting transaksi penerimaan artinya adalah melakukan proses push untuk semua jenis barang, menambah jumlah barang dalam stock dan mengubah status transaksi dari *open* menjadi *approved* (sudah diposting). Jika suatu transaksi sudah mempunyai status *approved,* maka transaksi tersebut tidak dapat diubah lagi, untuk mengubahnya harus dilakukan proses unPosting.

Algoritma untuk memposting setiap transaksi adalah sebagai berikut :

1. Mengambil detail dari transaksi yang akan diposting

   GetDetailTB()

2. Untuk masing – masing jenis barang dilakukan pengecekan dalam stock, apakah barang dalam gudang bersangkutan sudah ada dalam table stock. Jika belum ada maka dilakukan proses insert.

   Loop for each product

   If (checkSetup == false)

SetupNewProduct()

3. Mengambil nilai pada stock, yaitu nilai stok akhir, nilai rupiah stok akhir, debet untuk bulan bersangkutan dan nilai rupiah untuk bulan bersangkutan.

   GetQtyAmt()

4. Melakukan proses update pada stock, insert pada table WHFIFO dan update status transaksi dari *open* menjadi *approved.*

   AddStock()

5.2.2 UnPosting Transaksi Penerimaan

Unposting transaksi penerimaan digunakan untuk mengubah status transaksi dari *approved* menjadi *open.* Tidak semua transaksi penerimaan dapat diubah statusnya. Kondisi yang menjadi constraint adalah apakah barang dalam transaksi penerimaan tersebut sudah digunakan (jumlahnya sudah berkurang) atau belum. Jika belum maka transaksi penerimaan dapat dilakukan proses unPosting. Proses unPosting adalah menghapus record pada table WHFIFO sesuai dengan transaksi yang di unPosting, mengubah data stock (termasuk data stock bulanan) dan mengubah status transaksi penerimaan menjadi open. Berikut adalah algoritma untuk unPosting transaksi penerimaan :

1. Mengambil detail dari transaksi yang akan di-unposting

   GetDetailTB()

2. Untuk masing – masing jenis barang dilakukan pengecekan, apakah barang dalam gudang bersangkutan sudah berkurang. Jika belum ada maka dapat dilakukan proses unposting.

   Loop for each product

If (chekcFifoQty() == false)

DeleteFIFO()

UpdateDtStock()

UpdateHdStock()

UpdateStatus()

5.2.3 Posting Transaksi Pengeluaran

Posting untuk transaksi pengeluaran adalah mengurangi stock barang dan mengubah status transaksi dari *open* menjadi *approved.* Posting transaksi pengeluaran dapat dilakukan jika stock barang dalam gudang tersebut cukup. Proses pengambilan barang distock memberlakukan konsep FIFO, yaitu barang yang pertama kali masuk yang dikeluarkan terlebih dahulu. Hal ini juga digunakan untuk menghitung nilai transaksi pengeluaran yang diposting tersebut. Berikut adalah algoritma untuk memposting transaksi pengeluaran :

1. Mengambil detail dari transaksi yang akan diposting

   GetDetailBP()

2. Untuk masing – masing jenis barang dilakukan pengecekan dalam stock, apakah jumlah barang yang ada di stok cukup, jika cukup maka jalankan :
   a. Menjalankan prosedur POP yang mengembalikan nilai nominal barang dan jumlah sisa barang
   b. Mengambil nilai pada stock, yaitu nilai stok akhir, nilai rupiah stok akhir, debet untuk bulan bersangkutan dan nilai rupiah untuk bulan bersangkutan.

c. Melakukan proses update pada stock, table WHFIFO dan update status transaksi dari *open* menjadi *approved.*

Loop for each product

If (checkStock == false)

Jalankan prosedurPOP

CreateBPFIFO()

GetQtyAmt()

MinStock()

Prose penghitungan nilai nominal barang yang dikeluarkan dengan membuat sebuah stored procedure yang mengembalikan nilai nominal barang dan jumlah sisa barang (jika pengambilan barang membutuhkan lebih dari satu record pada WHFIFO). Berikut adalah script pembuatan stored procedure untuk mengurangi FIFO :

```
procedure pop (pWrhs whFifo.Wrhsid%type, pProd whfifo.prodcode%type,
          pQty IN whFifo.Fifoqty%type, pSisa OUT whFifo.Fifoqty%type,
          pAmt OUT whFifo.FifoAmt%type, pDate OUT Whfifo.Trxdate%type) is
  vDate     DATE;
  vQty      NUMBER;
  vPrice    NUMBER;
  vEndQty   NUMBER;
begin
  select min(a.trxdate) into vDate
    from whFifo a
    where a.wrhsid = pWrhs and a.prodcode = pProd
    and a.FifoEndQty > 0; -- yg belum kosong
  select a.FifoQty, a.FifoPrice, a.FifoEndAmt, a.FifoEndQty into vQty, vPrice, pAmt, vEndQty
    from whFifo a
    where a.wrhsid = pWrhs and a.prodcode = pProd and a.trxdate = vDate;
    pDate := vDate;
  if vEndQty >= pQty then
    update whFifo a
      set a.FifoEndQty = a.FifoEndQty - pQty,
```

```
            a.FifoEndAmt = a.FifoEndAmt - (pQty * vPrice)
          where a.trxdate = vDate and a.wrhsid = pWrhs and a.prodcode = pProd;
        pAmt := pQty * vPrice;
        pSisa := 0;
      else
        update whFifo a
          set a.FifoEndQty = 0,
            a.FifoEndAmt = 0
          where a.trxdate = vDate and a.wrhsid = pWrhs and a.prodcode = pProd;
        pSisa := pQty - vEndQty;
      end if;
    end;
```

Posting transaksi pengeluaran ini akan menginsertkan data ke table WHBPFIFO. Data yang diinsertkan diambil dari table WHFIFO, yaitu berisi

- Tanggal dipostingnya transaksi pemasukan barang (bertipekan datetime)
- Jumlah barang yang dikeluarkan
- Nilai nominal perbarang
- Nilai total dari 1 jenis barang

Semua perhitungan mengenai nilai nominal perbarang dan nilai nominal untuk satu jenis barang didapatkan berdasar FIFO. Proses posting transaksi pengeluaran ini juga akan mengubah data transaksi yaitu memasukkan total nilai transaksi untuk masing – masing barang, yang nilainya diperoleh dari total nominal pada table WHBPFIFO untuk masing – masing barang.

5.2.4 UnPosting Transaksi Pengeluaran

Proses unposting transaksi pengeluaran adalah menambah jumlah stock, mengubah data pada table WHFIFO dan mengubah status transaksi dari *approved* menjadi *open*. Berikut adalah algoritma untuk unPosting transaksi penerimaan :

1. Mengambil detail dari transaksi yang akan di-unposting

GetDetailBP()

2. Untuk masing – masing jenis barang dilakukan update FIFO dan stock

   Loop for each product

      UpdateFIFO()

      UpdateDtStock()

      UpdateHdStock()

3. Melakukan penghapusan pada BPFIFO dan pengubahan status transaksi

   DeleteBPFIFO()

   UpdateStatus()

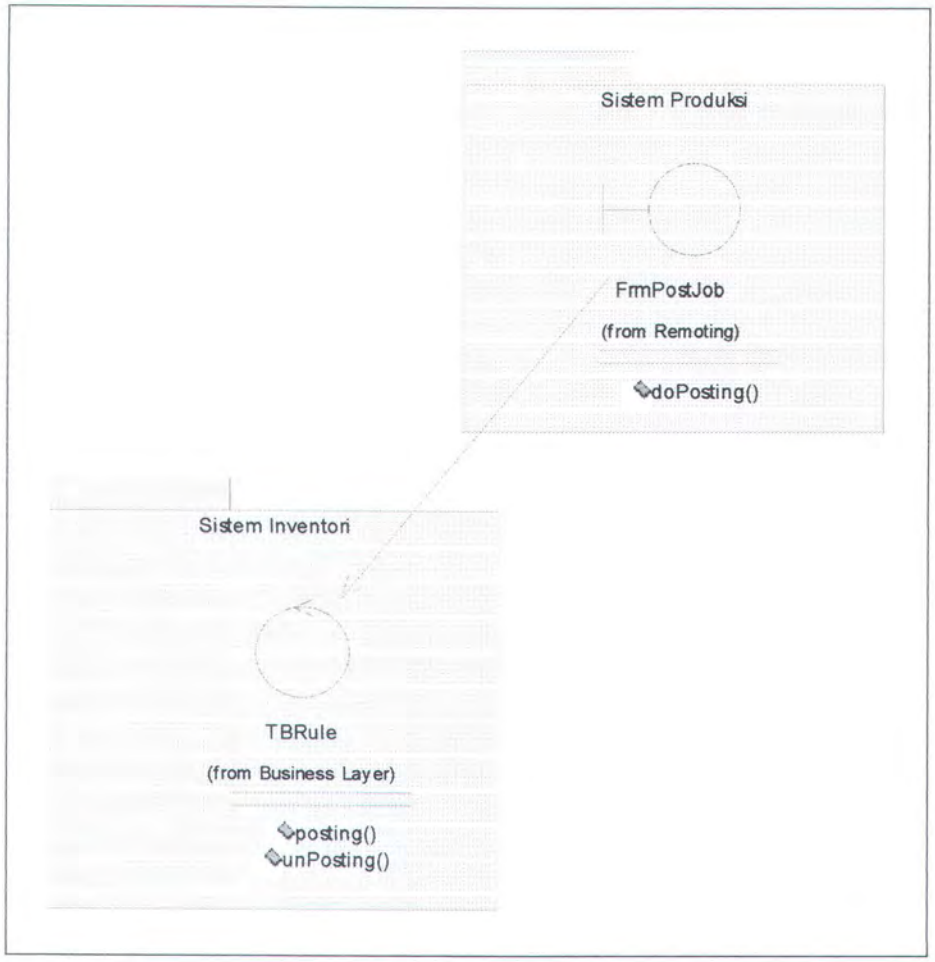
Layer bisnis ini akan diekspos untuk dapat digunakan oleh sistem lain sehingga sistem lain dapat menjalankan fungsi – fungsi yang berada di dalam sistem inventori ini. Gambar 5.1 menjelaskan pemakaian fungsi pada layer bisnis pada sistem inventori oleh sistem produksi. Sistem-sistem yang lain tidak diperkenankan melakukan perubahan atau penambahan data pada sistem inventori kecuali melalui pengaksesan method pada layer bisnis.

Berikut adalah contoh pemanfaatan method pada sistem inventori oleh sistem produksi :

```
private SIS.CAAssignment.BR.BRTB tbRule = new BRTB();

private bool DoPosting()
{
    // Fungsi untuk melakukan posting
    Cursor.Current = Cursors.WaitCursor;
    try
    {
        DataTable dt = new DataTable();
        DataRowView drv = ((DataRowView) this.BindingContext [dvJobProd].Current);
        string pJobId = drv[0].ToString().Trim();
```

```
            string sql, doc, strMonth;
            strMonth = System.DateTime.Now.ToString("yyyyMM");
            sql = "SELECT BPTBDocNmbr DOC FROM PDBPTBDocNmbr WHERE JobId = '" + pJobId + "' AND
BPTBType = 'T'";
            dt = dl.ExecuteQuery(sql);
            for (int i=0; i<dt.Rows.Count; i++)
            {
                        doc = dt.Rows[i][0].ToString();
                        tbRule.Posting(doc);
            }
                        RefreshGrid();
                        break;
              }
            }
      }
```



Gambar 5.1 Pemakaian method pada sistem inventori oleh sistem produksi

5.3 Pembuatan Layer Data Akses

Masing-masing layer data akses mengimplementasi interface IDataPersistence, yang berisi metode–metode untuk mendapatkan data dan menyimpan data dalam basis data.

5.3.1 Pembuatan Interface

Interface yang diimplementasi pada masing – masing *class* pada layer data akses hanya IDataPersistence. Interface IDataPersistence ini dideklarasikan pada IDataPersistence.cs.

```
public interface IDataPersistence
    {
            DataSet Get (string filterCriteria, string sortCriteria);
            DataSet Get (Hashtable filterCriteria, Hashtable sortCriteria);
            int Update (DataSet ds, string updateCriteria);
            DataTable ExecuteQuery (string sqlQuery);
    }
```

Fungsi Get() akan mengisikan dataset dengan data yang diambil berdasar query select dari OleDbDataAdapter, yaitu dengan metode fill() pada OleDbDataAdapter. Sementara fungsi Update() akan mengubah dataset, perubahan bisa terjadi karena insert, update ataupun delete yang kemudian oleDbDataAdapter akan membuat dataset kedua yang hanya berisi perubahan dari dataset asli (dengan memanggil fungsi GetChanges()). OleDbDataAdapter menjalankan *command* sesuai dengan perubahan yang terjadi, jika perubahan adalah penambahan data, maka oleDbDataAdapter mejalankan *insert command,* jika perubahan adalah pengubahan data maka oleDbDataAdabter menjalankan

update command dan jika perubahan adalah penghapusan data maka oleDbDataAdapter menjalankan *delete command.*

5.3.2 Implementasi Interface

Setiap *class* yang mengimplementasikan suatu interface harus mendeklarasikan semua metode yang dimililiki oleh *interface* tersebut.

Masing-masing vlayer data akses yang menimplementasi *interface* IDataPersistence juga mendeklarasikan metode – metode milik IDataPersistence. Berikut adalah script deklarasi metode Update pada FrmTBBeli.

```
public int Update (DataSet ds, string updateCriteria)
{
    //kode disini
}
```

5.4 Pembuatan Laporan

Pembuatan laporan ini dengan menggunakan komponen C1 Report. Untuk mempermudah pembuatan laporan diperlukan view untuk proses pengambilan data. Desain untuk laporan ini disimpan dalam bentuk file .XML. C1 Report akan me-load file XML ini dengan data source dari dataset yang telah diisi sesuai dengan filter yang dipilih oleh user.

5.5 Pembuatan Aplikasi Terdistribusi

Berikut adalah yang perlu dibuat untuk dapat mengimplementasikan aplikasi yang terdistribusi.

1. Hosting pada Internet Information Service (IIS)

   Dilakukan dengan membuat file konfigurasi web. Informasi yang dikonfigurasikan adalah :

   a. Nama obyek

   b. URI dari obyek

   c. Informasi leastime untuk obyek

```
<configuration>
 <system.runtime.remoting>
  <application>
  <lifetime leaseTime="0" />
   <service>
    <wellknown mode="Singleton" type="SIS.DLMaster.DLFactory, DLMaster"
      objectUri="DLFactory.soap" />
   </service>
  </application>
 </system.runtime.remoting>
</configuration>
```

2. Konfigurasi pada client

   Konfigurasi pada client hanya untuk mendeklarasikan nama computer dan virtual direktori dari obyek.

Class – class yang terlibat dalam proses remoting ini adalah :

1. ClientManager.cs : *class* ini berfungsi untuk mendapatkan proxy dari data layer atau bisnis layer yang akan dipanggil. ClientManager ini akan mengembalikan berupa *class* DLFactory dan BRFactory.
2. DLFactory.cs : *class* ini membuat *class* pada data layer yang dibutuhkan oleh aplikai layer ataupun bisnis layer dan akan menyimpan *class* yang telah dibuat dalam hashtable (colecction).

3. BRFactory.cs : *class* ini membuat *class* pada bisnis layer yang dibutuhkan oleh aplikasi layer dan akan menyimpan *class* yang telah dibuat dalam hashtable (collection)

BAB 6
UJI COBA DAN ANALISIS HASIL

Pada bab ini akan dilakukan uji coba terhadap posting transaksi pemasukan dan pengeluaran yang menerapkan metode FIFO, pencetakan laporan dan uji coba untuk aplikasi yang terdistribusi. Uji coba dilakukan untuk mengetahui apakah penerapan metode FIFO menghasilkan data yang valid dan untuk membuktikan penerapan konsep distribusi pada sistem ini.

6.1 Lingkungan Uji Coba

Ujicoba ini dilakukan pada 2 komputer (1 server dan 1 client), dengan spesifikasi sebagai berikut :

- Server Basis Data

  Komputer yang dijadikan sebagai server basis data mempunyai spesifikasi sebagai berikut :

  - Nama Komputer : Nitnot
  - IP Address : 192.168.0.81
  - Prosesor : Pentium 3 770 MHz
  - Memori fisik : 256 MB

- Client

  - Nama Komputer : Velocis
  - IP Address : 192.168.0.82

- Prosesor : Pentium 3 MMX 566 MHz
- Memori fisik : 320 MB

6.2 Skenario Uji Coba

Pada uji coba ini dilakukan proses memasukkan data transaksi yaitu transaksi penerimaan lain – lain dan transaksi pengeluaran lain – lain. Data – data transaksi yang telah dimasukkan kemudian di posting. Dalam ujicoba ini juga akan dicobakan unPosting untuk transaksi penerimaan dan pengeluaran. Tujuan dari uji coba ini adalah untuk mengetahui kevalidan data.

6.2.1 Uji Coba dan Analisis Transaksi Penerimaan

Dalam tahap ini dilakukan proses memasukkan data transaksi yaitu transaksi penerimaan lain – lain. Kemudian transaksi-transaksi tersebut di posting dan di unPosting untuk mengetahui kevalidan data pada proses penerimaan barang.

6.2.1.1 Persiapan Data Penerimaan

Membuat transaksi penerimaan lain-lain dengan menginputkan beberapa detail barang. Berikut adalah data yang diinputkan :

1. Test 1

   Gudang : D004 (Gudang Penolong)

   No Transaksi : 0303.06.PL.0001

   Tanggal transaksi : 12 Maret 2003

   Jenis transaksi : Penerimaan lain – lain (trxId = 06)

Jenis barang :

Kode Barang	Qty	Satuan	Harga	Harga Satuan
A3240015R	35	Box	70000	2000
A3240015Z	24	Carton	2400	100
A3240030R	3	Bks	90000	30000
A3240030Z	8	Bks	64000	8000

2. Test 2

Gudang : D004 (Gudang Penolong)

No Transaksi : 0303.06.PL.0002

Tanggal transaksi : 13 Maret 2003

Jenis transaksi : Penerimaan lain – lain (trxId = 06)

Jenis barang :

Kode Barang	Qty	Satuan	Harga	Harga Satuan
A3240015R	35	Box	70000	2000
A3240015Z	24	Carton	2400	100
A3240030R	3	Bks	90000	30000
A3240030Z	8	Bks	64000	8000

3. Test 3

Gudang : D004 (Gudang Penolong)

No Transaksi : 0303.01.PL.0001

Tanggal transaksi : 12 Maret 2003

Jenis transaksi : Penerimaan pembelian (trxId = 01)

Jenis barang :

Kode Barang	Qty	Satuan	Harga	Harga Satuan
A3240015R	43	Box	43000	1000
A3240015Z	30	Carton	15000	500
A3240030R	2	Bks	5000	2500
A3240030Z	4	Bks	4000	1000

6.2.1.2 Posisi Stok dan FIFO

Dalam tahap ini akan dilakukan posting terhadap masing – masing transaksi yang telah diinsertkan dengan data tersebut di atas.

1. Posisi Stock dan FIFO Awal

- Stock

Posisi awal barang dengan kode A3240015R, A3240015Z, A3240030R, A3240030R untuk gudang D004, masing-masing adalah 0. Untuk mengetahui posisi stock ini digunakan sub menu Monitor Stock dari menu Stock. Dalam ujicoba ini digunakan query dalam PLSQL.

```
select wrhsid, prodcode, stockqty, b.ivlmonth, b.ivlsaldo, b.ivldb, b.ivlcd, b.ivlsaldopri, b.ivldbpri, b.ivlcdpri
from whstock a, table(a.whmonthly)b order by prodcode
```

Hasil query tersebut menghasilkan 0 row.

- FIFO

Tiap barang yang diposting akan menginsertkan 1 row dalam table WHFIFO. Sebelum ada proses posting tidak ada row dalam table WHFIFO.

Untuk mengetahui data barang dalam table WHFIFO digunakan sub menu Monitor Stock dari menu Stock. Dalam ujicoba ini digunakan query:

```
select a.wrhsid, a.prodcode, a.trxdate, a.fifoqty, a.fifoprice, a.fifoendqty, a.fifoendamt--, c.sumqty, c.sumamt
from whfifo a,
(select wrhsid, prodcode, sum(fifoendqty) sumqty, sum(fifoendamt) sumamt from whfifo
group by prodcode, wrhsid) c
where a.wrhsid = c.wrhsid and a.prodcode = c.prodcode
order by a.prodcode
```

Hasil query tersebut juga menghasilkan 0 row.

2. Posisi Stock Setelah Posting

Dalam bagian ini akan diperlihatkan perubahan posisi stock dan FIFO setelah dilakukan posting atas transaksi – transaksi yang telah diinputkan.

