

Membuat Aplikasi Bisnis

Menggunakan bahasa Python dan
database berbasis SQL

Oleh:

Owo Sugiana <sugiana@rab.co.id>

JAKARTA

29 September 2002 - 7 Februari 2003

Daftar Isi

I	Pendahuluan	15
1	Pemrograman Komputer	17
1.1	Mengapa Kita Butuh Program	17
1.2	Bahasa Pemrograman	18
1.3	Siklus Pengembangan Program	18
2	Bahasa Pemrograman	21
2.1	Mengapa Python	21
2.2	Nama Python	23
2.3	Pemrograman Terstruktur	23
2.4	Dokumentasi	24
3	Teknik Penulisan	27
3.1	Gaya	27
3.2	Jenis Huruf dan Simbol	27

4	Persiapan	29
4.1	Paket Program	29
4.2	Text Editor	29
II	Python	31
5	Hello World	33
5.1	Aturan Penulisan	33
5.1.1	Indent	34
5.1.2	Baris Perintah	34
5.1.3	Keterangan Program	35
5.2	Variabel	35
5.3	Modus Interaktif	35
6	Tipe Data	37
6.1	Bilangan	37
6.1.1	Operator	38
6.1.2	Pengelompokan Operasi	38
6.1.3	Pembulatan	40
6.1.4	Bentuk Tampilan	40
6.2	String	41
6.2.1	Penjumlahan & Perkalian	42
6.2.2	String Format	42
6.2.3	String Menjadi Integer - <code>int()</code>	43
6.2.4	String Menjadi Float - <code>float()</code>	43
6.2.5	Panjang String - <code>len()</code>	43
6.3	List	43
6.3.1	Penambahan - <code>append()</code> & <code>insert()</code>	44

6.3.2	Penghapusan - <code>del</code>	44
6.3.3	String Sebagai List	45
6.3.4	Pemenggalan	45
6.3.5	Panjang List - <code>len()</code>	45
6.3.6	List Dalam List	45
6.3.7	Membalikkan Urutan - <code>reverse()</code>	47
6.3.8	Mengurutkan - <code>sort()</code>	47
6.3.9	Menyalin - <code>list()</code>	48
6.4	Dictionary - Key & Value	48
6.4.1	Daftar Kunci - <code>keys()</code>	49
6.4.2	Daftar Nilai - <code>values()</code>	49
6.4.3	Menambah Atau Mengubah Nilai - <code>update()</code>	49
6.4.4	Menghapus - <code>del</code>	49
6.5	Mengetahui Tipe Data - <code>type()</code>	50
6.6	Daftar Fungsi Suatu Objek - <code>dir()</code>	50
7	Kondisi	51
7.1	Bentuk Logika	52
7.2	Selain Itu - <code>else</code>	53
7.3	Selain Itu Jika - <code>elif</code>	53
7.4	Operator Perbandingan	54
7.5	Operator Logika	55
7.5.1	Bukan - <code>not</code>	55
7.5.2	Semua Kondisi Terpenuhi - <code>and</code>	55
7.5.3	Salah Satu Kondisi Terpenuhi - <code>or</code>	56
8	Perulangan - Loop	59
8.1	Jumlah Perulangan Ditetapkan - <code>for</code>	59
8.2	Selama - <code>while</code>	60

8.3	Keluar Dari Perulangan - <code>break</code>	62
9	Fungsi	63
9.1	Nilai Masukan	64
9.2	Nilai Keluaran - <code>return</code>	64
9.3	Memanggil Dirinya	65
9.4	Kepemilikan Variabel	65
9.5	Fungsi Interpreter - <code>exec()</code>	66
10	File	69
10.1	Baca Tulis	69
10.2	Printer	70
10.2.1	Ukuran Huruf	70
10.2.2	Ganti Halaman	70
10.2.3	Mencetak File	71
10.3	Direktori Aktif	72
11	Menangani Kesalahan - Exception	73
12	Proyek String	75
12.1	Membuat Nomor Baris	75
12.1.1	Awali Dengan Nol - <code>zfill()</code>	76
12.1.2	Penunjuk Pada File - <code>seek()</code>	76
12.2	File Sebagai Tabel	76
12.2.1	Membelah String - <code>splitfields()</code>	78
12.2.2	Hapus Karakter Tak Tampak - <code>strip()</code>	78
12.2.3	Rata Kiri dan Kanan - <code>ljust()</code> & <code>rjust()</code>	78
12.2.4	Kunci Pada Dictionary - <code>has_key()</code>	78

<i>DAFTAR ISI</i>	7
III Qt	79
13 Pendahuluan	81
14 Aplikasi Pertama	83
14.1 Berorientasi Objek	84
14.2 Program Utama	84
14.3 <code>self</code>	85
14.4 Fungsi Pada Objek	85
15 Visual Class	87
15.1 Buku Alamat	87
15.1.1 Parent dan Child	89
15.1.2 Parent dan Owner	89
15.1.3 Dengan Atau Tanpa <code>self</code>	90
15.1.4 Modul <code>qt</code>	90
15.1.5 String Atau <code>QString</code>	90
15.2 Sinyal	91
15.2.1 Keterkaitan Dengan <code>C++</code>	93
15.2.2 Sinyal atau Event	94
15.2.3 Membuat Event	95
15.3 Hiasan	96
15.3.1 Font - <code>QFont</code>	96
15.3.2 Warna - <code>QColor</code>	98
15.3.3 Parent Berpengaruh	99
15.4 Ya Atau Tidak - <code>QCheckBox</code>	99
15.5 Pilih Salah Satu - <code>QRadioButton</code>	101
15.6 Daftar Fluktuatif - <code>QComboBox</code>	105
15.7 Listbox	106

15.8	Widget Aktif - Enable	110
15.9	LCD	111
15.10	Hanya Keyboard	113
15.10.1	Tanpa Mouse	113
15.10.2	Tombol Keyboard	114
15.10.3	NumLock	117
16	Kasir I	119
17	Wadah - Container	123
17.1	Widget	123
17.2	Panel	123
17.3	Groupbox	123
17.4	Multigroup	127
18	Penataan	131
18.1	Fleksibilitas Ukuran	131
18.2	Fleksibilitas Posisi	133
18.3	Layout Dengan Metode Grid	136
19	Waktu	139
19.1	Jam	139
19.2	Tanggal - QDate	140
19.3	Tanggal dan Jam	142
19.4	Timer	143
20	Form Dialog	145
20.1	File Dialog	146
20.2	Pesan & Konfirmasi	149

20.3	Input	152
21	Tabel	155
21.1	Mengubah Sifat	157
21.2	Bentuk Tampilan	161
21.3	Form Pencarian	168
22	Kasir II	175
23	Database	179
23.1	Membuat Database	180
23.1.1	PostgreSQL	180
23.1.2	MySQL	181
23.2	Form Login	181
23.3	Membuat Tabel	185
23.4	Query	185
23.4.1	Current Record	189
23.4.2	Variant	190
23.5	Cara Lain Menangani Tabel	191
23.5.1	Browsing	192
23.5.2	Bentuk Tampilan	193
23.5.3	Pengganti Perintah SQL	196
23.5.4	NULL	197
23.5.5	Pengisian Data yang Lebih Nyaman	198
23.5.6	Urutkan - Sort	210
23.5.7	Nomor Urut - Autoincrement	213
24	Kasir III	217
24.1	Struktur Tabel	217

24.2	Daftar Barang	218
24.3	Penyimpanan Data	219
24.4	Pencetakan	219
24.5	Program	219
24.6	Laporan	225
24.6.1	Barang Terlaris	225
24.6.2	Total Penjualan Harian	226
24.6.3	Rata-rata Penjualan Harian	226
24.6.4	Jam Sibuk	226

Daftar Tabel

6.1	Operator Bilangan	39
6.2	Contoh pemenggalan list	46
7.1	Operator Perbandingan	54
7.2	Operator Logika	57

Daftar Gambar

1.1	Siklus Pengembangan Program (development cycle)	19
6.1	Nomor index pada list	45
14.1	Hello World !	84
15.1	Form Alamat	89
15.2	Checkbox	100
15.3	Radiobutton	103
15.4	Combobox	105
15.5	Listbox	107
15.6	Disable Widget	110
15.7	LCD	112
15.8	Tombol pintas (shortcut-key)	114
20.1	File Dialog	147
20.2	Text Editor	149

21.1	QTable	155
21.2	ValueGrid	161
23.1	Form Login	181
23.3	QDataTable	194
23.4	CursorTable	195
23.5	DBGrid	199
24.1	Kasir	218

Kata Pengantar

Pengembangan aplikasi bisnis di Indonesia sudah sangat berkembang dan memiliki potensi pasar yang masih sangat luas. Berbagai paket pemrograman - baik dari dalam maupun luar negeri - juga semakin memudahkan para programmer dalam membuatnya. Praktis kini pertumbuhan pengembang perangkat lunak (*software developer*) kian mengalami percepatan. Oleh karena itu dibutuhkan banyak tulisan - baik bersifat referensi maupun tutorial - yang dapat memenuhi kebutuhan para pengembang.

Kehadiran buku ini bermaksud untuk menumbuhkan minat bagi calon programmer atau sebagai referensi bagi para programmer lainnya agar dukungan lokal (*local support*) semakin tercapai terhadap perangkat lunak yang digunakan masyarakat.

Pembahasannya diupayakan melalui konsep terlebih dahulu, kemudian ulasan teknis pemrograman dibahas serinci mungkin. Banyaknya contoh program diharapkan semakin memudahkan pemahaman. Jadi selain membaca ulasan juga perhatikan alur programnya.

