Kylix dan PostgreSQL

Kylix termasuk salah satu RAD tool yang sangat memudahkan dalam pembuatan aplikasi GUI. Digabungkan dengan PostgreSQL, kita dapat membuat aplikasi berbasis database yang mudah digunakan sekaligus canggih.

Pemrograman di Linux memang gampang-gampang susah. Di satu sisi, Linux adalah surga bagi para programer. Banyak sekali pustaka bebas yang bisa digunakan. Belum lagi, dengan banyaknya aplikasi *open source*, seorang programer—asal rajin—dapat mengintip *source code*-nya untuk mendapatkan pengetahuan lebih. Semua kemudahan tersebut lantas ditunjang lebih lanjut lagi dengan kehadiran berbagai tool pengembangan aplikasi seperti ArgoUML untuk pemodelan UML, Eclipse untuk IDE, Cervisia sebagai CVS front end, dan lain sebagainya. Komplit. Seperti seolah-olah Linux hadir hanya untuk programer.

Sayangnya, perhatian para pengembang software untuk RAD sepertinya masih belum terlalu besar. Atau, budaya pengembangan aplikasi dengan sistem *drag*-sana-drag-sini lantas menulis kode sesedikit mungkin, sepertinya kurang populer di dunia free software. Sebagian besar developer lebih senang menyusun ribuan kepingan *puzzle*-nya sendiri untuk menghasilkan aplikasi besar. Termasuk menyusun GUI sendiri dengan berbagai pustaka GUI yang ada.

Menyusun sendiri memang sangat menggoda. Kita memiliki kontrol penuh atas kode-kode kita. Bahkan, kalau tidak puas, kita bisa mengubah sedikit-sedikit *binding* pustaka dengan bahasa pemrograman yang kita gunakan. Sekedar mengubah *constructor class* yang kita inginkan, misalnya.

Sayangnya, sistem susun sendiri memiliki kekurangan: waktu pengembangan, terutama dalam penyusunan GUI menjadi lebih lama. Hal tersebut belum lagi ditambah dengan kurangnya widget misalnya. Selain itu, kekurangan-kekurangan lain seperti kurang matangnya binding juga bisa menjadi masalah. Belum lagi dengan banyaknya pilihan pustaka GUI. Semuanya bisa menjadikan programer Windows yang berpindah ke Linux menjadi gila.

Beberapa RAD tool bagus memang tersedia. Seperti misalnya Netbeans untuk programmer Java, atau Qt Designer untuk programmer C++ dengan pustaka GUI Qt. Sayangnya, Qt Designer versi bebas tidak mengijinkan kita untuk membuat aplikasi proprietary. Ada pula Gambas, yang cukup ambisius, namun masih jauh dari matang. Beberapa lainnya masih ada, termasuk Lazarus untuk programer pascal, namun sebagian besar, tidaklah sematang dan semudah Visual Basic ataupun Delphi.

Borland kemudian merilis Kylix yang dapat digunakan sebagai RAD untuk bahasa Delphi ataupun C++. Kylix sendiri juga dirilis dalam beberapa edisi, yang salah satu di antaranya adalah *open edition*. Gratis download dan pakai, namun hanya bisa digunakan untuk membangun aplikasi GPL.

Seperti halnya Delphi, Kylix cukup matang dan lengkap. Kylix edisi *enterprise* bahkan menyediakan dukungan untuk konektifitas database besar seperti Oracle dan DB2. Walau dengan tidak adanya update dari Borland selama dua tahunan dan perkembangan Linux yang semakin cepat, Kylix masih dapat diandalkan untuk membangun aplikasi besar.

Di edisi lalu, kita membahas seluk beluk instalasi Kylix. Beberapa hal memang harus diperhatikan untuk menyiasati tidak adanya update resmi dari Borland untuk menyesuaikan diri dengan perkembangan Linux.

Di edisi ini, kita akan membahas konektifitas Kylix dengan PostgreSQL, sebagai ORDBMS yang populer dan semakin populer. Sebagai konsekuensi dari tidak adanya update Kylix tersebut, banyak kondisi telah menjadikan beberapa hal tidak dapat langsung digunakan. Butuh beberapa trik untuk menyesuaikan diri. Kita akan membahas tuntas trik yang diperlukan agar konektivitas Kylix dan PostgreSQL dapat dimungkinkan. Contoh aplikasi yang dapat digunakan untuk memahami Kylix dan database lebih lanjut juga akan diberikan.

Distro yang digunakan adalah SUSE 9.1, namun penerapan pada distro berbasis kernel 2.6.x.y lainnya seharusnya tidak akan terlalu berbeda. Kylix yang digunakan adalah Kylix 3 edisi enterprise. PostgreSQL yang digunakan adalah 7.3.4 yang penulis koneksikan secara remote ke server lain.

