IMPLEMENTASI KONEKSI DELPHI 7 TERHADAP PYTHON REMOTE OBJECT (PYRO) MELALUI TDATASET SECARA DATA AWARE

TUGAS AKHIR

Disusun Oleh :

FIRSTIAR NOORWINANTO

NRP. 5199 100 004

JURUSAN TEKNIK INFORMATIKA
FAKULTAS TEKNOLOGI INFORMASI
INSTITUT TEKNOLOGI SEPULUH NOPEMBER
SURABAYA

2006

IMPLEMENTASI KONEKSI DELPHI 7 TERHADAP PYTHON REMOTE OBJECT (PYRO) MELALUI TDATASET SECARA DATA AWARE

TUGAS AKHIR

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

Mengetahui / Menyetujui
Dosen Pembimbing

Wahyu Suadi, M.Kom.
NIP. 132 303 065

SURABAYA
JANUARI 2006

even if hopes turn into dispairs,
Live Goes On ...

ABSTRAK

Remote object memungkinkan suatu aplikasi client server dikembangkan dengan mudah. Client cukup melakukan koneksi ke sever, membuat semacam proxy dari remote object, dan memanggil fungsi-fungsi yang telah disediakan oleh server. Sehingga jika ada perubahan pada alur pemrosesan data pada server tanpa merubah argumen maupun data kembalian, client tidak perlu ikut dirubah.

Salah satu penerapan remote object adalah python remote object (pyro). Pyro menerapkan nameserver untuk menangani pengelompokan object berdasarkan grup. Lalu server mendaftarkan object yang disediakan untuk diakses oleh client kepada nameserver. Pyro cukup bagus mengingat modul ini adalah open source dan berukuran kecil dimana fasilitas remote object yang disediakan cukup memadai. Karena ditulis dalam bahasa python, kendala pyro yang terbesar adalah antarmuka.

Hal ini mendorong penulis untuk mencoba mengimplementasikan penggunaan dua bahasa untuk menangani masalah tersebut. Python di sisi remote object menggunakan pyro dan delphi di sisi antarmuka menggunakan konsep data aware. Dalam menangani tabel, di sisi delphi menggunakan TClientDataSet sebagai representasi dari tabel.

KATA PENGANTAR

Segala puji dan syukur Alhamdulillah semata ditujukan ke hadirat ALLAH SWT yang telah memberikan rahmat dan hidayah-Nya sehingga memungkinkan penulis untuk menyelesaikan Tugas Akhir yang berjudul "***IMPLEMENTASI KONEKSI DELPHI 7 TERHADAP PYTHON REMOTE OBJECT (PYRO) MELALUI TDATASET SECARA DATA AWARE***".

Mata Kuliah Tugas Akhir yang memiliki beban sebesar empat 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 mengemukakan tentang alternatif remote object dan integrasi python dengan delphi.

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 mengharapkan saran serta kritik yang membangun dari rekan-rekan pembaca.

Surabaya, Januari 2006

Penulis

UCAPAN TERIMA KASIH

Pengerjaan dan penyusunan Tugas Akhir ini tak lepas dari bantuan, bimbingan dan pengarahan dari berbagai pihak. Untuk itu perkenankanlah penulis mengucapkan banyak terima kasih kepada :

1 Allah SWT atas berkah dan rahmat serta hidayah yang telah diberikan kepada penulis hingga penulis bisa sampai sejauh ini.

2 Bunda beserta keluarga yang selama ini medukung baik secara moral maupun material hingga penulis dapat menyelesaikan Tugas Akhir ini dengan baik. Mohon maaf jika ternyata masa kuliah penulis berlarut-larut.

3 Bapak Wahyu Suadi M.Kom selaku pembimbing Tugas Akhir yang telah mencurahkan bimbingan dan ilmu yang berhubungan dengan Tugas Akhir.

4 Bapak Yudhi P. selaku Kepala Jurusan Teknik Informatika ITS, yang telah memberi banyak pelajaran dan contoh yang baik pada penulis pada khususnya dan pada mahasiswa Informatika pada umumnya demi keharuman nama Teknik Informatika ITS.

5 Bapak F.X. Arunanto, Royyana Muslim I. M.Kom dan Ary Mazharuddin Shiddiqi S.Kom selaku dosen penguji dalam seminar Tugas Akhir yang memberikan pengalaman, ilmu, dan kritik yang berharga bagi penulis.

6 Ibu Dr. Handayani Tjandrasa M.Sc dan bapak Wahyu Suadi M.Kom selaku dosen wali yang selama perkuliahan memberikan masukan kepada penulis baik ketika masih kuliah maupun ketika memasuki tahap penyelesaian Tugas Akhir.

7 Bapak dan Ibu Dosen Teknik Informatika ITS yang tidak dapat penulis sebutkan satu per-satu, yang telah banyak membantu penulis selama kuliah di Teknik Informatika hingga penulis merasa menyesal sekali apabila belum pernah mendapatkan bimbingan dari bapak dan ibu dosen.

8 Seluruh karyawan dan staff TU Teknik Informatika dan FTIF dalam pelayanannya yang baik dan ramah dalam hal birokrasi.

9 Prevan '00, Tyarso '00, Farid '00, Iwan san '99 dan Kris '01 yang memberikan dukungan teknis mulai penyediaan *library*, penyediaan data, hingga format buku Tugas Akhir.

10 Dh '99, Dark Wizard a.k.a D '99, in '99, skyfish '99, yo '99, zerkling '99, zzz '99, Quatermion '00, Open '01, Bejo '02, kebo giras '02, Udin '03, Leonhart '04 dan para DotA'ers dan Labprog'ers yang memberikan hiburan sehingga dalam masa penulisan Tugas Akhir ini penulis dapat menenangkan otak sejenak.

11 Seluruh mahasiswa Teknik Informatika ITS yang tidak dapat penulis sebutkan satu persatu karena keterbatasan tempat, yang telah memberikan pengalaman, ilmu serta lingkungan yang amat sangat penulis hargai.

Surabaya, Januari 2006

Penulis

DAFTAR ISI

DAFTAR GAMBAR

DAFTAR TABEL

BAB I

PENDAHULUAN

Dalam bab ini dijelaskan beberapa hal dasar yang meliputi latar belakang, permasalahan, batasan permasalahan, tujuan dan manfaat, metodologi pelaksanaan serta sistematika penulisan buku Tugas Akhir ini. Dari uraian tersebut diharapkan, gambaran umum permasalahan dan pemecahan yang diambil dapat dipahami dengan baik.

1.1 Latar Belakang

Dalam beberapa tahun terakhir, pemanfaatan teknologi jaringan benar-benar berkembang pesat. Salah satu metode koneksi yang digunakan adalah remote object. Berbagai tools telah tersedia di internet. Contohnya adalah CORBA, Java Remote Method Invocation (RMI). Bahkan bahasa scripting sekalipun juga menyediakan modul-modul tambahan untuk menangani remote object. Salah satunya adalah Python dengan Python Remote Object (PyRO).

Dengan menggunakan remote object, maka definisi proses dapat dilakukan pada server. Sehingga ketika terjadi perubahan alur proses maupun penambahan method lokal yang hanya dibutuhkan oleh server, maka client tidak perlu melakukan perubahan apapun selama argumen dalam request dan format data kembalian tidak berubah. Karena berbahasa script, pemrograman menggunakan script python lebih kearah pemrograman konsol. Kalaupun dapat menggunakan dialog form, fungsionalitasnya sangat terbatas.

Dalam delphi, tersedia komponen TClientDataSet yang merupakan turunan dari TDataSet. Komponen ini dapat dimanipulasi layaknya membuat tabel virtual dalam memori. Sehingga tabel yang direpresentasikan dalam TClientDataSet dapat dimanipulasi sesuai keinginan programer. Namun delphi tidak menyediakan fasilitas remote object.

1.2 Tujuan

Mengimplementasikan dua buah class yang berfungsi sebagai komunikator antara Python Remote Object (Pyro) dan TClientDataSet. Class pertama bertugas sebagai adapter dari pyro dan yang lain sebagai dataset manipulator.

Adapter tersebut memiliki proxy object dari remote object yang akan digunakan dan ditulis dalam bahasa python. Sedangkan dataset manipulator adalah class yang melakukan inisialisasi, penambahan record, perubahan record dan sebagainya terhadap dataset berdasarkan data dari adapter pyro client tersebut.

Arsitektur sistem ini bermanfaat untuk memberikan alternatif bagi pengembang dalam membuat sebuah aplikasi berbasis RPC yang berukuran kecil dan mudah digunakan sekaligus memiliki antarmuka yang mendukung fungsionalitas aplikasi tersebut. Selain itu juga untuk membuktikan bahwa python dapat dikombinasikan dengan bahasa pemrograman lain yang memiliki antarmuka yang lebih lengkap fasilitasnya seperti Delphi dalam contoh yang digunakan dalam sistem ini.

1.3 Permasalahan

Permasalahan yang dihadapi dalam pembuatan Tugas Akhir ini adalah:

a. Bagaimana merancang dan mengimplementasikan operasi database MySQL pada python dengan MySQLdb sebagai database adapter.
b. Bagaimana merancang dan mengimplementasikan sistem yang berbasis RPC dengan menggunakan Pyro, sehingga pyro client dapat mengakses database melalui *pyro server* yang menggunakan MySQLdb sebagai database adapter.
c. Bagaimana merancang dan mengimplementasikan konversi object dari python ke delphi dengan bantuan python for delphi sehingga aplikasi utama yang berbasis python dapat diintegrasikan dengan antarmuka yang berbasis delphi.
d. Bagaimana mengimplementasikan sebuah class yang menangani konversi data di atas mulai dari data yang berupa recordset pada pyro hingga memanipulasi TClientDataset pada delphi dengan data tersebut.
e. Bagaimana melakukan pengamanan data dengan menggunakan metode enkripsi dan session yang sederhana.

1.4 Batasan Masalah

Sejumlah batasan masalah dalam pembuatan Tugas Akhir ini adalah:

a. Uji coba dilakukan dengan menggunkan sebuah server database dengan menggunakan MySQL, sebuah server database sederhana yang tidak begitu rumit dan mudah dioperasikan, dan sebuah komputer yang

berlaku sebagai client yang menggunakan aplikasi yang dibangun, dimana uji coba ini dilakukan dalam lingkungan LAN jurusan Teknik Informatika ITS.

b. Dalam mengimplementasikan aplikasi yang akan dibangun ini, penulis menggunakan library-library atau modul-modul MySQLdb, python remote object (Pyro) dan pyCrypto yang berbasis Python 2.4 dan Deplhi 7 dengan TClientDataset untuk penanganan data aware dan TDBGrid dan TDBEdit untuk penanganan antarmuka.

c. Aplikasi yang akan dibangun akan menggunakan data pada database Ruang Baca Teknik Computer-Informatika berupa tabel anggota dan master koleksi.

d. Pengujian yang dilakukan berupa perbandingan kecepatan proses antara aplikasi yang dibangun dan aplikasi yang langsung terhubung ke database. Pengamanan data hanya sebagai contoh salah satu kelebihan dalam model aplikasi yang dibangun.

1.5 Metodologi

Pembuatan Tugas Akhir ini terdiri dari beberapa tahapan. Tahapan tersebut adalah sebagai berikut:

a. Studi Literatur

Pada tahap ini dilakukan pengumpulan informasi – informasi yang diperlukan dalam proses perancangan dan implementasi sistem yang akan dibangun. Adapun informasi – informasi yang diperlukan diantaranya

adalah spesifikasi database, spesifikasi database adapter, spesifikasi Pyro sebagai ekstensi modul remote object pada python, spesifikasi pycrypto sebagai ekstensi modul untuk melakukan enkripsi pada python, penerapan database adapter dan Pyro dengan bahasa pemrograman Python 2.4 sebagai client-server dan bahasa pemrograman Borland Delphi 7 sebagai GUI yang menggunakan pendekatan secara data aware.

b. Perancangan dan Implementasi Sistem

Pada tahap ini dilakukan perancangan dan implementasi sistem yang meliputi:

- Perancangan Arsitektur Sistem

  Perancangan arsitektur sistem adalah tahapan untuk menentukan model arsitektur sistem untuk mengimplementasikan koneksi server terhadap database, koneksi client terhadap server dengan berbasis RPC dengan tujuan untuk mengoptimalkan koneksi yang ada, dan konversi dari client yang berbasis python ke TDataSet yang berbasis delphi.

- Perancangan Data

  Yaitu perancangan database dengan format umum, format baku dalam pertukaran data, format kompresi dan enkripsi data, serta perancangan data-data yang digunakan dalam proses konversi dari python ke delphi.

- Perancangan Proses

Yaitu perancangan proses dari koneksi ke database oleh server yang dilakukan oleh object tertentu, yang selanjutnya object tersebut akan diakses secara remote oleh client dengan menggunakan *object proxy*. dilanjutkan proses konversi dari python ke delphi, yang dilanjutkan dengan perancangan class untuk mewakili datasource yang akan digunakan oleh TDataSet.

- Perancangan Antarmuka

Yaitu perancangan antarmuka untuk memudahkan pengoperasian perangkat – perangkat lunak di dalam sistem yang akan dibangun.

- Implementasi

Yaitu pembuatan perangkat lunak sesuai dengan perancangan yang telah dilakukan dengan menggunakan bahasa pemrograman Python 2.4 di sisi client-server dan Delphi 7 sebagai penyedia data aware dan GUI.

c. Uji Coba dan Analisa

Pada tahap ini dilakukan uji coba dengan skenario tertentu untuk membuktikan bahwa dengan impelementasi model tersebut, kecepatan proses data tidak kalah secara signifikan, dan bagaimana kemudahan dalam update jika ada perubahan di sisi proses bisnis.

d. Penyusunan Laporan Tugas Akhir

Penyusunan laporan Tugas Akhir ini berisi informasi – informasi mengenai sistem yang telah dibangun dan sekaligus merupakan tahapan akhir dari pelaksanaan Tugas Akhir.

1.6 Sistematika Penulisan

Buku Tugas Akhir ini terdiri dari beberapa bab yang tersusun secara sistematis, yaitu :

BAB I PENDAHULUAN

Bab ini merupakan gambaran umum yang membahas latar belakang dan tujuan pembuatan perangkat lunak, serta 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 DASAR TEORI

Bab ini membahas teori-teori dasar yang menunjang pengerjaan Tugas Akhir, meliputi *database adapter*, *Python Remote Object*, *Pyhton for delphi* sebagai penghubung antara python dan delphi, dan data aware secara umum.

BAB III PERANCANGAN SISTEM

Dalam bab ini diuraikan mengenai sistem yang akan dibangun yakni mengenai perancangan arsitektur sistem, perancangan data beserta

proses – proses yang ada, serta perancangan antarmuka untuk menjalankan perangkat – perangkat lunak yang berada dalam sistem.

BAB IV IMPLEMENTASI SISTEM

Dalam bab ini diuraikan implementasi dari perancangan yang telah dilakukan sebelumnya yakni implementasi arsitektur sistem dengan pengadaan komputer – komputer sebagai server dan client beserta perangkat – perangkat lunaknya.

BAB V UJI COBA DAN EVALUASI

Berisi tentang uji coba yang telah dilakukan terhadap sistem yang telah dibuat beserta evaluasi dari hasil uji coba tersebut.

BAB VI PENUTUP

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

BAB II

DASAR TEORI

Dalam bab ini dijelasakan mengenai teori – teori penunjang, khususnya mengenai database adapter, python remote object (pyro), pyhton for delphi, data aware dan implementasinya dengan menggunakan Python dan Borland Delphi 7, yang digunakan dalam menyelesaikan Tugas Akhir ini.

2.1 Database Adapter

Berikut ini adalah penjelasan beberapa konsep yang berkaitan dengan database adapter yang meliputi definisi dan library yang digunakan sebagai penunjang bagi penulis dalam pengerjaan Tugas Akhir ini.

2.1.1 Definisi Database Adapter

Database adapter merupakan adapter yang berinteraksi langsung dengan database. Database adapter ini menerjemahkan permintaan akses ke database dari *broker* ke SQL dan mengembalikan hasil dari query tersebut kedalam format yang dapat dimengerti oleh *broker*. Untuk server enterprise, database adapter menerjemahkan *request event* dan mengembalikan hasil dalam bentuk *reply event*.

2.1.2 MySQLdb Sebagai Database Adapter

MySQLdb adalah modul python yang berukuran kecil. Merupakan *wrapper* dari *_mysql*, modul dasar dari python untuk melakukan koneksi ke

database. _mysql menyediakan antarmuka yang umumnya merupakan implementasi dari MySQL C API.

_mysql telah melakukan wrapping MySQL C API ke bentuk orientasi object. Struktur data yang masih tetap digunakan hanya MYSQL sebagai penanganan koneksi database dan MYSQL_RES sebagai pengananan hasil. Semua fungsi yang tidak menggunakan struktur data MYSQL sebagai parameter akan didefinisikan sebagai fungsi modul. Penggunaan *_mysql* cukup menyulitkan karena untuk melakukan query, modul ini tidak menerima banyak argumen. Satu-satunya argumen yang diterima harus berupa string. Untuk mendapatkan hasil, _mysql menyediakan fungsi yang mengembalikan record satu per satu atau semua sekaligus. Selain itu juga tidak disediakan konversi data dari database ke python.