Secara teknis platform yang digunakan dalam contoh pro-

gram adalah berbasis Linux, bahasa Python, dengan database berbasis SQL yaitu PostgreSQL dan MySQL.

Pada bagian pertama, ulasan menyangkut bahasa Python beserta pustaka (*library*) standar yang dibawa dalam satu paket Python. Bagian ini mengulas hal-hal yang dijalankan pada modus teks (*console*).

Bagian keduanya berisi tentang aplikasi yang dijalankan dalam modus grafis (*graphical user interface*), yaitu menggunakan XWindow. Pustaka grafis untuk membentuk berbagai objek visual tidak disertakan dalam paket standar Python. Buku ini menggunakan Qt sebagai pustakanya.

Karena merupakan tutorial, maka ulasan bab demi bab dirangkai dengan tingkat keterkaitan yang tinggi. Bagi para pembaca sebaiknya membaca secara runut guna memudahkan pemahaman dalam setiap penjelasan.

Sebagai pelengkap, buku ini juga menyertakan beberapa pertanyaan dan latihan soal guna menggali potensi yang sudah dimiliki pembaca.

Ucapan terima kasih perlu ditujukan kepada pihak terkait yang sudah membantu kelancaran penyelesaian buku ini. Kepada keluarga penulis, Henry (NCS), Mahendra Putra, dan Henry (Bina Nusantara). Juga kepada majalah Infolinux yang telah memberikan bonus CD RedHat 7.3 di salah satu edisinya, yang praktis berperan memberikan kemudahan bagi para pembaca untuk memperoleh sumber daya yang dibutuhkan saat menjalankan contoh program dalam buku ini. Seluruh kebutuhan dalam menjalankan contoh program sudah tersedia dalam CD tersebut. Tidak lupa juga kepada Bapak I Made Wiryana (Gunadarma) yang bersedia memenuhi undangan penulis untuk di-

posisikan sebagai editor.

Sekali lagi, mudah-mudahan kehadiran buku ini di tengah masyarakat semakin menyemarakkan pasar IT (*information technology*) di Indonesia dan memperluas kesempatan kerja bagi masyarakat pada umumnya. Setidaknya melengkapi rak-rak perpustakaan kampus, kantor, rumah, dan perpustakaan lainnya agar dibaca siapa saja yang bersungguh-sungguh menumbuhkan IT di Indonesia.

Penulis

Bagian I

Pendahuluan

Bab 1

Pemrograman Komputer

Bagi mereka yang belum memahami dunia komputer, khususnya dunia pemrograman, terkadang masih dihadapkan pada pertanyaan tentang alasan-alasan kita membuat program, apa manfaat yang bisa diambil, bagaimana kelanjutannya, dan bagaimana arah teknologi komputer ke depan sehingga investasi waktu dan lainnya dalam mempelajari pemrograman tidak sia-sia.

Berikut ini sedikit ulasan seputar dunia pemrograman komputer agar memantapkan motivasi dan memudahkan pemahaman mengapa “suatu keputusan” telah diambil.

1.1 Mengapa Kita Butuh Program

Saat ini ada ribuan aplikasi yang siap pakai untuk berbagai keperluan. Mungkin Anda bertanya-tanya mengapa kita harus membuat program ? Bukankah program yang sudah ada dapat kita gunakan ?

Sebagaimana tujuan pembuatan komputer itu sendiri, program dibuat untuk:

- mendapatkan jawaban atas suatu masalah
- memperpendek langkah penyelesaian suatu masalah

Lalu mengapa kita perlu membuatnya ? Ya, karena:

- belum ada program yang dapat menyelesaikan permasalahan yang dihadapi
- program yang ada terlalu mahal
- program yang ada terlalu “berat” dimana spesifikasi komputer yang ada miliki tidak mampu untuk menjalankannya
- hanya untuk kesenangan

Apa yang kita lakukan dengan komputer untuk menyelesaikan masalah yang dihadapi ?

- Menggunakan program yang sudah ada
- Menggabungkan program yang sudah ada untuk membentuk suatu fungsi
- Membuat program baru

1.2 Bahasa Pemrograman

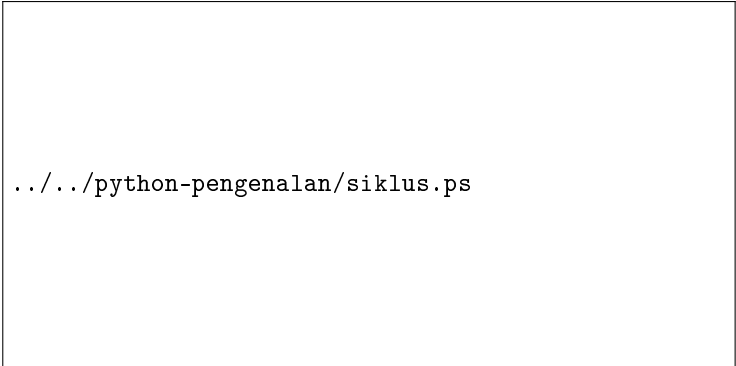
Bahasa pemrograman merupakan cikal bakal suatu program atau aplikasi komputer. Dengannya Anda bisa merangkai perintah-perintah yang sudah ditetapkan untuk membentuk suatu fungsi yang diinginkan. Ada begitu banyak bahasa pemrograman baik yang digunakan untuk kebutuhan khusus atau juga untuk penyelesaian masalah secara umum.

Bahasa pemrograman untuk kebutuhan khusus misalnya SQL (*structured query language*) yang khusus dibuat untuk penanganan database atau OpenGL yang diperuntukkan untuk membuat grafik.

Python sebagaimana C, C++, Pascal, atau Java, dibuat untuk membuat program dengan berbagai keperluan. Tidak hanya terbatas pada masalah perhitungan matematika, tapi juga dapat digunakan untuk menangani database, grafik, dan bahkan untuk membuat game.

1.3 Siklus Pengembangan Program

Development cycle atau siklus pengembangan program merupakan hal penting pemilihan suatu bahasa pemrograman. Pemrograman tradisional membutuhkan siklus pengembangan yang lebih lama karena harus melalui proses kompilasi dan *linking* ke sistem operasi. Dengan Python dua proses tersebut tidak dilakukan (lihat gambar 1.1).



```
../../python-pengenalan/siklus.ps
```

Gambar 1.1: Siklus Pengembangan Program (development cycle)

Bab 2

Bahasa Pemrograman

Ada banyak bahasa pemrograman yang sudah dibuat sejak diciptakannya komputer pertama kali. Bahasa-bahasa tersebut ada yang tergolong untuk pembuatan aplikasi umum, namun ada juga yang memang dirancang untuk suatu aplikasi tertentu.

2.1 Mengapa Python

Python dapat dijalankan di berbagai sistem operasi seperti Linux, Unix, dan juga Windows. Pengurangan *source* program secara besar-besaran juga merupakan tujuan dibuatnya bahasa ini. Adapun perbandingan Python dengan bahasa lain bisa dilihat di situs internet:

<http://www.python.org/doc/Comparisons.html>

<http://www.python.org/doc/essays/comparisons.html>

<http://www.sunworld.com/swol-10-1997/swol-10-scripting.html>

Atau untuk yang bergaya humor ada di:

<http://www.python.org/doc/Humor.html#vowels>

Mungkin bahasa pemrograman ini belum terdengar secara meluas di kalangan programmer Indonesia. Python pertama kali dikembangkan oleh Guido van Rossum pada tahun 1990 di CWI, Belanda. Bahasa ini dikategorikan sebagai bahasa pemrograman tingkat tinggi (*very-high-level language*) dan juga merupakan bahasa berorientasi objek yang dinamis (*object-oriented dynamic language*). Python bukan hanya “sekedar bahasa lain” untuk membuat aplikasi, tapi merupakan sebuah bahasa **jenis baru**. Secara umum Python menawarkan:

- Berorientasi objek
- Struktur pemrograman yang handal
- Arsitektur yang dapat dikembangkan (*extensible*) dan ditanam (*embeddable*) dalam bahasa lain
- Sintaks yang mudah dibaca

Sebagai contoh, ciri orientasi objeknya membuat Python dapat digabungkan dengan modul lain yang dibuat dengan C++. Sebagai *tools* yang berdiri sendiri, Python sudah dipakai untuk *system administrator tools*,¹ antarmuka grafis, *script* internet, dan pemrograman database.

¹Di RedHat Linux program instalasinya ditulis dengan Python

Python sering dibandingkan dengan bahasa lain seperti: Tcl, Perl, atau Java.

Tcl

Seperti Tcl, Python dapat digunakan sebagai bahasa yang digabungkan dengan bahasa lainnya. Tidak seperti Tcl, Python memiliki ciri bahasa pemrograman yang sesungguhnya. Secara umum, struktur datanya dan dukungan untuk pemrograman berskala besar membuatnya dapat diterapkan untuk ruang lingkup yang lebih besar. Tcl dirancang sebagai bahasa interpreter yang dapat digabungkan dengan modul yang dibuat dengan C, sedangkan Python dapat melakukan hal yang sama namun ditambah kemampuan orientasi objek, bukan sekedar *string processor*.

Perl

Seperti Perl, Python dapat digunakan untuk membuat *shell tools*. Tidak seperti Perl, Python ringkas dan lebih mudah dibaca. Bagi sebagian orang hal ini membuat Python lebih mudah digunakan dan pilihan yang lebih tepat untuk membuat program yang dapat dituliskan atau di-*maintenance* oleh pihak lain. Tanpa banyak tanya, Perl merupakan *tools* yang handal untuk sistem administrasi. Namun sekali kita berniat untuk melakukan lebih dari sekedar pengolahan teks dan file, kemampuan Python sangat menggoda.