- Posisi Stock dan FIFO Setelah Posting Test 1

Dengan menggunakan query yang sama diperoleh data :

WRHSID	PRODCODE	STOCKQTY	IVLMONTH	IVLSALDO	IVLDB	IVLCD	IVLSALDOPRI	IVLDBPRI	IVLCDPRI
D004	A3240015R	7	200303	7	7	0	6300	6300	0
D004	A3240015Z	10	200303	10	10	0	50000	50000	0
D004	A3240030R	9	200303	9	9	0	7200	7200	0
D004	A3240030Z	18	200303	18	18	0	36000	36000	0

WRHSID	PRODCODE	TRXDATE	FIFOQTY	FIFOPRICE	FIFOENDQTY	FIFOENDAMT
D004	A3240015R	5/28/2003 6:22:09 PM	7	900	7	6300
D004	A3240015Z	5/28/2003 6:22:09 PM	10	5000	10	50000
D004	A3240030R	5/28/2003 6:22:09 PM	9	800	9	7200
D004	A3240030Z	5/28/2003 6:22:09 PM	18	2000	18	36000

Gambar 6.1 Posisi stock dan FIFO setelah posting tes 1

- Posisi Stock dan FIFO Setelah Posting Test 2

Dengan menggunakan query yang sama diperoleh data :

WRHSID	PRODCODE	STOCKQTY	IVLMONTH	IVLSALDO	IVLDB	IVLCD	IVLSALDOPRI	IVLDBPRI	IVLCDPRI
D004	A3240015R	42	200303	42	42	0	76300	76300	0
D004	A3240015Z	34	200303	34	34	0	52400	52400	0
D004	A3240030R	12	200303	12	12	0	97200	97200	0
D004	A3240030Z	26	200303	26	26	0	100000	100000	0

WRHSID	PRODCODE	TRXDATE	FIFOQTY	FIFOPRICE	FIFOENDQTY	FIFOENDAMT
D004	A3240015R	5/28/2003 6:22:09 PM	7	900	7	6300
D004	A3240015R	5/28/2003 6:24:22 PM	35	2000	35	70000
D004	A3240015Z	5/28/2003 6:22:09 PM	10	5000	10	50000
D004	A3240015Z	5/28/2003 6:24:22 PM	24	100	24	2400
D004	A3240030R	5/28/2003 6:22:09 PM	9	800	9	7200
D004	A3240030R	5/28/2003 6:24:22 PM	3	30000	3	90000
D004	A3240030Z	5/28/2003 6:22:09 PM	18	2000	18	36000
D004	A3240030Z	5/28/2003 6:24:22 PM	8	8000	8	64000

Gambar 6.2 Posisi stock dan FIFO setelah posting tes 2

Pada posting test 2 diperoleh :

StockQty = 42 untuk kode barang A3240015R dan gudang D004. Nilai ini akan sama dengan jumlah FifoEndQty untuk kode barang A3240015R dan gudang D004 (7 + 35).

- Posisi Stock dan FIFO Setelah Posting Test 3

Dengan menggunakan query yang sama diperoleh data :

WRHSID	PRODCODE	STOCKQTY	IVLMONTH	IVLSALDO	IVLDB	IVLCD	IVLSALDOPRI	IVLDBPRI	IVLCDPRI
D004	A3240015R	85	200303	85	85	0	119300	119300	0
D004	A3240015Z	64	200303	64	64	0	67400	67400	0
D004	A3240030R	14	200303	14	14	0	102200	102200	0
D004	A3240030Z	30	200303	30	30	0	104000	104000	0

WRHSID	PRODCODE	TRXDATE	FIFOQTY	FIFOPRICE	FIFOENDQTY	FIFOENDAMT
D004	A3240015R	5/28/2003 6:22:09 PM	7	900	7	6300
D004	A3240015R	5/28/2003 6:24:22 PM	35	2000	35	70000
D004	A3240015R	5/28/2003 6:25:35 PM	43	1000	43	43000
D004	A3240015Z	5/28/2003 6:22:09 PM	10	5000	10	50000
D004	A3240015Z	5/28/2003 6:24:22 PM	24	100	24	2400
D004	A3240015Z	5/28/2003 6:25:35 PM	30	500	30	15000
D004	A3240030R	5/28/2003 6:22:09 PM	9	800	9	7200
D004	A3240030R	5/28/2003 6:24:22 PM	3	30000	3	90000
D004	A3240030R	5/28/2003 6:25:35 PM	2	2500	2	5000
D004	A3240030Z	5/28/2003 6:22:09 PM	18	2000	18	36000
D004	A3240030Z	5/28/2003 6:24:22 PM	8	8000	8	64000
D004	A3240030Z	5/28/2003 6:25:35 PM	4	1000	4	4000

Gambar 6.3 Posisi stock dan FIFO setelah posting tes 3

Pada posting test 3 diperoleh :

StockQty = 85 untuk kode barang A3240015R dan gudang D004. Nilai ini akan sama dengan jumlah FifoEndQty untuk kode barang A3240015R dan gudang D004 (7 + 35 + 43).

3. Posisi Stock Setelah UnPosting

Dalam tahap ini akan dicoba untuk melakukan proses unPosting transaksi penerimaan. Akan dicobakan transaksi test 2 (no transaksi = 0303.06.PL.0002). Setelah dilakukan proses unPosting, diperoleh data :

WRHSID	PRODCODE	STOCKQTY	IVLMONTH	IVLSALDO	IVLDB	IVLCD	IVLSALDOPRI	IVLDBPRI	IVLCDPRI
D004	A3240015R	50	200303	50	50	0	49300	49300	0
D004	A3240015Z	40	200303	40	40	0	65000	65000	0
D004	A3240030R	11	200303	11	11	0	12200	12200	0
D004	A3240030Z	22	200303	22	22	0	40000	40000	0

WRHSID	PRODCODE	TRXDATE	FIFOQTY	FIFOPRICE	FIFOENDQTY	FIFOENDAMT
D004	A3240015R	5/28/2003 6:22:09 PM	7	900	7	6300
D004	A3240015R	5/28/2003 6:25:35 PM	43	1000	43	43000
D004	A3240015Z	5/28/2003 6:22:09 PM	10	5000	10	50000
D004	A3240015Z	5/28/2003 6:25:35 PM	30	500	30	15000
D004	A3240030R	5/28/2003 6:22:09 PM	9	800	9	7200
D004	A3240030R	5/28/2003 6:25:35 PM	2	2500	2	5000
D004	A3240030Z	5/28/2003 6:22:09 PM	18	2000	18	36000
D004	A3240030Z	5/28/2003 6:25:35 PM	4	1000	4	4000

Gambar 6.4 Posisi stock dan FIFO setelah unPosting

Setelah dilakukan proses unPosting untuk transaksi 0303.06.PL.0002, maka nilai stock menjadi 50.

4. Posisi Stock Setelah Re-Posting

Dalam tahap ini akan dilakukan posting ulang transaksi yang telah diunposting, yaitu transaksi nomer 0303.06.PL.0002. Setelah re-posting, didapatkan data sebagai berikut :

WRHSID	PRODCODE	STOCKQTY	IVLMONTH	IVLSALDO	IVLDB	IVLCD	IVLSALDOPRI	IVLDBPRI	IVLCDPRI
D004	A3240015R	85	200303	85	85	0	119300	119300	0
D004	A3240015Z	64	200303	64	64	0	67400	67400	0
D004	A3240030R	14	200303	14	14	0	102200	102200	0
D004	A3240030Z	30	200303	30	30	0	104000	104000	0

WRHSID	PRODCODE	TRXDATE	FIFOQTY	FIFOPRICE	FIFOENDQTY	FIFOENDAMT
D004	A3240015R	5/28/2003 6:22:09 PM	7	900	7	6300
D004	A3240015R	5/28/2003 6:25:35 PM	43	1000	43	43000
D004	A3240015R	5/28/2003 6:27:50 PM	35	2000	35	70000
D004	A3240015Z	5/28/2003 6:22:09 PM	10	5000	10	50000
D004	A3240015Z	5/28/2003 6:25:35 PM	30	500	30	15000
D004	A3240015Z	5/28/2003 6:27:50 PM	24	100	24	2400
D004	A3240030R	5/28/2003 6:22:09 PM	9	800	9	7200
D004	A3240030R	5/28/2003 6:25:35 PM	2	2500	2	5000
D004	A3240030R	5/28/2003 6:27:50 PM	3	30000	3	90000
D004	A3240030Z	5/28/2003 6:22:09 PM	18	2000	18	36000
D004	A3240030Z	5/28/2003 6:25:35 PM	4	1000	4	4000
D004	A3240030Z	5/28/2003 6:27:51 PM	8	8000	8	64000

Gambar 6.5 Posisi stock dan FIFO setelah re-posting

Setelah proses posting dilakukan, maka stock kembali menjadi = 85. Namun terdapat perbedaan antara posting pertama dengan posting setelah proses unPosting. Pada posting pertama urutan data pada WHFIFO adalah sebagai berikut :

WRHSID	PRODCODE	TRXDATE	FIFOQTY	FIFOPRICE	FIFOENDQTY	FIFOENDAMT
D004	A3240015R	5/28/2003 6:22:09 PM	7	900	7	6300
D004	A3240015R	5/28/2003 6:24:22 PM	35	2000	35	70000
D004	A3240015R	5/28/2003 6:25:35 PM	43	1000	43	43000

Gambar 6.6 Urutan data pada WHFIFO dari hasil posting I

Setelah proses unPosting urutan data dalam WHFIFO adalah sebagai berikut :

WRHSID	PRODCODE	TRXDATE	FIFOQTY	FIFOPRICE	FIFOENDQTY	FIFOENDAMT
D004	A3240015R	5/28/2003 6:22:09 PM	7	900	7	6300
D004	A3240015R	5/28/2003 6:25:35 PM	43	1000	43	43000
D004	A3240015R	5/28/2003 6:27:50 PM	35	2000	35	70000

Gambar 6.7 Urutan data pada WHFIFO setelah proses unPosting

6.2.2 Uji Coba dan Analisis Transaksi Pengeluaran

Dalam tahap ini dilakukan proses memasukkan data transaksi yaitu transaksi pengeluaran lain – lain. Kemudian transaksi-transaksi tersebut di posting dan di unPosting untuk mengetahui kevalidan data pada proses pengeluaran barang.

6.2.2.1 Persiapan Data Pengeluaran

Membuat transaksi penerimaan lain-lain dengan menginputkan beberapa detail barang. Berikut adalah data yang diinputkan :

1. Test 4

   Gudang : D004 (Gudang Penolong)

   No Transaksi : 0303.19.PL.0001

   Tanggal transaksi : 18 Maret 2003

   Jenis transaksi : Pengeluaran lain-lain (trxId = 19)

   Jenis barang :

Kode Barang	Qty	Satuan
A3240015R	36	Box

2. Test 5

   Gudang : D004 (Gudang Penolong)

   No Transaksi : 0303.19.PL.0002

   Tanggal transaksi : 19 Maret 2003

   Jenis transaksi : Pengeluaran lain-lain (trxId = 19)

   Jenis barang :

Kode Barang	Qty	Satuan
A3240015R	15	Box

6.2.2.2 Posisi Stock dan FIFO

Dalam tahap ini akan dilakukan posting terhadap masing – masing transaksi yang telah diinsertkan dengan data tersebut di atas.

1. Posisi Stock Setelah Posting

Dalam tahap ini akan dilakukan posting untuk masing – masing transaksi dengan data seperti tersebut di atas.

- Posisi Stock dan FIFO Setelah Posting Test 4

Data yang diperoleh setelah melakukan posting test 4 adalah :

WRHSID	PRODCODE	STOCKQTY	IVLMONTH	IVLSALDO	IVLDB	IVLCD	IVLSALDOPRI	IVLDBPRI	IVLCDPRI
D004	A3240015R	49	200303	49	85	36	84000	119300	35300
D004	A3240015Z	64	200303	64	64	0	67400	67400	0
D004	A3240030R	14	200303	14	14	0	102200	102200	0
D004	A3240030Z	30	200303	30	30	0	104000	104000	0

WRHSID	PRODCODE	TRXDATE	FIFOQTY	FIFOPRICE	FIFOENDQTY	FIFOENDAMT
D004	A3240015R	5/28/2003 6:22:09 PM	7	900	0	0
D004	A3240015R	5/28/2003 6:25:35 PM	43	1000	14	14000
D004	A3240015R	5/28/2003 6:27:50 PM	35	2000	35	70000
D004	A3240015Z	5/28/2003 6:22:09 PM	10	5000	10	50000
D004	A3240015Z	5/28/2003 6:25:35 PM	30	500	30	15000
D004	A3240015Z	5/28/2003 6:27:50 PM	24	100	24	2400
D004	A3240030R	5/28/2003 6:22:09 PM	9	800	9	7200
D004	A3240030R	5/28/2003 6:25:35 PM	2	2500	2	5000
D004	A3240030R	5/28/2003 6:27:50 PM	3	30000	3	90000
D004	A3240030Z	5/28/2003 6:22:09 PM	18	2000	18	36000
D004	A3240030Z	5/28/2003 6:25:35 PM	4	1000	4	4000
D004	A3240030Z	5/28/2003 6:27:51 PM	8	8000	8	64000

Gambar 6.8 Posisi stock dan FIFO setelah posting tes 4

Setelah dilakukan posting didapatkan nilai stock = 49, dimana nilai ini sama dengan jumlah FIFOENDQTY untuk barang dan gudang yang sama (14 + 35) dan sama dengan IVLDB – IVLCD (85 - 36). Sementara IVLDBPRI, yaitu nilai nominal yang dikeluarkan adalah sebesar 35300.

Nilai nominal itu didapatkan dari: (7 * 900) + (29 * 1000) = 6300 + 29000 = 35300.

Detail penghitungan nominal ini disimpan dalam table WHBPFIFO untuk masing – masing barang.

TRXINDATE	TRXQTY	TRXPRICE	TRXAMT
5/28/2003 6:22:09 PM	7	900	6300
5/28/2003 6:25:35 PM	29	1000	29000

Gambar 6.9 Detail penghitungan nominal pada WHBPFIFO

Data tersebut di peroleh dari query :

```
select c.trxindate, c.trxqty, c.trxprice, c.trxamt
from whbphd a, table(a.whbpdt) b, table(b.bpfifo) c
where a.bpid = '0303.19.PL.0001'
```

- Posisi Stock dan FIFO Setelah Posting Test 5

Data yang diperoleh setelah melakukan posting test 5 adalah :

WRHSID	PRODCODE	STOCKQTY	IVLMONTH	IVLSALDO	IVLDB	IVLCD	IVLSALDOPRI	IVLDBPRI	IVLCDPRI
D004	A3240015R	34	200303	34	85	51	68000	119300	51300
D004	A3240015Z	64	200303	64	64	0	67400	67400	0
D004	A3240030R	14	200303	14	14	0	102200	102200	0
D004	A3240030Z	30	200303	30	30	0	104000	104000	0

WRHSID	PRODCODE	TRXDATE	FIFOQTY	FIFOPRICE	FIFOENDQTY	FIFOENDAMT
D004	A3240015R	5/28/2003 6:22:09 PM	7	900	0	0
D004	A3240015R	5/28/2003 6:25:35 PM	43	1000	0	0
D004	A3240015R	5/28/2003 6:27:50 PM	35	2000	34	68000
D004	A3240015Z	5/28/2003 6:22:09 PM	10	5000	10	50000
D004	A3240015Z	5/28/2003 6:25:35 PM	30	500	30	15000
D004	A3240015Z	5/28/2003 6:27:50 PM	24	100	24	2400
D004	A3240030R	5/28/2003 6:22:09 PM	9	800	9	7200
D004	A3240030R	5/28/2003 6:25:35 PM	2	2500	2	5000
D004	A3240030R	5/28/2003 6:27:50 PM	3	30000	3	90000
D004	A3240030Z	5/28/2003 6:22:09 PM	18	2000	18	36000
D004	A3240030Z	5/28/2003 6:25:35 PM	4	1000	4	4000
D004	A3240030Z	5/28/2003 6:27:51 PM	8	8000	8	64000

Gambar 6.10 Posisi stock dan FIFO setelah posting tes 5

Setelah proses posting test 5, stock qty adalah 34 yang sama dengan nilai FIFOENDQTY (=34), dan nilai nominal barang yang tersedia adalah

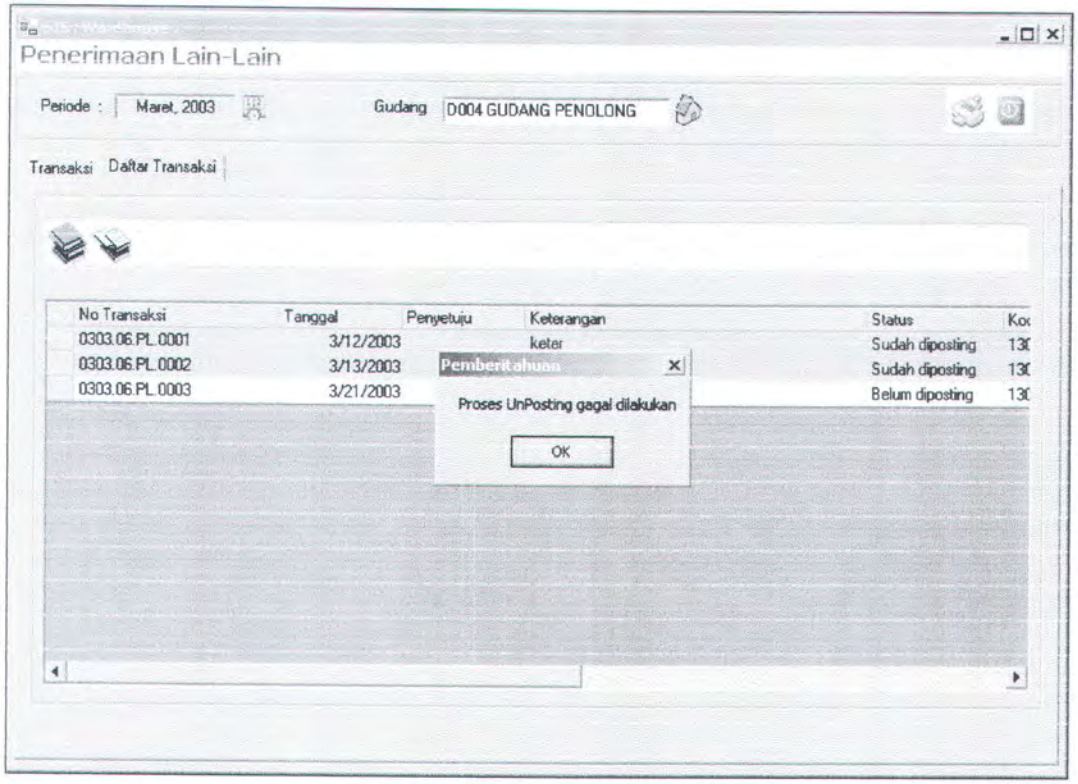
68000. Data penghitungan nominal hasil posting transaksi 0303.19.PL.0002 adalah :

TRXINDATE	TRXQTY	TRXPRICE	TRXAMT
5/28/2003 6:25:35 PM	14	1000	14000
5/28/2003 7:38:27 PM	1	2000	2000

Gambar 6.11 Posisi Penghitungan nominal pada WHBPFIFO

Nilai tersebut menghasilkan jumlah 16000. Sehingga IVLCDPRI = 51300 itu diperoleh dari jumlah nominal transaksi 0303.19.PL.0001 + jumlah nominal transaksi 0303.19.PL.0002 (= 35300 + 16000).

List ke 3 untuk barang A3240015R pada gudang D004 adalah berasal dari transaksi dengan nomer = 0303.06.PL.0002. Maka transaksi tersebut tidak dapat lagi di unposting karena barang sudah diambil walaupun yang diambil hanya 1.



Gambar 6.12 Pesan kegagalan proses unPosting

2. Posisi Stock dan FIFO Setelah UnPosting

Data yang tersedia setelah dilakukan proses unPosting tes 5 adalah :

WRHSID	PRODCODE	STOCKQTY	IVLMONTH	IVLSALDO	IVLDB	IVLCD	IVLSALDOPRI	IVLDBPRI	IVLCDPRI
D004	A3240015R	49	200303	49	85	36	84000	119300	35300
D004	A3240015Z	64	200303	64	64	0	67400	67400	0
D004	A3240030R	14	200303	14	14	0	102200	102200	0
D004	A3240030Z	30	200303	30	30	0	104000	104000	0

WRHSID	PRODCODE	TRXDATE	FIFOQTY	FIFOPRICE	FIFOENDQTY	FIFOENDAMT
D004	A3240015R	5/28/2003 6:22:09 PM	7	900	0	0
D004	A3240015R	5/28/2003 6:25:35 PM	43	1000	14	14000
D004	A3240015R	5/28/2003 6:27:50 PM	35	2000	35	70000
D004	A3240015Z	5/28/2003 6:22:09 PM	10	5000	10	50000
D004	A3240015Z	5/28/2003 6:25:35 PM	30	500	30	15000
D004	A3240015Z	5/28/2003 6:27:50 PM	24	100	24	2400
D004	A3240030R	5/28/2003 6:22:09 PM	9	800	9	7200
D004	A3240030R	5/28/2003 6:25:35 PM	2	2500	2	5000
D004	A3240030R	5/28/2003 6:27:50 PM	3	30000	3	90000
D004	A3240030Z	5/28/2003 6:22:09 PM	18	2000	18	36000
D004	A3240030Z	5/28/2003 6:25:35 PM	4	1000	4	4000
D004	A3240030Z	5/28/2003 6:27:51 PM	8	8000	8	64000

Gambar 6.13 Posisi stock dan FIFO setelah unPosting tes 5

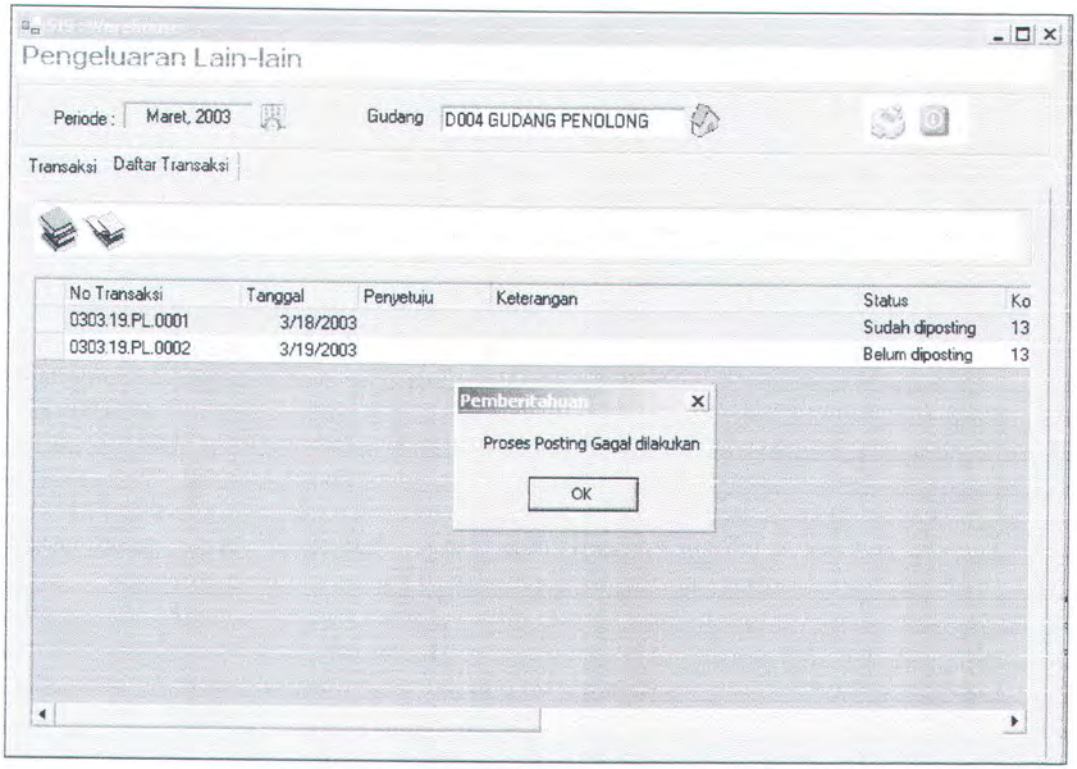
Posisi stock dan FIFO kembali seperti semula sebelum test 5 diposting. Dalam kondisi seperti ini, transaksi penerimaan dengan nomer 0303.06.PL.0002 bisa di unPosting, maka posisi stock dan FIFO setelah transaksi 0303.06.PL.0002 di unPosting adalah :

WRHSID	PRODCODE	STOCKQTY	IVLMONTH	IVLSALDO	IVLDB	IVLCD	IVLSALDOPRI	IVLDBPRI	IVLCDPRI
D004	A3240015R	14	200303	14	50	36	14000	49300	35300
D004	A3240015Z	40	200303	40	40	0	65000	65000	0
D004	A3240030R	11	200303	11	11	0	12200	12200	0
D004	A3240030Z	22	200303	22	22	0	40000	40000	0

WRHSID	PRODCODE	TRXDATE	FIFOQTY	FIFOPRICE	FIFOENDQTY	FIFOENDAMT
D004	A3240015R	5/28/2003 6:22:09 PM	7	900	0	0
D004	A3240015R	5/28/2003 6:25:35 PM	43	1000	14	14000
D004	A3240015Z	5/28/2003 6:22:09 PM	10	5000	10	50000
D004	A3240015Z	5/28/2003 6:25:35 PM	30	500	30	15000
D004	A3240030R	5/28/2003 6:22:09 PM	9	800	9	7200
D004	A3240030R	5/28/2003 6:25:35 PM	2	2500	2	5000
D004	A3240030Z	5/28/2003 6:22:09 PM	18	2000	18	36000
D004	A3240030Z	5/28/2003 6:25:35 PM	4	1000	4	4000

Gambar 6.14 Posisi stock dan FIFO setelah unPosting tes 2

Pada kondisi seperti ini, transaksi pengeluaran dengan nomer : 0303.19.PL.0002 tidak dapat diposting, karena stock tidak mencukupi.