Untuk mendukung artikel ini, sebaiknya Kylix telah terinstal di sistem Anda. Bacalah edisi yang lalu untuk seluk-beluk instalasinya. PostgreSQL juga sebaiknya telah terinstall, atau Anda memiliki hak akses ke server postgreSQL lain. Instalasi PostgreSQL di luar cakupan artikel ini.

Menyiapkan database

Pertama-tama, nyalakan terlebih dahulu PostgreSQL Anda, dan pastikan Anda memiliki hak akses ke server tersebut. Apabila belum, kita akan membuat sebuah user dan database dengan nama yang sama dengan nama username sistem.

Login-lah terlebih dahulu ke komputer server apabila database server Anda terpisah dengan komputer yang Anda gunakan. Berikut ini adalah perintah-perintah untuk membuat user dan membuat database:

```
nop@bios:~> su
Password:
bios:/home/nop # su postgres
postgres@bios:/home/nop>
createuser nop
Shall the new user be allowed to
create databases? (y/n) y
Shall the new user be allowed to
create more new users? (y/n) n
CREATE USER
postgres@bios:/home/nop> exit
exit
bios:/home/nop # exit
```

```
exit
nop@bios:~> createdb nop
CREATE DATABASE
```

Perintah-perintah tersebut dapat diartikan sebagai berikut:

- User nop telah login ke server.
- Memberikan perintah su untuk menjadi user root.
- Setelah menjadi user root, memberikan perintah su postgres untuk menjadi user postgres. Di SUSE (dan umumnya di distro lain), postgres adalah super user untuk database PostgreSQL.
- Setelah menjadi user postgres, kita memberikan perintah createuser nop untuk membuat user nop di database server PostgreSQL. User nop diijinkan untuk membuat database, namun tidak diizinkan untuk membuat user lain.
- Dengan perintah exit sebanyak dua kali, kita kembali ke user root dan kembali ke user nop.
- Sebagai nop, kita membuat database nop dengan perintah createdb nop.

Secara default, pengaturan server PostgreSQL umumnya memberi kemudahan (dan keamanan) untuk identifikasi berdasarkan user yang sedang login ketika konektifitas dilakukan. Pengaturan defaultnya berdasarkan *credential socket* untuk user yang sama dengan user yang sedang login di sistem. Untuk mengubah hal ini, aturlah isi file pg_hba.conf yang umumnya terletak di /var/lib/pgsql/data/.

Ujilah koneksi dengan memberikan perintah berikut ini:

```
psql -h 192.168.0.1 -U nop -d
nop
Welcome to psql 7.4.2, the Post
greSQL interactive terminal.

Type:  \copyright for
distribution terms
       \h for help with SQL
commands
       \? for help on internal
slash commands
       \g or terminate with
semicolon to execute query
       \q to quit

nop=>
```

Apabila Anda menjumpai tampilan tersebut, maka koneksi dengan database server PostgreSQL menggunakan client psql telah berhasil. Sebagai catatan, karena server PostgreSQL penulis berada pada komputer yang berbeda, maka opsi -h <host> harus diberikan.

Buatlah sebuah *table*, katakanlah dengan nama test. Ketikkanlah perintah berikut ini pada prompt nop=> sebelumnya:

```
nop=> create table test (no
serial, nama character
varying(64));
NOTICE:  CREATE TABLE will
create implicit sequence 'test_
no_seq' for SERIAL column 'test.
no'
CREATE TABLE
nop=>
```

Kini, Anda telah memiliki satu table dengan nama test yang terdiri dari dua field: no dan nama.

Untuk memudahkan administrasi PostgreSQL, Anda mungkin ingin menggunakan PHPPgAdmin (berbasis web, Anda membutuhkan Apache dan PHP terinstal, sedikit repot namun memiliki tampilan memikat. Dapatkan di *http://phppgadmin.sourceforge.net*) atau PgAccess (dibuat dengan Tcl/Tk, lengkap namun tampil dengan ciri khas Tk, dapatkan di *http://pgacces.org*).

Database telah siap digunakan. Kini, kita tinggal mengurus konektivitas Kylix dengan PostgreSQL.

Konektivitas Kylix dan PostgreSQL

Sebelum kita melanjutkan, pastikan Anda telah menginstall Kylix enterprise lengkap dengan dbexpress driver untuk PostgreSQL.