Java

Secara umum program Python memang lebih lambat ketimbang Java², tapi waktu yang diperlukan untuk membuatnya justru lebih cepat. Program Python tiga sampai lima kali lebih ringkas dibandingkan Java. Contohnya: Python tidak memerlukan deklarasi tipe data untuk suatu variabel, elemen *array*³ yang tipenya bisa beragam, dan dukungannya dengan apa yang disebut *dictionary*.⁴ Karena pemberian tipe data dilakukan pada saat *runtime*, program Python berjalan lebih lambat ketimbang Java. Contoh: ketika ekspresi $a+b$ dievaluasi, Python memeriksa tipe objek a dan b , yang sebenarnya tidak diketahui pada saat “kompilasi”. Java bisa menerapkan tipe *integer* dan *float* secara lebih efisien, namun membutuhkan deklarasi untuk a dan b .

2.2 Nama Python

Sekedar info, penamaan Python bukanlah dikaitkan dengan reptil yang biasa kita kenal, melainkan diberikan setelah pembuatnya menonton acara BBC “Monty Python’s Flying Circus”.

²Karena sifatnya yang *late-binding*, yaitu kode diproses ketika diperlukan.

³Python menyebutnya list

⁴Bisa juga disebut associative array, array yang memiliki kunci (key). Lihat halaman 48.

2.3 Pemrograman Terstruktur

Pembuatan program yang terstruktur merupakan tujuan dari Python. Hal ini dapat dilihat dari sifat Python itu sendiri seperti:

1. Tidak ada fasilitas *loncat ke baris tertentu*, sebagaimana yang bisa ditemukan dalam GOTO pada pemrograman Basic. Sebagai gantinya Anda dapat menerapkan fungsi agar program lebih mudah dibaca dan lebih sistematis.
2. Memperhatikan tipe data dalam setiap operasinya.
3. Dukungnya akan pemrograman berorientasi objek membuat Python dapat dipakai untuk mengembangkan aplikasi yang kompleks namun tetap konsisten.

Untuk sebuah hasil yang sama dapat ditempuh dengan berbagai cara (baca: algoritma). Prioritas berikut bisa dijadikan pedoman dalam menuliskan program:

Efisiensi Algoritma harus ringkas dan padat. Efisiensi juga mengarah pada cara penyimpanan data: semakin kecil semakin baik.

Kecepatan Program yang efisien biasanya berpengaruh pada kecepatan namun bisa juga sebaliknya. Penerapan rumus matematika dalam menyelesaikan suatu masalah bisa membuat efisiensi dalam penyimpanan data karena bisa menjadi lebih kecil, misalnya yang terjadi pada penyimpanan format gambar JPEG yang

lebih kecil dari BMP. JPEG sangat baik untuk faktor efisiensi tempat (*storage*), namun prosesor harus bekerja lebih keras karena proses penerjemahannya (dari data ke gambar) menerapkan rumus matematika yang rumit. BMP sangat *simple* dan prosesor tidak terlalu terbebani, namun membutuhkan alokasi memori yang cukup besar.

Modular Pilah permasalahan menjadi beberapa bagian agar mudah dalam menyelesaikannya.

Pengembangan Program atau algoritma yang baik harus mudah dikembangkan. *Reuseable code*⁵ merupakan kata yang akan dicapai. Anda mungkin sudah membuat algoritma yang ditulis dalam bahasa pemrograman tertentu. Bila Anda menuangkannya secara tepat, maka seharusnya dengan mudah Anda menuliskannya kembali untuk bahasa pemrograman yang lain.

2.4 Dokumentasi

Biasakan membuat dokumentasi dalam aktivitas pemrograman Anda. Hal ini berkaitan erat dengan kecepatan penyelesaian suatu aplikasi dengan manfaat:

1. Mempercepat proses debugging⁶

⁵Penggunaan kembali kode yang sama.

⁶Debugging: aktivitas dalam mencari kesalahan suatu program

2. Memudahkan pengembangan di kemudian hari.
3. Memudahkan orang lain untuk mempelajarinya sehingga Anda tidak perlu mengutarakannya berulang-ulang.

Dokumentasi bisa dalam berbagai bentuk tergantung “*audience*” yang Anda tuju, seperti:

1. Programmer: dokumentasi bisa berada dalam *script*⁷ yang Anda buat untuk menerangkan suatu algoritma, atau bisa juga berupa daftar *bug*⁸ yang sudah Anda ketahui tapi belum sempat diperbaiki.
2. System Analyst: tulisannya bisa berupa rancangan umum program, untuk apa, dan bagaimana langkah pengembangan selanjutnya.
3. Pengguna umum: dokumentasi bisa berupa langkah demi langkah penggunaan program, *option*⁹ untuk suatu tugas tertentu, dan juga sertakan alamat Anda (kalau berkenan) sebagai referensi bagi komunitas pengguna untuk bertanya.

⁷Script: baris-baris program

⁸Bug: sifat yang salah dari suatu program

⁹Option adalah pilihan atau alternatif tertentu dalam menjalankan program.

Bab 3

Teknik Penulisan

Sebelum lebih jauh membaca buku ini, ada baiknya Anda memahami beberapa hal teknis tentang bentuk tulisannya.

3.1 Gaya

Gaya penulisan yang diterapkan pada pembuatan buku ini mengutamakan hal-hal berikut:

1. Bahasan merupakan yang paling sering diterapkan dalam pembuatan aplikasi, terutama aplikasi database.
2. Memperbanyak contoh program dan sebisa mungkin mungkin menghindari gaya “buku referensi” untuk mempermudah

pemahaman, terutama bagi yang belum terbiasa dengan pemrograman komputer.

3. Meski begitu, daftar isi dan indeksnya sebisa mungkin bisa menjadi referensi ringkas.

Tulisan ini sendiri berusaha untuk menghindari hal-hal spesifik sistem operasi. Namun yang perlu Anda ketahui adalah bahwa contoh program yang ada di sini telah dicoba pada sistem operasi Linux.

3.2 Jenis Huruf dan Simbol

Perlu diperhatikan pula huruf dan font yang digunakan dalam buku ini yang mana berkaitan dengan kemudahan dalam memahami kalimat demi kalimat.

- Italic* perlu diperhatikan atau memperkenalkan istilah dalam bahasa Inggris untuk pertama kalinya.
- Courier** kode program seperti nama variabel, tipe data, dsb. Bisa juga berupa nama file atau program.
- Underline penegasan seperti moto atau kata kunci yang biasanya berkaitan dengan pembicaraan seputar konsep atau filosofi.

Pada script program yang cukup panjang dilengkapi dengan nomor baris untuk memudahkan rujukan dalam pengetikkan kembali (ketika Anda ingin mencobanya).

Karakter dolar (\$) yang mengawali suatu baris perintah menandakan perintah tersebut dijalankan di *console* atau sering disebut sebagai *command line*.¹

¹Karena banyak yang terbiasa dengan sistem operasi DOS, beberapa orang sering menyebutnya sebagai DOS Prompt.

Bab 4

Persiapan

Juga ada beberapa hal yang perlu dipersiapkan untuk menjalankan contoh program yang ada.

4.1 Paket Program

Ada beberapa paket program dalam bentuk RPM yang perlu dipasang (*install*) sebelum mengikuti pembahasan buku ini, yaitu:

python interpreter Python

qt library utama Qt, dalam C++

qt-PostgreSQL library untuk mengakses database PostgreSQL melalui Qt

qt-MySQL library untuk mengakses database MySQL melalui Qt

PyQt library untuk menggunakan Qt dalam Python

Nama paket di atas penulis dapatkan dari RedHat 7.3 yang terdiri dari 3 CD. Pada distribusi Linux lainnya bisa jadi namanya berbeda. Sebagai informasi saja bahwa - dengan CD tersebut - tidak ada yang perlu di-download lagi dari internet untuk mengikuti contoh program pada buku ini.

4.2 Text Editor

Anda dapat menggunakan `vi` atau `pico` dalam *console environment*. Untuk pengguna KDE dapat menggunakan `kate`, `kwrite` atau `gedit` pada Gnome. Sebagai tambahan `vi` dan `kwrite` dapat mengenail perintah Python dimana keduanya akan membedakan dengan warna objek dalam script Python seperti *reserved word*¹, string, integer, dsb.

¹kata resmi yang digunakan dalam bahasa Python sehingga tidak boleh digunakan sebagai nama variabel.

Bagian II

Python

Bab 5

Hello World

Hello world sudah merupakan “tema standar” dalam permulaan belajar sebuah bahasa pemrograman. Makna dibelakangnya adalah bagaimana menampilkan pesan tersebut dengan menggunakan bahasa yang tengah kita pelajari. Buatlah sebuah file `halo.py` dengan text editor pilihan Anda:

```
$ kwrite halo.py
```

Lalu ketikkan baris berikut:

```
print "Hello world !"
```

Simpan file ini dan jalankan:

```
$ python halo.py
```

```
Hello world !
```

dimana perintah `print` tersebut akan mencetak ke layar.

Selamat, kini Anda telah menjadi seorang programmer Python dan mari melanjutkannya dengan hal yang lebih menarik lagi.