MySQLdb dibuat untuk memudahkan pengembang dalam menggunakan modul *_mysql*. MySQLdb kompatibel dengan antarmuka python db API versi 2. Kode-kode yang mengimplementasikan API berada pada *_mysql*. Sehingga MySQLdb tidak perlu lagi mengimplementasikan API. Hal ini membuat modul *MySQLdb* menjadi cukup sederhana.

2.1.3 Object dalam MySQLdb

MySQLdb memiliki dua object utama. Yaitu object *Connection* dan *Cursor*. Object connection bertugas untuk menangani koneksi dengan database. Merupakan hasil kembalian ketika memanggil fungsi `connect()` dari modul.

Method-method yang dimiliki oleh object *connection* tertera pada tabel 2.1. Jika database dan tabel memiliki fasilitas *transaction*, maka object *connection*

dapat melakukan *commit* atau *rollback*. Setelah membuat object *Connection*, agar dapat melakukan query, dapat dipanggil fungsi *cursor* untuk membuat object *Cursor*.

Tabel 2.1 method dari object *Connection*

Method	Penjelasan
`commit()`	Jika database dan tabel memiliki fasilitas *transaction*, maka fungsi ini melakukan *commit* terhadap *transaction* yang sedang berlangsung. Jika tidak, maka fungsi ini tidak melakukan apa-apa.
`rollback()`	Jika database dan tabel memiliki fasilitas *transaction*, maka fungsi ini melakukan *rollback* terhadap *transaction* yang sedang berlangsung. Jika tidak, maka fungsi ini memanggil *NotSupportedError*.
`cursor()`	Untuk membuat object kursor yang nantinya digunakan untuk mengeksekusi query.

Method-method yang dimiliki oleh varian object *cursor* tertera pada tabel 2.2. Method-method umum tersebut berjumlah enam buah yaitu execute, executemany, nextset, fetchone, fetchmany dan fetchall. Execute dan executemany digunakan untuk mengeksekusi query. Sisanya digunakan untuk mengambil data hasil query. Yang membedakan masing-masing method adalah banyaknya data yang terlibat didalamnya. Dapat berupa satu atau lebih data yang terlibat.

Tabel 2.2 Method dari varian object *Cursor*

Method	Penjelasan
`execute()`	Mengeksekusi parameter sebagai query. Dapat berupa string maupun beberapa set parameter.
`executemany()`	Mengeksekusi parameter sebagai beberapa query sekaligus.
`nextset()`	Melanjutkan kursor ke record berikutnya. Membuang data yang sedang ditangani.
`fetchone()`	Mengambil satu record berikutnya dari hasil query. Mengembalikan *None* jika tidak ada lagi record yang dapat diambil.
`fetchmany()`	Mengambil beberapa record berikutnya dari hasil query sesuai dengan parameter yang disediakan. Mengembalikan tuple yang kosong jika tidak ada lagi record yang dapat diambil.
`fetchall()`	Mengambil semua record dari hasil query sesuai dengan parameter yang disediakan. Mengembalikan tuple yang kosong jika tidak ada lagi record yang dapat diambil.

Selain Cursor sebagai cursor default, MySQLdb juga menyediakan varian lain dari cursor. Varian-varian tersebut memiliki beberapa perbedaan. Varian tersebut adalah DictCursor, SSCursor, SSDictCursor.

a. DictCursor

Cursor ini pada prinsipnya sama dengan *Cursor*. Yang membedakan adalah hasil kembalian dari kedua *cursor* tersebut. Jika *Cursor* mengembalikan tuple sebagai hasil query, maka *DictCursor* mengembalikan *dictionary* sebagai hasil query.

b. SSCursor dan SSDictCursor

Kedua *Cursor* ini berbeda pendekatan dengan *Cursor* dan *DictCursor*. Jika kedua *cursor* awal mengambil semua kembalian hasil query dan menyimpan data pada client, maka kedua *cursor* terakhir ini menyimpan hasil query pada server. Sesuai dengan namanya (*Server Side Cursor*). Ketika user memanggil fungsi-fungsi *fetch*, data dikirim ke client. Sebelum data pada sisi server habis, *cursor* ini tidak dapat melakukan query lagi.

2.1.4 Penggunaan MySQLdb dalam Python

Untuk menggunakan modul MySQLdb, pada dasarnya hanya menggunakan beberapa langkah. *Import* modul MySQLdb ke aplikasi yang dibuat, buat satu koneksi, gunakan koneksi tersebut untuk membuat kursor. Selanjutnya cukup menggunakan kursor tersebut untuk mengeksekusi query. Contoh penggunaannya dapat dilihat pada gambar 2.1.

```
import MySQLdb
db=MySQLdb.connect(host='127.0.0.1',user='first',db='test')
```

Gambar 2.1 Koneksi ke mysql menggunakan MySQLdb

Db adalah object *connection* yang merupakan kembalian dari fungsi connect yang dipanggil. Untuk mengeksekusi query, dapat menggunakan kursor yang dibuat oleh db sebagai *connection*. Setelah membuat kursor, gunakan kursor tersebut untuk mengeksekusi query.

```
c=db.cursor()
id=5
c.execute("""SELECT * FROM person
WHERE id=%s""",(id,))
```

Gambar 2.2 Query pada MySQLdb

Dalam contoh pada gambar 2.2, id=5. Parameter menggunakan %s karena MySQLdb akan merubah nilai tersebut ke string sebelum meneruskan ke string query tersebut. Sedangkan pada parameter digunakan tuple karena *DB API* membutuhkan tuple sebagai parameter.

Setelah melakukan query, untuk mendapatkan hasil query, dapat menggunakan salah satu fungsi *fetch* yang tersedia. Contoh dapat dilihat pada gambar 2.3.

```
>>> c.fetchone()
(5L, 'Ryusei', 21L, 1L)
```

Gambar 2.3 Mengambil hasil query

Hasil dari fungsi *fetch* tersebut berupa *tuple*. Field yang bersesuaian dikonversikan sesuai dengan tipe data yang ditentukan, kecuali *integer*. Karena khusus *integer*, MySQLdb tidak mengkonversi langsung ke *integer*, melainkan

dikonversi ke *long integer* untuk menghindari nilai yang melebihi batas maksimum *integer*.

Untuk mendapatkan beberapa record sekaligus, dapat digunakan fungsi *fetchmany()* dan *fetchall()*.

Untuk melakukan beberapa query yang sejenis sekaligus, dapat menggunakan *executemany()*.

```
c.executemany(
      """INSERT INTO person
      (id, name, age, gender)
      VALUES (%s, %s, %s, %s)""",
      [
(2, 'Raidith', 25, 1),
(3, 'Aya', 19, 0),
(3, 'Ryusei', 21, 1)
```

Gambar 2.4 Melakukan beberapa query yang sejenis sekaligus

Seperti contoh pada gambar 2.4, dengan satu *formatter*, dapat digunakan untuk melakukan beberapa query yang sejenis sekaligus.

2.2 Python remote object

Python remote object (Pyro) merupakan sistim teknologi objek terdistribusi yang kuat dan terdepan. Mirip dengan Remote Method Invocation (RMI) milik Java, namun berbeda dengan CORBA yang independen terhadap sistem dan menawarkan beberapa kelebihan dari Pyro atau RMI. Akan tetapi, Pyro berukuran kecil, sederhana, mudah, dan free.

Dalam sistem object terdistribusi, ada dua bagian yang pasti dalam sistem. Client yang menginisialisasi pemanggilan method. Dan server yang mengeksekusi method. Diantara client dan server merupakan *middleware*, dalam kasus ini Pyro.

Pyro memiliki shell script yang merupakan tools untuk membangun aplikasi yang menggunakan Pyro. Salah satunya adalah nameserver yang menyimpan nama object yang didaftarkan oleh aplikasi server.

Client merupakan bagian dari sistem yang mengirim request ke server, untuk melakukan aktifitas tertentu. Ini adalah kode yang menggunakan remote object dengan cara memanggil method dari remote object tersebut. Untuk membuat instance object, client harus melalui dua langkah. Yang pertama adalah mencari identifier lokasi dari object yang dibutuhkan. Hal ini dilakukan oleh Pyro name server. Membuat object khusus yang memanggil remote object. Object khusus ini disebut *proxy*. Setelah client memiliki *proxy*, client dapat memanggil object tersebut sebagaimana object lokal.

Server adalah letak dari object yang akan diakses secara remote. Setiap instance harus merupakan bagian dari program python. Server memerlukan empat langkah untuk menyediakan instance. Yang pertama, membuat instance dengan sedikit tambahan kode Pyro. Memberi nama dan mendaftarkan ke Pyro name server. Memberi tahu Pyro bahwa dia harus menangani instance tersebut. Dan yang terakhir, meminta Pyro untuk menunggu pemanggilan method yang dapat datang sewaktu-waktu.

Remote object merupakan *instance* dari class turunan *Pyro.core.ObjBase*. Pada saat inisialisasi object juga melakukan inisialisasi *Pyro.core.ObjBase*.

2.2.1 Submodul dan Object Yang Perlu Diperhatikan Dalam Pyro

Pyro memiliki beberapa submodul yang peranannya sangat penting dalam membangun sebuah aplikasi berbasis RPC pada python. Submodul tersebut adalah *Pyro.core*, *Pyro.config*, *Pyro.naming*, dan *Pyro.error*.

Tabel 2.3 Beberapa object dan method yang penting dalam *Pyro.core*

Object / Method	Penjelasan
`ObjBase`	Merupakan class dasar untuk *remote object*. Ada dua cara menggunakan class ini. Yang pertama adalah diturunkan langsung, dan yang kedua adalah pendelegasian.
`Daemon`	Merupakan class yang bertugas melakukan *looping* menunggu *request* dari client.
`initServer()`	Method yang menginisialisasi aplikasi sebagai server.
`initClient()`	Method yang menginisialisasi aplikasi sebagai client.

Submodul *Pyro.core* merupakan inti dari modul pyro. Seperti tertera pada tabel 2.3, submodul ini memiliki class yang bernama *ObjBase*. Class inilah yang diturunkan menjadi object-object yang diakses secara remote. Object lain yang penting peranannya pada sisi server adalah *Daemon*. Object inilah yang melakukan *request loop*. Selain kedua object tersebut, Pyro.core juga menyediakan dua method yang menentukan apakah aplikasi yang dibangun

merupakan server atau client. Method-method tersebut tersebut adalah *initServer* dan *initClient*.

Submodul *Pyro.config* merupakan configurasi dari aplikasi yang menggunakan pyro, baik server maupun client. Configurasi yang harus diset adalah `Pyro.config.PYRO_NS_DEFAULTGROUP=group`. Group adalah nama grup yang digunakan oleh server dalam mengelompokkan object-object yang dibuka untuk dapat diakses secara remote.

Tabel 2.4 Beberapa object dan method yang penting dalam *Pyro.naming*

Object / Method	Penjelasan
`NameServerLocator`	Merupakan class yang fungsinya mencari nameserver.
`NameServerProxy`	Merupakan class yang membuat nama absolut agar dapat mengakses nameserver dengan benar. Class ini digunakan oleh *NameServerLocator*.
`detectNS()`	Method dari *NameServerLocator* yang digunakan untuk mendeteksi apakah pada ip address tertentu ada *nameserver* yang aktif.
`getNS()`	Method dari *NameServerLocator* yang digunakan untuk mengambil *nameserver* yang aktif. Jika ip address tidak disertakan, maka method ini akan menggunakan mekanisme broadcast.

Seperti disebutkan pada tabel 2.4, submodul *Pyro.naming* memiliki sebuah class yang bertugas menangani segala hal yang berhubungan dengan penamaan object. Class tersebut bernama *NameServerLocator*. Class ini memiliki satu method yang umum dipakai, yaitu *getNS*. Method ini bertugas mencari *nameserver* yang sedang online. Jika method ini dipanggil tanpa argumen, maka *locator* akan menggunakan mekanisme broadcast yang dimiliki *NameServerLocator*. Hal ini hanya dapat berfungsi jika jaringan mendukung mekanisme broadcast, dan server terjangkau oleh broadcast. Jika tidak, sebaiknya menggunakan *ip address* dan port (optional) dari *nameserver* sebagai argumen.

Submodul *Pyro.error* memiliki beberapa event yang digunakan untuk mendefinisikan error yang terjadi. Dua event tersebut adalah *PyroError* dan *NamingError*. Kedua error ini hanya diperlukan pada sisi server karena object yang dieksekusi hanya terletak pada server.

2.2.2 Penggunaan Pyro

a. Sisi server

Pada sisi server, yang harus dilakukan adalah mengimport modul-modul dasar dari pyro seperti disebut di atas termasuk *Pyro.error*. inisialisasi aplikasi sebagai server dengan memanggil *Pyro.core.initServer()*. Lalu set grup default dengan nama yang diinginkan. Contoh dapat dilihat pada gambar 2.5.

```
import sys
import Pyro.naming
import Pyro.core
from Pyro.errors import PyroError,NamingError
import testModule

group = ':testdb'

Pyro.core.initServer()
Pyro.config.PYRO_NS_DEFAULTGROUP=group
```

Gambar 2.5 Inisialisasi server

Setelah aplikasi terinisialisasi sebagai server, maka selanjutnya adalah mencari *nameserver* yang tersedia. Hal ini dilakukan dengan cara membuat *instance* dari class *NameServerLocator* dan memanggil fungsi *getNS()*.

Setelah *nameserver* didapat, maka dicoba untuk membuat grup baru dengan memanggil *createGroup* dari *nameserver* yang didapat sebelumnya. Jika sudah ada, maka lewati proses ini. Contoh seperti pada gambar 2.6.

```
locator = Pyro.naming.NameServerLocator()
ns = locator.getNS(host='127.0.0.1', port=9090)

try:
      ns.createGroup(group)
except NamingError:
      pass
```

Gambar 2.6 Mencari nameserver dan membuat grup baru

Setelah *nameserver* didapat dan nama grup telah ada atau selesai dibuat, maka langkah selanjutnya adalah membuat daemon dan memerintahkan daemon untuk menggunakan *nameserver* yang telah didapat seperti pada gambar 2.7.

```
daemon = Pyro.core.Daemon()
daemon.useNameServer(ns)
```

Gambar 2.7 Membuat daemon

Setelah membuat daemon, developer dapat melakukan registrasi object-object yang akan diakses secara remote. Adapun sebelum melakukan registrasi, ada baiknya jika mencoba menghapus terlebih dahulu object yang telah ada agar tidak terjadi duplikasi nama. Seperti yang dicontohkan pada gambar 2.8.

```
try:
      ns.unregister('testdb')
except NamingError:
      pass
daemon.connect(testModule.testObj(),'testObj')
# string 'testObj' pada argumen terakhir adalah
# nama object
```

Gambar 2.8 Meregistrasi *remote object*

Setelah semua object yang diiginkan teregistrasi, langkah terakhir yang harus dilakukan adalah memerintahkan daemon untuk melakukan

looping menunggu request yang akan datang dari sisi client seperti pada gambar 2.9.

```
try:
      daemon.requestLoop()
except KeyboardInterrupt:
      print 'shutdown gracefully!'
```

Gambar 2.9 Daemon request loop

b. Sisi client

Pada sisi client, yang harus dilakukan adalah mengimport modul-modul dasar dari pyro seperti disebut di atas kecuali *Pyro.error*. inisialisasi aplikasi sebagai client dengan memanggil *Pyro.core.initClient()*. Lalu set grup default dengan nama yang diinginkan.

Setelah aplikasi terinisialisasi sebagai client, maka selanjutnya adalah mencari *nameserver* yang tersedia. Hal ini dilakukan dengan cara membuat *instance* dari class *NameServerLocator* dan memanggil fungsi *getNS()*. Sampai langkah ini, persis sama dengan yang dilakukan pada server kecuali import Pyro.error yang ditiadakan pada client.

Setelah *nameserver* didapat, langkah selanjutnya adalah mencari string URI dari object yang akan diakses secara remote. Hal ini dilakukan oleh method *resolve* pada *nameserver*. Setelah mendapatkan URI dari object yang diinginkan, maka selanjutnya adalah membuat object proxy

dari remote object tersebut dengan perintah *getAttrProxyForURI*. Seperti pada gambar 2.10.

```
try:
    URI = ns.resolve('testObj')
except Pyro.core.PyroError,x:
    print 'dbconn object cannot be found:',x
    raise SystemExit
test = Pyro.core.getAttrProxyForURI(URI)
# contoh memanggil method dari remote object
test.myMethod(arguments)
```

Gambar 2.10 Mendapatkan URI dan Membuat object proxy

Setelah object proxy didapat, maka client dapat memanggil method-method dari remote object dengan cara sebagaimana memanggil method secara konvensional.