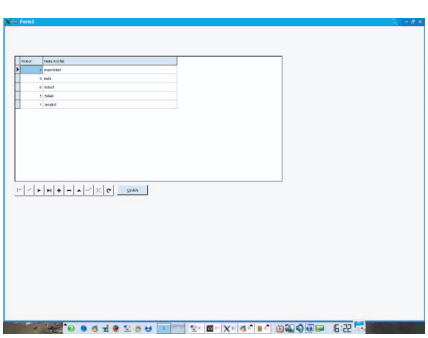
Konektifitas Kylix dan PostgreSQL menjadi masalah karena Kylix membutuhkan membutuhkan pustaka client yang kuno dan sudah tidak digunakan lagi saat ini (Kylix membutuhkan libpq.so.2, sementara umumnya yang ada saat ini adalah libpq.so.3).

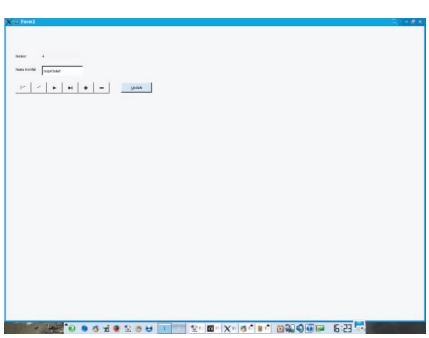
Dalam melakukan koneksi, Kylix memiliki mekanisme yang sedikit berbeda dengan bahasa-bahasa lainnya. Kylix menggunakan penengah untuk dapat terhubung ke database server PostgreSQL melalui libpq. Pustaka penengah tersebut adalah libsqlpg.so. Pustaka yang satu inilah yang nakal dan sangat setia dengan libpq.so.2 (libpq.so).

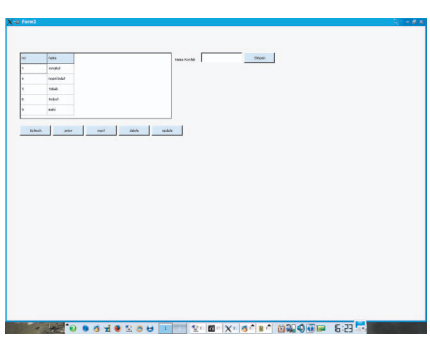
Solusinya sebenarnya mudah. Apabila Anda tidak memiliki libpq.so.2, copy-kan saja libpq.so.3 (bisa dari server remote atau dari server PostgreSQL lokal Anda, umumnya di /usr/lib) ke direktori binary kylix (umumnya di ~/kylix3/bin)

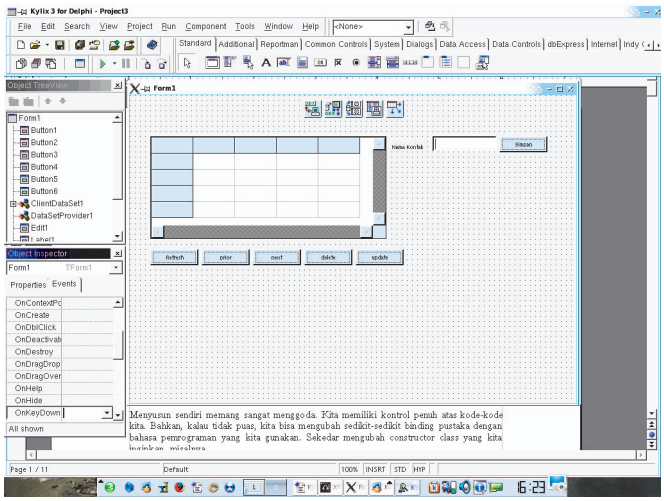
Kemudian, berikan perintah berikut ini untuk menguji ketergantungan libsqlpg.so:

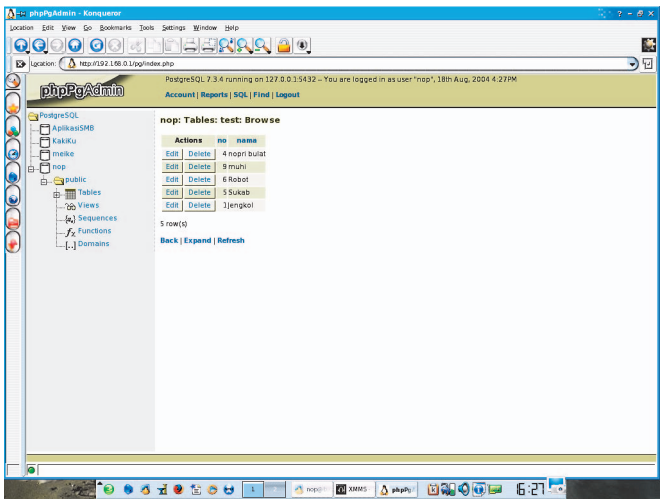
```
ldd libsqlpg.so
     linux-gate.so.1 =>
(0xffffe000)
     libpq.so.2 => not found
     libdl.so.2 => /lib/lib
dl.so.2 (0x40035000)
     libm.so.6 => /lib/tls/
```


Project1.