5.1 Aturan Penulisan

Keunggulan Python dibandingkan bahasa pemrograman sejenis lainnya adalah script-nya yang lebih mudah dibaca. Kemudahan ini bahkan berasal dari aturan main dalam penulisan sintaksnya seperti pembahasan berikut ini.

5.1.1 Indent

Sebagaimana bahasa pemrograman pada umumnya, Python mengenal apa yang disebut blok program. Baris-baris berikut dianggap sebuah blok program:

```
print "Hello world !"
print "Salam dunia !"
```

Anda tidak boleh menuliskannya seperti ini:

```
print "Hello world !"
print "Salam dunia !"
```

Karena pesan kesalahan berikut akan tampil:

```
File "halo.py", line 2
  print "Salam dunia"
  ^
SyntaxError: invalid syntax
```

Dimanakah letak perbedaannya ? Menurut Python, sebuah blok program harus dalam sebuah “margin”. Perhatikan baris kedua di atas dimana ada selisih satu karakter dengan baris sebelumnya.

Contoh tersebut adalah blok program utama. Anda dapat menjumpai sub-blok dalam pembahasan lainnya seperti pada Bab Kondisi di halaman 51.

5.1.2 Baris Perintah

Pergantian baris (dengan menekan Enter) merupakan pemisah antara dua baris perintah. Baris berikut

```
print "Hello world !"
```

dikatakan sebuah baris perintah, dan

```
print "Salam dunia !"
```

juga merupakan sebuah baris perintah. Mungkin Anda yang terbiasa dengan Pascal atau PHP jelas merasakan perbedaan dengan cara penulisan ini dimana kedua bahasa tersebut tidak akan menjalankan sebuah baris perintah kalau belum diakhiri titik koma (;).

Untuk sebuah baris yang terlalu panjang, Anda dapat menggunakan karakter *backslash* (\) untuk memberitahu Python bahwa baris tersebut belum berakhir.

```
print "Baris ini sangat panjang sehingga \  
perlu ditulis dalam beberapa baris agar \  
memudahkan pencetakan nanti."
```

5.1.3 Keterangan Program

Sebelumnya disebutkan bagaimana pentingnya dokumentasi dalam pemrograman. Anda dapat menggunakan karakter *cross* (#) untuk membuat komentar dalam script:

```
# Tanya kabar  
print "Apa kabar ?"
```

atau bisa juga dituliskan disampingnya:

```
print "Kabar baik." # Jawabannya
```

Selain itu bisa juga menggunakan tiga kutip ganda:

```
""" Oleh    : Owo Sugiana  
   Email   : sugiana@rab.co.id  
   Tanggal : 17-1-2003  
   Manfaat : Mencetak salam  
   """  
  
print "Salam"
```

5.2 Variabel

Python tidak membutuhkan deklarasi variabel secara khusus sebagaimana beberapa bahasa pemrograman lainnya. Deklarasi cukup dengan memberikan nilai awal seperti contoh berikut:

```
n = 1
```

yang menunjukkan variabel `n` bertipe integer (bilangan bulat). Python juga memberikan keluwesan dalam deklarasi variabel ini. Variabel yang sama dapat berubah-ubah tipe datanya meski dalam sebuah script.

```
n = 1
n = "Saya belajar Python."
```

Baris pertama mendeklarasikan `n` bertipe integer, dan pada baris berikutnya `n` berubah tipe menjadi string.

5.3 Modus Interaktif

Baris program juga dapat ditulis langsung dalam modus interaktif (*interactive mode*). Cara ini biasanya digunakan untuk melakukan eksperimen pendek.

```
$ python
```

```
Python 1.5.2 (#1, Aug 7 2000, 21:22:50)
[GCC 2.95.2 19991024 (release)] on linux2
```

Copyright 1991-1995 Stichting Mathematisch Centrum, Amsterdam

```
>>> print "Hello world !"  
Hello world
```

Jadi bila Anda menemui prompt `>>>` berarti baris program dijalankan melalui modus interaktif ini.

Sedikit perbedaan antara modus interaktif dengan penggunaan file, yaitu pada penampilan nilai. Penjumlahan berikut bisa langsung tampil hasilnya pada modus interaktif

```
>>> 5+7  
12
```

tapi tidak terjadi bila menggunakan file. Anda tetap memerlukan `print` untuk menampilkannya.

```
print 5+7
```

Bab 6

Tipe Data

Meski tidak memerlukan deklarasi secara khusus, Python sangat memperhatikan tipe data. Anda tidak diperkenankan melakukan hal ini:

```
>>> 5 + "1"
```

yang bisa menimbulkan pesan kesalahan berikut:

```
Traceback (innermost last):  
  File "<stdin>", line 1, in ?  
TypeError: number coercion failed
```

dimana sebuah bilangan ditambahkan dengan sebuah string. Selanjutnya akan diterangkan tipe data yang sering dipakai dalam pembuatan aplikasi nanti.

6.1 Bilangan

Bilangan atau angka merupakan hal yang biasa kita temui dalam matematika. Python mengenal dua tipe data bilangan: bilangan bulat (integer) dan bilangan pecahan (float).¹

```
>>> n=7
>>> n
7
```

menyatakan `n` bertipe integer dengan nilai 7.

```
>>> n=7.0
7.0
```

menyatakan bahwa `n` bertipe float karena mengandung titik desimal. Dikatakan juga bahwa `n` merupakan *variabel* dan 7 merupakan nilai.

6.1.1 Operator

Operator merupakan “pengolah” variabel atau nilai. Tabel 6.1 berisi beberapa operator bilangan yang sering dijumpai pada operasi matematika.

```
>>> 10+5
15
>>> 12*6
72
```

¹Python juga mengenal bilangan imajiner, namun tidak dibahas disini.

Catatan Dengan hasil yang langsung ditampilkan seperti di atas maka Anda juga dapat menjadikan modus interaktif ini sebagai kalkulator.

Sedikit catatan untuk operator bagi ($/$), bila kedua bilangan merupakan integer maka hasilnya pun integer.

```
>>> 8/5
1
```

Apabila ingin mendapatkan nilai 1,6 maka salah satunya harus float:

```
>>> 8.0/5
1.6
```

atau bisa juga menggunakan fungsi `float()`:

```
>>> float(8)/5
1.6
```

Fungsi matematika lainnya terdapat dalam modul `math`.

6.1.2 Pengelompokan Operasi

Sangat mungkin terdapat beberapa operasi terdapat dalam satu baris:

```
>>> 8 + 5 * 2
18
```

Operator		
Simbol		
Contoh		
tambah		
+		
a + b		
kurang		
-		
a - b		
kali		
*		
a * b		
bagi		
/		
a / b		
sisa bagi		
%		
a % b		
pangkat		
**		
a ** b		

Tabel 6.1: Operator Bilangan

namun Anda perlu memperhatikan prioritas masing-masing operator. Pada contoh di atas Python mendahulukan operasi perkalian, baru selanjutnya penjumlahan. Jika Anda ingin sebaliknya gunakan tanda kurung untuk pengelompokan operasi:

```
>>> (8 + 5) * 2
26
```

6.1.3 Pembulatan

Pembulatan bilangan pecahan bisa menggunakan fungsi `int()` dan `round()`. Perbedaannya adalah `int()` membulatkan ke bawah, sedangkan `round()` membulatkan ke bawah bila bagian pecahan (desimal) lebih kecil dari 0,5 namun bila sebaliknya akan membulatkan ke atas.

```
>>> n=17.5
>>> int(n)
17
>>> round(n)
18.0
>>> n=17.49
>>> int(n)
17
>>> round(n)
17.0
```

6.1.4 Bentuk Tampilan

Untuk mencetak laporan tentu diperlukan tampilan yang baik, misalnya:

1. Ada pemisah ribuan, contoh: 2.345
2. Hanya dua angka desimal di belakang koma, contoh: 2.345,67

Program kecil berikut akan menampilkan angka sesuai kriteria di atas.

```
>>> import locale
>>> locale.setlocale(locale.LC_ALL, "")
'id_ID'
```

`id_ID` berarti Indonesia, lalu gunakan `format()` untuk menampilkan angka dengan pemisah ribumannya.

```
>>> locale.format("%.2f", 1234.567, 1)
'1.234,57'
```

Angka 2 pada `"%.2f"` menunjukkan jumlah desimal yang akan tampil. Kalau Anda lihat hasilnya maka dapat diambil kesimpulan bahwa 1234.567 telah dibulatkan desimalnya dari 3 angka menjadi 2 dengan pembulatan ke atas. Cobalah ganti dengan angka terakhir lebih kecil dari 5:

```
>>> locale.format("%.2f", 1234.564, 1)
'1.234,56'
```

maka `format()` melakukan pembulatan ke bawah.

Angka 1 pada masukan terakhir pada fungsi `format()` menunjukkan apakah angka akan ditampilkan dengan pemisah ribuan.

```
>>> locale.format("%.2f", 1234.564)
'1234,56'
```

Modul locale

`locale` adalah modul yang berkaitan dengan setting negara. Maka tidak heran kalau pemisah ribuan dan pemisah desimal tampil sesuai dengan gaya Indonesia. Setting ini bisa Anda temui di awal instalasi sistem operasi.