2.3 Kriptografi pada python

Python tidak menyediakan modul-modul enkripsi secara default. Namun dalam internet tersedia beberapa ekstensi dari python dalam hal enkripsi. Salah satunya adalah pycrypto dari A.M. Kuchling yang menyediakan modul Crypto untuk python. Fasilitas pycrypto terbilang lengkap yaitu hash dan cipher. Pycrypto menyediakan object-object yang mewakili fungsi-fungsi hash dan cipher.

Fungsi hash menggunakan string sebagai masukan dan menghasilkan keluaran dalam ukuran yang tetap bergantung pada input yang diberikan. Fungsi

has termasuk fungsi yang nonreversible. Sehingga tidak mungkin mendapatkan string masukan hanya dengan menggunakan hasil dari fungsi hash.

Untuk fungsi hash, pycrypto telah mengimplementasikan MD2, MD4, MD5, RIPEMD dan SHA. MD2, MD4 dan MD5 menggunakan panjang digest 128 bit. Sementara RIPEMD dan SHA menggunakan panjang digest 160 bit. Cara menggunakan fungsi-fungsi hash tersebut, terlebih dahulu harus dibuatkan object yang menangani fungsi hash tertentu. Setelah object tersedia, nilai string dari object tersebut diupdate dengan nilai yang ingin di-hash. Dan terakhir memanggil method digest dari object hash tersebut.

Tabel 2.5 Property dan method dari object hash

Property / method	Penjelasan
`digest_size`	Merupakan nilai integer. Ukuran dari hasil digest dari object hash.
`copy()`	Menggandakan object hash. Update pada hasil kopi tidak merubah object yang asli.
`digest()`	Mengembalikan hasil hash dari object hash. Berupa string yang menggunakan data 8 bit per elemen.
`hexdigest()`	Seperti method digest, tapi mengembalikan string heksadesimal. Panjangnya dua kali panjang digest.
`update()`	Mengupdate nilai string dari object hash. Ketika memanggil method digest atau hexdigest, yang di-hash adalah string tersebut.

Contoh penggunaan dapat dilihat pada gambar berikut.

```
from Crypto.Hash import MD5
m = MD5.new()
m.update('Hello World')
# gunakan digest untuk unprintable result
m.digest()
# atau hexdigest untuk printable result
m.hexdigest()
```

Gambar 2.11 Penggunaan object hash

Algoritma enkripsi mentransformasi data masukan yang bergantung kepada kunci yang menghasilkan *ciphertext*. Transformasi ini dapat dilakukan dengan arah sebaliknya jika mengetahui kunci yang tepat (yang digunakan untuk mentransformasi data sebelumnya).

Block cipher mengambil multiblock sebagai masukan dengan panjang tertentu (umumnya kelipatan delapan atau enambelas) dan melakukan enkripsi terhadap masukan tersebut. Dengan demikian diharapkan pola-pola yang muncul pada enkripsi tanpa menggunakan blok dapat diminimalisasi. Sehingga keamanan data lebih terjamin.

Block cipher yang telah diimplementasikan pada pycrypto adalah ARC2, Blowfish, CAST, DES, DES3 (triple DES), IDEA, RC5. Semua block cipher tersebut menggunakan masukan dengan panjang kelipatan delapan. Kunci enkripsi ARC2, Blowfish, CAST dan RC5 menggunakan kunci yang panjangnya bervariasi. DES menggunakan kunci sepanjang 8 byte. Sedangkan DES, DES3 dan IDEA menggunakan kunci sepanjang 16 byte.

Tabel 2.6 Method pada object block cipher

Method	Penjelasan
`New()`	Membuat object cipher. Menerima argumen yang digunakan sebagai kunci
`decrypt()`	Dekripsi string yang diinputkan sebagai argumen. Menghasilkan plaintext.
`encrypt()`	Enkripsi string yang diinputkan sebagai argumen. Menghasilkan ciphertext.

Contoh penggunaan dapat dilihat pada gambar berikut.

```
from Crypto.Cipher import Blowfish
b = Blowfish.new('Hello Key')
string = 'Hello World'
# pemanggilan di bawah menghasilkan error
b.encrypt(string)
# pemanggilan di bawah benar. Len(string) dibulatkan
#menjadi kelipatan 8 (dalam hal ini 16)
b.encrypt(string + '      ')
```

Gambar 2.12 Penggunaan object block cipher

2.4 Python for delphi

Python for delphi merupakan satu set komponen yang mengkapsulasi dll dari python ke dalam delphi. Dengan komponen-komponen ini user delphi dapat mengeksekusi *script* python, membuat modul python, dan sebagainya. Python for

delphi ini dibuat oleh dua orang programmer yang bernama Dr. Dietmar Budelsky dan Morgan Martinet.

Umumnya, dalam menggunakan *library* ini, *class-class* akan diimplementasikan dalam python, dan menggunakan delphi untuk memanfaatkan *class-class* tersebut seperti class delphi pada umumnya. Hal ini dikarenakan pengembangan dalam python cukup cepat, dan tersedia banyak library yang umumnya bersifat *open source*.

Dengan demikian, python digunakan untuk mendesain proses bisnis , semetara delphi digunakan untuk mendesain GUI maupun *framework* baru.

2.4.1 Object Umum Dalam Python for Delphi

a. TPythonEngine

Merupakan *engine* python dalam delphi. Dasar dari penggunaan python dengan delphi. Object inilah yang melakukan eksekusi script-script python. Untuk dapat melakukan eksekusi script pyhton, object ini menentukan akan menggunakan file library eksternal dari python sendiri. Sebagai contoh adalah python24.dll pada direktori windows\system32. Object ini juga menentukan apakah menggunakan TPythonGUIInputOutput sebagai output atau tidak.

b. TPythonGUIInputOutput

Class yang digunakan untuk menghubungkan python engine dengan object tampilan pada delphi seperti TMemo sebagai input output. Penggunaan class ini bersifat optional.

c. TPyhtonDelphiVar

Class yang digunakan untuk konversi object dari python ke delphi. Selanjutnya dapat digunakan sebagai string, integer, ataupun object dengan tipe *OLEVariant*.

d. PythonAtom

Unit yang menyediakan fitur-fitur untuk menangkap object dalam python dengan delphi sehingga delphi dapat menggunakan object tersebut selayaknya object milik delphi sendiri.

2.4.2 Penggunaan Python for Delphi

Untuk dapat menggunakan python dalam delphi, buat sebuah object dari class TPythonEngine. Lalu set properti dari object tersebut. Properti utama yang perlu diset adalah file dll dari python yang digunakan, versi yang teregister, dan flag yang menandakan apakah menggunakan versi terakhir yang diketahui package Python for Delphi atau tidak.

Untuk GUI input output, sifatnya optional. Jika ingin menggunakan TMemo sebagai output secara otomatis, maka sebaiknya gunakan GUI input output. Untuk dapat menggunakan TMemo sebagai output, setelah membuat object dari class TPythonGUIInputOutput, set properti output ke object dari TMemo. Lalu dari object python engine, set properti IO ke object dari GUI input output.

Untuk menangkap object python, digunakan object dari class TPyhtonDelphiVar. Set properti engine ke object dari python engine, dan properti

varname ke nama variabel yang akan digunakan dalam script python. Lalu pada script python, gunakan nama pada varname sebagai nama object yang akan dipakai dalam delphi dengan tambahan .value sebagai properti pada object tersebut. Isi dari properti value inilah yang tangkap oleh delphi melalui TPyhtonDelphiVar.

```
# test adalah nilai yang diset pada properti
# varname milik TPyhtonDelphiVar
db = MySQLdb.Connection(db='test')
cursor = db.cursor()
cursor.execute('select * from person;')
test.value = cursor.fetchall()
```

Gambar 2.13 Contoh pada script python

```
var
  pObject: PPyObject;
  myAtom: OLEVariant;
begin
  pObject := PythonDelphiVar1.ValueObject;
  try
    myAtom := getAtom(pObject);
  finally
    GetPythonEngine.Py_XDecRef(pObject);
  end;
end;
// dengan demikian, myAtom berisi sequence hasil
// dari query
```

Gambar 2.14 Contoh pada aplikasi delphi

Pada contoh di atas, ketika python mengeksekusi perintah `test.value = cursor.fetchall()`, isi dari `PythonDelphiVar1.ValueObject` adalah

tuple hasil dari query pada script python dan dapat diakses sebagai array pada delphi. Hal ini berlaku pula untuk object dari class. Sehingga delphi dapat memanggil method-method dari object yang mengisi `test.value`.

2.5 Data Aware

Aplikasi database pada umumnya menggunakan akses langsung ke database. Dalam aplikasi web seperti asp dan php, untuk mengambil data, akan langsung melakukan query dan mendapatkan data. Begitu pula sebaliknya, untuk menyimpan data juga mengakses langsung ke database.

Dalam data aware, ketika user memanipulasi data dan melakukan *posting*, data tidak langsung disimpan ke database. Hal ini disebabkan karena manipulasi data dan proses posting tersebut sebenarnya hanya terjadi pada memori. Dalam data aware, umumnya setelah user melakukan *posting*, maka komponen data aware memanggil *class* atau fungsi khusus yang bertugas menlakukan *update* database dengan data baru tersebut. Contoh komponen data aware dalam delphi adalah TClientDataset.

2.5.1 Komponen TClientDataset

TClientDataset merepresentasikan dataset dalam memori. Client dataset dapat digunakan sebagai:

a. Dataset yang berbasis file yang berdiri sendiri dan berfungsi secara penuh. Ketika digunakan seperti demikian, client dataset

dinonaktifkan dan *FieldDefs* dibersihkan dari field sebelumnya. Setelah menambah beberapa field sesuai kebutuhan, diakhiri dengan memanggil *CreateDataset* dan mengaktifkan kembali client dataset.

Selain mendefinisikan field, client dataset juga dapat mendefinisikan index. Sama dengan field, index harus didefinisikan sebelum fungsi *CreateDataset* dipanggil.

```
ClientDataset1.IndexDefs.Clear;
index := ClientDataset1.IndexDefs.AddIndexDef;
index.Name := 'idIndex';
index.Fields := 'id';
index.Options := [ixPrimary, ixUnique];
...
ClientDataset1.IndexName := 'indexId';
```

Gambar 2.17 Menambahkan index pada TClientDataset

Untuk menggunakan index yang telah dibuat, set properti *IndexName* dengan nama index yang digunakan sebagai *primary key*. Dapat dilakukan sebelum maupun sesudah *CreateDataset* dipanggil.

```
ClientDataset1.Append;
ClientDataset1.FieldValues['id'] := 5;
ClientDataset1.FieldValues['name'] := 'Ryusei';
ClientDataset1.FieldValues['age'] := 21;
ClientDataset1.FieldValues['gender'] := True;
ClientDataset1.Post;
```

Gambar 2.18 Menambahkan record pada *TClientDataset*

Untuk menambahkan record pada client dataset, client dataset harus terlebih dahulu memiliki field yang akan digunakan. Setelah penataan field dari tabel lengkap, maka record dapat mulai ditambahkan dengan menggunakan fungsi *Append* seperti pada gambar 2.18.

Setelah memanggil fungsi Append, data dapat diisikan ke masing-masing field. Setelah menambahkan data pada field masing-masing, panggil fungsi Post agar tabel pada client dataset diupdate. Atau memanggil fungsi Cancel untuk membatalkan update.

```
ClientDataset1.IndexFieldNames := 'name';
ClientDataset1.FindKey(['Raidith']);
```

Gambar 2.19 Mencari data pada *TClientDataset*

Setelah terisi beberapa data, untuk memudahkan pencarian, client dataset menyediakan fungsi *FindKey*. Secara default fungsi ini menggunakan index yang didefinisikan pada *IndexName*. Jika ingin menggunakan field lain sebagai *key*, set terlebih dahulu *IndexFieldNames* ke nama field yang akan digunakan sebagai kunci pencarian.

```
// set field id menjadi read only
ClientDataset1.FindField('id').ReadOnly := True;

// set field id menjadi writeable
ClientDataset1.FindField('id').ReadOnly := False;
```

Gambar 2.20 Merubah sifat data ke mode *read only*

Untuk mencegah user memanipulasi data yang sifatnya otomatis yang tidak boleh dimanipulasi, client dataset menyediakan sebuah property yang menandakan apakah sebuah field bersifat baca saja atau tidak.

Jika menggunakan baca saja pada salah satu field, hal yang tidak boleh dilupakan adalah sebelum sistem atau aplikasi memanipulasi data, field yang bersifat baca saja tersebut harus dirubah terlebih dahulu ke mode baca tulis.

BAB III

PERANCANGAN SISTEM

Bab ini menjelaskan tentang tahapan proses perancangan perangkat lunak mulai dari deskripsi secara umum sampai perancangan data, perancangan aplikasi dan perancangan antar muka.

3.1 Deskripsi Umum

Dalam tugas akhir ini akan dibangun suatu sistem yang terdiri dari database, server, client dan aplikasi antarmuka untuk menunjukan pemisahan antara perancangan proses bisnis sebagai dasar dari aplikasi dan perancangan antarmuka sebagai jembatan antara user dengan aplikasi.

Pada sisi perancangan proses bisnis yang akan dibangun, developer merancang bagaimana cara koneksi ke database berikut empat fungsi dasar yaitu select, insert, update dan delete. Selain merancang koneksi database, sisi ini juga merancang bagaimana proses bisnis berlangsung.

Pada sisi perancangan antarmuka, developer merancang tampilan yang akan digunakan. Selain itu juga mengimplementasikan sebuah class yang merupakan kumpulan-kumpulan fungsi yang menjadi perantara antara client dan aplikasi antarmuka.

Pada awal perencanaan, antara kedua sisi diharuskan menyepakati apa saja yang diperlukan sebagai antarmuka client dengan aplikasi desktop. Setelah desain

antarmuka yang menjembatani antara client dengan aplikasi desktop ini selesai, baru kemudian kedua sisi melanjutkan pekerjaan masing-masing.

Studi kasus yang diangkat dalam tugas akhir ini adalah Sistem Informasi Ruang Baca Teknik Computer (RBTC).

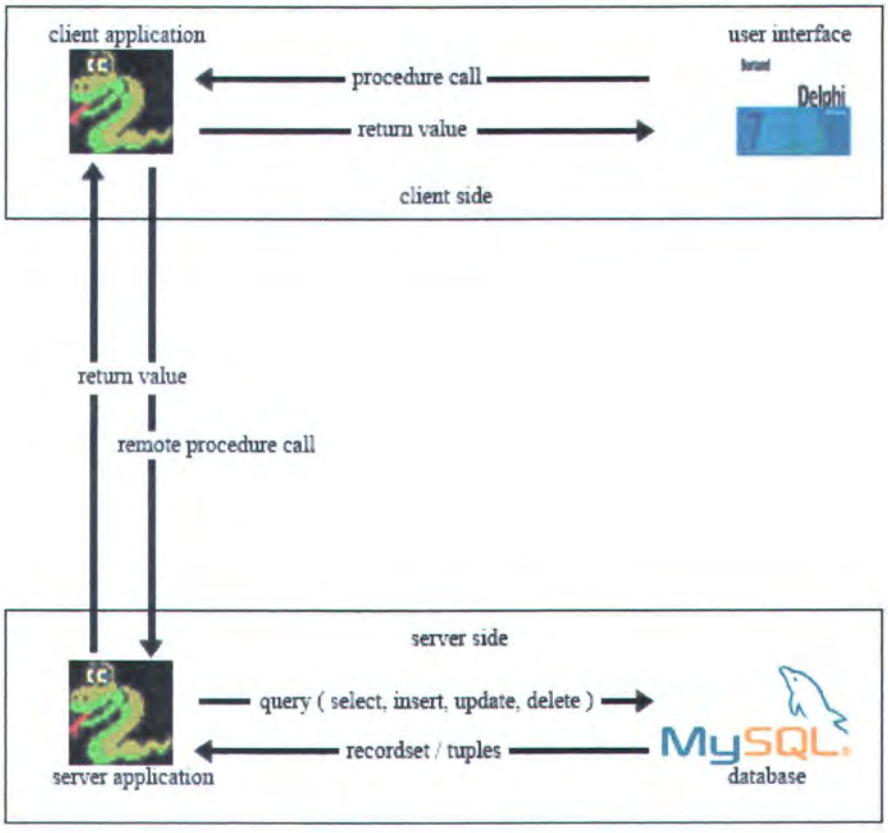
3.1.1 Arsitektur Sistem

Perangkat lunak yang akan dibangun dalam tugas akhir ini merupakan aplikasi berbasis desktop pada sisi client secara keseluruhan (client dan antarmuka), dan konsol pada sisi server. Kebutuhan minimum pada aplikasi ini adalah satu komputer yang sekaligus berfungsi sebagai database, server dan client. Namun pada tugas akhir ini digunakan dua komputer sebagai contoh jaringan.