Project2.


Project3.


Kylix untuk Delphi.


Phppgadmin.

```
libm.so.6 (0x40039000)
     libc.so.6 => /lib/tls/
libc.so.6 (0x4005b000)
     /lib/ld-linux.so.2 =>
/lib/ld-linux.so.2 (0x80000000)
```

Anda bisa melihat betapa nakalnya libsqlpg.so.

Karena Anda sudah memiliki libpq.so.3, maka apa yang perlu Anda lakukan adalah membuat link file dari libpq.so.3 ke libpq.so.2 dan libpq.so dengan perintah berikut ini:

```
ln -sf libpq.so.3 libpq.so.2
ln -sf libpq.so.3 libpq.so
```

Kemudian, uji kembali libsqlpg.so dan pastikan Anda tidak menjumpai tulisan *not found* lagi. Berikut ini adalah keluaran di komputer penulis:

```
ldd libsqlpg.so
     linux-gate.so.1 =>
(0xffffe000)
     libpq.so.2 => libpq.so.2
(0x40024000)
     libdl.so.2 => /lib/libdl.
so.2 (0x4004a000)
     libm.so.6 => /lib/tls/libm.
so.6 (0x4004e000)
     libc.so.6 => /lib/tls/libc.
so.6 (0x40070000)
     libssl.so.0.9.7 => /usr/
lib/libssl.so.0.9.7 (0x40185000)
     libcrypto.so.0.9.7 =>
/usr/lib/libcrypto.so.0.9.7
(0x401b5000)
     libcrypt.so.1 => /lib/
libcrypt.so.1 (0x402a5000)
     libresolv.so.2 => /lib/
libresolv.so.2 (0x402d6000)
     libnsl.so.1 => /lib/
libnsl.so.1 (0x402e9000)
     /lib/ld-linux.so.2 =>
/lib/ld-linux.so.2 (0x80000000)
```

Kini, Anda sebenarnya sudah siap untuk melakukan koneksi ke PostgreSQL dari Kylix. Hanya, perhatikan juga apakah libpq.so.2 dan libpq.so terdapat dalam library path Anda. Sebagian besar dari Anda akan mengatur variabel sistem LD_LIBRARY_PATH untuk ikut mengarah ke direktori binary kylix3 (umumnya ~/kylix3/bin) karena pustaka runtime aplikasi Anda disimpan di direktori tersebut. Sayangnya, apabila psql di sistem Anda harus membutuhkan libpq.so.3, maka Anda harus mengatur pencarian lokasi pustaka, atau kalau tidak ingin melakukan hal tersebut, Anda harus mengunset LD_LIBRARY_PATH setiap kali ingin menjalankan psql.

Hal ini seharusnya tidak diperlukan apabila Anda menggunakan pgacces ataupun phppgadmin.

Untuk percobaan koneksi, bukalah Kylix (Delphi) dan lakukan langkah-langkah berikut ini:

- Buka proyek baru, simpan dengan nama default sebagai Project1.
- Masukkan komponen SQL Connection di Form1. Komponen ini didapat dari tab dbExpress (komponen paling kiri). Karena SQLConnection adalah non visual component, maka Anda bisa meletakkannya di mana saja. Aturlah nilai *LoginPrompt* menjadi *False.*
- Klik ganda komponen tersebut. Sebuah dialog akan terbuka.
- Pilihlah PostgreSQL Connection dan aturlah nilai yang benar pada *Connection Settings.*
- Klik tombol *Test Connection* (tombol kedua dari kanan di deretan tombol bagian atas dialog). Apabila tertulis *Successfully Connected,* maka koneksi berhasil. Apabila gagal, ulangi langkah-langkah sebelumnya ketika kita memberikan link dari libpq.so.3 ke libpq.so.2 dan libpq.so.

Simpanlah Project1 Anda. Kita akan menggunakannya untuk contoh-contoh aplikasi kita selanjutnya.

Contoh 1: Penggunaan DBGrid

Contoh pertama kita adalah aplikasi yang dapat menampilkan data tabel test dalam sebuah grid. Kita juga dapat menambahkan data, atau menghapus data. Kita juga dapat melakukan navigasi sederhana di dalam grid tersebut. Dan hebatnya, kita hanya perlu menuliskan satu atau dua baris kode. Begitu mudah, begitu cepat!