6.2 String

String merupakan kalimat yang diawali dan diakhir dengan kutip, baik kutip ganda maupun tunggal. Selain angka, Python juga dapat menangani string dengan berbagai cara. String dapat diapit baik oleh kutip tunggal maupun kutip ganda.

```
>>> 'telur dadar'  
'telur dadar'  
>>> "bolu kukus"  
'bolu kukus'
```

Menuliskan kutip dalam suatu string harus diawal *backslash* (`\`) bila jenis kutipnya sama:

```
>>> 'doesn\'t'  
"doesn't"
```

atau bisa juga tanpa *backslash* bila kutipnya berbeda:

```
>>> "doesn't"  
"doesn't"
```

String juga dapat ditulis dengan awalan tiga buah kutip (baik kutip tunggal maupun ganda) dimana akhir string juga dinyatakan dengan tiga kutip juga.

```
>>> print """
... Cara pakai: kontak [OPTIONS]
...   -h                Tampilkan bantuan ini
...   -H nama-host     Nama host yang akan dihubungi
... """

Cara pakai: kontak [OPTIONS]
  -h                Tampilkan bantuan ini
  -H nama-host     Nama host yang akan dihubungi

>>>
```

Keuntungan cara ini adalah memberikan kebebasan penulisan string bila lebih dari satu baris, karena tidak memerlukan backslash (\).

6.2.1 Penjumlahan & Perkalian

Beberapa string dapat digabung dengan menggunakan operator + (penjumlahan) berikut:

```
>>> a = 'Bahasa saya'
>>> b = 'adalah Python.'
>>> c = a + ' ' + b
>>> print c
Bahasa saya adalah Python.
```

dan juga digandakan dengan * (perkalian):

```
>>> '-' * 10
'-----'
```

6.2.2 String Format

Anda sudah mengetahui cara menggabungkan string. Tapi bagaimana kalau angka digabungkan dengan string? Python menyediakan fungsi `str()` untuk mengubah variabel bertipe apa saja menjadi string.

```
>>> a = 1
>>> b = 2
>>> c = a + b
>>> s = str(a) + " + " + str(b) + " = " + str(c)
>>> print s
1 + 2 = 3
```

Meski cara di atas sudah benar, namun pengisian variabel `s` menyulitkan pembacaan program. Gantilah menjadi seperti ini:

```
>>> s = "%s + %s = %s" % (a,b,c)
>>> print s
1 + 2 = 3
```

dan dengan demikian juga mempercepat proses debugging.

6.2.3 String Menjadi Integer - `int()`

Kebalikan dari fungsi `str()`, `int()` digunakan untuk mengubah string menjadi bilangan integer.

```
>>> int("9")
9
```

Memang, fungsi `int()` yang dibahas sebelumnya digunakan untuk pembulatan. Contoh di atas menunjukkan `int()` memiliki kegunaan lain.

6.2.4 String Menjadi Float - `float()`

`float()` mirip dengan `int()` di atas, hanya saja ia mengembalikan bilangan bertipe float.

```
>>> float("8.9")
8.9
```

Proses perubahan dari suatu tipe data ke tipe data lainnya ini sering disebut sebagai konversi atau *cast*.

6.2.5 Panjang String - `len()`

Fungsi `len()` akan mengembalikan integer yang menunjukkan jumlah karakter dalam suatu string:

```
>>> len("siap")
4
```


6.3 List

List sering disebut sebagai array atau larik (dalam bahasa Indonesia) adalah kumpulan data dimana setiap data memiliki nomor index yang dimulai dari 0.

```
>>> a = [9,8,7,6,5]
>>> a[0]
9
>>> a[1]
8
```

Python juga memperbolehkan setiap elemen data memiliki tipe yang berbeda:

```
>>> b = ['Python',1991]
>>> b[0]
'Python'
>>> b[1]
1991
```

dimana `b[0]` bertipe string, dan `b[1]` bertipe integer.

6.3.1 Penambahan - `append()` & `insert()`

List sebenarnya merupakan objek dimana ia memiliki beberapa fungsi. Misalnya fungsi untuk menambah elemen data.

```
>>> c = []
```

c merupakan list hampa dimana tidak ada elemen data di dalamnya. Untuk menambahkannya gunakan fungsi `append()`.

```
>>> c.append(100)
>>> c.append('persen')
>>> c
[100, 'persen']
```

Perintah `insert()` dapat digunakan untuk menyisipkan:

```
>>> a = [0,1,2,3,4]
>>> a.insert( (2,999) )
>>> a
[0,1,999,2,3,4]
```

Kalau nomor index melampaui nomor index maksimum maka hasilnya sama saja dengan penggunaan `append()`.

```
>>> a.insert( (10,888) )
>>> a
[0,1,999,2,3,4,888]
```

6.3.2 Penghapusan - del

Perintah `del` digunakan untuk menghapus elemen data:

```
>>> a = [1,2,3,4,5]
>>> del a[3]
>>> a
[1,2,3,5]
```

yang berarti menghapus elemen data pada nomor index 3.

6.3.3 String Sebagai List

String juga dapat diperlakukan sebagaimana list, atau lebih tepatnya *list of character* (list yang elemen datanya terdiri dari satu karakter).

```
>>> d = 'Python'
>>> d[0]
'p'
>>> d[5]
'n'
```

6.3.4 Pemenggalan

Pemenggalan (*slice*) suatu list bertujuan untuk mengambil sejumlah elemen data yang ada di dalamnya. Lihat tabel 6.2 sebagai contoh.

Cara terbaik untuk mengingat bagaimana pemenggalan bekerja adalah dengan berasumsi bahwa angka index berada diantara dua karakter.²

6.3.5 Panjang List - len()

Anda dapat menggunakan fungsi len() untuk mendapatkan jumlah elemen dalam list:

```
>>> d = ["Jeruk", "manis"]
>>> len(d)
2
```

²Lihat gambar 6.1.

Variabel	
	<p>Hasil</p> <p>Keterangan</p> <p>d</p> <p>'Python'</p> <p>nilai awal</p> <p>d[0]</p> <p>'P'</p> <p>karakter pertama dari kiri</p> <p>d[2:]</p> <p>'thon'</p> <p>mulai karakter ketiga dari kiri</p> <p>d[:2]</p> <p>'Py'</p> <p>dua karakter pertama dari kiri</p> <p>d[-1]</p> <p>'n'</p> <p>karakter pertama dari kanan</p> <p>d[-2:]</p> <p>'on'</p> <p>dua karakter pertama dari kanan</p> <p>d[:-2]</p> <p>'Pyth'</p> <p>mulai karakter pertama dari kiri hingga sebelum karakter ke dua dari kanan</p> <p>d[2:5]</p> <p>'tho'</p> <p>mulai karakter ketiga hingga kelima dari kiri</p>

Tabel 6.2: Contoh pemenggalan list

```

+---+---+---+---+---+---+
| P | y | t | h | o | n |
+---+---+---+---+---+---+
0   1   2   3   4   5   6
    -6  -5  -4  -3  -2  -1

```

Gambar 6.1: Nomor index pada list

6.3.6 List Dalam List

Tipe data untuk setiap elemen list bisa apa saja, termasuk tipe list itu sendiri. Sehingga kita dapat membuat list dalam list atau disebut juga list multidimensi.

```
>>> mhs = [[2001,"Hery"],[2002,"Yuli"],[2003,"Gani"]]
```

Struktur `mhs` di atas menyerupai sebuah tabel dimana terdapat ID dan nama mahasiswa. Menurut Anda berapa jumlah mahasiswa di atas? Tentu Anda tidak ingin menjawab 6 (total jumlah elemen data).

```
>>> len(mhs)
3
```

Gunakan nomor index 0 untuk mendapatkan “record” pertama:

```
>>> mhs[0]
[2001, 'Hery']
```

sedangkan untuk mendapatkan “field” kedua (nama) dalam record tersebut:

```
>>> mhs[0][1]
'Hery'
```

List dua dimensi ini dapat juga dikatakan sebagai struktur data tabel yang terdiri dari baris dan kolom. Jadi pada contoh di atas [0] adalah nomor baris, dan [1] adalah nomor kolom.

6.3.7 Membalikkan Urutan - reverse()

Urutan elemen dalam list bisa dibalik dengan menggunakan fungsi reverse().

```
>>> a = [9,4,8,3]
>>> a.reverse()
>>> a
[3, 8, 4, 9]
```

6.3.8 Mengurutkan - sort()

sort() digunakan untuk mengurutkan list dari yang terkecil ke yang terbesar (*ascending*):

```
>>> a = [9,4,8,3]
>>> a.sort()
```

```
>>> a
[3, 4, 8, 9]
```

Sedangkan untuk mengurutkan dari yang terbesar ke yang terkecil (*descending*) bisa menggunakan kombinasi `sort()` dan `reverse()`.

```
>>> a = [9,4,8,3]
>>> a.sort()
>>> a.reverse()
>>> a
[9, 8, 4, 3]
```

6.3.9 Menyalin - list()

Menyalin suatu variabel list ke variabel lainnya tidak bisa dengan cara seperti ini:

```
>>> a = [9,4,8,3]
>>> b = a
```

karena itu hanya membuat `b` sebagai nama lain dari `a` saja. Ini bisa dibuktikan setiap perubahan pada `a` maka `b` juga berubah

```
>>> a.append(7)
>>> a
[9, 4, 8, 3, 7]
>>> b
[9, 4, 8, 3, 7]
```

Fungsi `list()` dapat digunakan agar `b` benar-benar merupakan list baru hasil salinan dari `a`.

```
>>> a = [9,4,8,3]
>>> b = list(a)
>>> a.append(7)
>>> a
[9, 4, 8, 3, 7]
>>> b
[9, 4, 8, 3]
```

6.4 Dictionary - Key & Value

Dictionary sering disebut sebagai *associative arrays* dimana - tidak seperti array - ia tidak diurutkan berdasarkan index angka (0,1,2,3). Setiap elemen data dalam dictionary memiliki kunci (*key*) dimana satu sama lain bersifat unik.

```
>>> mhs = { 1001:"Andre", 1002:"Rusly", 1003:"Endro"}
```

1001, 1002, dan 1003 disebut sebagai *keys* (kunci), sedangkan "Andre", "Rusly", dan "Endro" merupakan *values* (nilai). Untuk mendapatkan nilai-nilai tersebut perlu mengetahui *key*-nya:

```
>>> mhs[1001]
'Andre'
```

Tipe data *keys* tidak harus integer, string juga bisa, begitu juga dengan *values* yang bisa berupa integer, float, atau list. Bahkan tipe data untuk *keys* dan *values* tidak harus seragam antara elemen yang satu dengan yang lainnya.


```
>>> data = { "a":1990, 9:"sembilan", "test":[1,2,"c","x"] }
>>> data["a"]
1990
```

Sebagaimana list, dictionary juga merupakan objek yang memiliki fungsi.