Gambar 6.15 Pesan kegagalan proses posting

6.2.3 Uji Coba dan Analisis Pencetakan Laporan

Tujuan dari uji coba ini adalah apakah hasil laporan yang akan dicetak sudah sesuai dengan hasil yang diharapkan.

1. Laporan Kartu Stock

Berdasarkan transaksi yang sudah dimasukkan, maka dapat dicetak laporan kartu stock yang menunjukkan alur penerimaan dan pengeluaran barang sebagai berikut :

SYSTEM COST- ACCOUNTING

KARTU STOK FINISH GOOD

Periode: Maret 2003

Gudang : GUDANG PENCLONG

Produk : 2 JOIN WING 30/40 (FRESH)

Tanggal	No.Bukti	(Masuk)			(Keluar)			(Saldo)		
		Kg	Piece	Rp	Kg	Piece	Rp	Kg	Piece	Rp
12/03/2003	0303.01.PL.0001	43		43,000.00	0	0	0.00	43	0	43000
12/03/2003	0303.06.PL.0001	7		6,300.00	0	0	0.00	50	0	49300
13/03/2003	0303.06.PL.0002	35		70,000.00	0	0	0.00	85	0	119300
18/03/2003	0303.19.PL.0001	0	0	0.00	36	0	35,300.00	49	0	84000
	Saldo Akhir							49	0	84000

Gambar 6.16 Format laporan kartu stock

2. Laporan Posisi Persediaan Barang

Laporan ini ditujuakan untuk menunjukkan posisi persediaan barang adalah sebagai berikut :

SYSTEM COST- ACCOUNTING

POSISI PERSEDIAAN BARANG

Gudang : GUDANG PENOLONG

Kode Produk	Nama Produk	Saldo Akhir	Satuan	Rupiah	Umur
A8240015R	2 JOIN WING 30/40 (FRESH)	49	Kg	84000	2 hari
A8240015Z	2 JOIN WING 30/40 (FROZEN)	64	Kg	67400	2 hari
A8240030R	2 JOIN WING 40/50 (FRESH)	14	Kg	102200	2 hari
A8240030Z	2 JOIN WING 40/50 (FROZEN)	30	Kg	104000	2 hari
		Total :		357600	

Gambar 6.17 Format laporan persediaan barang

Jumlah akhir untuk barang dengan kode A3240015R untuk gudang D004 (Gudang Penolong) pada laporan posisi persediaan barang adalah sama dengan kartu stock untuk barang kode A3240015R dan untuk gudang D004 (Gudang Penolong), yaitu jumlah barang = 49 dan jumlah nominal = 84000.

BAB 7
PENUTUP

Bab ini berisi beberapa kesimpulan dari tugas akhir ini. Beberapa kesimpulan yang dapat diambil dari tugas akhir ini adalah sebagai berikut :

1. ORDBMS dapat dimanfaatkan pada sistem inventori terdistribusi.
2. Fitur ORDBMS pada Oracle 9i versi 9.2.0.1 ini masih prematur karena ada *error* yang belum bisa ditemukan solusinya.
3. Penangananan mekanisme FIFO dalam sistem inventori terdistribusi ini dilakukan dengan meletakkan metode pada layer bisnis (memakai C#).
4. Adanya keterbatasan Microsoft ADO.NET dalam mengakomodir struktur Oracle ORDBMS juga menyebabkan kekuatan masing-masing kurang bisa termanfaatkan secara penuh.
5. *Interface* antar sistem dibuat melalui *class* dari aplikasi yang berada pada layer bisnis, yang dapat digunakan oleh sistem lain sehingga sistem lain dapat menjalankan fungsi – fungsi yang berada di dalam sistem inventori ini.

DAFTAR PUSTAKA

[1] Connolly, Thomas. *Database System. A Practical Design to Design, Implementation & Management*. Addison-Wesley. 1999.

[2] Dorsey, D.R Paul. *Oracle8 Design Using UML Object Modeling*. McGraw-Hill. California : 1999

[3] Richter, Charles. *Designing Flexible Object-Oriented System With UML*.. Macmillan Technical Publishing.1999

[4] Liberty, Jesse. *Programming C#*. O'Reilly. 2001

[5] Gietz, William. *Oracle9i Application Developer's Guides - Object Relational Features, Release 2(9.2)*. Oracle Corporation, 1996

[6] Lee, Geof. *Simple Strategies for Complex Data: Oracle9i Object-Relational Technology*. Oracle Corporation, January 2002

[7] Devarakonde, Ramakanth S. *Object – Relational Database Systems – The Road Ahed,* http://www.acm.org/crossroads /xrds7-3/ordbms.html, Februay 2001

[8] Microsoft Corporation, *Microsoft ® Developer Netwotk Library*, October 2002

LAMPIRAN A

Pada lampiran A ini akan menjelaskan istilah – istilah yang dipakai dalam system inventori ini.

Berikut ini adalah daftar istilah yang digunakan dalam sistem inventori ini.

1. **Gudang Unloading**

   Gudang penerimaan ayam hidup

2. **Gudang**

   Gudang-gudang yang menerima dan mengeluarkan barang selain ayam hidup

3. **Killing**

   Proses dalam produksi yang menyembelih ayam hidup, sampai dengan pembersihan kulit dan pemotongan kaki ayam

4. **Eviscerating**

   Proses dalam produksi yang memisahkan ayam dengan jeroannya

5. **Chilling Tank**

   Proses dalam produksi untuk pentirisan ayam dan seleksi grade ayam

6. **Cut Up**

   Proses produksi untuk memotong ayam, misal sayap, paha, boneless dll

7. **Rendering**

   Proses produksi untuk menghancurkan bulu ayam agar dapat dimanfaatkan ulang

8. LPAH

Kepanjangan dari Laporan Penerimaan Ayam Hidup, istilah ini dapat disamakan dengan penerimaan ayam hidup

9. Mati tunggu

Ayam yang mati pada waktu menunggu proses produksi

10. Produk

Barang – barang yang terlibat dalam proses penerimaan dan pengeluaran barang

11. Unit of Measurement (UOM)

Satuan untuk barang

12. Kode Transaksi (TrxId)

Kode transaksi untuk masing-masing transaksi penerimaan dan pengeluaran barang. Masing – masing kode transaksi memiliki penanda apakah transaksi tersebut merupakan transaksi penerimaan atau transaksi pengeluaran barang.

13. Kode Perkiraan Transaksi (GLM)

Kode Cost of Accounting (COA) dari masing-masing transaksi penerimaan dan pengeluaran. Setiap transaksi penerimaan maupun pengeluaran dilengkapi dengan kode perkiraan ini dimaksutkan untuk keperluan Cost Accounting (General Ledger)

14. Responsibility Center (GLS)

Informasi unit penanggung jawab dari masing – masing transaksi yang terjadi.

15. GL Maping (GLMAP)

Maping antara kode perkiraan dengan unit yang bertanggung jawab atasnya

16. Relasi

Data dari relasi, baik itu customer maupun supplier

17. Purchase Order

Informasi mengenai pemesanan barang

18. SPM

Surat Perintah Muat, yaitu surat perintah pengiriman barang

LAMPIRAN B

Use Case Specification: Meng-inputkan Penerimaan Barang Pembelian

1. Meng-inputkan Penerimaan Barang Pembelian

1.1 Brief Description:

Tujuan dari tugas ini adalah mencatat seluruh transaksi penerimaan pembelian barang

2. Flow of Event

2.1 Basic Flow:

1. User memilih kode gudang
2. Sistem men-generate nomor Penerimaan Barang
3. Status penerimaan menjadi 'OPEN'
4. Sistem memasukkan tanggal default penerimaan
5. User Gudang memilih nomor PO
6. User Gudang memasukkan data barang dan jumlah yang diterima.
7. User Gudang mencetak BPB dan memintakan persetujuan ke bagian yang berwenang.
8. User Gudang memasukkan nama penyetuju.
9. Status penerimaan menjadi 'APPROVED'
10. Sistem menambah stok barang

2.2 Alternate Flow:

[4.a] Tanggal penerimaan tidak sama dengan tanggal default

1. User Gudang memasukkan tanggal penerimaan

[7.a] BPB harus di-edit

1. User Gudang memperbarui BPB
2. User Gudang mencetak ulang BPB dan memintakan persetujuan ke pihak yang berwenang.

[7-10.a] BPB tidak disetujui

1. User Gudang membatalkan penerimaan barang dan BPB.
2. Status penerimaan menjadi 'OPEN'
3. Sistem memperbarui stok barang.
4. User menghapus data transaksi

3. Special Requirement

None.

4. Pre-Condition

Kualitas barang memenuhi standar.

5. Post-Condition

BPB telah dibuat dan siap diberikan kepada supplier dan salinannya didistribusikan kepada pihak yang terkait.

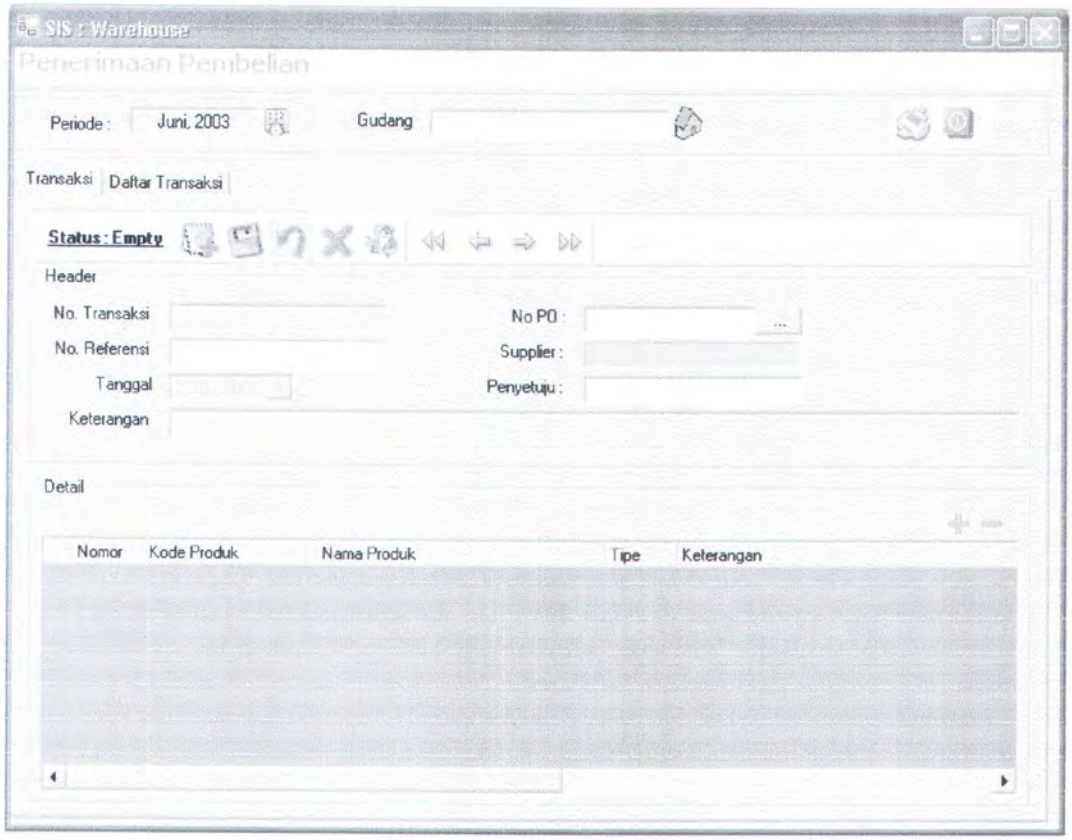
6. Extension Points

None.

7. Document Sources

None.

8. Screen Layout



Use Case Specification: Menerima LPAH

1. Menerima LPAH

1.1 Brief Description :

Tujuan dari tugas ini adalah untuk menyimpan informasi transaksi Penerimaan ayam hidup.

2. Flow of Event

2.1 Basic Flow:

1. Sistem memasukkan kode gudang
2. Sistem men-generate nomor penerimaan barang other.
3. Status bukti menjadi 'OPEN'
4. Sistem memasukkan tanggal default penerimaan
5. User Gudang memasukkan data penerimaan
6. User Gudang memilih data barang yang diterima
7. User Gudang memasukkan jumlah barang yang akan diterima

User Gudang mengulangi langkah 6-7 sampai seluruh data barang dimasukkan.

8. User memasukkan data timbang untuk masing – masing barang

User Gudang mengulangi langkah 8 sampai seluruh data timbang barang dimasukkan.

9. User Gudang mencetak Bukti LPAH dan memintakan persetujuan ke pihak yang berwenang.
10. User Gudang memasukkan nama penyetuju
11. Status bukti menjadi 'APPROVED'
12. Sistem menambah stok

2.2 Alternate Flow:

[4.a] Tanggal penerimaan tidak sama dengan tanggal default Sistem

1. User Gudang memasukkan tanggal penerimaan

[9-12.a] Bukti tidak disetujui

1. User Gudang menghapus penerimaan LPAH

[9.a] Bukti harus di-edit

1. User Gudang meng-edit Bukti LPAH
2. User Gudang mencetak ulang Bukti LPAH dan memintakan persetujuan ke bagian yang berwenang

3. Special Requirement

None.

4. Pre-Condition

None.

5. Post-Condition

None.

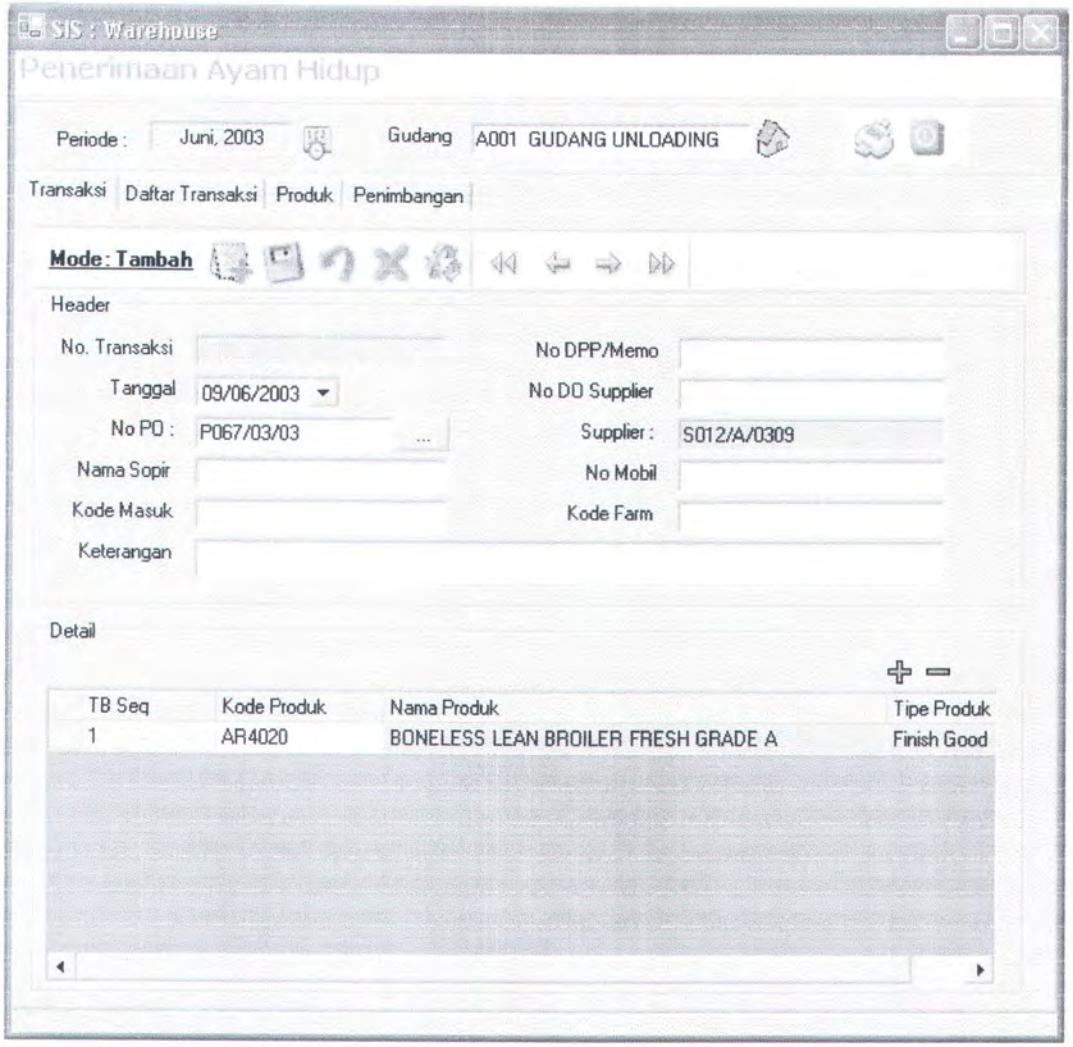
6. Extension Points

None.

7. Document Sources

None.

8. Screen Layout



Use Case Specification: Menerima Barang Lain-lain

1. Menerima Barang Lain-lain

1.2 Brief Description :

Tujuan dari tugas ini adalah untuk menyimpan informasi transaksi Penerimaan Barang Other.

2. Flow of Event

2.1 Basic Flow:

3. User memasukkan kode gudang
4. Sistem men-generate nomor penerimaan barang other.
5. Status bukti menjadi 'OPEN'
6. Sistem memasukkan tanggal default penerimaan
7. User Gudang memasukkan alasan penerimaan
8. User Gudang memilih data barang yang diterima
9. User Gudang memasukkan jumlah barang yang akan diterima

User Gudang mengulangi langkah 6-7 sampai seluruh data barang dimasukkan.

10. User Gudang mencetak Bukti Transaksi Other Penerimaan dan memintakan persetujuan ke pihak yang berwenang.
11. User Gudang memasukkan nama penyetuju
12. Status bukti menjadi 'APPROVED'
13. Sistem menambah stok

2.2 Alternate Flow:

[4.a] Tanggal penerimaan tidak sama dengan tanggal default Sistem

14. User Gudang memasukkan tanggal penerimaan

[8-12.a] Bukti tidak disetujui

15. User Gudang menghapus penerimaan barang other

[8.a] Bukti harus di-edit

16. User Gudang meng-edit Bukti Transaksi Other Penerimaan
17. User Gudang mencetak ulang Bukti Transaksi Other Penerimaan dan memintakan persetujuan ke bagian yang berwenang

3. Special Requirement

None.

4. Pre-Condition

None.

5. Post-Condition

None.