Lakukan langkah-langkah berikut ini:

- Bukalah Project1 sebelumnya. Anda telah memiliki komponen TSQLConnection di Form1 Anda. Aturlah agar peletakan komponen ini tidak menghalangi komponen lain di dalam *form.*

- Siapkan komponen TSQLDataSet yang dapat diambil dari tab dbExpress (berada di sebelah kanan TSQLConnection). Berikan nama default SQLDataSet1. Aturlah nilai properti SQLConnection menjadi SQLConnection1. Kemudian, berikan perintah berikut ini sebagai nilai properti CommandText: "Select * from test". Terakhir, ubahlah nilai properti Active menjadi True. Anda telah terhubung dengan database server pada database nop.
- Siapkan komponen TdataSetProvider yang dapat diambil dari tab Data Access (komponen ketiga dari kiri). Berikan nama default DataSetProvider1. Ubahlah nilai properti DataSet menjadi SQLDataSet1.
- Siapkan komponen TclientDataSet yang dapat diambil dari tab Data Access (komponen kedua dari kiri). Berikan nama default ClientDataSet1. Ubahlah nilai properti ProviderName menjadi DataSetProvider1 dan nilai properti Active menjadi True.
- Siapkan komponen TdataSource yang dapat diambil dari tab Data Access (komponen paling kiri). Berikan nama default DataSource1. Ubahlah nilai properti DataSet menjadi ClientDataSet1.
- Semua komponen nonvisual telah kita letakkan dan persiapan koneksi telah selesai kita lakukan. Kini, kita akan berhubungan dengan komponen visual. Pindahlah ke tab Data Controls dan tariklah sebuah TDBGrid (komponen paling kiri) ke dalam form. Berikan nama default DBGrid1. Aturlah agar peletakannya enak dipandang. Kemudian, ubahlah nilai properti DataSource menjadi DataSource1. Anda akan melihat isi dari DBGrid1 menjadi isi dari tabel test berkat perintah "select * from test".
- Kita akan menambahkan sebuah navigator ke dalam form kita. Masih di tab Data Controls, tariklah TDBNavigator (komponen kedua dari kiri) ke dalam form. Berikan nama default DBNavigator1. Ubahlah nilai properti DataSource menjadi DataSource1.
- Selesai. Simpan Project1 dan jalankanlah dengan menekan tombol F9. Anda telah selesai membuat aplikasi tanpa menulis satu baris kode pun.

Kini, kita akan mencoba Project1 kita tersebut. Anda bisa menambahkan data, mengubah data atau mengurangi data, atau melakukan navigasi sederhana. Hanya, satu kekurangan yang ada yaitu tidak otomatisnya update pada database.

Agar setiap perubahan langsung dikirim dan diberlakukan, tariklah sebuah button ke dalam form1 dan berikan kode-kode berikut ini pada event OnClick:

```
if ClientDataSet1.State in
[dsEdit, dsInsert] then Client
DataSet1.Post;
ClientDataSet1.ApplyUpdates(-1);
```

Sebagai sentuhan terakhir, perindahlah tampilan form Anda. Berikut ini adalah beberapa hal yang dapat dilakukan:

- Ubahlah caption Button1 menjadi '&Update' Misalnya.
- Ubahlah agar Dbgrid dapat dipilih per baris dengan mengubah properti Options.dgRowSelect menjadi True. Namun, hal ini akan menyusahkan pemasukan data baru.
- Anda dapat menampilkan button yang hanya Anda inginkan pada DBNavigator1 dengan mengubah properti VisibleButtons.
- Anda dapat mengubah hint pada tombol-tombol DBNavigator1 menjadi bahasa Indonesia dengan mengubah properti Hints.

Satu hal yang cukup menarik untuk diperhatikan. Umumnya, nama field dibuat sedemikian rupa agar ringkas. Namun bagi user, hal yang satu ini malah merugikan. Ada baiknya, kita mengubah field nama misalnya, menjadi Nama Kontak dan no menjadi Nomor. Berikanlah kode-kode berikut ini pada event OnShow Form1:

```
ClientDataSet1.
FieldByName('no').DisplayLabel
:= 'Nomor';
ClientDataSet1.
FieldByName('nama').DisplayLabel
:= 'Nama Kontak';
```

Sekarang, masih dalam tahapan memperindah aplikasi, cobalah menekan kombinasi tombol CTRL-DEL di dalam DBGrid. Apa yang terjadi? Ya. Sebuah konfirmasi penghapusan data akan ditampilkan. Bagaimanakah kalau kita ingin mencegah penghapusan data dengan cara demikian? Bloklah penekanan tombol tersebut. Berikan kode-kode berikut untuk menangani event OnKeyDown Dbgrid1:

```
if (key = key_delete) and
(ssCtrl in Shift) then key := 0;
```

Harap diperhatikan, untuk dapat mengompile kode tersebut (konstanta key_delete), Anda harus menggunakan unit tambahan, yaitu unit Qt. Tambahan pada bagian atas unit Anda.