6.4.1 Daftar Kunci - keys()

Gunakan fungsi `keys()` untuk memperoleh *keys* dari dictionary.

```
>>> mhs.keys()
[1003, 1002, 1001]
```

6.4.2 Daftar Nilai - values()

`values()` digunakan untuk memperoleh keseluruhan nilai dalam dictionary.

```
>>> mhs.values()
['Endro', 'Rusly', 'Andre']
```

6.4.3 Menambah Atau Mengubah Nilai - update()

`update()` digunakan untuk menambah atau mengubah elemen data.

```
>>> mhs = {}
```

`{}` adalah dictionary hampa

```
>>> mhs.update({1004: 'Eko'})
>>> mhs
{1004: 'Eko'}
```

6.4.4 Menghapus - del

Untuk menghapus elemen dalam dictionary bisa menggunakan perintah del.

```
>>> del mhs[1004]
>>> mhs
{}
```

6.5 Mengetahui Tipe Data - type()

Fungsi type() digunakan untuk mengetahui tipe suatu nilai:

```
>>> type(7.0)
<type 'float'>
```

6.6 Daftar Fungsi Suatu Objek - dir()

Beberapa variabel pada contoh sebelumnya merupakan objek. Hal ini terlihat dari banyaknya fungsi yang dimiliki oleh suatu variabel. Untuk mengetahui fungsi apa saja yang ada dapat dilihat dengan menggunakan fungsi dir():

```
>>> a = []
```

```
>>> dir(a)
['append', 'count', 'extend', 'index', 'insert', 'pop',
 'remove', 'reverse', 'sort']
```


Bab 7

Kondisi

`if` menyatakan bila pada saat kondisi tertentu tercapai maka apa yang akan dilakukan, sehingga tema ini sering disebut sebagai “jika-maka”.

Karena penulisan `if` melibatkan blok program lebih dari satu, ada baiknya kini Anda menggunakan *text editor* seperti `vi` atau `kwrite` untuk menulis script program, tidak lagi dalam *interactive interpreter*.

Sekarang mulailah menuliskan program pendek berikut untuk memahami bagaimana `if` bekerja.

```
print "Jika Anda tidak yakin saya diam saja."  
n = raw_input("Yakin (y/t) ? ")
```

```
if n == "y":
    print "Baiklah"
```

`raw_input()` digunakan untuk menerima masukan dari pemakai. Fungsi ini mengembalikan nilai string.

Operator `==` menyatakan persamaan, sehingga `n == "y"` dibaca *n sama dengan "y"*. Jangan lupa untuk menyertakan titik dua (`:`) pada akhir baris `if`.

Sub-Blok Program

Perhatikan baris `print "Baiklah"` tampak menjorok ke dalam. Baris ini disebut sebagai *sub-blok* dimana `if` merupakan blok utamanya. Jika Anda ingin menyertakan baris program lainnya masih termasuk dalam sub-blok `if`, maka posisi kolomnya harus sama dengan posisi sebelumnya seperti pada contoh berikut:

```
if n == "y":
    print "Baiklah"
    print "Siapkan perlengkapannya"
```

Baris lain yang sejajar dengan `if` menandakan baris tersebut merupakan blok utama.

```
if n == "y":
    print "Baiklah"
    print "Siapkan perlengkapannya"
print "..."
```

Sehingga `print "..."` selalu dijalankan meski `n == "y"` tidak terpenuhi.

7.1 Bentuk Logika

Secara umum semua tipe data yang menyatakan kehampaan dianggap memiliki logika *false* (salah). Misalnya angka 0, string "", objek `None`, list `[]`, dsb. Kondisi sebaliknya berarti bermakna *true* (benar) seperti angka 1, -100, string "a", list `[1,2,3]`, dst. Perhatikan script berikut ini:

```
if 0:
    print "Benar"
else:
    print "Salah"
```

yang akan menampilkan pesan "Salah", dan baris berikut

```
if 1:
    print "Benar"
else:
    print "Salah"
```

akan menampilkan pesan "Benar". Jadi contoh program sebelumnya dapat ditulis seperti ini:

```
print "Jika Anda tidak yakin saya diam saja."
n = raw_input("Yakin (y/t) ? ")
yakin = n == "y"
if yakin:
    print "Baiklah"
```

dan kalau Anda tampilkan variabel `yakin` dengan `print yakin` maka Anda mendapatkan angka 1 atau 0, tergantung nilai `n`.

7.2 Selain Itu - else

`else` digunakan untuk menyatakan kondisi perlawanan dari `if`. Bisa juga dikatakan bahwa jika suatu kondisi tidak terpenuhi maka jalankan perintah lainnya.

```
n = raw_input("Apakah Anda yakin (y/t) ? ")
if n == "y":
    print "Baiklah, kita berangkat"
else:
    print "Kita perlu latihan lagi"
```

Jadi kalau `n` tidak sama dengan "y" maka baris

```
print "Kita perlu latihan lagi"
```

yang akan dijalankan.

7.3 Selain Itu Jika - elif

Pemakai bisa jadi menjawab selain huruf `y` dan `t` dimana Anda perlu mengantisipasi hal ini.

```
n = raw_input("Apakah Anda yakin (y/t) ? ")
if n == "y":
    print "Baiklah, kita berangkat"
else:
    if n == "t":
        print "Kita perlu latihan lagi"
    else:
        print "Pilihannya y atau t"
```


Namun dengan `elif` script di atas bisa ditulis seperti ini:

```
n = raw_input("Apakah Anda yakin (y/t) ? ")
if n == "y":
    print "Baiklah, kita berangkat"
elif n == "t":
    print "Kita perlu latihan lagi"
else:
    print "Pilihannya y atau t"
```

`elif` juga bisa ditulis kembali untuk memeriksa kondisi berikutnya.

```
n = raw_input("Apakah Anda yakin (y/t/d) ? ")
if n == "y":
    print "Baiklah, kita berangkat"
elif n == "t":
    print "Kita perlu latihan lagi"
elif n == "d":
    print "Diam ni yee"
else:
    print "Pilihannya y, t, atau d"
```

7.4 Operator Perbandingan

Operator `==` dikategorikan sebagai operator perbandingan untuk menyatakan *sama dengan*. Operator perbandingan lainnya dapat Anda lihat di tabel 7.1.


```
n = raw_input("Yakin (y/t) ? ")
if n in ["y","Y"]:
    print "Baiklah"
```

Catatan Meski pertanyaannya meminta pemakai untuk memasukkan huruf `y` atau `t` dalam huruf kecil namun kita perlu mengantisipasi *capslock* pada keyboard sedang aktif.

7.5 Operator Logika

Sebagaimana bilangan atau string, suatu kondisi juga memiliki operator yang disebut operator logika yaitu `not`, `and`, dan `or`.

7.5.1 Bukan - not

`not` digunakan untuk negasi (perlawanan) suatu bentuk logika.¹

```
n = raw_input("Masukkan huruf : ")
if not n:
    print "Kok kosong ?"
else:
    print "Anda memasukkan huruf " + n
```

Pada saat program dijalankan Anda diminta memasukkan huruf apa saja dan diakhiri dengan penekanan Enter. Apabila tidak ada huruf yang dimasukkan berarti `n` merupakan string hampa.

¹Lihat halaman 52 mengenai bentuk logika.

7.5.2 Semua Kondisi Terpenuhi - and

Dengan operator `and` semua kondisi harus terpenuhi untuk mencapai *true*. Contoh berikut meminta pemakai memasukkan bilangan ganjil.

```
n = raw_input("Masukkan bilangan ganjil : ")
if n and (int(n) % 2):
    print "Benar"
else:
    print "Bukan"
```

Kondisi pertama adalah apabila `n` terisi (bukan string hampa). Sedangkan kondisi kedua adalah apabila `int(n)` habis dibagi 2.

Catatan Penggunaan `int(n)` juga bisa menimbulkan kesalahan yang tidak dapat ditangani script di atas, yaitu apabila proses konversi dari string ke integer gagal, misalnya dengan memasukkan huruf. Untuk menangkap kesalahan konversi tersebut Anda bisa menggunakan `try` yang dibahas pada halaman 73.