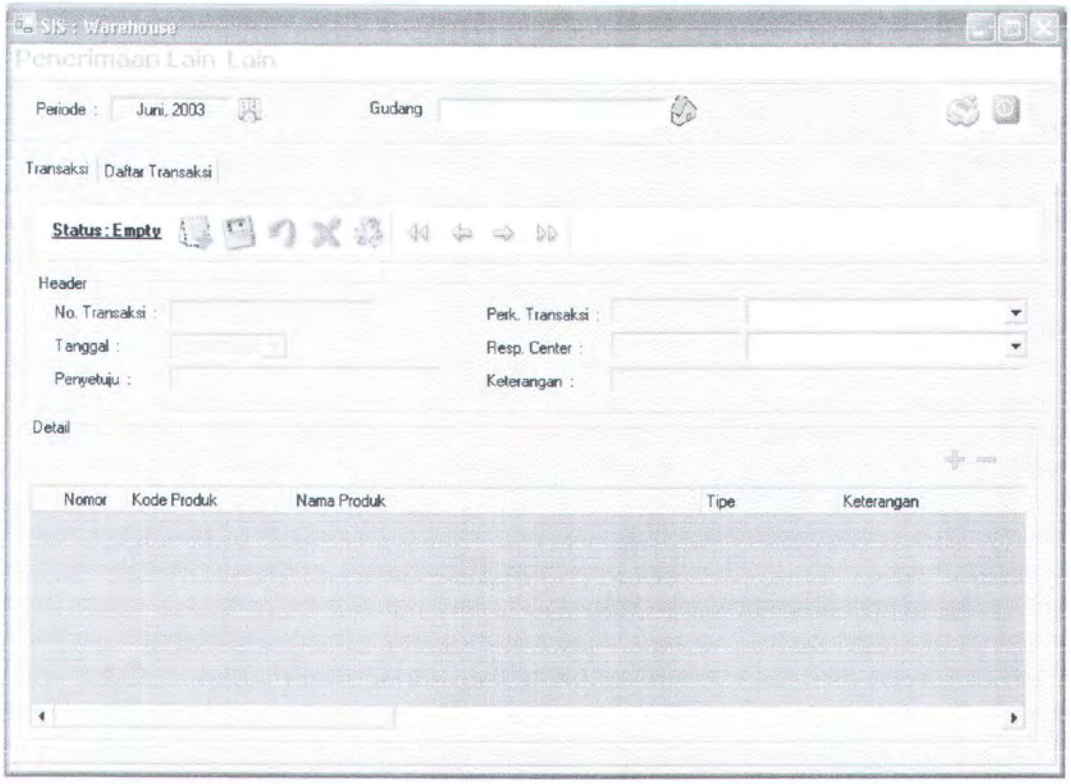
6. Extension Points

None.

7. Document Sources

None.

8. Screen Layout



Use Case Specification: Transfer In

1. Transfer In

1.1 Brief Description:

Tujuan dari tugas ini adalah untuk menyimpan transaksi penerimaan transfer barang dari gudang lain.

2. Flow of Event

2.1 Basic Flow :

1. User memasukkan kode gudang
2. Sistem men-generate nomor transfer in.
3. Status bukti menjadi 'OPEN'
4. Sistem memasukkan tanggal default transfer in
5. User Gudang memilih nomor Bukti Transfer Out.
6. User Gudang mencetak Bukti Transfer In dan memintakan persetujuan bukti ke pihak yang berwenang.
7. User Gudang memasukkan nama penyetuju
8. Status bukti menjadi 'APPROVED'
9. Sistem menambah stok

2.2 Alternate Flow :

[4.a] Tanggal penerimaan tidak sama dengan tanggal default dari Sistem

1. User Gudang memasukkan tanggal penerimaan

[5.a] Penerimaan tidak sama dengan Bukti Transfer Out

1. User Gudang memasukkan data aktual penerimaan

[6.a] Bukti harus di-edit

1. User Gudang meng-edit Bukti Transfer In
2. User Gudang mencetak ulang Bukti Transfer In dan memintakan persetujuan.

[6-10.a] Bukti tidak disetujui

1. User Gudang menghapus transaksi

3. Special Requirement

None

4. Pre-Condition

None

5. Post-Condition

Barang telah diterima

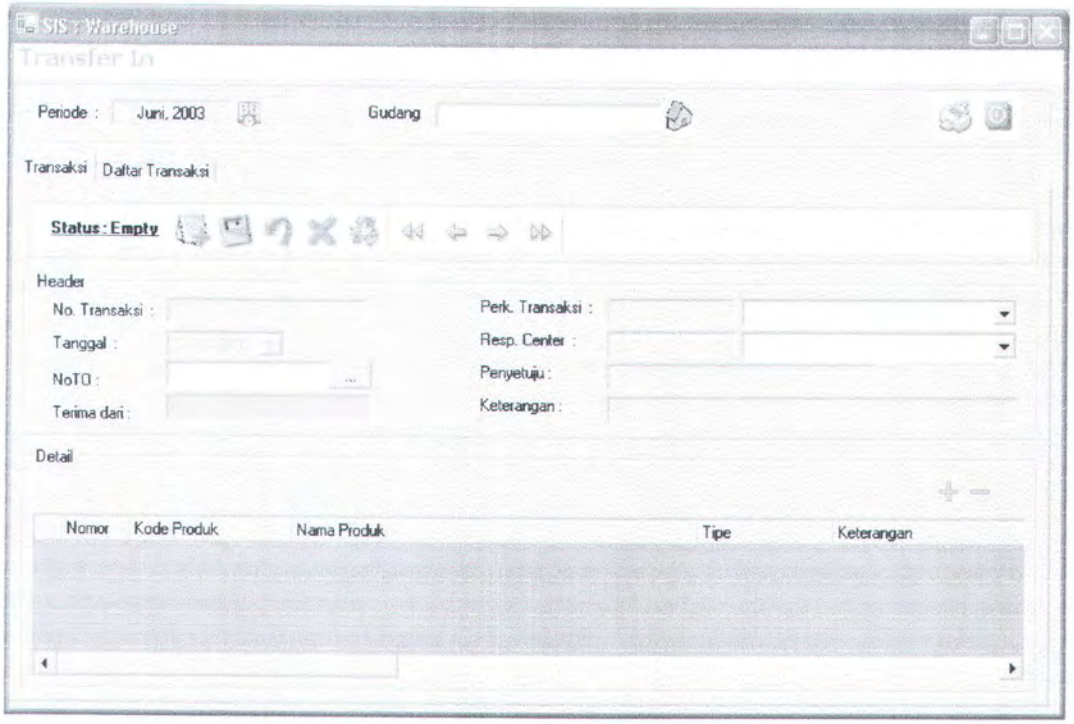
6. Extension Points

None

7. Document Sources

None

8. Screen Layout



Use Case Specification: Meng-inputkan Pengeluaran Ayam Mati

1. Meng-inputkan Pengeluaran Ayam Mati

1.1 Brief Description:

Tujuan dari tugas ini adalah menyimpan informasi yang berhubungan dengan proses pengeluaran ayam mati.

2. Flow of Event

2.1 Basic Flow:

1. Sistem memasukkan kode gudang
2. Sistem men-generate nomor pengeluaran pemusnahan
3. Sistem memasukkan tanggal default pengeluaran pemusnahan
4. User Gudang memilih barang yang akan dikeluarkan
5. User Gudang memasukkan jumlah barang
6. User Gudang mencetak Surat Pengeluaran Barang
7. Sistem mengurangi stok barang.
8. Status pengeluaran menjadi 'APPROVED'

2.2 Alternate Flow:

[3.a] Tanggal Usulan tidak sama dengan tanggal default dari Sistem

1. User Gudang memasukkan tanggal Usulan

3. Special Requirement

None

4. Pre-Condition

None

5. Post-Condition

None

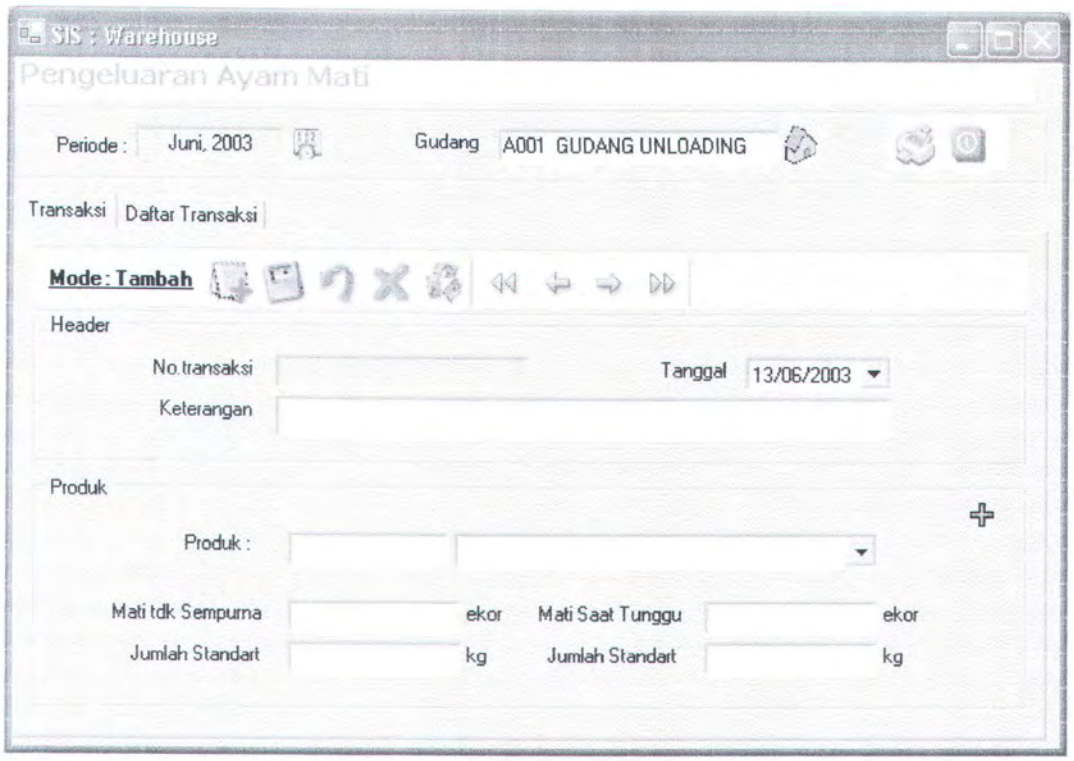
6. Extension Points

None

7. Document Sources

None

8. Screen Layout



Use Case Specification: Meng-inputkan Pengeluaran Pemusnahan (Afkir)

1. Meng-inputkan Pengeluaran Pemusnahan (Afkir)

1.1 Brief Description:

Tujuan dari tugas ini adalah menyimpan informasi yang berhubungan dengan proses pemusnahan stok afkir di gudang.

2. Flow of Event

2.1 Basic Flow:

2. User memasukkan kode gudang
3. Sistem men-generate nomor pengeluaran pemusnahan
4. Sistem memasukkan tanggal default pengeluaran pemusnahan
5. User Gudang memilih barang yang akan dimusnahkan
6. User Gudang memasukkan jumlah barang

User Gudang mengulangi langkah 4-5 sampai seluruh data barang dimasukkan.

7. User Gudang mencetak Surat Pengeluaran Barang
8. Sistem mengurangi stok barang.
9. Status pengeluaran menjadi 'APPROVED'

2.2 Alternate Flow:

[3.a] Tanggal Usulan tidak sama dengan tanggal default dari Sistem

10. User Gudang memasukkan tanggal Usulan

3. Special Requirement

None

4. Pre-Condition

Surat Ijin pemusnahan telah dibuat dan disetujui secara manual.

5. Post-Condition

Barang siap diserahterimakan dengan bagian GA.

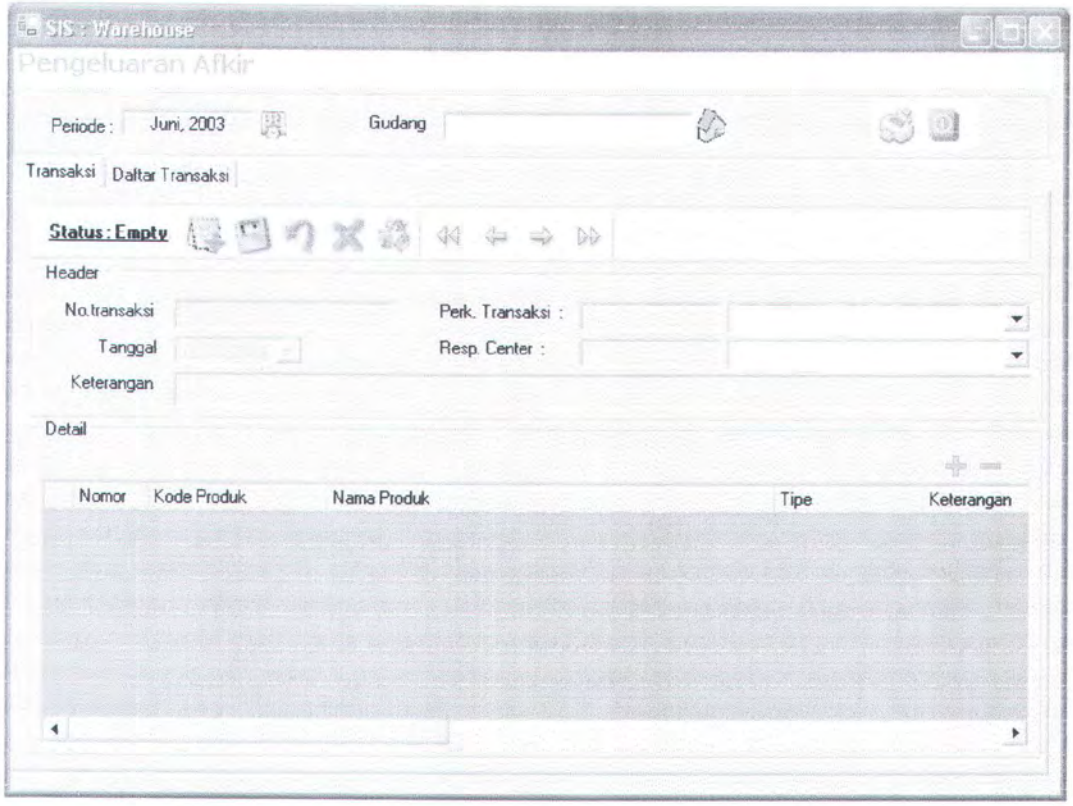
6. Extension Points

None

7. Document Sources

None

8. Screen Layout



Use Case Specification: Meng-inputkan Pengeluaran Umum

1. Meng-inputkan Pengeluaran Umum

1.1 Brief Description:

Tujuan dari tugas ini adalah menyimpan informasi yang berhubungan dengan proses pengeluaran barang umum.

2. Flow of Event

2.1 Basic Flow:

11. User memasukkan kode gudang
12. Sistem men-generate nomor pengeluaran barang
13. Sistem memasukkan tanggal default pengeluaran barang
14. User Gudang memilih barang yang akan dikeluarkan
15. User Gudang memasukkan jumlah barang

User Gudang mengulangi langkah 4-5 sampai seluruh data barang dimasukkan.

16. User Gudang mencetak Surat Pengeluaran Barang
17. Sistem mengurangi stok barang.
18. Status pengeluaran menjadi 'APPROVED'

2.2 Alternate Flow:

[3.a] Tanggal Usulan tidak sama dengan tanggal default dari Sistem

19. User Gudang memasukkan tanggal Usulan

3. Special Requirement

None

4. Pre-Condition

None

5. Post-Condition

None

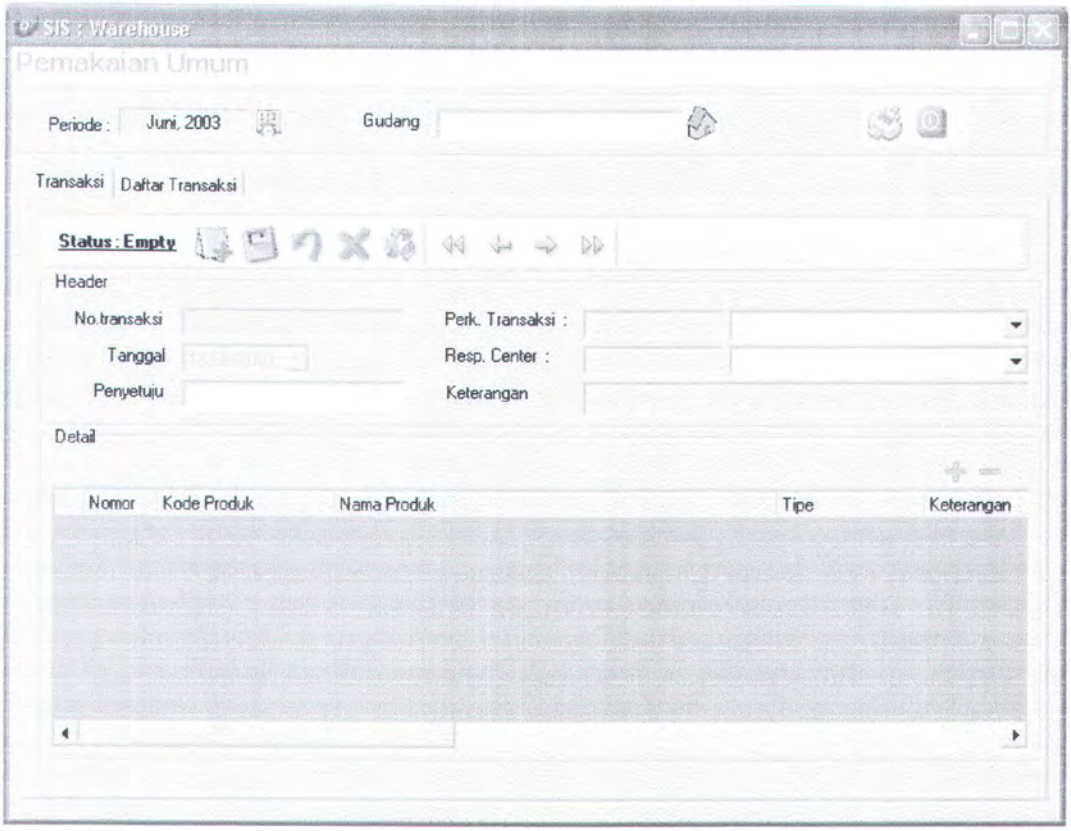
6. Extension Points

None

7. Document Sources

None

8. Screen Layout



Use Case Specification: Mengeluarkan Barang Other

1. Mengeluarkan Barang Other

1.1 Brief Description :

Tujuan dari tugas ini adalah untuk menyimpan informasi transaksi Pengeluaran Barang Other.

2. Flow of Event

2.1 Basic Flow:

1. User memasukkan kode gudang
2. *Sistem* men-generate nomor pengeluaran barang other.
3. Status bukti menjadi 'OPEN'
4. *Sistem* memasukkan tanggal default pengeluaran
5. *User Gudang* memasukkan alasan pengeluaran
6. *User Gudang* memilih data barang yang akan dikeluarkan
7. *User Gudang* memasukkan jumlah barang yang akan dikeluaran

   User Gudang mengulangi langkah 6-8 sampai seluruh data barang dimasukkan.

8. *User Gudang* mencetak Bukti Transaksi Other Pengeluaran dan memintakan persetujuan ke pihak yang berwenang.
9. *User Gudang* memasukkan nama penyetuju
10. Status bukti menjadi 'APPROVED'
11. *Sistem* mengurangi stok

2.2 Alternate Flow:

[4.a] Tanggal pengeluaran tidak sama dengan tanggal default dari *Sistem*

1. *User Gudang* memasukkan tanggal pengeluaran

[8-13.a] Bukti tidak disetujui

1. *User Gudang* menghapus transaksi pengeluaran barang other

[9.a] Bukti harus di-edit

1. *User Gudang* meng-edit Bukti Transaksi Other Pengeluaran
2. *User Gudang* mencetak ulang Bukti Transaksi Other Pengeluaran dan memintakan persetujuan ke bagian yang berwenang

3. Special Requirement

None

4. Pre-Condition

1. Ada DPB manual dari unit untuk pengeluaran barang other.

5. Post-Condition

None

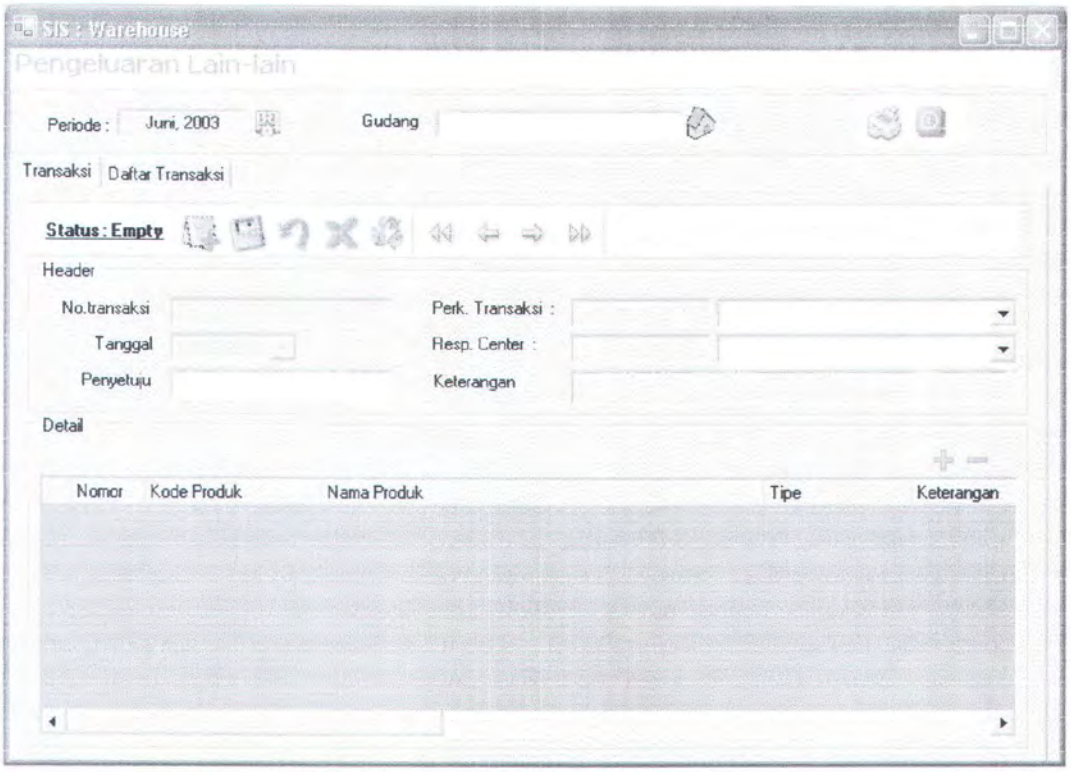
6. Extension Points

None

7. Document Sources

None

8. Screen Layout



Use Case Specification: Transfer Out

1. Transfer Out

1.1 Brief Description :

Tujuan dari tugas ini adalah untuk menyimpan transaksi pengeluaran transfer barang ke gudang lain.