Aplikasi yang dibangun terdiri dari class yang akan disediakan untuk diakses secara remote, sebuah server yang meregistrasikan object-object dari class tersebut, sebuah client yang membuat sebuah object proxy untuk mengakses object yang telah diregistrasikan. Ketiga bagian ini seluruhnya ditulis dengan bahasa python (Python2.4) dengan menggunakan pyro sebagai ekstensi dari python dalam urusan remote object.

Selain itu juga dibuat sebuah aplikasi desktop sebagai antarmuka antara user dengan aplikasi client yang merupakan aplikasi konsol. Aplikasi desktop ini menangkap salah satu object pada client untuk dianggap sebagai objectnya sendiri yang kemudian digunakan untuk memberikan data pada dataset yang akan ditampilkan kepada user. Aplikasi desktop ini ditulis dengan bahasa delphi (Delphi7).

Prinsip kerja aplikasi desktop ini adalah menangkap salah satu object python yang akan digunakan dalam merepresentasikan data pada user dengan menggunakan *python for delphi*. Setelah object python diterima oleh delphi sebagai *OLEVariant*, maka delphi akan memanipulasi client dataset dengan mengimplementasikan data yang diterima dari object tersebut. Selanjutnya, setiap user melakukan manipulasi data, client dataset memanggil fungsi-fungsi yang sudah disediakan pada class yang merupakan kumpulan fungsi yang mendukung pertukaran data dari python ke delphi. Sedangkan untuk sisi database, pada tugas akhir ini menggunakan mysql yang berjalan pada platform windows. Desain arsitektur secara dasar adalah seperti pada gambar 3.1.



Gambar 3.1 Arsitektur Sistem

3.2 Perancangan Data

Implementasi dari sistem menggunakan Ruang Baca Teknik Computer sebagai studi kasus. Storage (database) yang digunakan merujuk pada database yang sudah ada yang dimiliki oleh RBTC. Namun begitu, terjadi perpindahan database. Bila pada RBTC digunakan SQL Server 2000, sistem menggunakan MySQL.

Seperti yang sudah dijelaskan dalam arsitektur sistem, client tidak langsung melakukan akses ke database. Yang melakukan akses ke database adalah object yang diakses secara remote oleh client. Bagaimana mekanisme pemanggilan method secara remote, semua diatur oleh pyro sebagai modul penyedia fitur RPC. Client cukup memanggil method dari object proxy.

3.2.1 Format Data Argumen

Dalam melakukan panggilan method, client antarmuka menggunakan beberapa data bertipe string sebagai argumen. Data-data tersebut selanjutnya dikonversi sebagai tuple, yang kemudian digunakan sebagai argumen ketika pyro client memanggil method remote object.

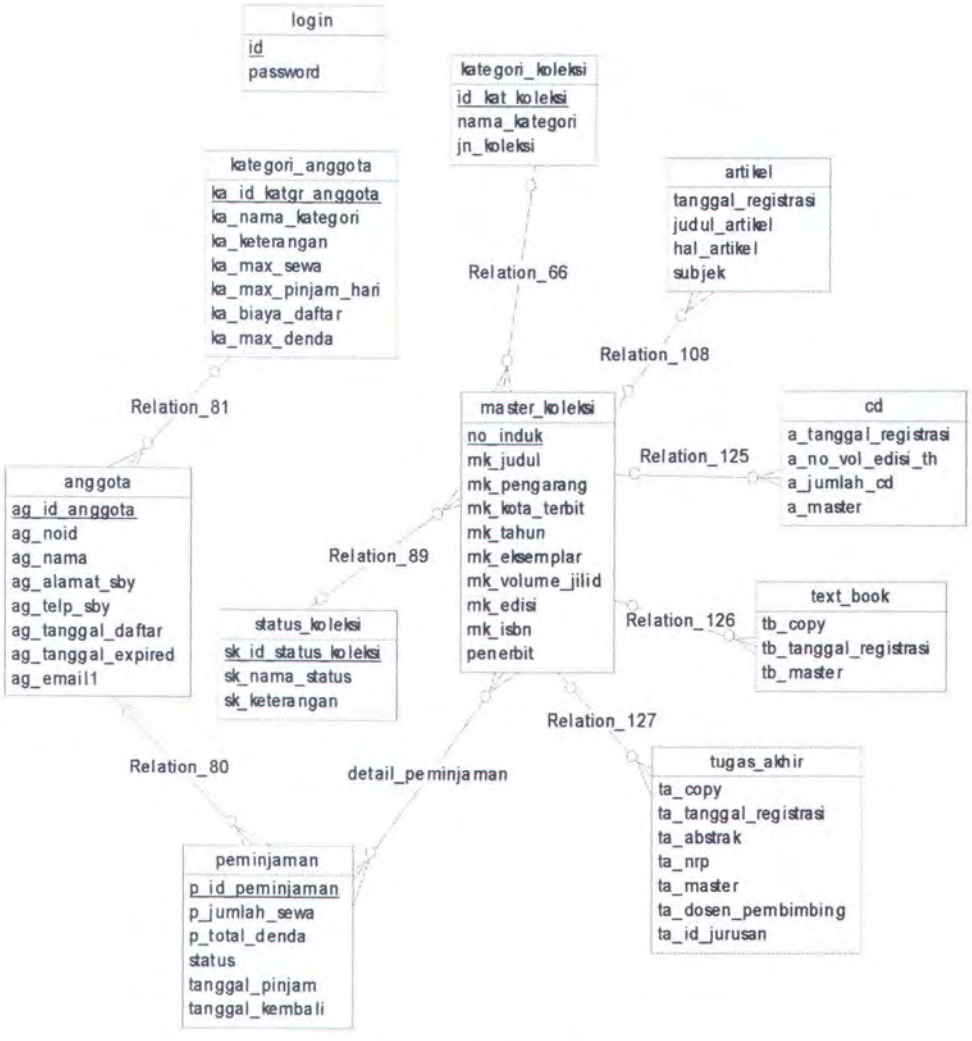
3.2.2 Format Data Kembalian

Data hasil proses yang merupakan hasil kembalian dari method yang dipanggil, oleh pyro akan dikirim kepada pyro client sebagai hasil kembalian object proxy. Adapun format data yang digunakan dalam aplikasi merupakan tuple. Tuple itu sendiri merupakan object yang menyimpan array dari bermacam-

macam object. Selanjutnya, client antarmuka menerjemahkan tuple tersebut sebagai array yang selanjutnya digunakan sesuai keperluan.

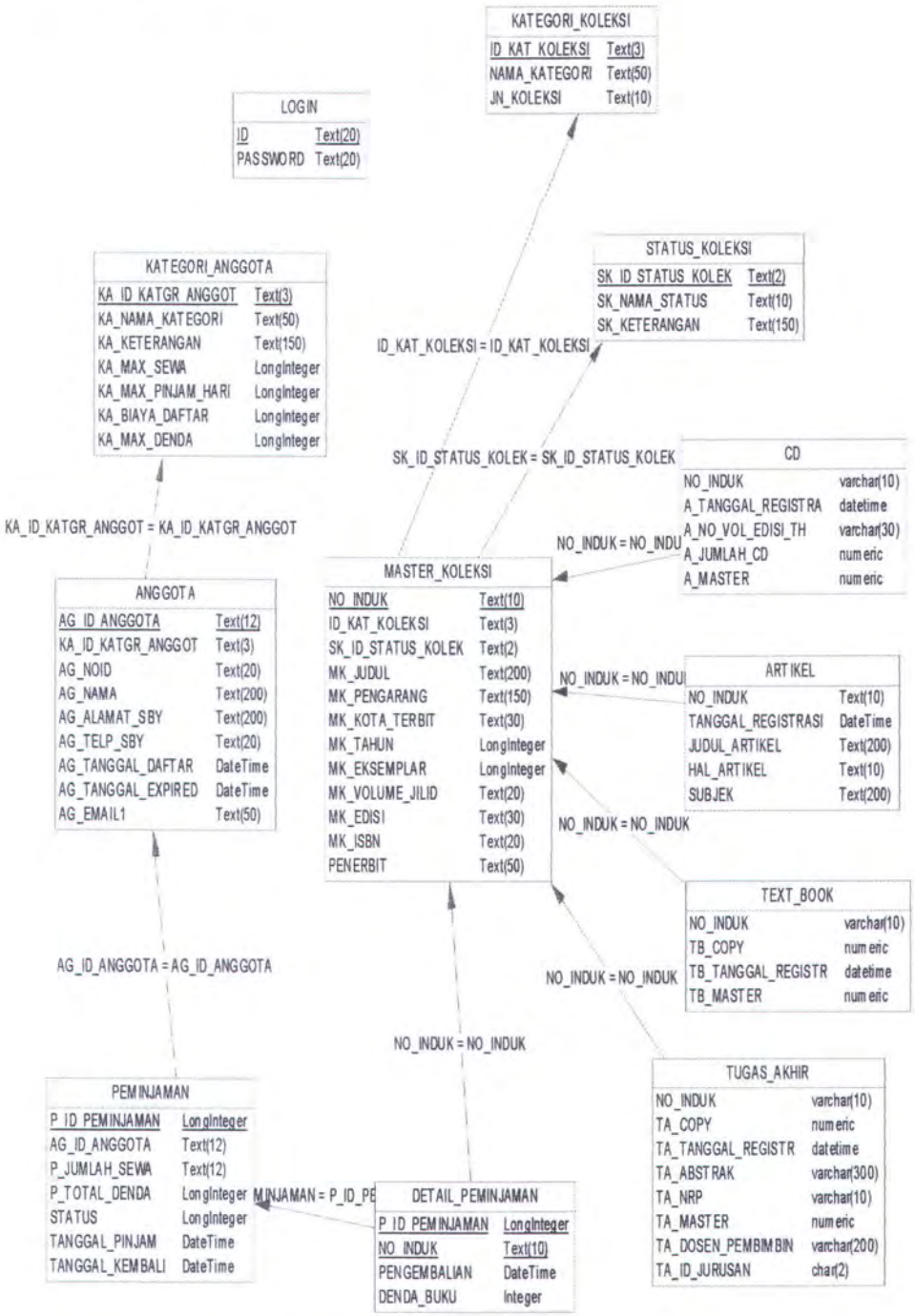
3.2.3 Struktur Database

Diagram ER (CDM) yang digunakan dalam RBTC adalah seperti pada gambar 3.2.



Gambar 3.2 Desain CDM

Dan berikut adalah PDM Database hasil *generate* dari Power Designer 9 sesuai pada gambar 3.3.



Gambar 3.3 Desain PDM

Berikut ini adalah struktur database yang dibuat di dalam MySQL:

a. Tabel login

Menyimpan data username dan password dari user yang dapat menggunakan client pada aplikasi yang akan dibangun.

Tabel 3.1 login

Method	Penjelasan
Id	Id dari user. Digunakan untuk login. Bertipe karakter variabel / string
Password	Password dari user yang bersangkutan. Digunakan untuk login. Bertipe karakter variabel / string

b. Tabel kategori_ anggota

Berisi pembedaan tipe anggota dalam ruang baca. Tiap kategori mempunyai data meliputi jumlah maksimal peminjaman, biaya pendaftaran dan maksimal denda.

Tabel 3.2 kategori_anggota

Method	Penjelasan
ka_id_katgr_anggota	Id dari kategori anggota. Bertipe karakter variabel / string
ka_nama_kategori	Nama dari kategori anggota. Bertipe karakter variabel / string
ka_keterangan	Keterangan dari kategori anggota.

	Bertipe karakter variabel / string
ka_max_sewa	Maksimal jumlah sewa kategori anggota. Bertipe numerik
ka_max_pinjam_hari	Maksimal lama meminjam kategori anggota. Bertipe numerik
ka_biaya_daftar	Biaya pendaftaran kategori anggota. Bertipe numerik
ka_max_denda	Maksimal denda kategori anggota. Bertipe numerik

c. Tabel anggota

Berisi daftar anggota ruang baca. Data yang ada meliputi nama, NRP/NIM, kategori, alamat, nomor telepon, tanggal pendaftaran, tanggal expired, email. Tabel inilah yang dipakai untuk menunjukkan data anggota.

Tabel 3.3 anggota

Method	Penjelasan
ag_id_anggota	Id dari anggota. Bertipe karakter variabel / string
ka_id_katgr_anggota	id kategori dari anggota. Bertipe karakter variabel / string
ag_noid	Nomor identitas anggota(NRP / NIP).

	Bertipe karakter variabel / string
ag_nama	Nama anggota. Bertipe karakter variabel / string
ag_alamat_sby	Alamat anggota di surabaya. Bertipe karakter variabel / string
ag_telp_sby	Telepon anggota di surabaya. Bertipe karakter variabel / string
ag_tanggal_daftar	Tanggal daftar anggota. Bertipe date
ag_tanggal_expired	Tanggal expired anggota. Bertipe date
ag_email1	Email anggota. Bertipe karakter variabel / string

d. Tabel kategori_koleksi

Berisi kategori-kategori dari koleksi yang dimiliki ruang baca. Berisi id, nama dan jenis koleksi. Jenis koleksi yang dimaksud di sini adalah apakah koleksi tersebut berseri atau tidak dan sebagainya.

Tabel 3.4 kategori_koleksi

Method	Penjelasan
id_kat_koleksi	Id dari kategori koleksi. Bertipe karakter variabel / string
nama_kategori	Nama dari kategori koleksi. Bertipe karakter variabel / string

jn_koleksi	Jenis dari kategori koleksi. Bertipe karakter variabel / string

e. Tabel status_koleksi

Tabel ini merupakan tabel yang menyimpan status dari sebuah koleksi. Tabel ini untuk menyatakan apakah kondisi dari koleksi tersebut baik, rusak, atau bahkan hilang.

Tabel 3.5 status_koleksi

Method	Penjelasan
sk_id_status_koleksi	Id dari status koleksi. Bertipe karakter variabel / string
sk_nama_status	Nama dari status koleksi. Bertipe karakter variabel / string
sk_keterangan	Keterangan dari status koleksi. Bertipe karakter variabel / string

f. Tabel master_koleksi

Berisi daftar koleksi yang dimiliki ruang baca. Menyimpan nomor induk koleksi, status koleksi, kategori koleksi, pengarang, penerbit dan sebagainya. Tabel inilah yang dipakai untuk mencari data tentang koleksi yang ada pada ruang baca fakultas teknologi informasi. Status koleksi dan kategori koleksi merupakan id yang merujuk pada tabel-tabel yang bersangkutan.

Tabel 3.6 master_koleksi

Method	Penjelasan
no_induk	Nomor induk dari master koleksi. Berupa tahun dan nomor urut. Bertipe karakter variabel / string
sk_id_status_koleksi	id dari status master koleksi. Bertipe karakter variabel / string
id_kat_koleksi	Kategori dari master koleksi Bertipe karakter variabel / string
mk_judul	Judul master koleksi. Bertipe karakter variabel / string
mk_pengarang	Nama pengarang master koleksi. Bertipe karakter variabel / string
mk_kota_terbit	Kota tempat master koleksi terbit. Bertipe karakter variabel / string
mk_tahun	Tahun master koleksi terbit. Bertipe numeric
mk_eksemplar	Jumlah eksemplar master koleksi. Bertipe numeric
mk_volume_jilid	Jenis jilid dari master koleksi. Bertipe karakter variabel / string
mk_edisi	Edisi dari master koleksi. Bertipe karakter variabel / string

mk_indeks	Halaman indeks master koleksi. karakter variabel / string
mk_isbn	Nomor isbn master koleksi. Bertipe karakter variabel / string
penerbit	Nama penerbit dari master koleksi. Bertipe karakter variabel / string
status	status dari master koleksi. Bertipe boolean

3.3 Perancangan Proses

Seperti yang telah disebut sebelumnya. Model aplikasi yang akan dibangun memisahkan dua bagian besar. Sisi manajemen object remote dan sisi penangkapan object python untuk digunakan dalam delphi yang selanjutnya memanipulasi client dataset untuk menampung data dari operasi pada server.

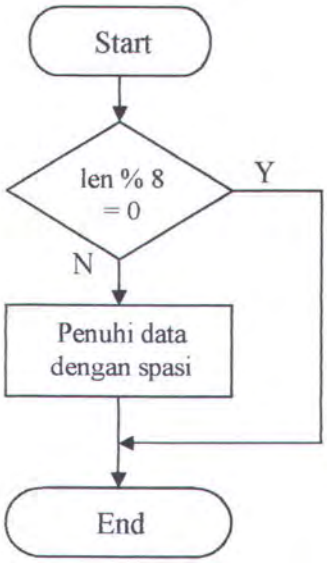
3.3.1 Sisi Manajemen Object Remote

Pada sisi ini, aplikasi memiliki tiga fungsi global dan lima class yang memegang peranan penting. Tiga fungsi tersebut adalah fill, encrypt dan decrypt.

a. fill

memeriksa apakah data yang diberikan memiliki panjang kelipatan nilai yang diinputkan. Jika tidak, maka untuk memenuhi kriteria tersebut,

ditambahkan spasi pada akhir data sebanyak sisanya hingga kelipatan yang diinginkan terpenuhi.