7.5.3 Salah Satu Kondisi Terpenuhi - or

Operator `or` menyatakan bahwa *true* tercapai bila salah satu kondisi terpenuhi.

```
n = raw_input("Apakah Anda yakin (y/t) ? ")
if n == "y" or n == "Y":
```

```
    print "Bagus."  
else:  
    print "Kita cari alternatif lain."
```

a					
b					
not a					
a and b					
a or b					
1					
1					
0					
1					
1					
1					
0					
0					
0					
1					
0					
1					
1					
0					
1					
0					
0					
1					
0					
0					

Tabel 7.2: Operator Logika

Bab 8

Perulangan - Loop

Perulangan (*loop*) adalah menjalankan proses sampai suatu kondisi terpenuhi. Kondisi yang dimaksud - tentunya - akan mempengaruhi seberapa banyak proses tersebut dijalankan. Banyaknya perulangan ini bisa ditentukan di awal *loop* maupun selama *loop* berlangsung.

8.1 Jumlah Perulangan Ditetapkan - for

`for` merupakan perulangan sebanyak elemen data dalam suatu list. Perhatikan contoh berikut:

```
for nomor in [1,2,3]:  
    print nomor
```

`for` akan menghitung satu persatu setiap elemen dalam `[1,2,3]`. Selama proses menghitung itu, ia:

1. Memberi nilai dari setiap elemen data ke variabel nomor.
2. Menjalankan `print` nomor sesaat setelah point 1 dijalankan.

Dengan demikian script di atas menghasilkan output sebagai berikut:

```
1
2
3
```

`for` memang bekerja dengan list, sehingga list `[1, 2, 3]` bisa merupakan list berisi data lainnya.

```
record = [ 2769, "Jeruk", 5500 ]
for nilai in record:
    print nilai,
print
```

Koma pada `print` berarti tidak berganti baris.

Untuk menampilkan list multidimensi kita membutuhkan *for di dalam for* sebanyak dimensi tersebut. Misalkan struktur tabel yang merupakan list dua dimensi, maka diperlukan dua `for` untuk menampilkan isi setiap elemen datanya.

```
tabel = [
    [ 2769, "Jeruk", 5500 ],
    [ 8205, "Mangga", 3750 ]
```



```

]
for record in tabel:
    for nilai in record:
        print nilai,
    print

```

`range()`

`range()` merupakan fungsi yang mengembalikan daftar bilangan integer secara berurutan dimulai dari nol, banyaknya sesuai dengan jumlah yang diinginkan. `range(3)` berarti `[0,1,2]` dan `range(5)` berarti `[0,1,2,3,4]`, begitu seterusnya.

Fungsi ini sering dipakai dalam `for` untuk mendapatkan jumlah perulangan yang diinginkan.

```

for nomor in range(3):
    print nomor

```

dimana output yang dihasilkan adalah:

```

0
1
2

```

Bila kisaran yang diinginkan adalah 1-3 maka gunakan `range(1,4)`.

Latihan Tampilkan nilai faktorial¹ n dimana n dimasukkan pada saat *runtime*. Contoh: faktorial 5 adalah $5 \times 4 \times 3 \times 2 \times 1 = 120$. Namun faktorial 0 atau lebih kecil lagi (negatif) hasilnya tetap 1.

¹Bilangan faktorial n biasa ditulis dengan $n!$

8.2 Selama - while

`while` adalah bentuk perulangan dimana jumlah perulangannya ditentukan oleh suatu kondisi. Ikuti langkah berikut untuk hasil yang sama dengan contoh `for` sebelumnya, namun kini menggunakan `while`:

```
i = 0
while i < 3:
    i = i + 1
    print i
```

Script di atas memang lebih pas bila dengan `for`. Contoh berikut akan menampilkan bilangan acak yang dibuat terus-menerus sampai ditemukan bilangan yang lebih besar dari 0.5:

```
import random

i = 0
while i <= 0.5:
    i = random.random();
    print i
```

dengan contoh hasil sebagai berikut:

```
0.135330567072
0.321183281272
0.42870775016
0.335857508686
0.447331794014
```

```
0.287945799635
0.854292081945
```

Hasil di atas sangat mungkin berbeda dengan yang Anda peroleh - karena sifat bilangan acak itu sendiri. Perhatikan nilai setiap barisnya dimana program tidak akan berhenti sampai ditemukan kondisi yang dimaksud yaitu 0,854292081945 lebih kecil dari 0,5. Coba jalankan sekali lagi, pasti Anda akan menemukan hasil yang berbeda.

Kini sudah jelas, untuk menerapkan algoritma tersebut perulangan `for` tidak dapat digunakan karena jumlah perulangan yang tidak menentu.

Modul random

`random` adalah modul tambahan untuk hal-hal yang berkaitan dengan bilangan acak. Ada banyak modul lainnya yang disertakan dalam paket Python seperti `time` untuk penanganan waktu dan `string` untuk penanganan string. Anda dapat melihat daftar fungsi maupun konstanta yang terdapat dalam suatu modul dengan menggunakan fungsi `dir()`. Contoh:

```
import random
print dir(random)
```

Kalau Anda beruntung, dalam suatu modul terdapat konstanta `__doc__` yang merupakan dokumentasi.

```
import time
print time.__doc__
```

8.3 Keluar Dari Perulangan - break

Perulangan juga bisa dihentikan “dari dalam”, yaitu menggunakan perintah `break`.

```
import random

while 1:
    i = random.random();
    print i
    if i > 0.5:
        break
    print "selesai"
```

Perhatikan angka 1 diatas yang merupakan kondisi benar². Dengan kata lain perulangan akan dijalankan terus-menerus tanpa henti, karena akan dihentikan dengan perintah `break`. `break` hanya menghentikan perulangan, selanjutnya proses kembali ke blok utama. Pada contoh di atas dibuktikan dengan dijalanannya `print "selesai"`.

Latihan

1. Tampilkan satu bilangan acak yang lebih kecil dari 0,5.
2. Melanjutkan sebelumnya, tampilkan n bilangan acak yang lebih kecil dari 0,5. n diisi pada saat runtime. Tentu saja Anda boleh menggunakan kombinasi `for` dan `while`.

²Lihat pembahasan bentuk logika di halaman 52.

Bab 9

Fungsi

Pada contoh-contoh program sebelumnya, Anda sudah menggunakan fungsi seperti `len()`, `range()`, dan `random()` dimana tujuan pembuatan fungsi adalah mempersingkat penulisan suatu blok program yang dipakai berulang-ulang. Misalnya pada program berikut:

```
print "-" * 10
print "Setiap pesan diawali garis."
print "-" * 10
print "Juga diakhiri garis."
print "-" * 10
```

Tampak dalam script di atas bahwa

```
print "-" * 10
```

ditulis berulang-ulang. Contoh berikut akan kita ganti dengan pemanggilan fungsi:

```
# Deklarasi fungsi
def garis():
    print "-" * 10

# Program utama
garis()
print "Setiap pesan diawali garis."
garis()
print "Juga diakhiri garis."
garis()
```

`def` menyatakan awal deklarasi suatu fungsi. Perhatikan posisi kolom source fungsi di atas yang menjorok ke dalam.

9.1 Nilai Masukan

Fungsi tersebut dikatakan fungsi statis karena ia tidak bisa menerima “masukan” dari baris program yang memanggilmnya. Misalnya kita ingin membuat garis dengan panjang berbeda-beda namun fungsi yang digunakan tetap sama.

```
def garis(n):
    print "-" * n

garis(10)
print "Setiap pesan diawali garis."
```

```
garis(3)
print "Juga diakhiri garis."
garis(10)
```

Fungsi juga dapat menerima masukan lebih dari satu. Misalkan karakternya bisa diganti-ganti, tidak hanya "-".

```
def garis(k,n):
    print k * n

garis("-",10)
print "Setiap pesan diawali garis."
garis(".",3)
print "Juga diakhiri garis."
garis("*",10)
```

9.2 Nilai Keluaran - return

Fungsi juga dapat mengembalikan suatu nilai sebagaimana yang pernah kita temui pada `len()`, `range()`, dan `random()` sebelumnya. `return` digunakan untuk tujuan tersebut.

```
def garis(k,n):
    return k * n

print garis("-",10)
print "Setiap pesan diawali garis."
print garis(".",3)
print "Juga diakhiri garis."
```

```
print garis("*",10)
```

Sebagai tambahan, perintah lain di bawah baris `return` (masih dalam fungsi yang sama) tidak akan diproses, sebagaimana “baris mubazir” yang terdapat pada contoh berikut ini:

```
def garis(k,n):  
    return k * n  
    print "Selesai"
```

Baris `print "Selesai"` tidak akan dijalankan karena sudah didahului oleh `return` yang menyebabkan proses keluar dari fungsi.

Fungsi yang tidak menggunakan `return` sebenarnya tetap mengembalikan nilai, yaitu `None`. `None` merupakan objek hampa.

9.3 Memanggil Dirinya

Rekursif adalah istilah untuk fungsi yang memanggil dirinya sendiri. Contohnya pada fungsi `faktorial()` berikut ini:

```
def faktorial(n):  
    if n <= 1: return 1  
    else: return n * faktorial(n-1)  
  
print faktorial(5)
```


Rekursif memang membuat algoritma menjadi lebih pendek. Namun perlu diingat teknik ini tergolong lebih boros memori. Jadi sebisa mungkin tetap gunakan perulangan. Namun jika algoritma menjadi terlalu rumit silahkan terapkan rekursif.