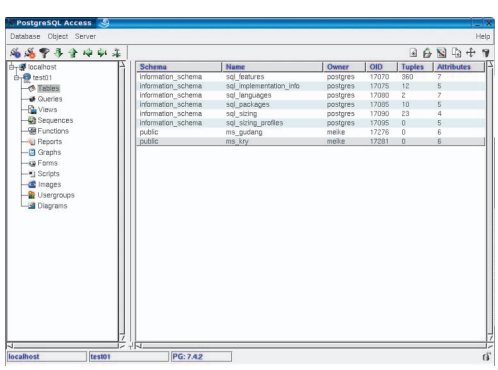
Demikianlah contoh pertama kita. Simpanlah Project1. Kita akan segera melanjutkan ke contoh berikutnya.

Contoh 2: Penggunaan DB Control Lain

Bukalah Project baru dan simpanlah dengan nama default Project2. Di Project2 ini, kita akan melihat lebih lanjut berbagai DB Control lain. Kita masih akan menggunakan table test database nop. Apabila di contoh pertama kita menggunakan DBGrid untuk segala-galanya, kali ini kita akan menggunakan komponen-komponen lainnya.

Lakukan langkah-langkah berikut ini:

- Seperti biasa, kita akan meletakkan berbagai komponen agar koneksi ke PostgreSQL dapat dilakukan.
- Tambahkan komponen Tlabel ke form dan ganti captionnya menjadi Nomor. Tambahkan sekali lagi Tlabel dan ubah captionnya menjadi Nama Kontak. Posisikan vertikal.
- Tambahkan komponen TDBText di sebelah kanan Label dengan caption Nomor dan TDBEdit di sebelah kanan Label dengan caption Nama Kontak. Berikan keduanya nama default Dbtext1



pgaccess.

dan Dbedit1. Ubahlah properti DataSource Dbtext1 menjadi DataSource1 dan properti DataField menjadi no. Kemudian, Ubahlah properti DataSource DbEdit1 menjadi DataSource1 dan properti DataField menjadi nama.

- Tambahkan komponen TDBNavigator ke dalam form dan aturlah sedemikan sehingga button yang aktif hanyalah button first, prior , last, next, insert, dan delete.
- Tambahkan pula button Update seperti contoh sebelumnya. Berikan kode yang sama untuk menangani event OnClick:

```
if ClientDataSet1.State in
[dsEdit, dsInsert] then Cli
entDataSet1.Post;
ClientDataSet1.ApplyUpdates(-1);
```

- Kini, data Anda tidak lagi ditampilkan tabular lagi.

Contoh kedua ini siap dijalankan tanpa penambahan kode-kode lainnya. Contoh ini ingin memperlihatkan integrasi antara setiap komponen db control. Pembuatan program akan menjadi sangat mudah.

Ada sebuah lelucon tentang Delphi. Anda menjalankan Delphi dan tidak tahu apapun soal pemrograman. Anda membaca tutorial langkah demi langkah untuk membuat aplikasi database. Setelah aplikasi selesai, dan Anda dapat menggunakannya, Anda tetap tidak mengerti apa-apa.

Pada dasarnya, Delphi sangat memanjakan usernya. Namun, seperti yang kita ketahui, manja berlebihan tidaklah baik. Dalam kedua contoh sebelumnya, Anda sebenarnya bisa menggunakan komponen-komponen biasa untuk menampilkan data dari database. Hanya, lebih susah.

Pada contoh ketiga, kita akan melihat contoh bagaimana menampilkan data menggunakan komponen-komponen biasa. Tanpa menggunakan komponen database control.

Contoh 3:

Contoh 1 dengan komponen biasa

Contoh ketiga ini adalah pembuatan ulang contoh pertama dengan komponen-komponen biasa, yang tidak database-aware. Simpanlah dengan nama Project3.