2. Flow of Event

2.1 Basic Flow :

1. User memasukkan kode gudang
2. *Sistem* men-generate nomor transfer out.
3. Status bukti menjadi 'OPEN'
4. *Sistem* memasukkan tanggal default transfer out
5. *User Gudang* memasukkan data gudang tujuan.
6. *User Gudang* memasukkan alasan pengeluaran
7. *User Gudang* memilih data barang yang akan dikeluarkan
8. *User Gudang* memasukkan jumlah barang yang akan dikeluarkan

   User Gudang mengulangi langkah 7-8 sampai seluruh data barang dimasukkan.

9. *User Gudang* mencetak Bukti Transfer Out dan memintakan persetujuan bukti ke pihak yang berwenang.
10. *User Gudang* memasukkan nama penyetuju
11. Status bukti menjadi 'APPROVED'
12. *Sistem* mengurangi stok

2.2 Alternate Flow :

[4.a] Tanggal pengeluaran tidak sama dengan tanggal default dari *Sistem*

1. *User Gudang* memasukkan tanggal pengeluaran

[11.a] Bukti harus di-edit

1. *User Gudang* meng-edit Bukti Transfer Out
2. *User Gudang* mencetak ulang Bukti Transfer Out dan memintakan persetujuan.

[11-12.a] Bukti tidak disetujui

1. *User* Gudang membatalkan transfer out

3. Special Requirement

None

4. Pre-Condition :

None

5. Post-Condition :

1. Barang siap dikirim

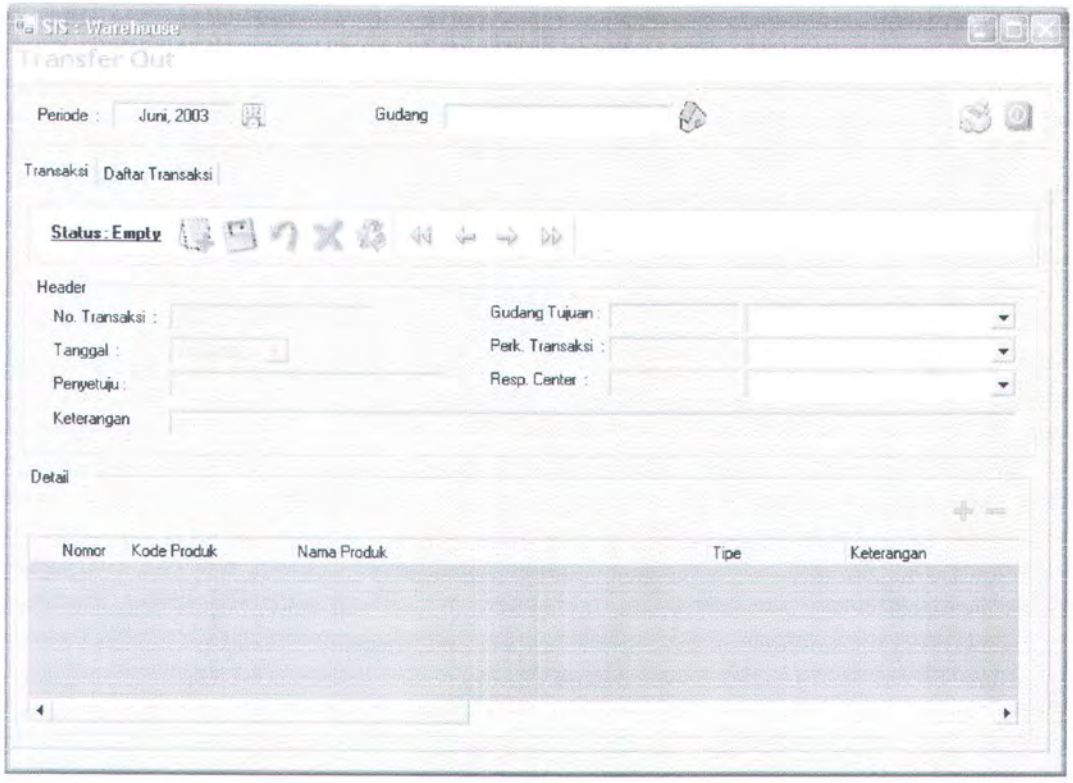
6. Extension Points

None

7. Document Sources

None

8. Screen Layout



Use Case Specification: Meng-inputkan Pengeluaran Barang Penjualan

1. Meng-inputkan Pengeluaran Barang

1.1 Brief Description :

Tujuan dari tugas ini adalah mencatat seluruh transaksi pengeluaran barang per SPM.

2. Flow of Event

2.1 Basic Flow :

1. User memasukkan kode gudang
2. Sistem men-generate nomor pengeluaran barang
3. Sistem memasukkan tanggal default pengeluaran barang
4. Sistem memasukkan initiator default pengeluaran barang
5. User Gudang memilih SPM.
6. User Gudang memasukkan data muat aktual.
7. User Gudang melakukan Packing List untuk mengurangi stok barang.

2.2 Alternate Flow:

[3.a] Tanggal pengeluaran barang tidak sama dengan tanggal default dari Sistem

1. User Gudang memasukkan tanggal pengeluaran barang

3. Special Requirement

None

4. Pre-Condition

None

5. Post-Condition

None

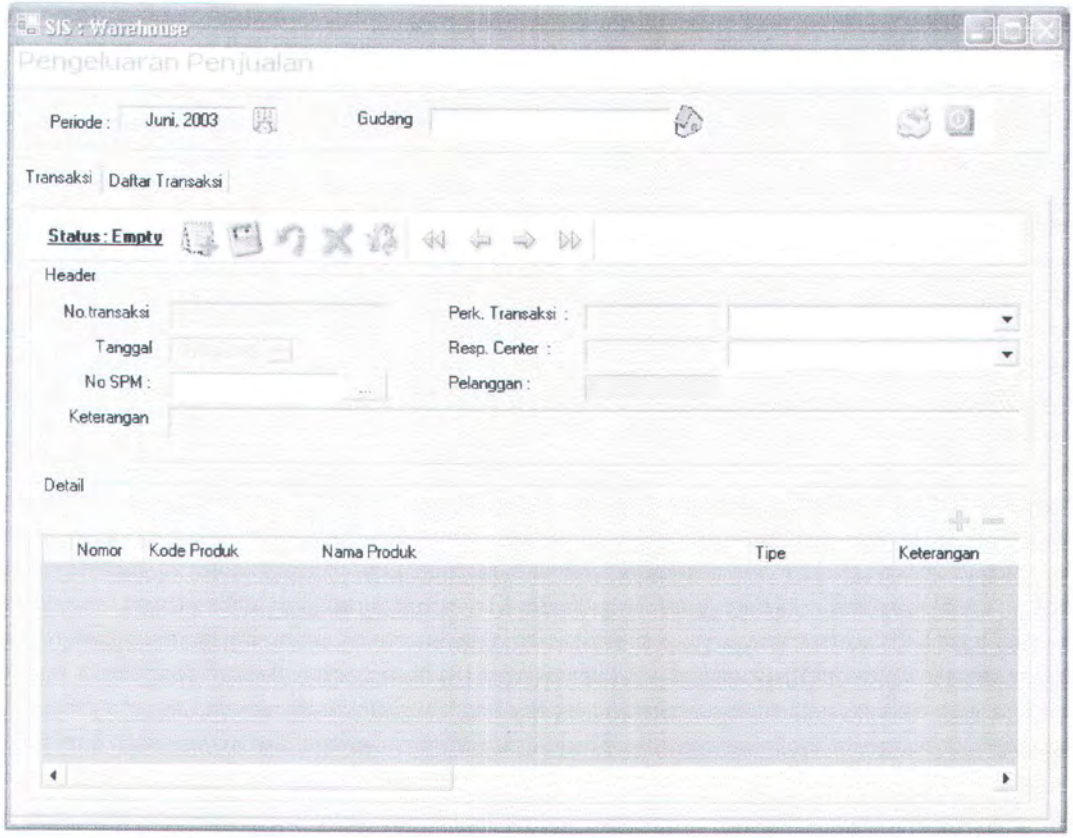
6. Extension Points

None

7. Document Sources

None

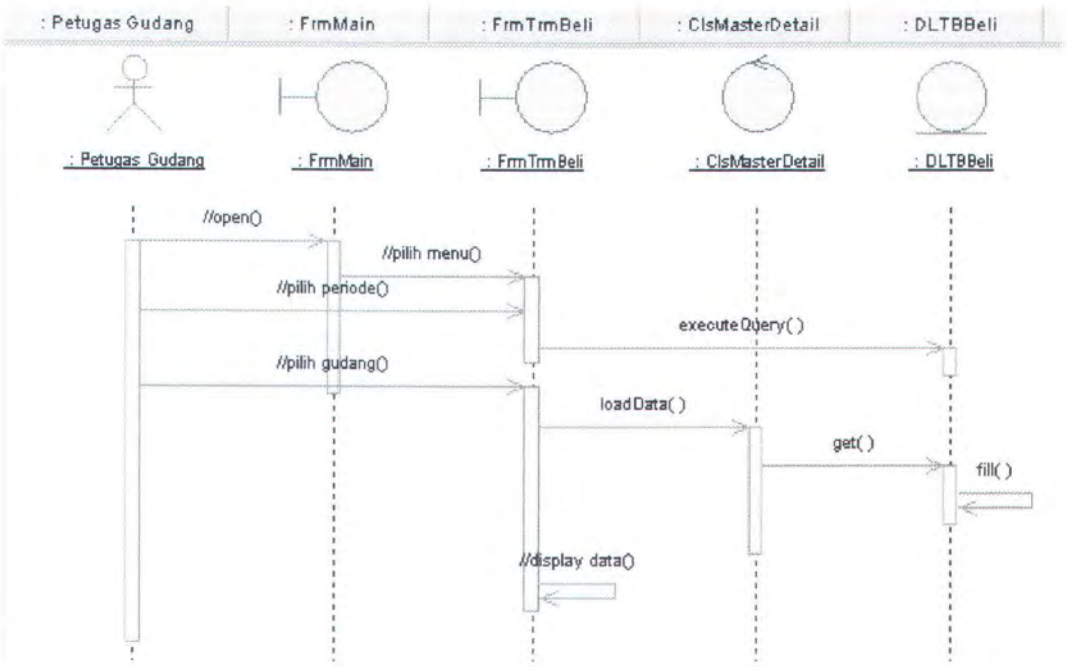
8. Screen Layout



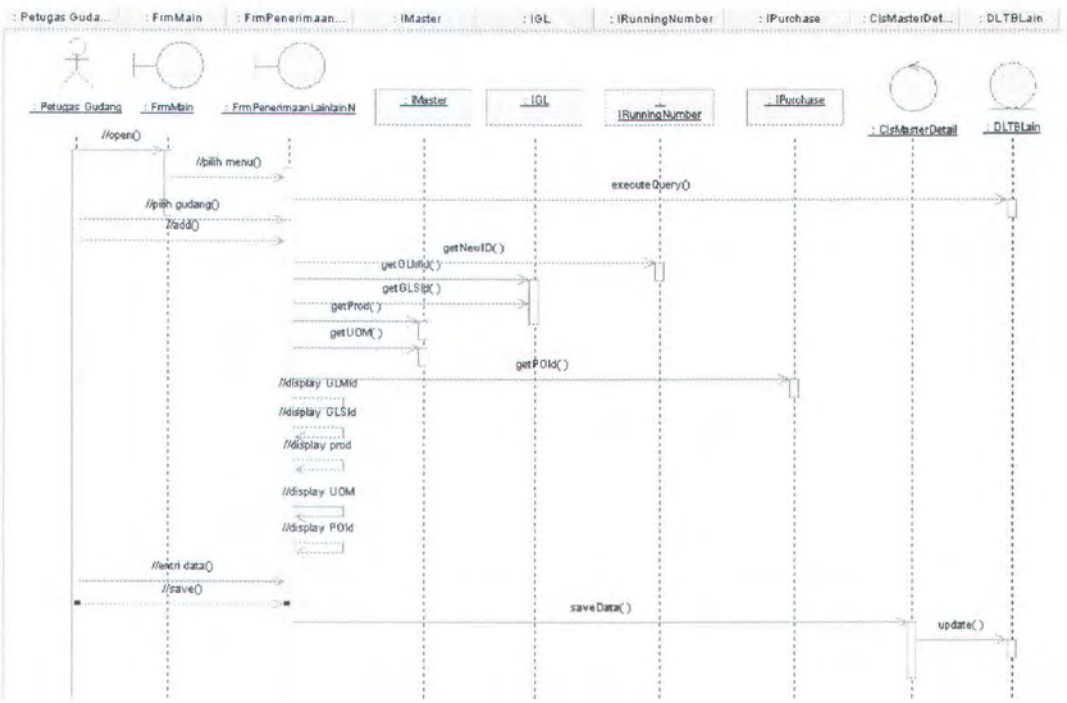
LAMPIRAN C

1. Use Case Realization Penerimaan Pembelian

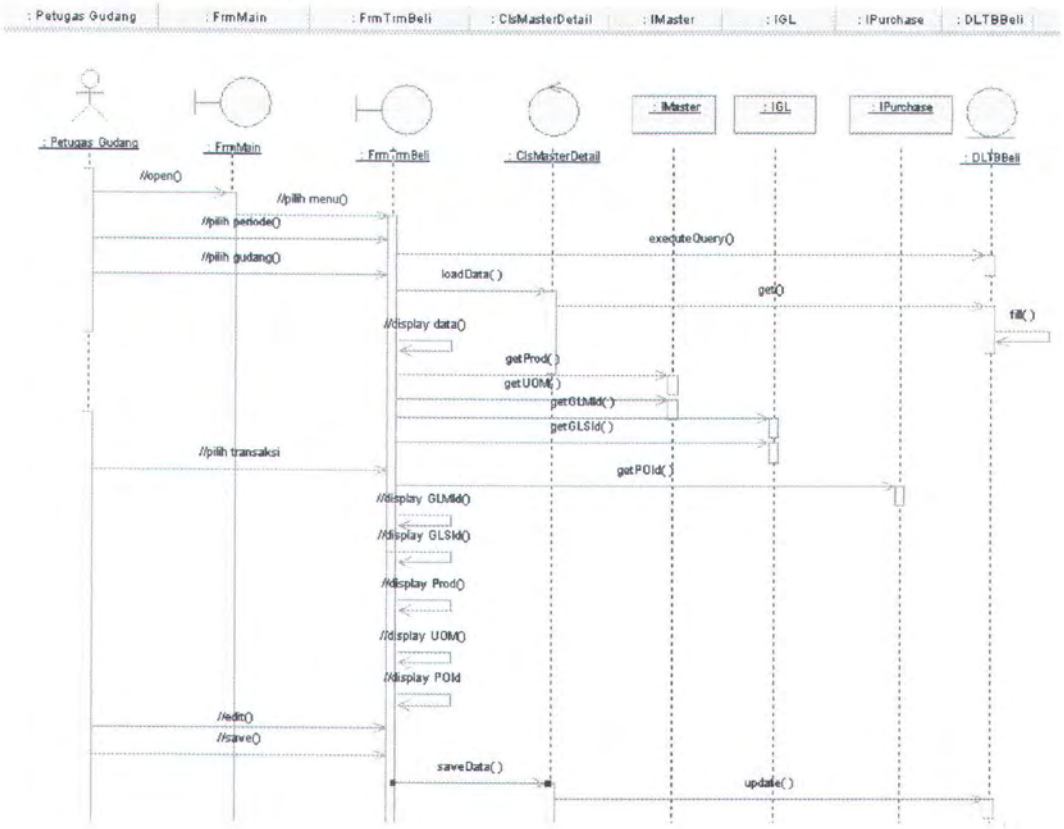
- **Sequence Diagram Basic Flow (view)**