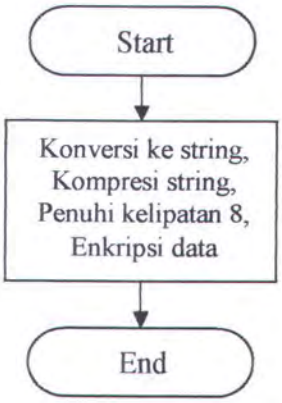
Gambar 3.4 Memenuhi string hingga kelipatan delapan

b. encrypt

Proses penting dalam enkripsi adalah kompresi dan enkripsi. Modul zlib pada python dapat memenuhi kebutuhan kompresi. Namun argumen yang diterima hanya berupa string.

Untuk memenuhi persyaratan tersebut diperlukan konversi dari data yang masuk ke string dimana data tersebut harus dapat dikembalikan lagi seperti aslinya. Python memiliki modul bernama pickle yang dapat digunakan untuk menkonversi dari object ke string dan dari string ke object. Untuk menkonversi dari object ke string, digunakan *pickle.dumps*. Baru setelah itu dilakukan kompresi string dengan zlib.

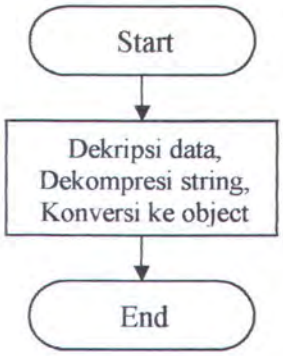
Langkah selanjutnya adalah menggunakan hasil kompresi string dari zlib tersebut untuk dienkripsi. Dalam proses ini, aplikasi menggunakan ekstensi dari python yang bernama Crypto. Enkripsi yang digunakan adalah blowfish. Karena menggunakan enkripsi model blok, maka diperlukan data yang panjangnya merupakan kelipatan delapan. Karena itu sebelum melakukan kompresi, fungsi ini terlebih dahulu menggunakan fungsi fill untuk mendapatkan string yang panjangnya tepat kelipatan delapan. Baru setelah itu dilakukan enkripsi.



Gambar 3.5 Enkripsi data

c. decrypt

Kebalikan dari enkripsi, fungsi decrypt melakukan dekripsi terlebih dahulu. Setelah data terdekripsi, maka selanjutnya adalah melakukan dekompresi dengan zlib. Hasil dekompresi tersebut diteruskan ke *pickle.loads* untuk mendapatkan object yang akan diolah. Object tersebut selanjutnya digunakan sebagaimana data tanpa enkripsi.



Gambar 3.6 Dekripsi data

Pada proses dekripsi ini, ada satu langkah yang tidak dilakukan. Yaitu mengembalikan panjang string ke nilai sebelum dilakukan proses fill. Hal ini dikarenakan, fungsi decompress pada zlib membaca header dari data terkompresi untuk menentukan pembacaan sampai batas tertentu walaupun panjangnya melebihi yang ditentukan header.

Class-class pada sisi manajemen object remote adalah clientHandler, clientIdentifier, timeoutThread, timerThread, clientManagement dan remoteClient. Empat class pertama adalah class yang akan dijalankan secara lokal dan dua class terakhir merupakan class yang akan didaftarkan untuk diakses secara remote.

a. clientHandler

Class ini bertugas untuk menangani segala permintaan client. Mulai dari query data, edit data dan sebagainya bergantung pada jenis server yang diinginkan (tidak harus databaase). Memiliki satu atribut yang bernama timeout. Atribut ini digunakan untuk mengukur seberapa

lama sebuah client boleh idle sehingga dapat mengidentifikasi session timeout untuk menghapus id client dari list.

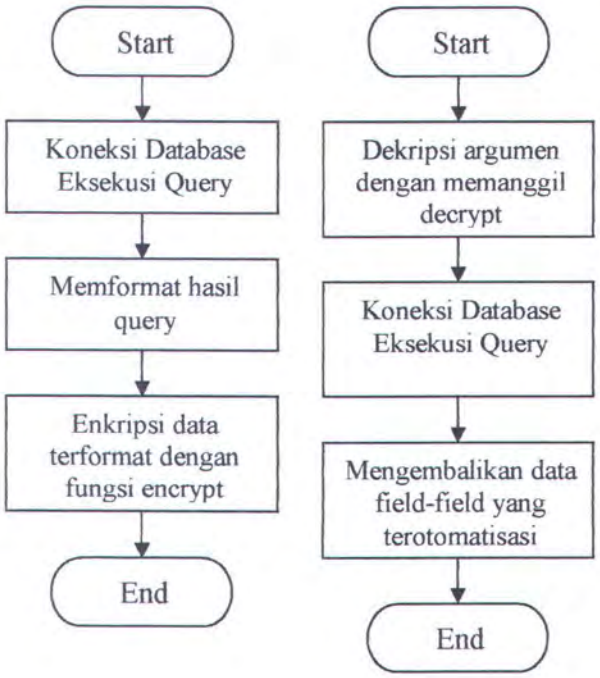
Dalam contoh kasus aplikasi ini, clientHandler menangani query data yang diminta oleh client. Yang ditangani adalah select, insert, update, delete. Namun secara garis besar, ada dua jenis proses. Yaitu select dan manipulasi tabel (insert, update, delete).

Ketika method select dipanggil, yang dilakukan adalah melakukan koneksi ke database, lalu mengeksekusi query. Jika hasil eksekusi mengembalikan nilai lebih dari nol, maka selanjutnya adalah memformat data hasil query dengan format berupa tuple yang berisi empat elemen. Yaitu jumlah field, jumlah record hasil query, metadata dari masing-masing field dan terakhir adalah recordset hasil query.

Setelah hasil format didapat, maka selanjutnya adalah melakukan enkripsi untuk mengacak data yang akan dikirim kembali ke client. Enkripsi ini dilakukan oleh fungsi encrypt yang dibuat terlebih dahulu.

Ketika melakukan manipulasi tabel, yang pertama dilakukan adalah melakukan dekripsi argumen dengan memanggil fungsi decrypt. Setelah data sesuai dengan object saat data dibuat, baru kemudian melakukan koneksi database dan mengeksekusi query.

Setelah melakukan eksekusi query, method akan mengembalikan nilai dari field-field yang terisi atau terupdate secara otomatis baik dari sisi database maupun dari sisi server.



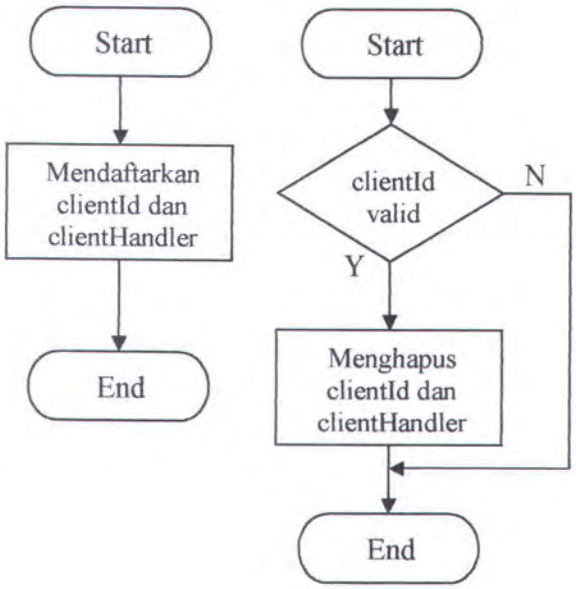
Gambar 3.7 Select dan manipulasi tabel (insert, update, delete)

b. clientIdentifier

Class ini bertugas menyimpan daftar clientId dan clientHandler dari session yang bersangkutan. Memiliki sebuah object dictionary yang menyimpan clientHandler dan clientId sebagai kunci pencarian. Memiliki dua buah method yang bertugas mendaftarkan dan menghapus pasangan clientId dan clientHandler dari dictionary sebagai session.

Proses registrasi cukup sederhana. Hanya melakukan penambahan pasangan clientId dan clientHandler pada dictionary yang ada. Sedangkan unregistrasi lebih panjang. Karena ketika akan menghapus sebuah session, harus memeriksa apakah clientId masih ada pada dictionary atau tidak. Ini diperlukan jika ada client yang logout hampir bersamaan dengan session

timeout. Sehingga hanya salah satunya saja yang melakukan penghapusan pasangan. Yang lainnya tidak perlu lagi melakukan penghapusan.



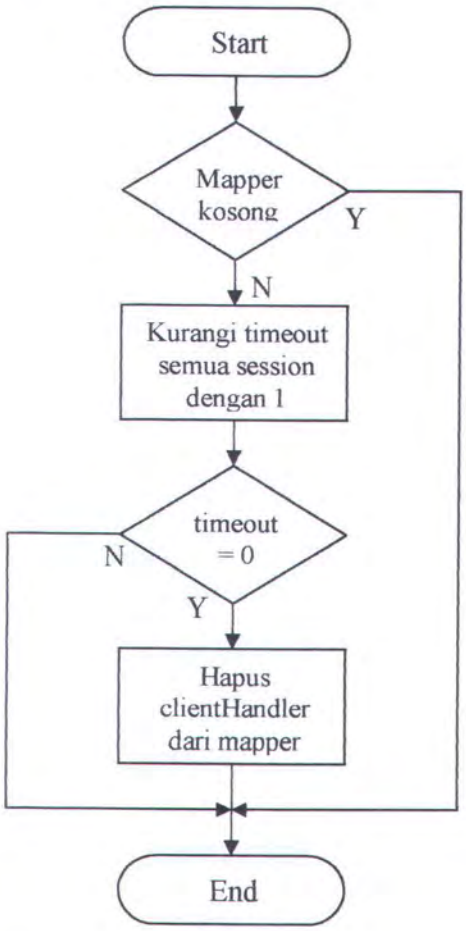
Gambar 3.8 Register dan unregister clientHandler

c. timeoutThread dan timerThread

Class ini bertugas menghitung timeout dari tiap session yang ada. Tiap detik, melakukan pengurangan terhadap timeout dari masing-masing session. Dan jika ada yang mencapai nol, maka session tersebut dianggap berakhir. Selanjutnya adalah melakukan unregistrasi session yang dianggap berakhir tersebut.

Object timerThread berjalan pada thread sendiri sehingga perhitungan waktu tiap detik tidak terganggu oleh pengurangan data timeout dan penghapusan session. Proses yang dilakukan timerThread

hanya menunggu satu detik kemudian beritahukan ke timeoutThread. Sedangkan timeoutThread memiliki proses sebagai berikut.



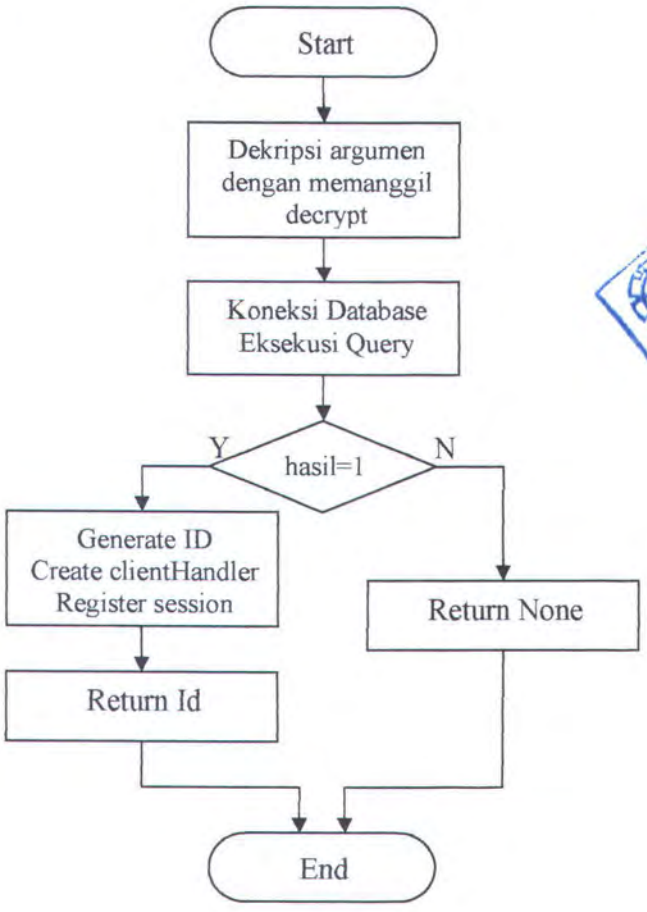
Gambar 3.9 Proses pada timeoutThread

d. clientManagement

Class ini menangani login dan logout dari client. Class ini didaftarkan ke nameserver agar dapat diakses secara remote oleh client. Memiliki dua method yaitu login dan logout.

Ketika client memanggil method logout, object ini menghapus session dari clientIdentifier yang memiliki clientId sesuai dengan argumen yang diberikan oleh client.

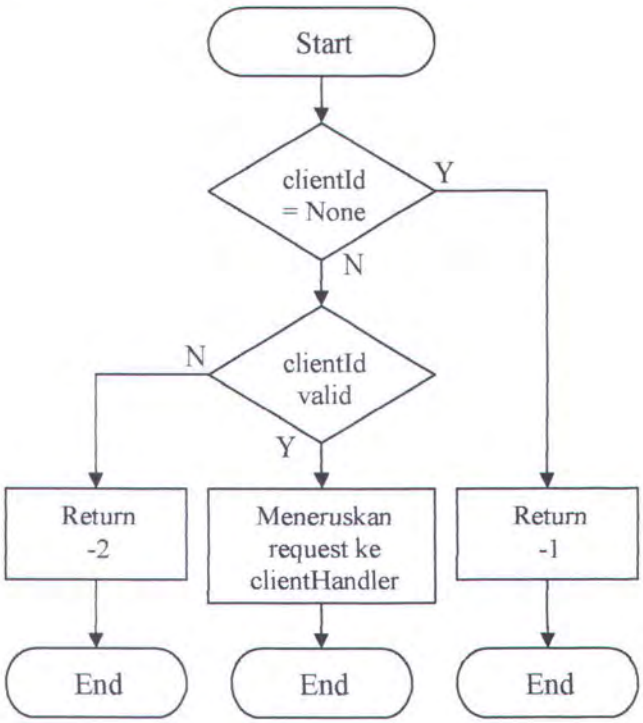
Ketika client memanggil method login, yang dilakukan object adalah memeriksa apakah user dan password valid atau tidak. Jika tidak, maka method ini mengembalikan nilai None. Jika valid, maka method ini akan membuat id baru dan object dari class clientHandler. Id dan object tersebut didaftarkan ke clientIdentifier dengan memanggil method register.



Gambar 3.10 Proses login

e. remoteClient

Class ini menangani request dari client. Ketika ada request masuk, yang dilakukan pertama kali oleh method yang memproses request tersebut adalah memeriksa apakah request tersebut menyertakan clientId atau tidak. Jika tidak, maka method yang bersangkutan akan mengembalikan nilai -1 sebagai elemen pertama tuple.



Gambar 3.11 Proses method pada remoteClient

Jika clientId disertakan oleh client, maka tugas method selanjutnya adalah memeriksa apakah clientId tersebut valid atau tidak dengan memeriksa apakah dictionary pada clientIdentifier memiliki key bernilai clientId atau tidak. Jika tidak, maka method akan mengembalikan nilai -2 sebagai elemen pertama tuple.

Jika clientId valid, baru kemudian method meneruskan request kepada method clientHandler yang bersesuaian dari session yang memiliki clientId sebagai key. Dan mengembalikan apapun hasil kembalian dari clientHandler tersebut.

3.3.2 Sisi Konversi Python ke Delphi Untuk Antarmuka

Sisi ini terdiri dari dua bagian. Kontrol object remote dan manipulator dataset serta antarmuka database dengan pengguna. Kontol object berupa class clientCtl. Dan manipulator dataset dan antarmuka database berupa class TpythonDatasetClient.

a. clientCtl

Client memiliki class yang melakukan kontrol terhadap object remote. Class ini ditulis dengan bahasa python. Hal ini diperlukan karena delphi memerlukan data yang disimpan dimemori sampai data tersebut dihapus. Hal ini tidak berlaku pada hasil kembalian dari method milik remote object. Untuk menyiasati hal tersebut, maka setelah memanggil method dari suatu remote object, hasil kembalian dari pemanggilan tersebut disimpan di variabel lokal. Dalam hal ini properti dari class kontrol.

Selain menyimpan data kembalian yang akan dibaca oleh delphi, class control juga menyimpan data clientId yang digunakan untuk mengidentifikasi diri saat melakukan pemanggilan method remote object.

Selain melakukan login, pada saat memanggil remote object, class control mengelompokkan argumen-argumen yang digunakan dan melakukan enkripsi tuple argumen tersebut. Dan setelah menerima hasil dari remote object, class control melakukan dekripsi data dari hasil kembalian tersebut.

b. TPythonDatasetClient

Class ini ditulis dengan bahasa delphi. Bertugas menangani manipulasi dataset, sebagai jembatan antara dataset dengan class kontrol pada python dan menangani manipulasi antarmuka database dengan pengguna.