Lakukan langkah-langkah berikut ini:

- Seperti biasa, kita akan meletakkan berbagai komponen agar koneksi ke PostgreSQL dapat dilakukan.
- Tambahkan StringGrid ke dalam Form1. Berikan nama default StringGrid1.
- Tambahkan sebuah button, berikan nama default Button1, ganti captionnya menjadi 'Refresh'. Berikut ini adalah prosedur lengkap untuk menangani event OnClick:

```
procedure TForm1.
Button1Click(Sender: TObject);
var i : integer;
begin
SQLDataSet1.Close;
ClientDataSet1.Close;
SQLDataSet1.CommandText :=
'Select * from test  order by
no';
SQLDataSet1.Open;
ClientDataSet1.Open;

StringGrid1.ScrollBars :=
ssAutoBoth;
StringGrid1.FixedCols := 0;
StringGrid1.FixedRows := 1;
StringGrid1.ColCount := Data
Source1.DataSet.FieldCount;
StringGrid1.RowCount :=
DataSource1.DataSet.Record
Count+1;


for i:= 0 to DataSource1.
DataSet.FieldCount -1 do
Begin
        StringGrid1.Cells[i,0]
:= DataSource1.DataSet.Field
Defs.Items[i].Name;
End;

i := 1;
while not DataSource1.Data
Set.Eof do
Begin
        StringGrid1.Cells[0,i]
:= DataSource1.DataSet.
Fields[0].AsString;
        StringGrid1.Cells[1,i]
:= DataSource1.DataSet.
Fields[1].AsString;
        DataSource1.DataSet.
Next;
        inc(i);
End;
end;
```

Pertama-tama, kita menutup dan membuka kembali dataset yang kita miliki. Setelah itu, kita menentukan jumlah baris dan kolom StringGrid1. Setelah itu, kita mengisikan nama field ke kolom 0 baris 0 dan kolom1 baris 0 StringGrid1. Dan, yang terakhir, kita mengisikan isi setiap *record* ke dalam StringGrid1.

Bisa kita lihat, untuk mendapatkan hasil seperti ini, kode yang diperlukan sebenarnya cukup panjang (menurut ukuran Delphi). Namun, dengan memahami kode-kode seperti ini, kita setidaknya bisa menguasai kode-kode kita sendiri untuk membuat aplikasi database. Membuat aplikasi database tidak harus selalu menggunakan komponen database control.

- Berikutnya, ketika kali pertama form1 ditampilkan (event OnShow), jalankan prosedur sebelumnya. Berikut ini adalah kode untuk menangani event OnShow tersebut:

```
Button1.Click;
```

Dengan demikian, pada saat pertama kali form1 ditampilkan, kita sudah bisa menikmati tampilan data terbaru tanpa harus klik pada tombol *Refresh* terlebih dahulu.

- Selanjutnya, kita akan memasukkan data ke dalam tabel test dengan komponen non-database control. Tariklah satu TLabel ke sebelah kanan StringGrid1 dan posisikan vertikal. Berikan caption 'Nama Kontak'.
- Letakkan komponen Tedit, dengan nama edit1 ke sebelah kanan label 'Nama Kontak'. Kosongkan properti Text. Edit1 akan menampung nama yang akan kita masukkan ke tabel test. Kita tidak perlu memasukkan no sendiri karena *field no* bertipe serial (*auto increment*).
- Di sebelah kanan edit1, tariklah sebuah button, beri nama default button2, dan ubah captionnya menjadi 'Simpan'.
- Tuliskan kode-kode berikut untuk menangani event OnClick Button2:

```
SQLDataSet1.Close;
ClientDataSet1.Close;
SQLDataSet1.CommandText :=
```

```
'insert into test (nama)
values(:nama)';
SQLDataSet1.Params[0].AsString
:= Edit1.Text;
SQLDataSet1.ExecSQL;
Button1.Click;
```

Sebagai catatan, kita perlu langsung memasukkan isi edit1.text ke dalam CommandText SQLdataSet1. Sebagai gantinya, kita menggunakan parameter :nama, yang nantinya akan digantikan dengan Edit1.Text.

- Berikutnya, kita akan membuat navigator sederhana yang bisa melakukan navigasi *prior* dan *next*. Tariklah dua Button ke bawah StringGrid1 dan berilah nama default masing-masing Button3 dan Button4. Ganti Caption Button3 menjadi 'prior' dan Caption Button4 menjadi 'next'.
- Isikan kode-kode berikut ini untuk menangani event OnClick Button3:

```
if StringGrid1.Row > 1 then
StringGrid1.Row := StringGrid1.
Row - 1;
```

- Isikan kode-kode berikut ini untuk menangani event OnClick Button4:

```
if StringGrid1.Row < String
Grid1.RowCount - 1  then String
Grid1.Row := StringGrid1.Row +
1;
```

- Berikutnya, kita akan membuat fasilitas penghapusan data. Tariklah sebuah Button ke form1. Beri nama default Button5. Gantilah Captionnya menjadi 'Delete'. Berikan kode-kode berikut ini untuk menangani event OnClick Button5:

```
SQLDataSet1.Close;
ClientDataSet1.Close;
SQLDataSet1.CommandText :=
'Delete from test where no=:no';
SQLDataSet1.Params[0].AsString
:= StringGrid1.Cells[0, String
Grid1.Row];;
SQLDataSet1.ExecSQL;
Button1.Click;
```

Sebagai catatan, kita akan menggunakan field no sebagai penanda record yang akan dihapus. Oleh karena itu, dimanapun kursor berada di dalam StringGrid1, kita selalu mengambil informasi kolom 0, yang digunakan untuk menampung isi field no.