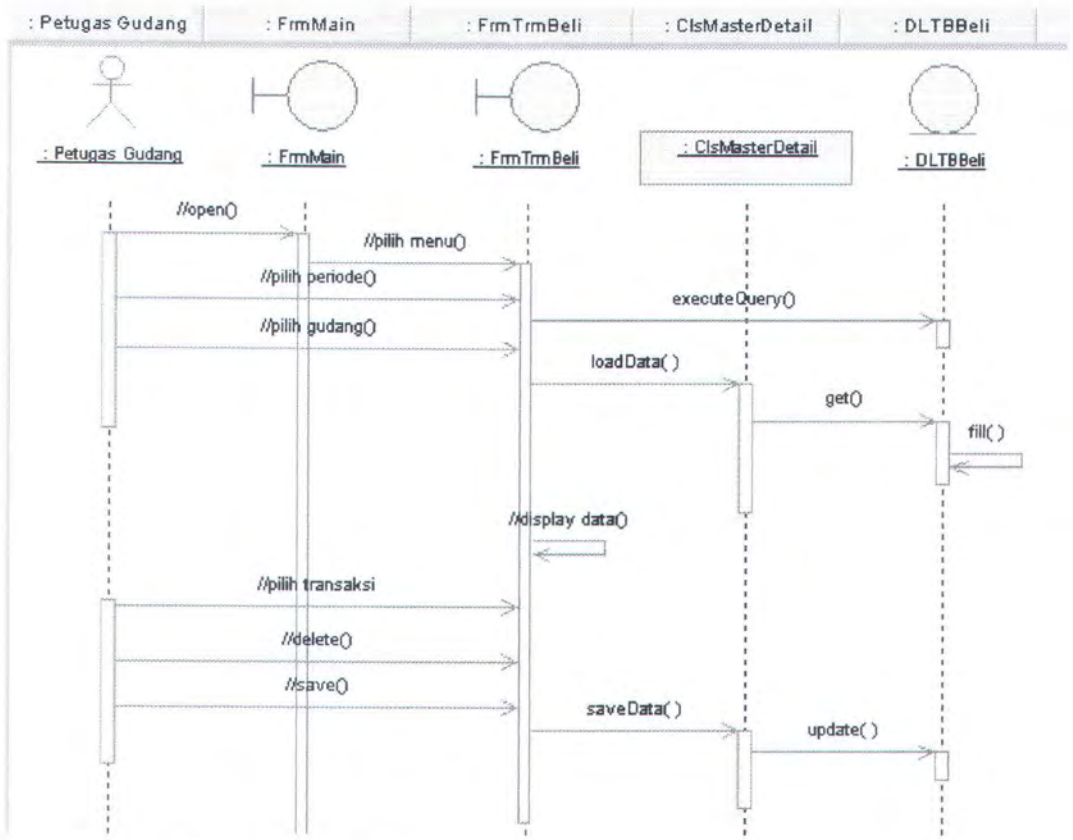
- **Sequence Diagram Add**



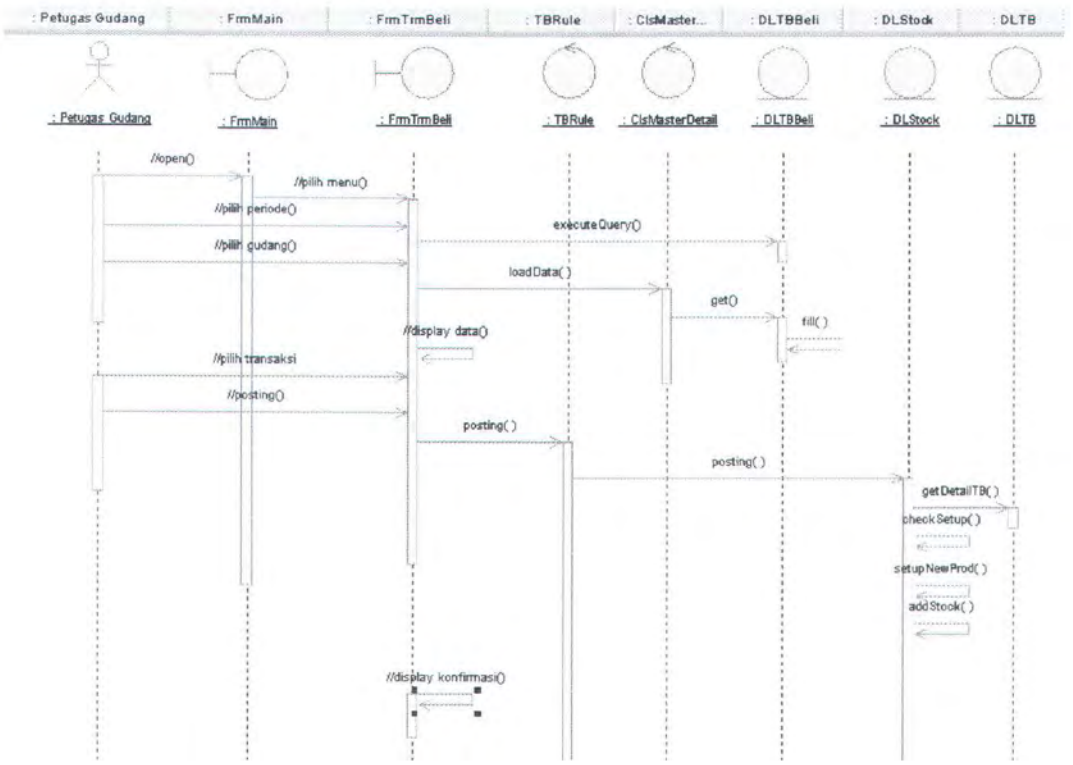
- **Sequence Diagram Edit**



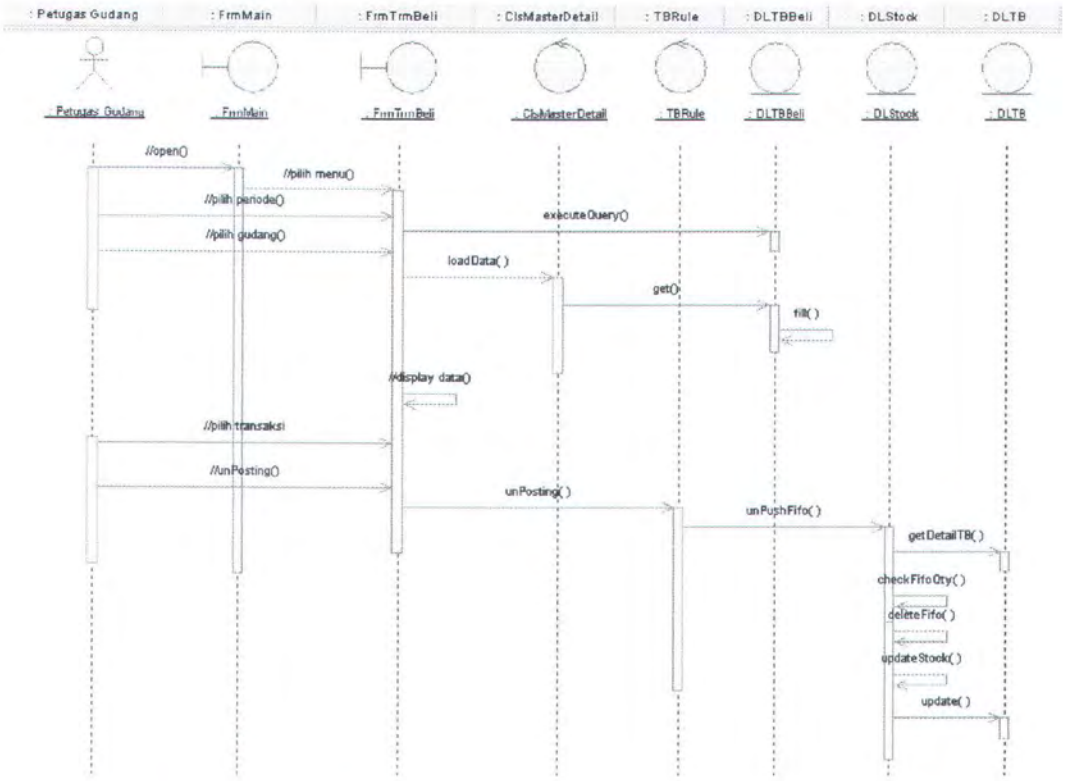
- **Sequence Diagram Delete**



- **Sequence Diagram Posting**

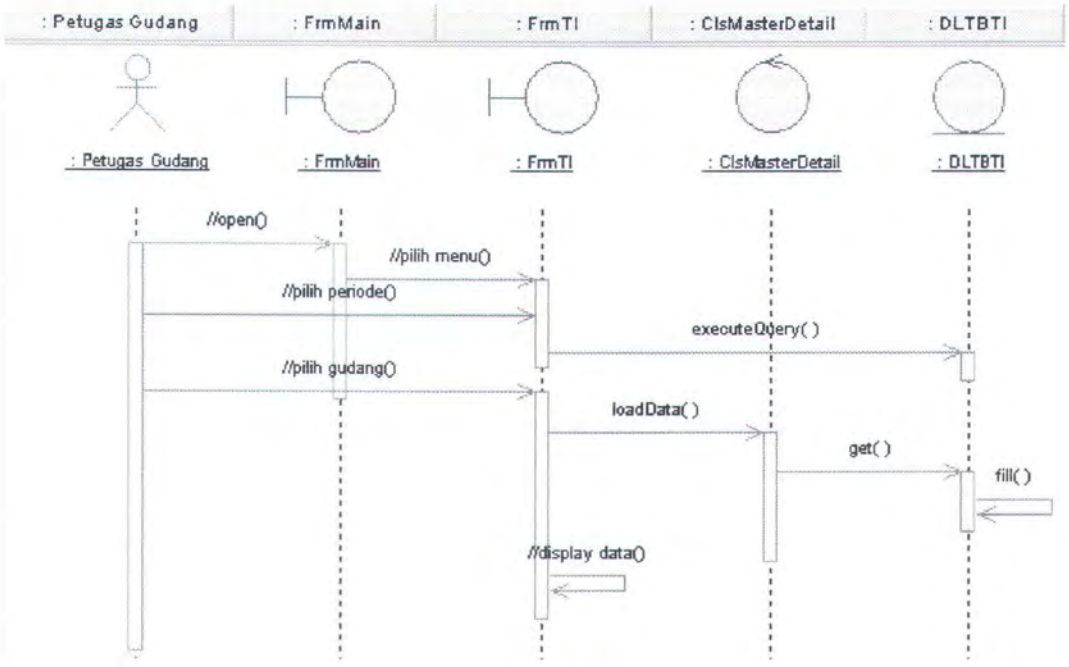


- **Sequence Diagram UnPosting**

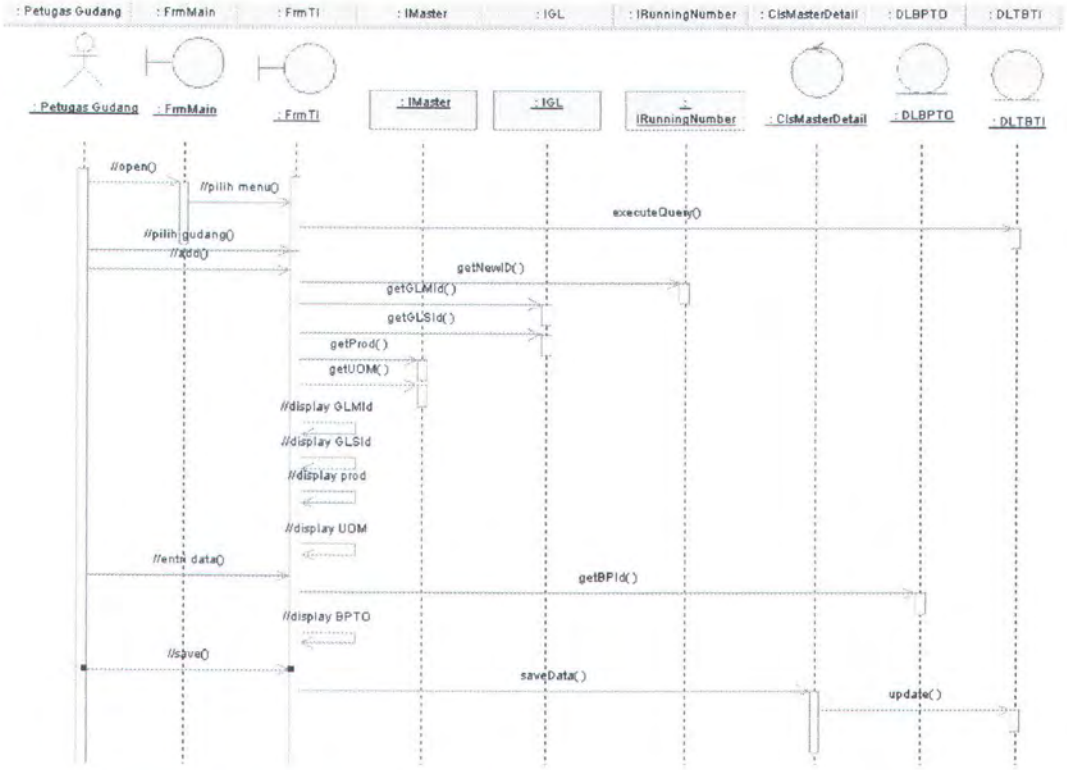


2. Use Case Realization Transfer In

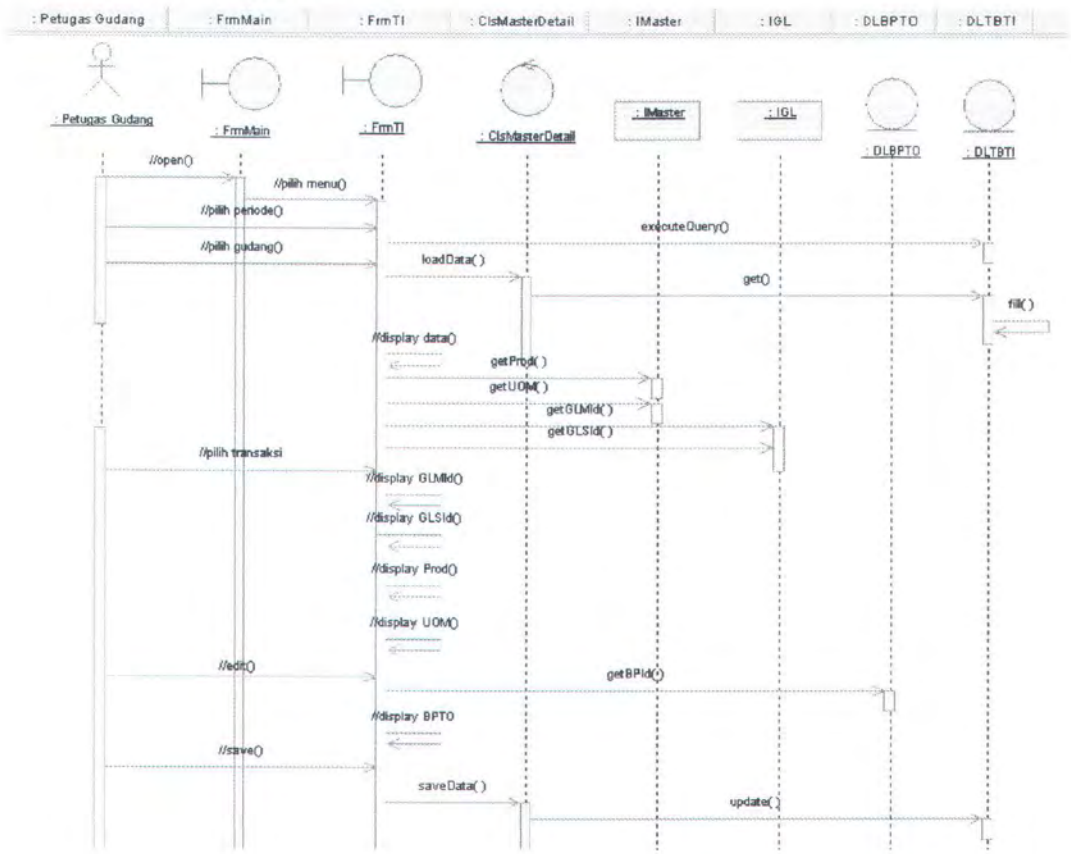
- **Sequence Diagram Basic Flow (view)**