Tugas pertama dari class ini adalah memanipulasi dataset. Adapun manipulasi yang dilakukan adalah inisialisasi dataset berupa membersihkan field dan mengatur field baru, menginputkan data hasil query. Saat melakukan pergantian tabel, yaitu ketika pengguna memilih untuk menampilkan tabel lain, maka semua referensi dari semua komponen yang merujuk kepada field dari dataset harus dikosongkan. Jika tidak, hal ini menyebabkan error ketika kontrol field berusaha membaca field yang telah dihapus dari dataset.

Tugas kedua adalah menangkap object python dan meneruskan request ketika dataset melakukan post data setelah manipulasi baik insert, update, maupun delete. Ketika melakukan insert, class ini juga melakukan

update dari field yang diinisialisasi dengan nilai *reserved* karena field tersebut terisi secara otomatis.

Tugas ketiga adalah melakukan pengaturan terhadap komponen-komponen visual sebagai antarmuka antara data dari database dengan pengguna. Komponen-komponen tersebut adalah DBGrid dan DBEdit.

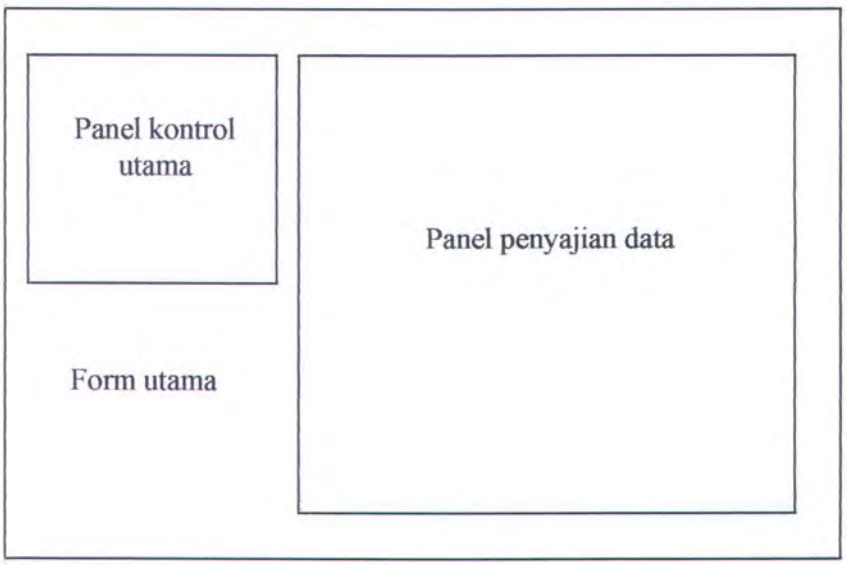
Ketika form utama diinisialisasi, komponen-komponen ini tidak ikut dibuat. Setelah melakukan login, dan memilih untuk menampilkan salah satu data, baru komponen-komponen tersebut dibuat, dan diatur posisinya.

Saat pengguna menampilkan tabel lain yang jumlah fieldnya lebih banyak dari tabel sebelumnya, maka array dari komponen yang menangani data masing-masing field harus diperpanjang. Dalam hal ini Label sebagai nama field dan DBEdit sebagai data.

3.4 Perancangan Antarmuka

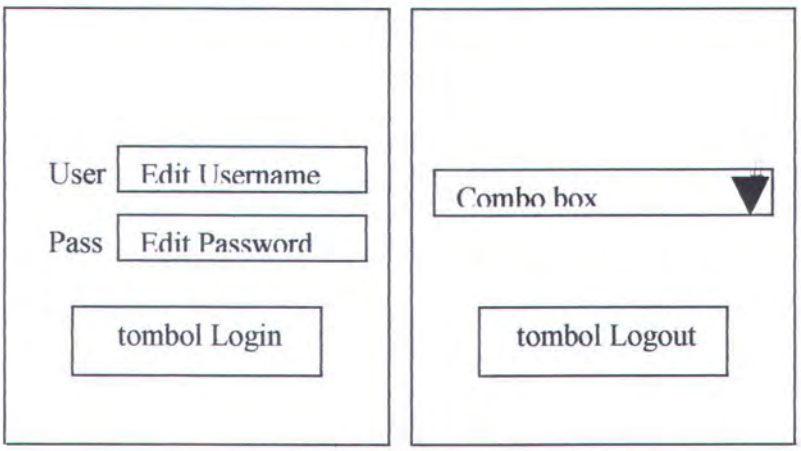
Antarmuka terbagi atas dua panel. Panel kontrol utama dan panel penyajian data. Panel kontrol utama adalah tempat dimana pengguna melakukan login dan memilih tabel atau operasi yang ingin dilakukan. Sedangkan panel penyajian data adalah tempat dimana kontrol-kontrol tampilan dan manipulasi data berada.

Ketika aplikasi dijalankan, pada panel kontrol utama harus menampilkan kontrol untuk login berupa editbox untuk username dan password serta tombol login. Sedangkan kontrol penyajian data belum dibuat.



Gambar 3.12 Rancangan desain form utama

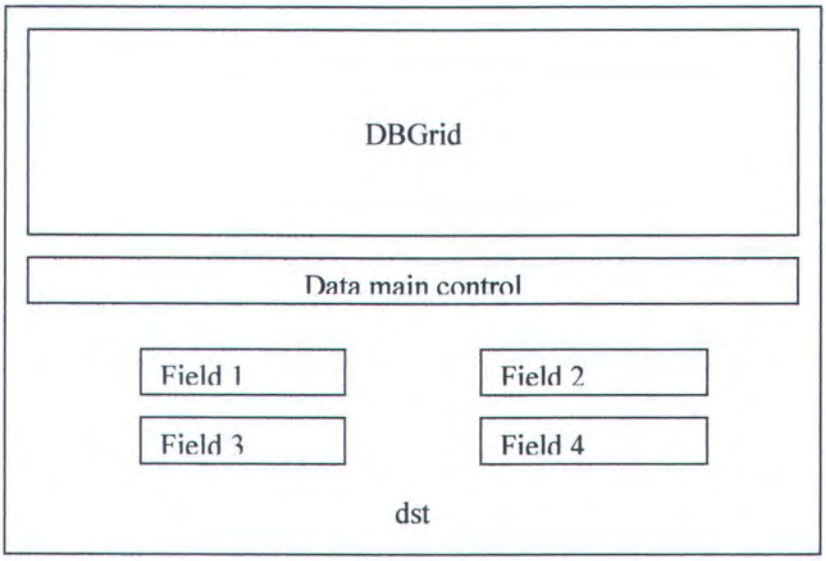
Setelah pengguna melakukan login, maka kontrol login tersebut disembunyikan kecuali tombol login yang berubah fungsi menjadi tombol logout. Selain itu, form harus menampilkan sebuah combobox yang memiliki daftar-daftar tabel dan operasi yang dapat dilakukan client.



Gambar 3.13 Rancangan desan panel kontrol

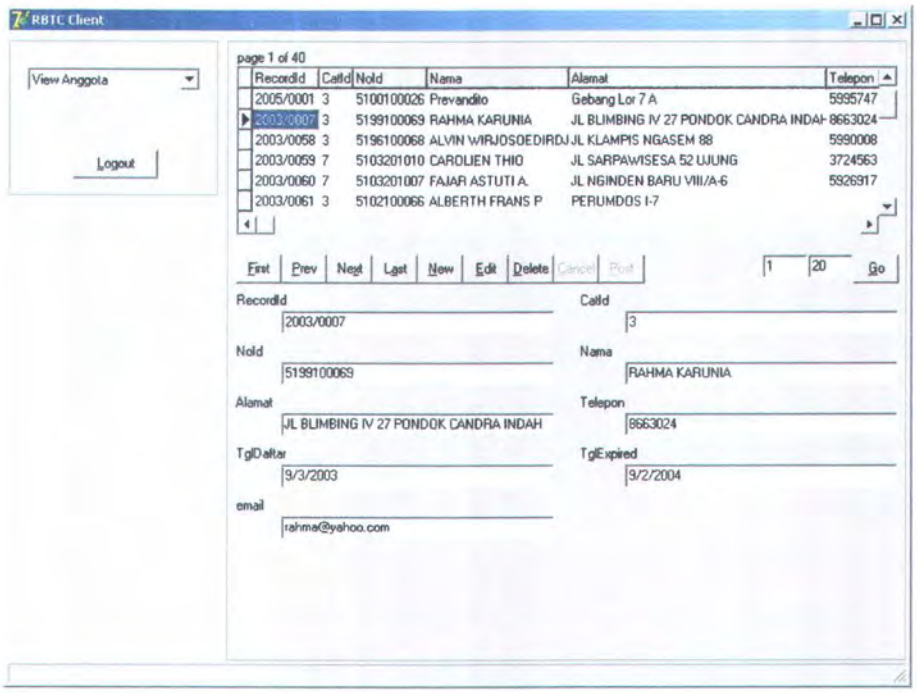
Pada panel penyajian data, ketika aplikasi dijalankan, panel ini masih kosong. Ketika pengguna memilih salah satu pilihan yang disediakan pada

combobox, maka pada panel penyajian data bermunculan berbagai macam kontrol yang berhubungan dengan operasi tabel.



Gambar 3.14 Rancangan desain panel penyajian data

Berikut contoh form dalam keadaan sedang menyajikan data anggota.



Gambar 3.15 Form aplikasi

BAB IV

IMPLEMENTASI PERANGKAT LUNAK

Berdasar pada perancangan aplikasi yang telah dijelaskan pada bab tiga, bab ini menjelaskan tentang implementasi dari rancangan tersebut. Implementasi ini dibagi menjadi dua bagian besar. Yaitu implementasi data dan implementasi sistem. Implementasi data mencakup bagaimana bentuk data pada aplikasi yang dibuat. Sedangkan implementasi sistem adalah bagaimana sistem dibangun menggunakan bahasa python maupun delphi.

4.1 Implementasi Data

Data pada aplikasi ini umumnya berupa tuple atau list. Pada saat melakukan pemanggilan method secara remote, data yang disertakan sebagai argumen hanya dua. Sebuah tuple yang berisi array argumen dan clientId sebagai identitas client yang melakukan pemanggilan.

Tuple adalah array dari kumpulan object yang bertipe apapun. Tuple tersebut kemudian diteruskan ke fungsi enkripsi yang prosesnya melalui beberapa langkah. Baru kemudian class kontrol remote object melakukan pemanggilan method dengan menggunakan tuple yang telah terenkripsi menjadi satu kesatuan data sebagai argumen. Berikut adalah contoh pengelompokan data sebagai tuple.

```
args = (page, recsPerPage)
# args merupakan tuple yang berisi page dan rpp
```

Gambar 4.1 Membuat tuple yang terdiri dari berbagai object

Pada sisi server, data yang akan dikembalikan juga melalui proses enkripsi. Bentuk data bergantung pada tipe data apa yang akan dikembalikan dari suatu method. Pada proses insert, data yang dikembalikan adalah nilai dari field id. Karena field tersebut terisi secara otomatis sehingga diperlukan untuk merubah data pada dataset. Sedangkan pada proses select, data merupakan list dari bermacam-macam jenis. Berikut contohnya.

```
result = []
result.append(fnum)
result.append(rnum)
result.append(pnum)
result.append(meta)
result.append(recordset)
```

Gambar 4.2 Nilai kembalian method select

Seperti pada contoh, *result* merupakan sebuah list yang anggotanya terdiri dari bermacam-macam jenis. Object fnum, rnum dan pnum bernilai integer dimana fnum adalah jumlah field, rnum adalah jumlah record, dan pnum adalah jumlah halaman maksimal dengan kalkulasi record per halaman yang diberikan. Sedangkan meta dan recordset bertipe list. object meta berisi data-data tentang metadata dari hasil query dan recordset merupakan list data hasil query.

4.2 Implementasi Fungsi Bantu

Seperti pada perancangan, modul classes terdapat tiga fungsi bantu. Fungsi-fungsi tersebut adalah fill, encrypt dan decrypt. Pada implementasi ini,

selain ketiga fungsi tersebut juga ditambahkan satu buah fungsi yang bertugas mengubah dari string '*True*' atau '*False*' menjadi angka 1 atau 0.

Fungsi fill menambahkan spasi sebanyak sisa dari panjang string untuk memenuhi kelipatan tertentu. Langkah-langkahnya adalah memeriksa apakah panjang dari string habis dibagi angka tertentu. Jika ya, maka tidak perlu ditambahkan spasi pada belakang string. Jika tidak, maka dihitung nilai tertentu tersebut dikurangi dengan sisa pembagian dan menambahkan spasi sebanyak hasil perhitungan tersebut. Berikut pseudocode dari fungsi fill.

```
def fill(data, ln):
    cek apakah modulus data dari ln = 0
    jika ya:
        kembalikan data asli
    jika tidak:
        looping sebanyak ln - hasil modulus
            tambahkan spasi dibelakang data
        kembalikan data termodifikasi
```

Gambar 4.3 Pseudocode fungsi fill pada modul classes

Pada fungsi *encrypt* dan *decrypt*, merupakan fungsi pendek yang masing-masing hanya terdiri dari dua baris. Namun dalam satu baris tersebut berisi beberapa pemanggilan fungsi yang dijadikan satu. Berikut pseudocode dari kedua fungsi tersebut.

Pada fungsi enkripsi, terdiri dari empat langkah fungsi yang memproses input/output secara berurutan. Sedangkan pada fungsi dekripsi hanya terdiri dari tiga langkah fungsi. Hal ini disebabkan karena fungsi *compress* dan *decompress* pada modul zlib memberikan tanda berupa header pada hasil kompresi. Sehingga

walaupun data hasil kompresi ditambah dengan spasi sebanyak apapun, dengan mengacu pada header, dekompresi dari data terkompresi tersebut tetap berhenti sesuai pada nilai panjang data pada header.

```
def encrypt(key, data):
    Buat object yang akan melakukan enkripsi
    Dump data ke bentuk string menggunakan pickle
    Kompresi data menggunakan zlib
    Buat hasil kompresi menjadi kelipatan 8
        menggunakan fill
    Enkripsi hasil fill
    Kembalikan hasil enkripsi

def decrypt(key, data):
    Buat object yang akan melakukan dekripsi
    Dekripsi data
    Dekompresi hasil dekripsi menggunakan zlib
    Load string hasil dekripsi menggunakan pickle
    Kembalikan object hasil load pickle
```

Gambar 4.4 Pseudocode fungsi encrypt dan decrypt

Fungsi terakhir adalah fungsi boolConverter yang bertugas melakukan konversi dari *boolean* pada delphi ke nilai angka karena mysql menggunakan angka (*tinyint*) sebagai ganti *boolean*. Data dari delphi bukan asli *boolean*, melainkan string yang mewakili nilai *boolean*, yaitu '*True*' dan '*False*'. Sebab pada sisi delphi, dataset mengembalikan nilai masing-masing field dengan nilai string.

4.3 Implementasi Sistem

Dalam memenuhi rancangan pada bab terdahulu, maka pada implementasi ini juga terdapat lima class pada sisi server, satu class pada sisi python client, dan satu class pada sisi delphi client.

4.3.1 Sistem di sisi server

Class-class berikut dibangun sesuai dengan rancangan pada bab sebelumnya. Class-class tersebut adalah clientHandler, clientIdentifier, timerThread, clientManagement dan remoteClient.

a. clientHandler

Memiliki atribut yang bernama timeout dan sequence. Timeout merupakan object integer yang menandakan apakah session client masih aktif atau tidak. Session akan dihapus jika timeout mencapai nilai nol.

Sequence merupakan object random yang bertugas melakukan pengacakan kunci untuk enkripsi dan dekripsi. Object random ini menggunakan clientId sebagai seed. Dengan demikian, kunci untuk enkripsi dan dekripsi selalu berubah-ubah. Hanya client yang dapat mengikuti urutan request dan result dengan benar.

Ketika terjadi pemanggilan method, masing-masing method melakukan hal yang sama, yaitu dekripsi argumen, pengambilan data argumen dari tuple argumen, pemrosesan data, memformat data kembalian ke bentuk tuple jika diperlukan, dan enkripsi data kembalian. Berikut pseudocode secara umum pada method milik clientHandler.

```
def method(self, args):
    args = decrypt(kunci acak, args)
    arg1 = args[0]
    arg2 = args[1]
    arg3 = args[2]
    ...
    argn = args[n]
    '''
    dilakukan pemrosesan data sesuai kebutuhan
    '''
    # format nilai kembalian jika perlu
    result = (res1, res2, res3, ..., resn)
    result = encrypt(kunci acak, args)
    return result
```

Gambar 4.5 Pseudo pada method class clientHander

Seperti dijelaskan sebelumnya dan tertuang pada gambar di atas, proses enkripsi dan dekripsi menggunakan kunci yang berbeda-beda. Tiap kali melakukan enkripsi dan dekripsi selalu mengacak kunci terlebih dahulu. Mekanisme dari pencocokan kunci untuk enkripsi-dekripsi client-server akan dijelaskan pada subbab 4.3.3.