- Selanjutnya, giliran fasilitas update data yang akan kita buat. Cara untuk melakukan update adalah dengan mengubah langsung data di dalam StringGrid. Tariklah sebuah Button ke dalam form1 dan posisika di bawah StringGrid1. Beri nama default Button6 dan gantilah Captionnya menjadi 'Update'. Berikut ini adalah kode-kode untuk menangani event OnClick Button6:

```
if StringGrid1.Col  = 1 then
begin
        StringGrid1.Options :=
[goFixedVertLine, goFixedHor
zLine, goVertLine, goHorzLine,
goRangeSelect, goEditing];
end;
```

Sebagai catatan, kita perlu memeriksa kolom aktif karena kita hanya mengizinkan update untuk field nama. Hanya ketika kursor pada kolom 1 lah kita akan menambahkan opsi goEditing pada StringGrid1 sehingga memungkinkan editing. Hal yang seperti ini harus diperhatikan untuk memastikan kita tidak memperbolehkan user mengisikan nilai atau perubahan pada kolom yang tidak diizinkan. Pendekatan lain untuk pengisian pada *stringgrid* apabila Anda tidak menyukai pengisian langsung seperti ini adalah penggunaan *inputbox*. Caranya, periksalah tombol keyboard yang ditekan. Apabila tombol ENTER ditekan, tampilkanlah InputBox.

- Fasilitas *update* belum selesai. Kita perlu memroses event OnKeyDown milik StringGrid1. Berikut ini adalah kode-kodenya:

```
if key = key_return then
Begin
        StringGrid1.Options :=
[goFixedVertLine, goFixedHor
zLine, goVertLine, goHorzLine,
goRangeSelect];
        SQLDataSet1.Close;
        ClientDataSet1.Close;
        SQLDataSet1.CommandText
:= 'update test set nama=:nama
where no=:no';
        SQLDataSet1.Params[0].
AsString := StringGrid1.
Cells[1,StringGrid1.Row];
        SQLDataSet1.Params[1].
AsString := StringGrid1.
Cells[0,StringGrid1.Row];
        SQLDataSet1.ExecSQL;
        Button1.Click;
End;
```

Jangan lupa tambahkan Qt sebagai unit yang digunakan (penggunaan konstanta key_return) pada bagian atas unit kita. Ketika user menekan tombol ENTER, kita akan segera memproses perubahan yang dilakukan dan mengirimkannya ke database server. Kita juga tidak lupa untuk segera menonaktifkan opsi editing pada StringGrid1 sehingga pengeditan lebih lanjut tidak lagi dimungkinkan.

Sampai di sini, kita sudah selesai membuat fungsi sendiri untuk fasilitas-fasilitas yang sediakan oleh Delphi seperti pada Contoh 1. Apabila pada contoh 1 kita hanya mengetikkan satu atau dua baris, maka di contoh 3 ini, kita mengetikkan lebih dari 50 baris. Jauh berbeda. Namun, cobalah perhatikan ukuran *executable*-nya. Dengan lebih banyak komponen, kita mendapatkan ukuran yang lebih kecil.

Dan, yang pasti, pemahaman kita akan jauh lebih meningkat, dibandingkan ketika kita meminta Delphi untuk melakukan segalanya untuk kita.

Bagaimana menurut Anda untuk contoh ketiga ini? Sejujurnya, cara seperti ini mungkin belum cara yang dapat dikategorikan sebagai cara *low level*, masih banyak lagi cara yang lebih low level, namun jelas lebih susah dan sayangnya, Delphi akan kehilangan kekuatannya apabila kita menggunakan cara yang sangat low level.

Sampai di sini dulu perkenalan kita dengan Kylix dan Database. Di edisi-edisi mendatang, kita akan terus membahas bagaimana meningkatkan usability Linux, dari berbagai bidang, termasuk pemrograman.

Karena, agar Linux digunakan lebih meluas, kita harus menjadikan Linux sebagai lingkungan yang cukup ramah dan produktif. Kita serap hal bagus yang dimiliki oleh Windows atau Mac OS dan kita terapkan di Linux. Setuju?

Noprianto *(noprianto@infolinux.co.id)*