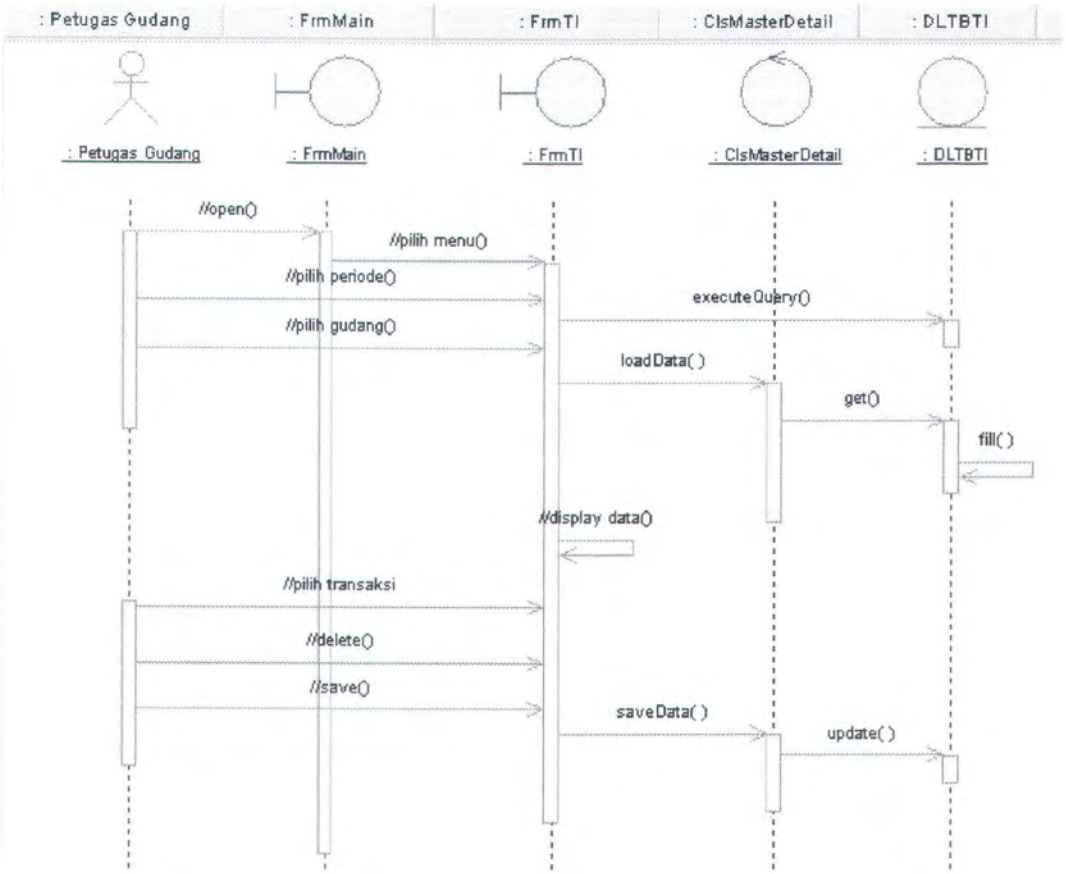
- **Sequence Diagram Add**



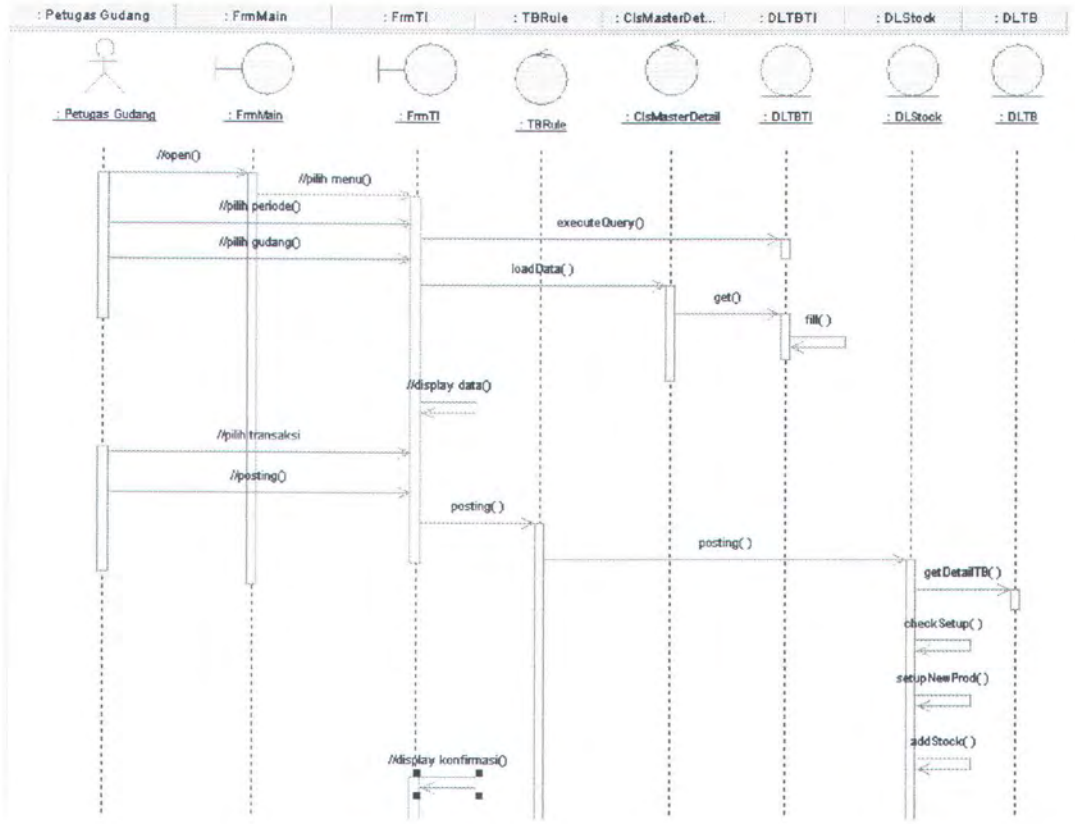
- **Sequence Diagram Edit**



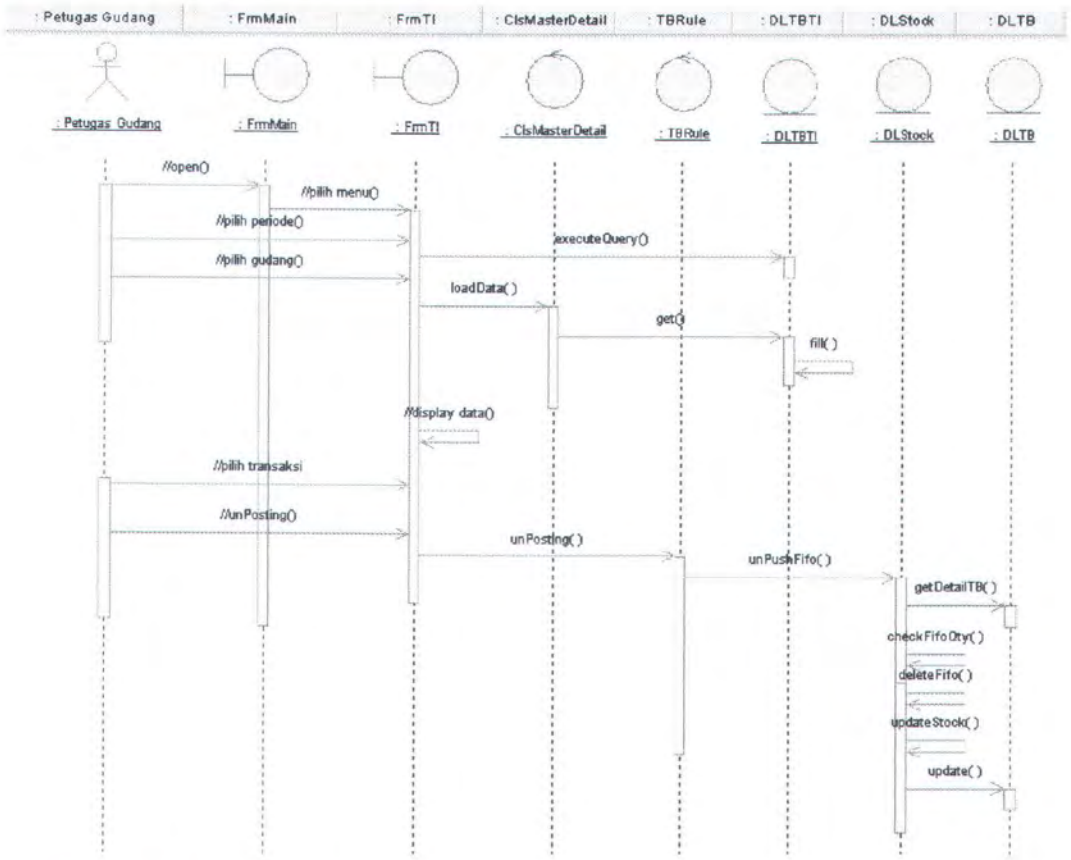
- **Sequence Diagram Delete**



- **Sequence Diagram Posting**

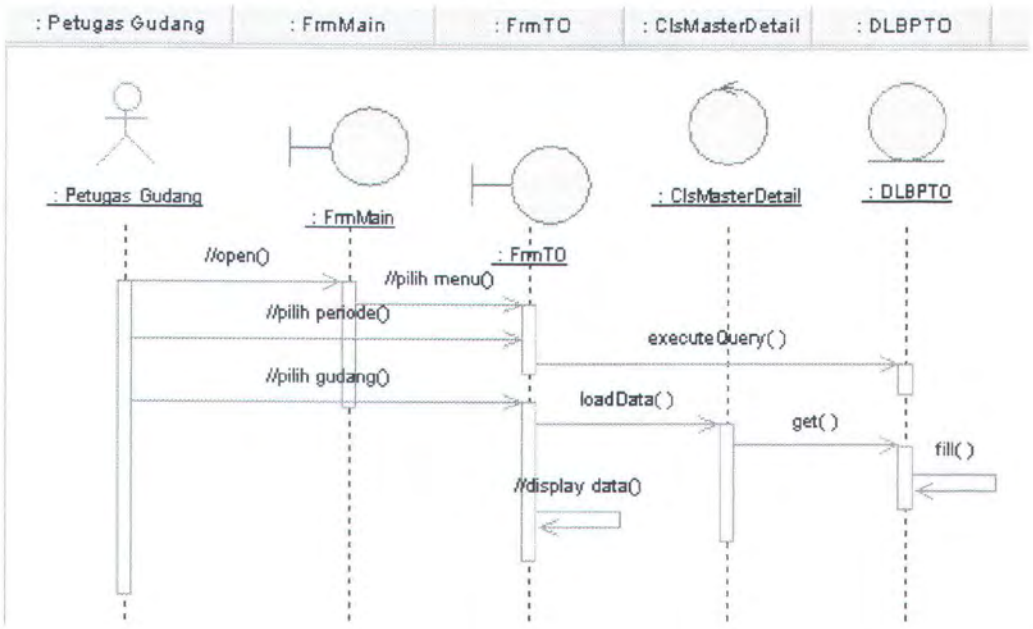


- **Sequence Diagram UnPosting**

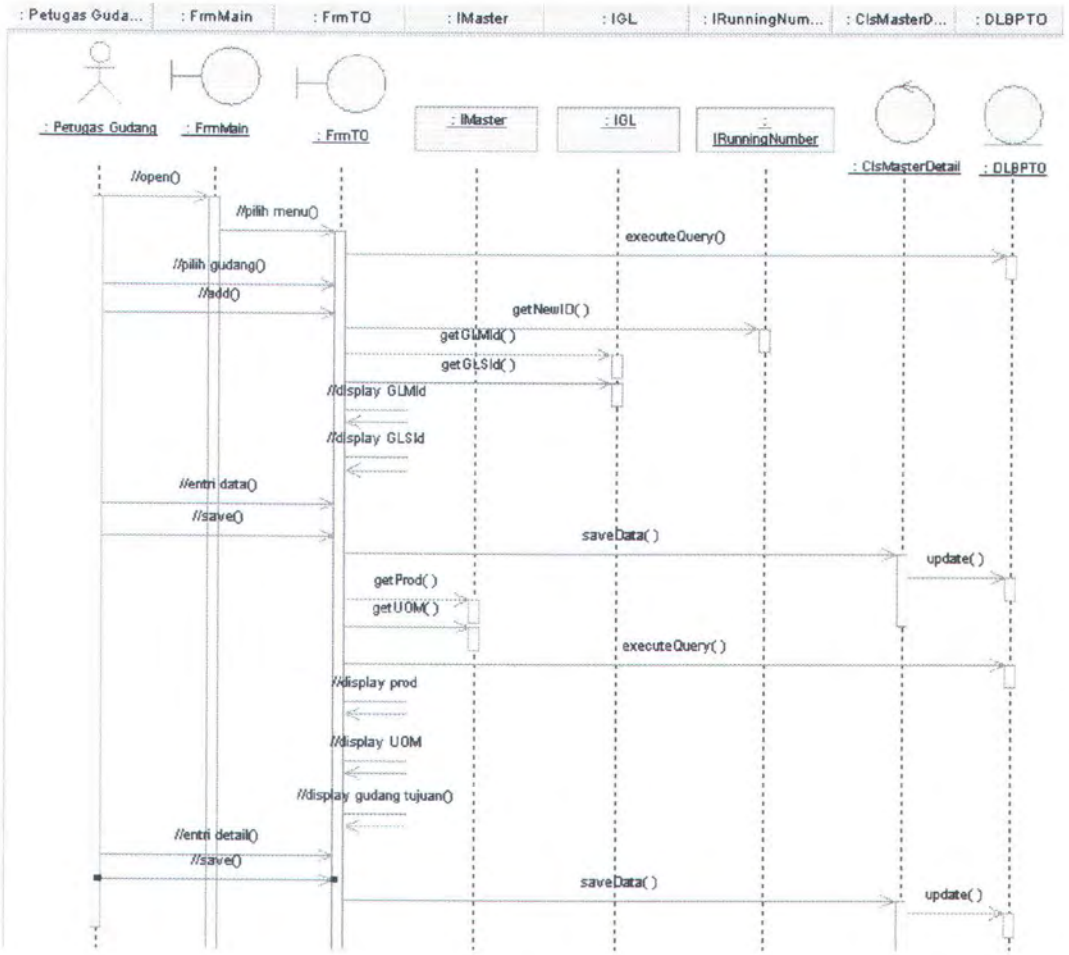


3. Use Case Realization Transfer Out

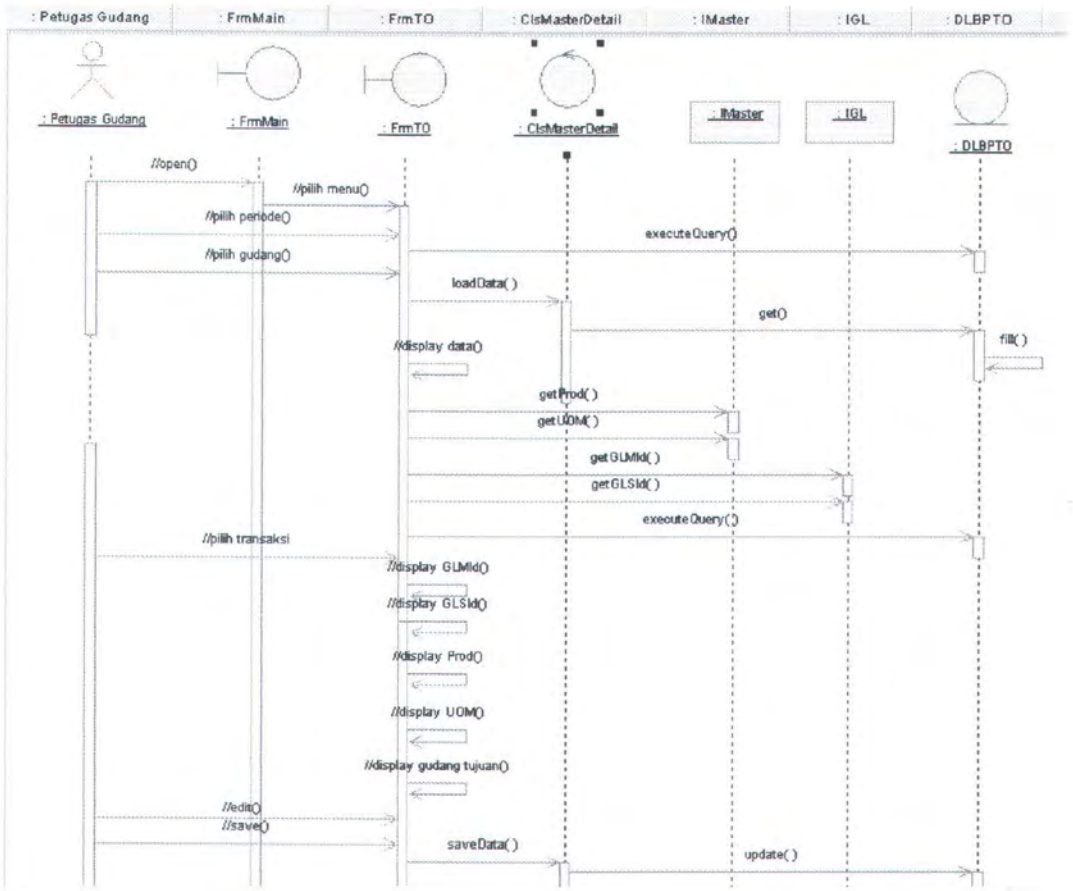
- **Sequence Diagram Basic Flow (view)**