Pada method select (baik selectAnggota maupun selectKoleksi) selalu menghasilkan data kembalian yang terformat dalam bentuk tuple. Isi dari tuple tersebut adalah fnum, rnum, pnum, meta, dan recordset seperti dijelaskan pada subbab 4.1 pada gambar 4.2.

Meta merupakan object list yang elemennya juga merupakan list. Pertama kali object meta dibuat berupa list kosong. Lalu melakukan pengulangan pada cursor description untuk mendapatkan metadata dari

hasil query. Cursor description merupakan tuple dengan tujuh elemen berupa satu string dan enam integer. Elemen-elemen tersebut adalah nama, tipe data, ukuran penampilan data, ukuran internal, presisi, skala, boleh null atau tidak. Dalam aplikasi kali ini yang diambil hanya nama, tipe data, dan ukuran internal. Berikut implementasinya.

```
meta = [0]
for field in cursor.description:
      meta_detail = []
      meta_detail.append(field[0])
      meta_detail.append(field[1])
      meta_detail.append(field[3])
      meta.append(meta_detail)
```

Gambar 4.6 Mendapatkan metadata

Setelah mendapatkan metadata, selanjutnya adalah memformat recordset. Dalam proses ini terjadi pengulangan bertingkat. Tingkat pertama untuk tiap record, dan tingkat berikutnya untuk tiap data dalam record. Data lalu dirubah ke bentuk string untuk menghindari terjadinya ketidak cocokan format data dengan delphi. Berikut implementasinya.

```
res = cursor.fetchall()
recordset = []
for row in res:
      record = []
      for field in row:
            record.append(str(field))
      recordset.append(record)
```

Gambar 4.7 Mendapatkan recordset

b. clientIdentifier

Seperti telah dijelaskan sebelumnya, class ini memiliki sebuah dictionary yang memetakan clientId dengan clientHandler yang menangani id tersebut. Object dictionary pada python sama dengan object map dalam namespace std milik c++. Merupakan pasangan *key:value* dimana key adalah indeks untuk mencari value.

```
class clientIdentifier:
    def __init__(self):
        self.idMapper = {}
    def register(self, clientId, clientHdl):
        self.idMapper[clientId] = clientHdl
    def unregister(self, clientId):
        if self.idMapper.has_key(clientId):
            del self.idMapper[clientId]
            return 0
        else:
            return -1
```

Gambar 4.8 Implementasi clientIdentifier

Method dalam class ini berjumlah dua buah. Digunakan untuk mendaftarkan dan menghapus session. Ketika melakukan pendaftaran, cukup dengan menggunakan key layaknya menggunakan indeks dalam array. Dan untuk menghapus pasangan session, harus diperiksa terlebih dahulu apakah key tersebut masih ada atau tidak dengan menggunakan method `has_key()`. Sebab ada dua class yang memanggil penghapusan ini. Yaitu timerThread dan clientManager. Setelah dipastikan key ada pada

dictionary, baru dilakukan penghapusan dengan perintah `del dictionary[key]`. Implementasi class tersebut ada pada gambar 4.8.

c. timerThread dan timeoutThread

```
class timerThread(threading.Thread):
    def __init__(self):
        threading.Thread.__init__(self)
    def run(self):
        self.setName('timerThread')
        while(not done):
            time.sleep(1)
            eventTimeout.set()
        return
#map adalah dictionary pada clientIdentifier
class timeoutThread(threading.Thread):
    def __init__(self):
        threading.Thread.__init__(self)
    def run(self):
        self.setName('timeoutThread')
        while(not done):
            eventTimeout.wait()
            eventTimeout.clear()
            if len(map) > 0:
                tempId = []
                for first, second in map.iteritems():
                    second.timeout = second.timeout - 1
                    if second.timeout == 0:
                        tempId.append(first)
                for clientId in tempId:
                    res = clients.unregister(clientId)
                    if res == 0:
        return
```

Gambar 4.9 Implementasi timerThread dan timeoutThread

Kedua class ini melakukan pengulangan selama tidak diperintahkan untuk berhenti. Class timerThread hanya menjalankan dua perintah dalam tiap putarannya yaitu menunggu satu detik dengan `sleep(1)` dan memberitahu timeoutThread bahwa satu detik telah terlewati.

Sedangkan timeoutThread melakukan tugas mengurangi semua data timeout pada masing-masing session dengan satu. Jika data timeout tersebut ada yang mencapai nol, maka session tersebut akan dihapus. Dalam menjalankan pengulangan, timeoutThread menunggu pemberitahuan dari timerThread melalui object Event. Ketika pemberitahuan tersebut datang, maka timeoutThread akan mereset event tersebut untuk timeout berikutnya. Setelah itu timeoutThread melakukan pekerjaan sesuai dengan yang didefinisikan pada saat perancangan. Implementasi kedua class ada pada gambar 4.9.

d. clientManagement

Class ini bertugas melayani login dan logout dari client. Agar client dapat mengakses class ini, maka class ini harus didaftarkan ke nameserver dari pyro. Untuk itu, class ini harus merupakan turunan dari class Pyro.core.ObjBase.

Memiliki tiga method. Yang pertama adalah untuk membuat id baru berupa angka random antara nol sampai satu juta (0 sampai 10^6). Yang kedua menangani login. Ketika ada client yang meminta login,

method ini mencari dalam database untuk mencocokkan username dan password dari client. Jika ditemukan, maka method ini akan memanggil method pertama untuk mendapatkan id baru, membuat object dari class clientHandler, dan mendaftarkan id serta clientHandler tersebut ke clientIdentifier. Setelah itu id baru tersebut dikembalikan kepada client untuk digunakan dalam mengakses remoteClient. Yang ketiga menangani logout. Method ini memanggil method unregister milik clientIdentifier.

```
class clientManagement(Pyro.core.ObjBase):
    def __init__(self):
        Pyro.core.ObjBase.__init__(self)
    def _generateId(self):
        return random.randint(0,1000000)
    def login(self, user, passwd):
        db = MySQLdb.connect(connection_string)
        cursor = db.cursor()
        res = cursor.execute(query, (user, passwd))
        cursor.close()
        db.close()
        if res == 1:
            tempId = self._generateId()
            tempHdl = clientHandler(tempId)
            clients.register(tempId, tempHdl)
            print 'client id(%d), logged in' %(tempId)
            return tempId
        else:
            return None
    def logout(self, clientId):
        res = clients.unregister(clientId)
        if res == 0:
            print 'id:%s logged out' %(clientId)
```

Gambar 4.10 Implementasi class clientManager

e. remoteClient

Class remoteClient bertugas meneruskan request dari client kepada clientHandler. Sehingga memiliki method yang sama dengan clientHandler dengan tambahan satu argumen berupa clientId untuk menentukan apakah client tersebut berhak melakukan pemanggilan terhadap method-method dari class ini atau tidak.

Sama seperti clientManager, class ini juga merupakan turunan dari Pyro.core.ObjBase. Masing-masing method memiliki kesamaan dalam memproses request sebelum meneruskan ke clientHandler. Yang pertama dilakukan adalah memeriksa apakah clientId disertakan. Jika tidak, kembalikan nilai -1 dalam tuple. Jika disertakan, maka method akan memeriksa ke clientIdentifier apakah session dengan id tersebut ada. Jika tidak, kembalikan nilai -2 dalam tuple. Jika id tersebut valid, maka data timeout pada clientHandler direset ke nilai awal dan request diteruskan ke clientHandler dan mengembalikan apapun kembalian dari clientHandler berupa tuple bersamaan dengan nilai 0.

```
def namaFungsi(self, args, clientId=None)
    if clientId == None:
        return (-1,)
    elif not clientIdentifier.map.has_key(clientId):
        return (-2,)
    else:
        clientIdentifier.map[clientId].timeout = 600
        return (0,
         clientIdentifier.map[clientId].namaFungsi(args))
```

Gambar 4.11 Pseudocode dari method pada remoteClient

4.3.2 Sistem di sisi client

Pada sisi client, terdapat class clientCtl dan TPythonDatasetClient. ClientCtl merupakan class lokal yang menjembatani antara remote object dengan delphi. Sedangkan TpythonDatasetClient bertugas menangani format tampilan GUI yang menggunakan TClientDataset sebagai perwakilan tabel.

a. clientCtl

Class clientCtl menangani kontrol remote object. Class inilah yang ditangkap oleh delphi dan dipanggil method-methodnya untuk diteruskan ke object pada server. Namun diantara pengiriman request dan penerimaan data kembalian terdapat proses enkripsi dan dekripsi.

Class ini memiliki tujuh property. Dua buah property merupakan object proxy yang mewakili object pada server, dua buah object integer yang menyimpan jumlah field dan jumlah record yang didapat, dua buah object kosong yang nantinya diisi oleh integer untuk clientId dan macam-macam object untuk menyimpan data kembalian dari pemanggilan method dan sebuah object random dengan menggunakan clientId sebagai seed yang bertugas menjaga session dengan clientHandler pada server.

Penjagaan urutan session adalah dengan menggunakan seed yang sama pada kedua sisi. Ketika client melakukan request, maka sebuah angka random didapat baik dari sisi server maupun client. Pada sisi client digunakan untuk enkripsi dan pada sisi server digunakan untuk dekripsi. Begitu pula sebaliknya.

Method login dan logout bersesuaian dengan login dan logout pada class clientManagement. Ketika login mendapat kembalian berupa id, id tersebut disimpan pada property id. Pada method logout hanya mengirimkan request untuk menghapus session dengan id yang dikirimkan tanpa memeriksa data kembalian. Berikut pseudocode dari class dan method login.

```
class clientCtl:
    def __init__(self):
        self.client =
            Pyro.core.getAttrProxyForURI(URILogin)
        self.remote =
            Pyro.core.getAttrProxyForURI(URIRemote)
        self.id = None
        self.res = None
        self.fnum = 0
        self.rnum = 0
        self.sequence = random.Random()
    def login(self, user, passwd):
        self.id = self.client.login(user, passwd)
        if self.id == None:
            return 'Invalid Login'
        else:
            self.sequence.seed(self.id)
```

Gambar 4.12 Pseudocode class clientCtl

Sedangkan method-method yang diteruskan kepada remoteClient menyatukan data argumen menjadi tuple dan melakukan enkripsi kepada object tuple tersebut. Setelah menerima data kembalian, dicek object dengan indeks nol untuk memeriksa status pemanggilan method. Jika

status bernilai nol (id dinyatakan valid), maka object pada indeks satu didekripsi dan disimpan pada property res dan selanjutnya, object-object dalam res yang bernilai integer dikonversikan kembali kedalam bentuk integer karena setelah proses enkripsi-dekripsi, integer tersebut tidak dapat dibaca oleh delphi. Berikut pseudocode dari method-method tersebut.

```
def namaFungsi(self, arg1, arg2, ..., argn, self.id):
    args = (self, arg1, arg2, ..., argn)
    args = encrypt(kunci acak, args)
    res = self.remote.namaFungsi(args, self.id)
    if res[0] == -1:
        return -1
    if res[0] == -2:
        return -2
    else:
        self.res = decrypt(kunci acak, res[1])
        self.res[x] = int(self.res[x])
        return 0
```

Gambar 4.13 Pseudocode dari method utama pada clientCtl

b. TPythonDatasetClient

Class inilah yang memanipulasi dataset, mengatur komponen-komponen tampilan dan menangkap object dari class clientCtl pada python. Class ini memiliki banyak property dan method karena mengatur tampilan komponen-komponen seperti TDBGrid, TLabel, TButton dan TtDBEdit. Diantara method-method tersebut, ada beberapa yang perlu diperhatikan.

Yang pertama adalah method capturePyObject. Method ini bertugas menangkap object python yang dijalankan oleh delphi ini. Langkah-langkahnya sama persis dengan contoh pada gambar 2.14.

```
procedure TPythonDatasetClient.initDataset(var dataset:
TClientDataSet);
var
  metadata : OleVariant;
  i : integer;
begin
  metadata := client.res[3];
  clearEditorReference;
  dataset.Active := False;
  dataset.FieldDefs.Clear;
  for i := 0 to client.res[0]-1 do
  begin
    with dataset.FieldDefs.AddFieldDef do
    begin
      Name := metadata[i][0];
      DataType := fieldtype(metadata[i][1]);
      if DataType = ftString then
        if metadata[i][2] > 0 then
          Size := metadata[i][2]
        else Size := 4;
      FieldNo := i + 1;
    end;
  end;
  dataset.CreateDataSet;
  dataset.Active := True;
end;
```

Gambar 4.14 Implementasi initDataset

Yang kedua adalah initDataset. Method ini bertugas menghapus data pada dataset, kemudian membuat definisi field yang baru untuk data

pada object clientCtl. Method ini melakukan pengulangan sebanyak jumlah field yang kemudian mengisikan nama, tipe data, ukuran data dan posisi field pada tabel. Implementasi initDataset ada pada gambar 4.14.

Yang ketiga adalah fetchRecord. Bertugas mengisikan data hasil query pada dataset yang baru saja diinisialisasi menggunakan initDataset. Melakukan perulangan dua tingkat dengan mengisikan data yang berupa array dua dimensi. Berikut implementasinya.

```
procedure TPythonDatasetClient.fetchRecord(var dataset:
TClientDataSet);
var
  metadata : OleVariant;
  recordset : OleVariant;
  i, j : integer;
begin
  metadata := client.res[3];
  recordset := client.res[4];
  dataset.EmptyDataset;
  for j := 0 to client.res[1]-1 do
  begin
    With dataset do
    begin
      Append;
      for i := 0 to client.res[0]-1 do
      begin
        FieldValues[metadata[i][0]] := recordset[j][i];
      end;
      Post;
    end;
  end;
end;
```

Gambar 4.15 Implementasi fetchRecord

Yang ketiga adalah method yang memanggil method dari clientCtl. Method ini merupakan fungsi yang meneruskan request baik dari class ini sendiri maupun dari form kepada clientCtl.

c. Method-method pada form delphi

Method-method yang perlu diperhatikan pada class form adalah pada event datasetAfterPost. Pada method inilah form memanggil method manipulasi record pada dataset yang dimiliki TPythonDatasetClient. Method ini menggunakan data pada field menjadi argumen pada pemanggilan method-method tersebut. Berikut Pseudocode dari event ini.

```
var
  strlist : array of string;
  i, len: integer;
begin
  len := mainForm.dset.FieldCount;
  SetLength(strlist, len);
  for i := 0 to len-1 do
    strlist[i] := dset.Fields.Fields[i].AsString;
  if dbstate = 1 then
    insertMethod
  else if dbstate = 2 then
    updateMethod;
end;
```

Gambar 4.16 Pseudocode event datasetAfterPost

Delete tidak ditangani pada event ini karena delete tidak memerlukan method post pada dataset dan langsung menghapus record yang sedang aktif.

BAB V

UJI COBA DAN EVALUASI

Pada bab ini akan dilakukan uji coba terhadap sistem yang telah dibuat. Uji coba dilakukan untuk mengetahui barapa lama waktu yang diperlukan bila suatu transaksi dilakukan. Waktu proses dihitung mulai suatu request dikirim kepada server hingga jawaban dari server selesai diolah. Hasil uji coba dibandingkan dengan aplikasi yang langsung terhubung ke database. Diharapkan

5.1 Deskripsi Hardware Uji Coba

Uji coba terhadap sistem yang telah dibangun dilakukan dengan menggunakan lingkungan dan spesifikasi sebagai berikut:

Spesifikasi platform server:

- Windows XP service pack 2
- Python 2.4 dengan ekstensi MySQLdb, pyro dan pycripto
- MySQL 4.1.13
- Apache 2.0.54

Spesifikasi hardware server:

- Pentium IV 2.80 GHz
- RAM 504 MB

Spesifikasi platform client:

- Microsoft Windows 2000 Service Pack 4
- Python 2.4 dengan ekstensi pyro dan pycripto

Spesifikasi hardware client:

- Pentium IV 2.40 GHz
- RAM 479 MB

5.2 Skenario Uji Coba

Uji coba akan dilakukan dengan mengukur waktu yang diperlukan saat suatu transaksi dilakukan. Waktu proses dihitung mulai dari request dikirim kepada server, respond diterima dari server, pengolahan data lebih lanjut oleh kontrol client, pembacaan data pada kontrol client hingga pengolahan hasil pembacaan selesai dilakukan.

Tabel yang digunakan untuk uji coba adalah tabel master_koleksi dari Aplikasi Sistem Informasi Ruang Baca FTIF. Tabel dipresentasikan dalam bentuk dataset yang ditampilkan dengan DBGrid. Sedangkan detail dari record yang sedang aktif ditampilkan oleh dipresentasikan oleh dataset detail dan ditampilkan oleh sepasang array dinamis berupa label dan DBEdit. Jumlah data yang berupa record juga divariasikan untuk mendapatkan variasi waktu. Uji coba juga dilakukan dengan menghilangkan fungsi enkripsi data untuk mencari perbedaan durasi proses antara data terenkripsi dan tak terenkripsi.