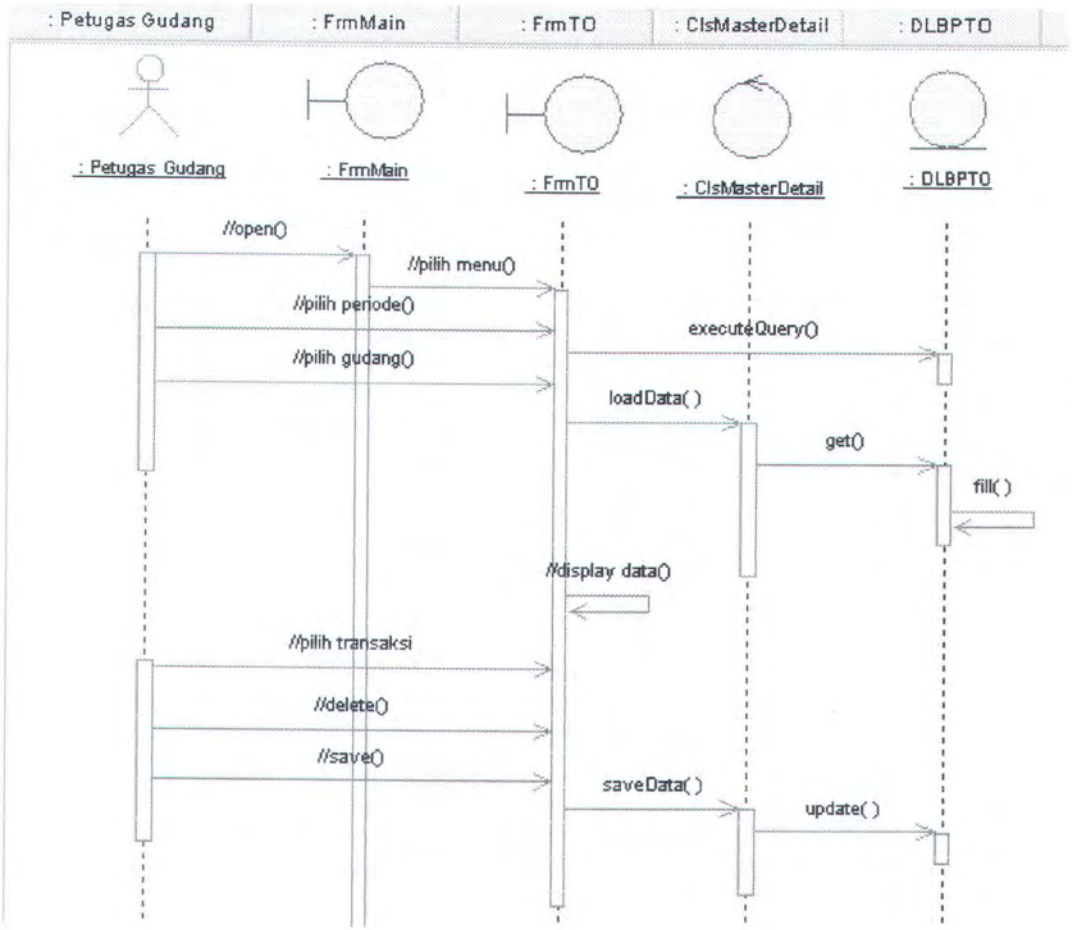
- **Sequence Diagram Add**



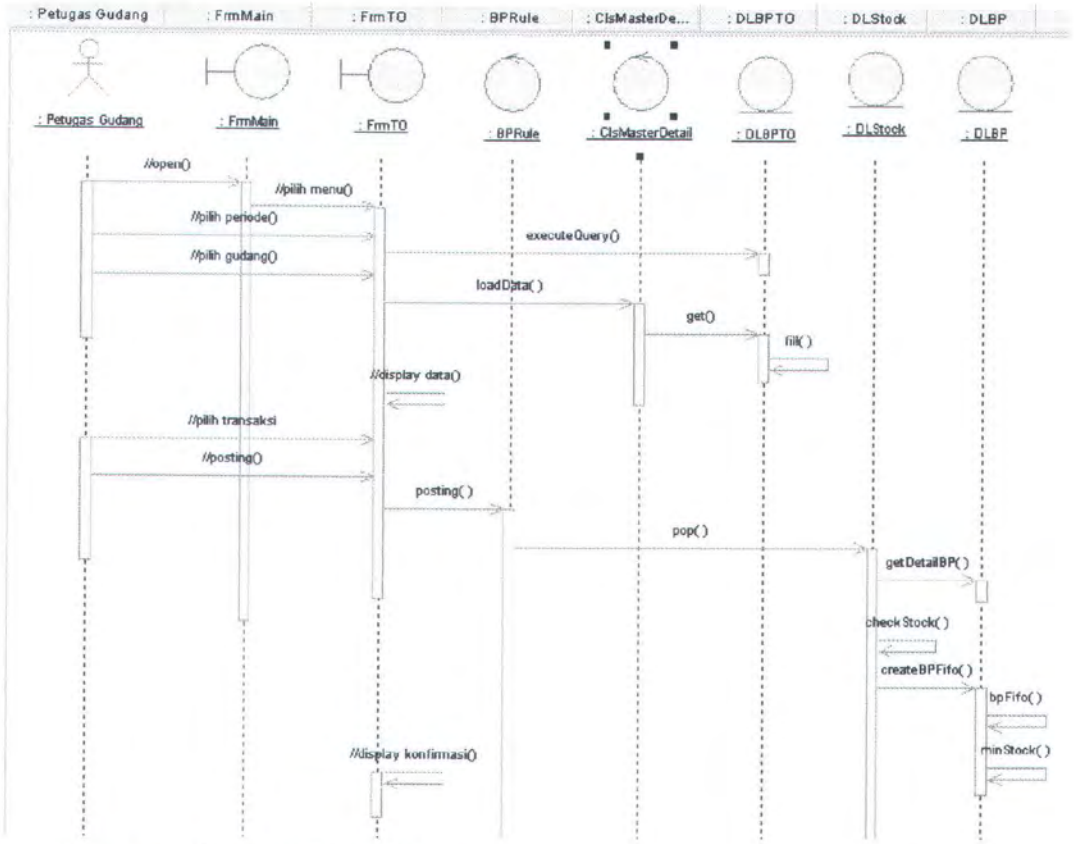
- **Sequence Diagram Edit**



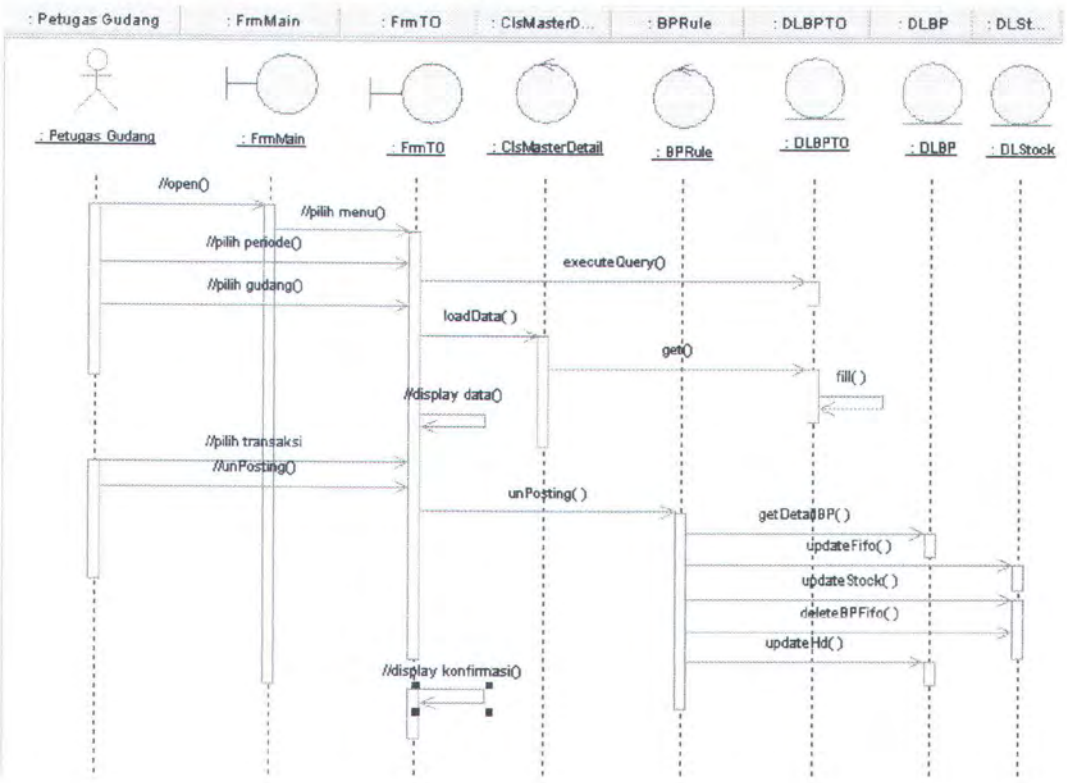
- **Sequence Diagram Delete**



- **Sequence Diagram Posting**



- **Sequence Diagram UnPosting**



LAMPIRAN D

D.1 SCRIPT DATA DEFINITION LANGUAGE (DDL)

```
/*==============================================================*/
/* Database name:  SIS                                          */
/* DBMS name:      ORACLE Version 9i                            */
/* Created on:     1/20/2003 2:12:51 PM                         */
/*==============================================================*/

create or replace  type APPROVAL_TYP as OBJECT
(
   APPROVER1 VARCHAR(25),
   APPROVER2 VARCHAR(25),
   APPROVER3 VARCHAR(25),
   APPROVER4 VARCHAR(25),
   APPROVER5 VARCHAR(25)
)
/

 create or replace  type AUDITLOG_TYP as OBJECT
(
   UPDUSER VARCHAR2(30),
   UPDTIME DATE,
   UPDTYPE CHAR(1)
)
/

create or replace type MSUOM_TYP as OBJECT  (
  UOMCODE             VARCHAR2(10),
  UOMCONVTO           VARCHAR2(10),
  UOMALIAS            VARCHAR2(10),
  UOMDESC             VARCHAR2(128),
  UOMCONVFACTOR   NUMBER(12,6)
)
/

create table MSUOM  of MSUOM_TYP(
  primary key (UOMCODE),
  constraint FK_MSUOM_KONVERSIS_MSUOM foreign key (UOMCONVTO)
      references MSUOM (UOMCODE)
)
  object ID primary key
/

create index KONVERSISATUAN_FK on MSUOM (
  UOMCONVTO ASC
)
/

create or replace type MSWHGROUP_TYP as OBJECT (
  WHGRPID              CHAR(2),
  WHGRPNAME         VARCHAR2(20)
)
```

```
/

create table MSWHGROUP of MSWHGROUP_TYP (
  primary key (WHGRPID),
  WHGRPNAME          not null
)
  object ID primary key
/

create or replace type MSWAREHOUSE_TYP as OBJECT (
  WRHSID                    VARCHAR2(10),
  CITYCODE                  VARCHAR2(10),
  WHGRP                     REF MSWHGROUP_TYP,
  GLSID                     CHAR(8),
  WRHSPARENT                VARCHAR2(10),
  WRHSNAME                  VARCHAR2(30),
  WRHSALIAS                 VARCHAR2(100),
  WRHSADDR1                 VARCHAR2(80),
  WRHSADDR2                 VARCHAR2(50),
  WRHSZIPCODE               VARCHAR2(8),
  WRHSPHONE                 VARCHAR2(20),
  WRHSFAX                   VARCHAR2(30),
  WRHSTYPE                  VARCHAR2(1),
  FGSTATUS                  VARCHAR2(1)
)
/

create table MSWAREHOUSE of MSWAREHOUSE_TYP (
  primary key (WRHSID),
  foreign key (WHGRP) REFERENCES MSWHGROUP,
  WRHSNAME          not null,
  constraint FK_MSWAREHO_ANGGOTA_D_MSWAREHO foreign key (WRHSPARENT)
      references MSWAREHOUSE (WRHSID),
  constraint FK_MSWAREHO_KOTA_WARE_MSCITY foreign key (CITYCODE)
     references MSCITY (CITYCODE),
  constraint FK_MSWAREHO_WHGLS_GLS foreign key (GLSID)
     references GLS(GLSID)
)
  object ID primary key
/

create index ANGGOTA_DARI_FK on MSWAREHOUSE (
  WRHSPARENT ASC
)
/

create index GROUP_WAREHOUSE_FK on MSWAREHOUSE (
  WHGRP ASC
)
/

create index KOTA_WAREHOUSE_FK on MSWAREHOUSE (
  CITYCODE ASC
)
/

create or replace type WhFIFO_TYP as object
```

```
(
  WRHSID                  VARCHAR2(10),
  PRODCODE                VARCHAR2(25),
  TRXDATE                 DATE,
  FIFOQTY                 NUMBER(15,2),
  FIFOAMT                 NUMBER(16,2),
  FIFOPRICE               NUMBER(16,2),
  FIFOENDQTY              NUMBER(15,2),
  FIFOENDAMT              NUMBER(16,2),
  TBID_FIFO               VARCHAR(15),
  TBSEQ_FIFO              NUMBER(4),
  NCASHMOHAMT             NUMBER(16,2),
  NCASHMOHPRICE           NUMBER(16,2)
)
/

create or replace type MSPRODUCT_TYP as OBJECT (
  PRODCODE                VARCHAR2(25),
  PRODGROUP               VARCHAR2(25),
  UOM                     REF MSUOM_TYP,
  GLMID                   CHAR(10),
  PRODDLPRICE             NUMBER(16,2),
  PRODUMUR                NUMBER(5),
  PRODSATUMUR             CHAR(1),
  ISGROUP                 NUMBER(1),
  GRPMEMBERS              NUMBER(5),
  PRODTYPE                VARCHAR2(1),
  PRODNAME                VARCHAR2(50),
  PRODALIAS               VARCHAR2(100),
  PRODARTICLE             VARCHAR2(25),
  PRODBARCODE             VARCHAR2(25),
  PRODCONVERSIONX         NUMBER(15,2),
  PRODCONVERSIONY         NUMBER(15,2),
  PRODMINSTOCK            NUMBER(15,2),
  PRODREORDERPOINT        NUMBER(15,2),
  PRODTOLERANCE           NUMBER(15,2),
  PRODLEADTIME            NUMBER(5),
  PRODSTDPRICE            NUMBER(16,2),
  PRODRMPRICE             NUMBER(16,2),
  PRODFOHPRICE            NUMBER(16,2),
  PRODSTATUSDATE          DATE,
  STRAINTYPE              VARCHAR2(1),
  DOCTYPE                 VARCHAR2(15),
  EGGNAME                 VARCHAR2(15),
  FGPPN                   VARCHAR2(1),
  FGPACKAGE               VARCHAR2(1),
  FGBOM                   VARCHAR2(1),
  FGROYALTY               VARCHAR2(1),
  FGTYPE                  NUMBER(3),
  FGSTOCK                 VARCHAR2(1),
  FGSTATUS                VARCHAR2(1),
  PRODDIFFLEVEL           NUMBER(1),
  SALESGLM                CHAR(10),
  DISCGLM                 CHAR(10),
  COGSGLM                 CHAR(10),
  INRETURGLM              CHAR(10),
  AUDLOG                  AuditLog_Typ
```

```
)
/

create table MSPRODUCT of MSPRODUCT_TYP (
  primary key (PRODCODE),
  foreign key (UOM) REFERENCES MSUOM,
  foreign key (GLMID) REFERENCES GLM,
  PRODSATUMUR       default '*' not null,
  ISGROUP           default 0  not null,
  PRODTYPE          not null,
  PRODNAME          not null,
  constraint CKC_PRODDIFFLEVEL_MSPRODUC check (PRODDIFFLEVEL is null or (
PRODDIFFLEVEL in (1,2,3) )),
  constraint CKC_PRODSATUMUR_MSPRODUC check (PRODSATUMUR in ('*','J','H','B','T')),
  constraint FK_MSPRODUC_PRODUKGLM_GLM foreign key (GLMID)
    references GLM (GLMID),
  constraint FK_MSPRODUC_GROUPPROD_MSPRODUC foreign key (PRODGROUP)
    references MSPRODUCT (PRODCODE)
)
/

create table WhFIFO of WhFIFO_TYP(
  WRHSID not null,
  TRXDATE not null,
  PRODCODE not null,
  NCASHMOHAMT       default 0  not null,
  NCASHMOHPRICE     default 0  not null,
  constraint PK_WHFIFO primary key (WRHSID, PRODCODE, TRXDATE),
  constraint FK_WHFIFO_FIFO4WH_MSWAREHO foreign key (WRHSID)
    references MSWAREHOUSE (WRHSID),
  constraint FK_WHFIFO_FIFOPRODU_MSPRODUC foreign key (PRODCODE)
    references MSPRODUCT (PRODCODE)
)
/

create index PRODUKGLM_FK on MSPRODUCT (
  GLMID ASC
)
/

create index GROUPPRODUK_FK on MSPRODUCT (
  PRODGROUP ASC
)
/

create index PRODUKUOM_FK on MSPRODUCT (
  UOM ASC
)
/

create table MSPRODUCTFOTO  (
  PRODCODE          VARCHAR2(25)              not null,
  PRODFOTO          ORDSYS.ORDIMAGE,
  constraint PK_MSPRODUCTFOTO primary key (PRODCODE),
  constraint FK_MSPRODUC_FOTOPRODU_MSPRODUC foreign key (PRODCODE)
    references MSPRODUCT (PRODCODE)
)
```

```
/

create or replace type TBTIMBANG_TYP as OBJECT (
  TIMBANGSEQ          NUMBER(4),
  UOMCODE             REF MSUOM_TYP,
  TIMBANGQTY          NUMBER(15,2),
  TIMBANGQTY1         NUMBER(15,2),
  AUDLOG              AuditLog_Typ
)
/

create type TBTIMBANG_NTABTYP as TABLE of TBTIMBANG_TYP
/

create or replace type WHTBDT_TYP as OBJECT (
  TBSEQ               NUMBER(4),
  UOMNONSTD           REF MSUOM_TYP,
  UOMSATPKG           REF MSUOM_TYP,
  PRODCODE            REF MSPRODUCT_TYP,
  CURRCODE            CHAR(3),
  GLMID               CHAR(10),
  GLSID               CHAR(8),
  AMTORG              NUMBER(17,2),
  EXCHRATE            NUMBER(11,5),
  AMT                 NUMBER(16,2),
  TRANSINQTY          NUMBER(15,2),
  TRANSINQTY1         NUMBER(15,2),
  INQTYACT            NUMBER(15,2),
  INQTYACT1           NUMBER(15,2),
  DLVRQTY             NUMBER(15,2),
  DLVRQTY1            NUMBER(15,2),
  DWQTY               NUMBER(15,2),
  DWQTY1              NUMBER(15,2),
  CONTROLQTY          NUMBER(15,2),
  CONTROLQTY1         NUMBER(15,2),
  SLSHTANGKAP         NUMBER(15,2),
  SLSHCONTROL         NUMBER(15,2),
  BRUTOQTYSTD         NUMBER(15,2),
  QTYPACKAGEIN        NUMBER(15,2),
  TRXPRICE            NUMBER(16,2),
  TBTRANSPORT         NUMBER(16,2),
  TBHRGSAT            NUMBER(11,2),
  TBHRGSATPRI         NUMBER(16,2),
  TBRETUR             NUMBER(11,2),
  CNDNPrice           NUMBER(16,2),
  TBHRGSEMENTARA      NUMBER(1),
  JNSRETUR            CHAR(1),
  TRANSDESC           VARCHAR2(128),
  DLVRTIME            DATE,
  TBGRPA              NUMBER(5,2),
  BERATKRJ            NUMBER(5,2),
  FGBASAH             CHAR(1),
  TRANSRANGE          VARCHAR2(10),
  NCASHMOHAMT         NUMBER(16,2),
  NCASHMOHPRICE       NUMBER(16,2),
  TIMBANG             TBTIMBANG_NTABTYP,
  AUDLOG              AuditLog_Typ,
```

```
  member function sumTimbangQty return number,
  member function sumTimbangQtyStd return number
)
/

create type WHTBDT_NTABTYP as TABLE of WHTBDT_TYP
/
create or replace type WHTBHD_TYP as OBJECT (
  TBID                    VARCHAR2(15),
--  BPID                  VARCHAR2(15),
  RELID                   VARCHAR2(15),
  TRXID                   CHAR(2),
  WRHSID                  REF MSWAREHOUSE_TYP,
  EXPDID                  VARCHAR2(5),
  WRHSORG                 REF MSWAREHOUSE_TYP,
  POID                    VARCHAR2(15),
  POSEQ                   NUMBER(4),
  GLMID                   CHAR(10),
  GLSID                   CHAR(8),
  TRXDATE                 DATE,
  TRXPRICE                NUMBER(16,2),
  TBDOCNMBR1              VARCHAR2(15),
  TBDOCNMBR2              VARCHAR2(15),
  INITIATOR               VARCHAR2(30),
  POSTDATE                DATE,
  JOURNALNMBR      VARCHAR2(15),
  POSTDATEAP              DATE,
  FARMCODE                VARCHAR2(10),
  FARMNAME                VARCHAR2(30),
  DISCRESPCODE            VARCHAR2(8),
  DISCSUBLEDGER           VARCHAR2(18),
  PPNRESPCODE             VARCHAR2(8),
  PPNSUBLEDGER            VARCHAR2(18),
  ADTRESPCODE             VARCHAR2(8),
  ADTSUBLEDGER            VARCHAR2(18),
  APRESPCODE              VARCHAR2(8),
  APSUBLEDGER      VARCHAR2(18),
  TRXSTAT                 CHAR(1),
  TRANSDESC               VARCHAR2(128),
  SOPIR                   VARCHAR2(30),
  NOPOL                   VARCHAR2(10),
  KODEMSK                 VARCHAR2(10),
  NODO                    VARCHAR2(10),
  WHTBDT                  WHTBDT_NTABTYP,
  APPROVAL                Approval_Typ,
  member function sumActPrice return number
)
/

create type WHBPFIFO_TYP as OBJECT (
  TRXINDATE          DATE,
  TRXQTY             NUMBER(15,2),
  TRXPRICE           NUMBER(16,2),
  TRXAMT             NUMBER(15,2),
  NCASHMOHAMT        NUMBER(16,2),
  NCASHMOHPRICE      NUMBER(16,2)
)
```

```
/

create type WHBPFIFO_NTABTYP as TABLE of WHBPFIFO_TYP
/

create or replace type WHBPDT_TYP AS OBJECT (
  BPSEQ                 NUMBER(4),
  CURRCODE              CHAR(3),
  TBID                  REF WHTBHD_TYP,
  TBSEQ                 NUMBER(4),
  GLMID                 CHAR(10),
  GLSID                 CHAR(8),
  UPERFECTQTY           NUMBER(15,2),
  UPERFECTQTY1          NUMBER(15,2),
  PRODCODE              REF MSPRODUCT_TYP,
  UOMNONSTD             REF MSUOM_TYP,
  UOMSATPKG             REF MSUOM_TYP,
  AMTORG                NUMBER(17,2),
  EXCHRATE              NUMBER(11,5),
  AMT                   NUMBER(16,2),
  TRANSOUTQTY           NUMBER(15,2),
  TRANSOUTQTY1          NUMBER(15,2),
  QTYPACKAGEOUT    NUMBER(15,2),
  TRXPRICE              NUMBER(16,2),
  BPHRGSAT              NUMBER(11,2),
  BPHRGSATPRI           NUMBER(16,2),
  STOREDV               NUMBER(16,2),
  TRANSDESC             VARCHAR2(128),
  PRID                  VARCHAR(15),
  PRSEQ                 NUMBER(4),
  NCASHMOHAMT           NUMBER(16,2),
  NCASHMOHPRICE         NUMBER(16,2),
  BPFIFO                WHBPFIFO_NTABTYP,
  AUDLOG                AuditLog_Typ,
  member function sumFifoAmt return number
)
/

create type WHBPDT_NTABTYP as TABLE of WHBPDT_TYP
/

create or replace type WHBPHD_TYP as OBJECT (
  BPID                  VARCHAR2(15),
  WRHSID                REF MSWAREHOUSE_TYP,
  WRHSDEST              REF MSWAREHOUSE_TYP,
  RELID                 VARCHAR2(15),
  TRXID                 CHAR(2),
  GLMID                 CHAR(10),
  GLSID                 CHAR(8),
  TRXDATE               DATE,
  BPDOCNMBR1            VARCHAR2(15),
  BPDOCNMBR2            VARCHAR2(15),
  INITIATOR             VARCHAR2(30),
  TRXPRICE              NUMBER(16,2),
  JOURNALNMBR      VARCHAR2(15),
  POSTDATE              DATE,
  APPOSTDATE            DATE,
```

```
  TRXSTAT                       CHAR(1),
  TRANSDESC                     VARCHAR2(128),
  WHBPDT                        WHBPDT_NTABTYP,
  APPROVAL                      Approval_Typ,
  member function sumActPrice return number
)
/

alter type WHTBHD_TYP
  add attribute BPID ref WHBPHD_TYP cascade
/

create table WHTBHD of WHTBHD_TYP (
  primary key (TBID),
  foreign key (WRHSID) REFERENCES MSWAREHOUSE,
  foreign key (WRHSORG) REFERENCES MSWAREHOUSE,
  POID          REFERENCES POHD (POID),
  GLMID         REFERENCES GLM (GLMID),
  GLSID         REFERENCES GLS (GLSID),
  EXPDID        REFERENCES MsExpedisi(Expdid),
  TRXID         REFERENCES RnTRx(TRXID),
  RELID         REFERENCES MsRelasi(Relid),
  TRXDATE           default sysdate  not null,
  TRXSTAT           default '0'  not null,
  constraint FK_WHTBHD_TRANS_IN_RNTRX foreign key (TRXID)
      references RNTRX (TRXID),
  constraint FK_WHTBHD_TBRELASI_MSRELASI foreign key (RELID)
      references MSRELASI (RELID),
  constraint FK_WHTBHD_TBPUNYAPO_PODT foreign key (POID, POSEQ)
      references PODT (POID, POSEQ),
  constraint FK_WHTBHD_WHTBGL_GLMAP foreign key (GLMID, GLSID)
      references GLMAP (GLMID, GLSID),
  constraint FK_WHTBHD_LPAHEXPED_MSEXPEDI foreign key (EXPDID)
      references MSEXPEDISI (EXPDID)
)
  object id primary key
  NESTED TABLE WHTBDT STORE AS WHTBDT_NTAB
   ((PRIMARY KEY(NESTED_TABLE_ID, TBSEQ))
   ORGANIZATION INDEX COMPRESS
     NESTED TABLE TIMBANG STORE AS TBTIMBANG_NTAB)
/

alter table WHTBDT_NTAB
  add scope for(UOMNONSTD) is MSUOM
/

alter table WHTBDT_NTAB
  add scope for(UOMSATPKG) is MSUOM
/

alter table WHTBDT_NTAB
  add scope for(prodCode) is MSPRODUCT
/

alter table TBTIMBANG_NTAB
  add scope for(UOMCODE) is MSUOM
/
```

```
create index TIMBANG_UOMSEKUNDER_FK on TBTIMBANG_NTAB (
  UOMCODE ASC
)
/

create index TB_SATNONSTANDAR_FK on WHTBDT_NTAB (
  UOMNONSTD ASC
)
/

create index WHTBDTGL_FK on WHTBDT_NTAB (
  GLMID ASC,
  GLSID ASC
)
/

create index WHTBDT_FK on WHTBDT_NTAB (
  CURRCODE ASC
)
/

create table WHBPHD of WHBPHD_TYP (
  primary key (BPID),
  foreign key (WRHSID) REFERENCES MSWAREHOUSE,
  foreign key (WRHSDEST) REFERENCES MSWAREHOUSE,
  GLMID   REFERENCES GLM (GLMID),
  GLSID   REFERENCES GLS (GLSID),
  TRXDATE          default sysdate,
  TRXSTAT          default '0'  not null,
  constraint FK_WHBPHD_TRANS_OUT_RNTRX foreign key (TRXID)
     references RNTRX (TRXID),
  constraint FK_WHBPHD_RBSUPPLIE_MSRELASI foreign key (RELID)
     references MSRELASI (RELID),
  constraint FK_WHBPHD_WHBPGL_GLMAP foreign key (GLMID, GLSID)
      references GLMAP (GLMID, GLSID)
)
  object id primary key
  NESTED TABLE WHBPDT STORE AS WHBPDT_NTAB
  ((PRIMARY KEY (NESTED_TABLE_ID, BPSEQ))
  ORGANIZATION INDEX COMPRESS
    NESTED TABLE BPFIFO STORE AS WHBPFIFO_NTAB)
/

alter table whtbhd
    add scope for (bpid) is WHBPHD
/

alter table WHBPDT_NTAB
  add scope for(UOMNONSTD) is MSUOM
/

alter table WHBPDT_NTAB
  add scope for(UOMSATPKG) is MSUOM
/

alter table WHBPDT_NTAB
```

```
  add scope for(prodCode) is MSPRODUCT
/

create index TRANS_OUT_FK on WHBPHD (
  TRXID ASC
)
/

create index BPWH_FK on WHBPHD (
  WRHSID ASC
)
/

create index WHBPGL_FK on WHBPHD (
  GLMID ASC,
  GLSID ASC
)
/

create index TOWH_FK on WHBPHD (
  WRHSDEST ASC
)
/

create index RBSUPPLIER_FK on WHBPHD (
  RELID ASC
)
/

create or replace type WHDOWNGRADEDT_TYP as OBJECT (
  DGSEQ          NUMBER(4),
  PRODCODE       REF MSPRODUCT_TYP,
  DGQTY          NUMBER(15,2),
  UOMCODE        REF MSUOM_TYP,
  TRXSTAT        CHAR(1),
  AUDLOG         AuditLog_Typ
)
/

create type WHDOWNGRADEDT_NTABTYP as TABLE of WHDOWNGRADEDT_TYP
/

create or replace type WHDOWNGRADEHD_TYP as OBJECT  (
  DGID                VARCHAR2(15),
  PRODCODE            REF MSPRODUCT_TYP,
  WRHSID              REF MSWAREHOUSE_TYP,
  TRXDATE             DATE,
  DGQTY               NUMBER(15,2),
  WHDOWNGRADEDT       WHDOWNGRADEDT_NTABTYP,
  APPROVAL            Approval_Typ
)
/

create table WHDOWNGRADEHD of WHDOWNGRADEHD_TYP (
  primary key (DGID),
  foreign key (PRODCODE) REFERENCES MSPRODUCT,
  foreign key (WRHSID) REFERENCES MSWAREHOUSE,
```

```
  TRXDATE          default sysdate
)
  object id primary key
  NESTED TABLE WHDOWNGRADEDT STORE AS WHDOWNGRADEDT_NTAB (
   (PRIMARY KEY(NESTED_TABLE_ID, DGSEQ))
   ORGANIZATION INDEX COMPRESS)
  RETURN AS LOCATOR
/

alter table WHDOWNGRADEDT_NTAB
  add scope for(PRODCODE) is MSPRODUCT
/

alter table WHDOWNGRADEDT_NTAB
  add scope for(UOMCODE) is MSUOM
/

create index DNGRADEWH_FK on WHDOWNGRADEHD (
  WRHSID ASC
)
/

create or replace type WHMONTHLY_typ as OBJECT
(
   lvlMonth            CHAR(6),
   lvlSaldo            NUMBER(15,2),
   lvlDb               NUMBER(15,2),
   lvlCd               NUMBER(15,2),
   lvlSaldoPri         NUMBER(16,2),
   lvlDbPri            NUMBER(16,2),
   lvlCdPri            NUMBER(16,2)
)
/

create type WHMONTHLY_ntabtyp as table of WHMONTHLY_typ
/

create or replace type WhStock_typ as OBJECT
(
   WrhsID              VARCHAR2(10),
   ProdCode            VARCHAR2(25),
   StockQty            NUMBER(15,2),
   StockCommit         NUMBER(15,2),
   StockOff            NUMBER(15,2),
   StockLocked         NUMBER(1),
   WHMONTHLY           WHMONTHLY_ntabtyp
)
/

create table WhStock of WHStock_typ
(
   WrhsID              not null,
   ProdCode            not null,
   StockQty            default 0 not null,
   StockCommit         default 0 not null,
   StockOff            default 0 not null,
   StockLocked         default 0 not null,
```

```
    primary key (WrhsID, ProdCode)
)
 nested table WHMONTHLY  store as WHMONTHLY_ntab(
     (primary key(nested_table_id, lvlMonth))
     organization index compress)
  return as locator
/

create type SKOPNDT_TYP as OBJECT(
  SKOPNSEQ          NUMBER(4),
  PRODCODE          REF MSPRODUCT_TYP,
  AUDLOG            AuditLog_Typ
)
/

create type SKOPNDT_NTABTYP as table of SKOPNDT_TYP
/

create type WhOpname_typ as OBJECT
(
  SKOpnID           VARCHAR2(15),
  InitDate          DATE,
  StartDate         DATE,
  EndDate           DATE,
  TrxStat           CHAR(1),
  Initiator         VARCHAR2(30),
  SKOPNDT           SKOPNDT_NTABTYP,
  Approval          Approval_Typ,
  Audlog            AuditLog_Typ
)
/

create table WHOpname of WHOpname_typ(
     primary key (SKOpnID)
)
 nested table SKOPNDT  store as SKOPNDT_NTAB(
     (primary key(nested_table_id, SKOPNSEQ))
     organization index compress)
  return as locator
/

alter table SKOPNDT_NTAB
  add scope for(PRODCODE) is MSPRODUCT
/

create type WhOpnameDt_typ as OBJECT
(
  OpnSeq            NUMBER(4),
  ProdCode          ref msProduct_typ,
  SoQtyBaru         NUMBER(15,2),
  SoQtyLama         NUMBER(15,2),
  SoEksekusi        NUMBER(1),
  SoDisetujui       NUMBER(1),
  SoHrgSatRp        NUMBER(16,2),
  KondisiBarang     CHAR(1),
  Audlog            AuditLog_Typ
)
```

```
/

create type WHOpnameDt_ntabtyp as table of WHOpnameDt_typ
/

create type WhOpnameHd_typ as OBJECT
(
   OpnID              VARCHAR2(15),
   WrhsID             ref msWarehouse_typ,
   SKOpn              ref whOpname_typ,
   TrxDate            DATE,
   Petugas            VARCHAR2(30),
   FgOpname           VARCHAR2(1),
   TrxStat            CHAR(1),
   TransDesc          VARCHAR2(200),
   OpnameDt           WHOpnameDt_ntabtyp
)
/

create table WHOpnameHD of WHOpnameHd_typ(
      primary key (OpnID),
      foreign key (WrhsID) references msWarehouse
)
      object ID primary key
   nested table OpnameDt store as WHOpnameDt_ntab(
               (primary key(nested_table_id, OpnSeq))
               organization index compress)
      return as locator
/

alter table WHOpnameDt_ntab
      add scope for (ProdCode) is MsProduct
/

alter table whOpnameHd
 add scope for (skOpn) is WhOpname
/
```

D.2 PENDEFINISIAN METHOD

```
create or replace type body WhTbDt_Typ as
 member function sumTimbangQty return number is
      i integer;
      total number := 0;
 begin
    for i in 1..self.timbang.count loop
      total := total + timbang(i).timbangQty;
    end loop;
    return total;
 end;

 member function sumTimbangQtyStd return number is
      i integer;
      total number := 0;
 begin
```

```
      for i in 1..self.timbang.count loop
        total := total + timbang(i).timbangQty1;
      end loop;
      return total;
  end;
END;
/

create or replace type body whtbhd_typ as
  member function sumActPrice return number is
       i integer;
       total number := 0;
  begin
      for i in 1..self.whtbdt.count loop
        total := total + whtbdt(i).trxPrice;
      end loop;
      return total;
  end;
END;
/

create or replace type body whBphd_typ as
  member function sumActPrice return number is
       i integer;
       total number := 0;
  begin
      for i in 1..self.whBpdt.count loop
        total := total + whBpdt(i).trxPrice;
      end loop;
      return total;
  end;
END;
/

create or replace type body Whbpdt_Typ as
  member function sumFifoAmt return number is
       i integer;
       total number := 0;
  begin
      for i in 1..self.bpFifo.count loop
        total := total + bpFifo(i).trxAmt;
      end loop;
      return total;
  end;
END;
/
```

LAMPIRAN E

Berikut adalah email dari Oracle yang menyatakan bahwa error ***ORA-00600: [qctcfx : len], [0], [0], [], [], [], [], []*** masih belum ditemukan solusinya :

From: Oracle, Ken Robinson 30-Jul-02 22:14

Subject: Re : ORA-00600: [qctcfx : len], [0], [0], [], [], [], [], []

COMPILING PLSQL WITH NESTED TABLE

I'm not finding an exact match, but there is a similar problem with nested tables that could be related filed in bug 2412956. The bug is still under investigation and it would be best to file an iTAR to get your diagnostics information (associated trace file(s)) into support. Something in your scenario may help development track down this problem more quickly.

Regards,

Ken Robinson
Oracle Server EE Analyst