Uji coba dilanjutkan dengan melakukan uji coba aplikasi lain yang juga dibangun menggunakan delphi. Aplikasi ini melakukan koneksi langsung dengan database melalui koneksi ADO. Uji coba ini dilakukan untuk membandingkan waktu yang diperlukan kedua aplikasi untuk melakukan proses query tabel master_koleksi sebanyak 50, 100, 200, 500, 1000.

Berikut ini tiga skenario yang akan dilakukan dalam uji coba:

a. Uji coba skenario view data koleksi

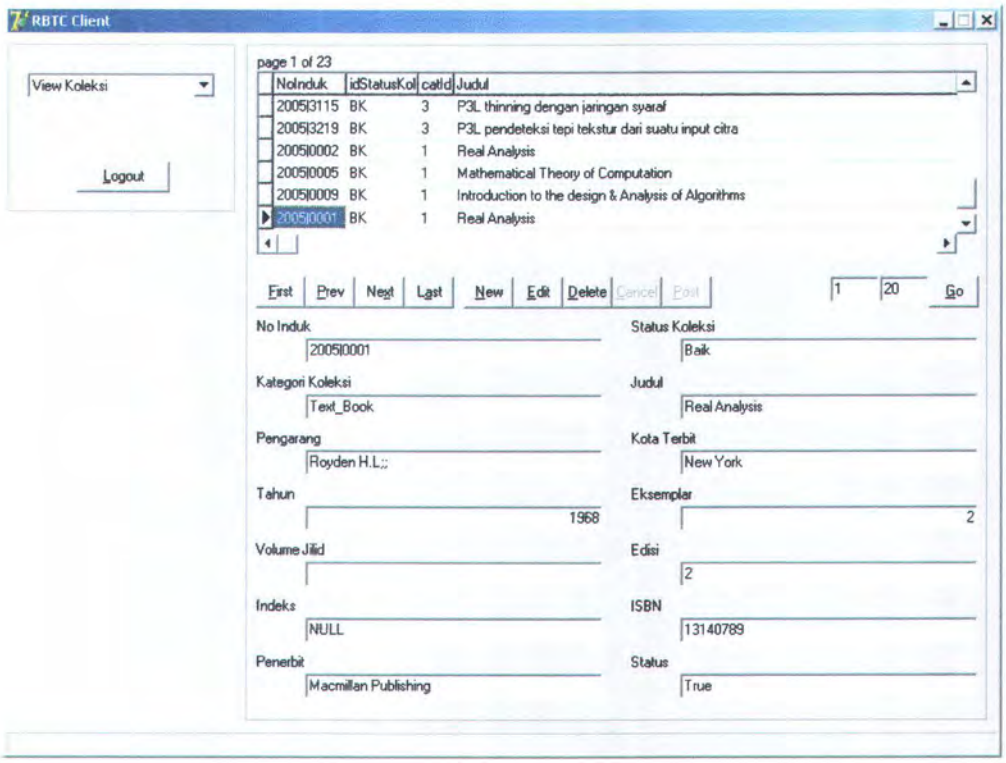
b. Uji coba skenario view data koleksi tanpa enkripsi

c. Uji coba skenario view data koleksi dengan koneksi langsung

d. Uji coba skenario pembandingan besaran transfer data

5.2.1 Uji Coba Skenario View Data Koleksi

Pada uji coba ini akan dilakukan view dari tabel master_koleksi pada DBGrid dan array DBEdit. Jumlah data yang diambil untuk ditampilkan bervariasi mulai 50, 100, 200, 500 hingga 1000 data.



Gambar 5.1 Proses view data master_koleksi oleh sistem

Evalasi terhadap skenario view data koleksi

Hasil uji coba dengan data sebanyak 50 record:

Tabel 5.1 Hasil uji coba 1 dengan 50 record

Lama Waktu Proses	
Waktu request dikirim	17:50:11.578
Clock saat request diterima server	21.468
Clock saat request selesai ditangani server	21.489
Durasi proses pada server	0 jam 0 menit 0.021 detik
Durasi client menunggu response	0 jam 0 menit 0.034 detik
Waktu pemrosesan jawaban oleh client selesai	17:50:11.687
Waktu total	0 jam 0 menit 0.109 detik

Hasil uji coba dengan data sebanyak 100 record:

Tabel 5.2 Hasil uji coba 1 dengan 100 record

Lama Waktu Proses	
Waktu request dikirim	17:51:14.937
Clock saat request diterima server	84.822
Clock saat request selesai ditangani server	84.859
Durasi proses pada server	0 jam 0 menit 0.037 detik
Durasi client menunggu response	0 jam 0 menit 0.059 detik
Waktu pemrosesan jawaban oleh client selesai	17:51:15.125
Waktu total	0 jam 0 menit 0.188 detik

Hasil uji coba dengan data sebanyak 200 record:

Tabel 5.3 Hasil uji coba 1 dengan 200 record

Lama Waktu Proses	
Waktu request dikirim	17:52:14.500
Clock saat request diterima server	144.384
Clock saat request selesai ditangani server	144.456
Durasi proses pada server	0 jam 0 menit 0.072 detik
Durasi client menunggu response	0 jam 0 menit 0.113 detik
Waktu pemrosesan jawaban oleh client selesai	17:52:14.812
Waktu total	0 jam 0 menit 0.312 detik

Hasil uji coba dengan data sebanyak 500 record:

Tabel 5.4 Hasil uji coba 1 dengan 500 record

Lama Waktu Proses	
Waktu request dikirim	17:52:51.265
Clock saat request diterima server	181.148
Clock saat request selesai ditangani server	181.323
Durasi proses pada server	0 jam 0 menit 0.175 detik
Durasi client menunggu response	0 jam 0 menit 0.286 detik
Waktu pemrosesan jawaban oleh client selesai	17:52:52.031
Waktu total	0 jam 0 menit 0.766 detik

Hasil uji coba dengan data sebanyak 1000 record:

Tabel 5.5 Hasil uji coba 1 dengan 1000 record

Lama Waktu Proses	
Waktu request dikirim	17:53:30.062
Clock saat request diterima server	219.945
Clock saat request selesai ditangani server	220.294
Durasi proses pada server	0 jam 0 menit 0.349 detik
Durasi client menunggu response	0 jam 0 menit 0.569 detik
Waktu pemrosesan jawaban oleh client selesai	17:53:31.531
Waktu total	0 jam 0 menit 1.469 detik

Pengukur waktu yang digunakan pada server adalah clock sejak aplikasi server dijalankan dalam hitungan detik. Waktu proses relatif lebih lama adalah pada proses penanganan data recordset hasil kembalian dari remote object. Hal ini terlihat dari hasil uji coba di atas dimana total waktu hampir tiga kali lipat dari waktu dimana client menunggu response dari server.

5.2.2 Uji Coba Skenario View Data Koleksi tanpa enkripsi

Pada uji coba ini tetap akan dilakukan view dari tabel master_koleksi pada DBGrid dan array DBEdit. Jumlah data yang diambil untuk ditampilkan bervariasi mulai 50, 100, 200, 500 hingga 1000 data. Dalam uji coba kali ini, perintah yang melakukan enkripsi dan dekripsi dihilangkan untuk mengetahui durasi pemrosesan data tanpa pengamanan enkripsi.

Evalasi terhadap skenario view data koleksi tanpa enkripsi

Hasil uji coba dengan data sebanyak 50 record:

Tabel 5.6 Hasil uji coba 2 dengan 50 record

Lama Waktu Proses	
Waktu request dikirim	18:11:20.437
Clock saat request diterima server	17.702
Clock saat request selesai ditangani server	17.712
Durasi proses pada server	0 jam 0 menit 0.010 detik
Durasi client menunggu response	0 jam 0 menit 0.015 detik
Waktu pemrosesan jawaban oleh client selesai	18:11:20.531
Waktu total	0 jam 0 menit 0.094 detik

Hasil uji coba dengan data sebanyak 100 record:

Tabel 5.7 Hasil uji coba 2 dengan 100 record

Lama Waktu Proses	
Waktu request dikirim	18:12:25.343
Clock saat request diterima server	82.599
Clock saat request selesai ditangani server	82.616
Durasi proses pada server	0 jam 0 menit 0.017 detik
Durasi client menunggu response	0 jam 0 menit 0.023 detik
Waktu pemrosesan jawaban oleh client selesai	18:12:25.484
Waktu total	0 jam 0 menit 0.141 detik

Hasil uji coba dengan data sebanyak 200 record:

Tabel 5.8 Hasil uji coba 2 dengan 200 record

Lama Waktu Proses	
Waktu request dikirim	18:13:00.859
Clock saat request diterima server	118.117
Clock saat request selesai ditangani server	118.147
Durasi proses pada server	0 jam 0 menit 0.030 detik
Durasi client menunggu response	0 jam 0 menit 0.040 detik
Waktu pemrosesan jawaban oleh client selesai	18:13:01.093
Waktu total	0 jam 0 menit 0.234 detik

Hasil uji coba dengan data sebanyak 500 record:

Tabel 5.9 Hasil uji coba 2 dengan 500 record

Lama Waktu Proses	
Waktu request dikirim	18:13:41.968
Clock saat request diterima server	159.226
Clock saat request selesai ditangani server	159.295
Durasi proses pada server	0 jam 0 menit 0.069 detik
Durasi client menunggu response	0 jam 0 menit 0.092 detik
Waktu pemrosesan jawaban oleh client selesai	18:13:42.515
Waktu total	0 jam 0 menit 0.547 detik

Hasil uji coba dengan data sebanyak 1000 record:

Tabel 5.10 Hasil uji coba 2 dengan 1000 record

Lama Waktu Proses	
Waktu request dikirim	18:14:17.187
Clock saat request diterima server	194.453
Clock saat request selesai ditangani server	194.587
Durasi proses pada server	0 jam 0 menit 0.133 detik
Durasi client menunggu response	0 jam 0 menit 0.189 detik
Waktu pemrosesan jawaban oleh client selesai	18:14:18.296
Waktu total	0 jam 0 menit 1.109 detik

Dengan hasil uji coba di atas semakin terlihat bahwa pada proses penanganan data recordset hasil kembalian dari remote object memakan waktu paling banyak. Bahkan lebih dari lima kali dari waktu client menunggu response jika tanpa menggunakan enkripsi.

5.2.3 Uji Coba Skenario View Data Koleksi dengan koneksi langsung

Pada uji coba ini juga akan dilakukan view dari tabel master_koleksi pada DBGrid dan array DBEdit namun pada aplikasi baru yang lebih sederhana. Jumlah data yang diambil untuk ditampilkan tetap bervariasi mulai 50, 100, 200, 500 hingga 1000 data. Dalam uji coba kali ini, koneksi database dilakukan langsung melalui TADOQuery dengan driver MySQL ODBC 3.51 Driver. Berikut hasil uji coba tersebut.

Evalasi terhadap skenario view data koleksi dengan koneksi langsung

Hasil uji coba dengan data sebanyak 50 record:

Tabel 5.11 Hasil uji coba 3 dengan 50 record

Lama Waktu Proses	
Waktu request dikirim	18:07:39.796
Waktu response diterima client	18:07:39.812
Waktu total	0 jam 0 menit 0.016 detik

Hasil uji coba dengan data sebanyak 100 record:

Tabel 5.12 Hasil uji coba 3 dengan 100 record

Lama Waktu Proses	
Waktu request dikirim	18:08:02.578
Waktu response diterima client	18:08:02.593
Waktu total	0 jam 0 menit 0.015 detik

Hasil uji coba dengan data sebanyak 200 record:

Tabel 5.13 Hasil uji coba 3 dengan 200 record

Lama Waktu Proses	
Waktu request dikirim	18:08:44.343
Waktu response diterima client	18:08:44.375
Waktu total	0 jam 0 menit 0.032 detik

Hasil uji coba dengan data sebanyak 500 record:

Tabel 5.14 Hasil uji coba 3 dengan 500 record

Lama Waktu Proses	
Waktu request dikirim	18:09:26.718
Waktu response diterima client	18:09:26.765
Waktu total	0 jam 0 menit 0.047 detik

Hasil uji coba dengan data sebanyak 1000 record:

Tabel 5.15 Hasil uji coba 3 dengan 1000 record

Lama Waktu Proses	
Waktu request dikirim	18:10:05.046
Waktu response diterima client	18:10:05.109
Waktu total	0 jam 0 menit 0.063 detik

Dengan hasil tersebut, terlihat jelas bahwa koneksi langsung memang jauh lebih cepat. Namun dengan menggunakan remote object dan datasetpun durasi proses untuk mengambil data sebanyak 1000 dan terenkripsi hanya mencapai 1.469 detik sehingga masih dapat ditoleransi oleh pengguna. Bahkan umumnya aplikasi database hanya menampilkan maksimal 100 record tiap halaman.

Namun dengan koneksi langsung, maka proses bisnis harus didefinisikan pada client sehingga untuk pengembangan lebih lanjut kurang fleksibel. Selain itu pengamanan data dengan enkripsi tidak dapat dilakukan. Dengan koneksi tidak

langsung, maka selain pengembangan minor masih dapat dilakukan dengan fleksibel. Pengamanan juga dapat dilakukan dengan baik.

5.2.4 Uji Coba Skenario Pembandingan besaran transfer data

Hasil uji coba dengan 1000 data:

Tabel 5.16 Besaran data yang terjadi pada saat proses berjalan

Besaran transfer data masuk dan keluar	
Pada aplikasi dengan koneksi ADO (langsung)	
Rata-rata esaran data keluar saat request terjadi	78 paket
Rata-rata Besaran data keluar saat response terjadi	106 paket
Pada aplikasi dengan remote object dan data aware	
Rata-rata Besaran data keluar saat request terjadi	25 paket
Rata-rata Besaran data keluar saat response terjadi	78 paket

Dari hasil uji coba 1000 data, dengan koneksi langsung, aplikasi mengirimkan request sebesar kira-kira 78 paket dan menerima response sebesar 106 paket. Jika dibandingkan dengan aplikasi yang menggunakan remote object, request yang dikirim sebesar kira-kira 25 paket dan menerima response sebesar 78 paket. Aplikasi tersebut membutuhkan trafik lebih kecil daripada koneksi langsung karena pada aplikasi remote object yang dibangun, sebelum melakukan enkripsi, data terlebih dahulu dikompres.

BAB VI

KESIMPULAN DAN SARAN

Bab ini menyimpulkan hasil uji coba yang telah dilakukan. Selanjutnya diberikan beberapa saran yang mungkin dapat dijadikan pertimbangan untuk mengembangkan hasil yang diperoleh pada tugas akhir ini.

6.1 Kesimpulan

Setelah dilakukan serangkaian uji coba dan analisa terhadap sistem yang dibuat, maka dapat diambil kesimpulan sebagai berikut:

a. Aplikasi sistem sebagai prototipe remote object menggunakan python dengan delphi sebagai antarmuka menggunakan mendekatan secara data aware telah berhasil diimplementasikan.

b. Durasi proses data berbanding lurus dengan besar data. Besar data bergantung pada jumlah record dan jumlah field tiap record yang ditangani.

c. Sequence enkripsi-dekripsi data pada tiap session telah diimplementasikan dengan baik. Proses enkripsi-dekripsi tidak mempengaruhi secara signifikan terhadap durasi proses secara keseluruhan.

d. Dibandingkan dengan aplikasi yang melakukan koneksi langsung dengan database, aplikasi yang telah dibangun berjalan lebih lamban. Namun dapat memberikan tambahan fitur seperti pengamanan data.

6.2 Saran

Berdasarkan hasil evaluasi yang dilakukan terhadap sistem, ada beberapa saran yang perlu dipertimbangkan dalam pengembangan teknologi ini, yaitu :

a. Fungsi pengamanan data pada aplikasi tugas akhir ini masih sangat sederhana. Untuk digunakan dalam jalur internet, pengamanan data perlu dikembangkan lebih lanjut.

b. Untuk keamanan aplikasi itu sendiri, file script client.py dan classes.py pada sisi client dapat dikompres dengan algoritma sendiri yang diimplementasi langsung secara *hard-coded* pada aplikasi. Sehingga file script tidak dapat digunakan oleh aplikasi lain.

DAFTAR PUSTAKA

[1] Tyarso Bhari Permana, PPPL Thin Client Memanfaatkan XML-RPC Sebagai Sarana Transaksi Data Fom Berbentuk XML Dengan Server Python dan Client Menggunakan Delphi, FTIF/ITS, 2005

[2] Mark Lutz, David Ascher, Learning Python, O'Reilly, 1999

[3] Mark Lutz, Programming Python, 2nd Edition, O'Reilly, 2001

[4] Wesley J. Chun, Core Python Programming, Prentice Hall PTR, 2000

[5] Marc-Andre Lemburg, Python Database API Specification v2.0, http://www.python.org/peps/pep-0249.html

[6] Pyro homepage, http://pyro.sourceforge.net/

[7] A.M. Kuchling, Python Cryptography Toolkit, http://www.amk.ca/python/code/crypto

[8] Andy Bulka, Using Delphi and Python Together, http://www.atug.com/andypatterns/pythonDelphiTalk.htm