9.4 Kepemilikan Variabel

Fungsi dapat mengenal variabel yang dibuat pada blok utama. Sedangkan tidak sebaliknya.

```
def hurufDalamNama():
    huruf = {}
    for h in nama:
        if huruf.has_key(h):
            jml = huruf[h] + 1
        else:
            jml = 1
        huruf.update( {h:jml} )
    return huruf

nama = raw_input("Nama Anda: ")
print hurufDalamNama()
```

Variabel `nama` dibuat pada blok utama, namun dapat dikenal oleh fungsi di atas. Namun blok utama tidak dapat mengenal variabel yang dibuat di dalam fungsi tersebut. Sehingga kalau Anda mencoba menambahkan baris berikut pada akhir program:

```
print huruf
```

akan tampil pesan kesalahan:

```
Traceback (innermost last):
  File "kepemilikan.py", line 13, in ?
    print huruf
NameError: huruf
```

Sehingga dikatakan kalau huruf dimiliki oleh fungsi `hurufDalamNama()`.

Latihan

`hurufDalamNama()` adalah fungsi yang mengembalikan dictionary sebagai keluarannya. Tampilkan isinya seperti contoh berikut:

```
Nama Anda: sugiana
a = 2
g = 1
i = 1
n = 1
s = 1
u = 1
```

tanpa perlu mengubah fungsinya.

Petunjuk Gunakan `keys()` dan `sort()`.

9.5 Fungsi Interpreter - exec()

`exec()` adalah fungsi interpreter yang akan menerjemahkan string sebagai perintah Python.

```
>>> exec("a = 5")
>>> a
5
```

Berikut ini program kalkulator yang akan berhenti bila hanya tombol Enter saja yang ditekan.

```
from math import *

while 1:
    r = raw_input("x = ")
    if r == "":
        break
    exec("x = " + r)
    print x
```

Anda dapat menuliskan rumus matematika sesuai dengan ketentuan Python. Bahkan modul `math` sudah disertakan, memudahkan penggunaan fungsi tambahan.

```
x = 8*3
24
x = 7+3
10
x = sin(45)
```

```
0.850903524534
x = pi*(10**2)
314.159265359
x =
```

Bab 10

File

Bab ini membahas beberapa hal mengenai operasi file dan direktori. Pembuatan file pada pemrograman aplikasi database biasanya untuk mencetak laporan dimana file tersebut berfungsi ganda: sebagai *preview*¹ (ke layar) dan juga untuk pencetakan ke printer.

10.1 Baca Tulis

Untuk membuat atau membuka suatu file dapat menggunakan fungsi `open()`.

```
f = open("test.txt", "w")
```

¹Preview atau pracetak, istilah untuk melihat hasil yang akan dicetak.

```
f.write("coba")
f.close()
```

Option “w” (*write*) dipakai untuk menulis ke file, sedangkan untuk membacanya gunakan option “r” (*read*). Contoh berikut akan menampilkan isi file `/etc/hosts`.

```
f = open("/etc/hosts","r")
for baris in f.readlines():
    print baris,
f.close()
```

`readlines()` mengembalikan list dimana elemennya merupakan string yang mewakili setiap baris file. String ini diakhiri karakter Enter, namun belum tentu untuk string pada baris terakhir. Oleh karena itu `print` diberi tambahan koma agar tidak terjadi pencetakan karakter enter (“\n”) sebanyak dua kali.

10.2 Printer

Sebuah device printer dapat dianggap sebuah file, biasanya ada di `/dev/lp0`.² Oleh karena itu untuk melakukan pencetakan langsung ke printer dapat memanfaatkan fungsi `open()` ini.

```
f = open("/dev/lp0","w")
f.write("test\n")
f.close()
```

²DOS menyebutnya sebagai LPT1

Jangan lupa, pastikan printer siap (*ready*) sebelum program di atas dijalankan, juga pastikan Anda memiliki hak tulis pada `/dev/lp0` ini. Kalau belum hubungi *system administrator* (user root). Atau bila Anda orangnya jalankan perintah berikut:

```
$ su -c "chmod 666 /dev/lp0"
```

Bila printer belum siap maka program akan terhenti pada `close()` hingga printer siap atau Ctrl-C ditekan.

10.2.1 Ukuran Huruf

Sertakan `chr(15)` pada awal pencetakan untuk memperkecil ukuran huruf, dan ini cukup digunakan sekali saja. Selama printer belum dimatikan maka perintah pengecilan masih berlaku.

```
f = open("/dev/lp0", "w")
f.write( chr(15) )
f.close()
```

Setelah program di atas selesai dijalankan, perintah pengecilan masih tersimpan dalam memori printer. Jadi bila

```
$ cat /etc/hosts > /dev/lp0
```

dijalankan maka pencetakan sudah menggunakan font yang kecil.

10.2.2 Ganti Halaman

Mengganti halaman bisa juga diartikan mengeluarkan kertas dari printer. Untuk kertas *continues* mengganti halaman berarti menuju ke lembar berikutnya. Gunakan karakter "\f" digunakan untuk tujuan tersebut.

```
f = open("/dev/lp0","w")
f.write( "\f" )
f.close()
```

Karakter ASCII

`chr()` sebenarnya fungsi untuk mendapatkan karakter ASCII berdasarkan nomor yang dimasukkan, sedangkan fungsi `ord()` sebaliknya. Contoh:

```
print chr(80), ord("P")
```

Anda juga bisa mencari tahu sebenarnya "\n" dan "\f" nomor ASCII-nya berapa:

```
print chr("\n"), ord("\f")
```

10.2.3 Mencetak File

Dengan perintah `cat` di atas kita sudah dapat mencetak file. Namun untuk mengecilkan ukuran huruf, karakter nomor 15 perlu dikirimkan ke printer terlebih dahulu. Program kecil berikut akan menggabungkan fungsi keduanya:


```
import sys
import os

namafile = sys.argv[1]
printer = "/dev/lp0"

f = open(printer,"w")
f.write( chr(15) )
f.close()

perintah = "cat %s > %s &" % (namafile, printer)
os.system( perintah )
```

Simpan program di atas dengan nama `cetak.py`, lalu cobalah dengan cara berikut:

```
$ python cetak.py /etc/hosts
```

Bila melihat contoh sebelumnya, pencetakan file sebenarnya bisa sepenuhnya dilakukan dengan fungsi-fungsi pada Python, tanpa menggunakan program eksternal seperti `cat`. Cara di atas dilakukan untuk mengantisipasi kegagalan printer³ yang dapat menyebabkan program terhenti dan menunggu printer siap kembali. Perhatikan karakter `&` yang menyebabkan pencetakan tetap dilakukan meski program Python yang memanggilnya sudah selesai.

³Misalnya: kertas habis (out of paper)

Modul `sys`

Ruang lingkup modul `sys` adalah hal-hal yang berkaitan sekali dengan interpreter. Sedangkan list `argv` pada contoh di atas berisi parameter pada baris perintah (di console), sehingga `argv` itu berisi [`'cetak.py'`, `'/etc/hosts'`].

Modul `os`

Modul ini memiliki banyak fungsi yang berkaitan dengan sistem operasi. Fungsi `system()` digunakan untuk menjalankan program lain.

10.3 Direktori Aktif

Current directory atau direktori aktif pada saat program dijalankan. Jadi file `test.txt` pada baris ini

```
f = open("test.txt","w")
```

akan ditulis pada direktori aktif. Untuk mengetahui direktori apa yang sedang aktif atau bagaimana pindah ke direktori tertentu, cobalah program berikut:

```
import os
os.chdir("/tmp")
print os.getcwd()
```

Bab 11

Menangani Kesalahan - Exception

Suatu kesalahan dapat dijumpai dalam penulisan program dimana bisa kita kelompokkan menjadi dua:

1. Kesalahan penulisan sintaks yang disebut sebagai *syntax error*.
2. Kesalahan logika atau *exception*.

Yang pertama sangat mudah ditemui karena Python terlebih dahulu memeriksa kebenaran penulisan sintaks sebelum menjalankan program, seperti contoh berikut:

```
while 1
```

```
print "sip"
```

dimana akan muncul pesan kesalahan karena titik dua (:) belum disertakan dalam `while`:

```
File "pesan.py", line 1
  while 1
      ^
SyntaxError: invalid syntax
```

Sedangkan *exception* terjadi pada saat program dijalankan karena *kesalahan logika*. Misalnya perintah berikut ini:

```
1 / 0
```

yang akan menimbulkan pesan kesalahan:

```
ZeroDivisionError: integer division or modulo
```

Perhatikan contoh berikut untuk “menangkap” suatu kesalahan:

```
n = raw_input("Mencari akar n, n = ")
try:
    print float(n) ** 0.5
except:
    print n, "bukan angka"
```

Blok di dalam `try` akan “diawasi” hingga apabila terjadi kesalahan maka blok `except` dijalankan.

Bab 12

Proyek String

Paket Python menyertakan modul string yang memiliki berbagai fungsi untuk penanganan string. Meski pokok bahasannya pengolahan string, namun banyak hal baru yang juga dibahas seputar tipe data lainnya.

12.1 Membuat Nomor Baris

Bila Anda penulis buku tentang bahasa pemrograman tentu sering memberikan contoh-contoh program yang jumlah barisnya bisa mencapai ratusan. Dengan memberikan nomor baris pada contoh tersebut tentu memudahkan mereka yang hendak menulis ulang pada komputer. Oleh karena itu Anda perlu membuat program kecil untuk kebutuhan tersebut dengan con-

toh hasil sebagai berikut:

```
nomorbaris.py
-----
01| import sys
02| import string
03|
04| namafile = sys.argv[1]
05| print namafile
06| print "-" * len(namafile)
07|
08| file      = open(namafile,"r")
09| jumlahbaris = len(file.readlines())
10| lebar     = len(str(jumlahbaris))
11| file.seek(0)
12| n = 0
13| for baris in file.readlines():
14|     n = n + 1
15|     nomor = string.zfill(n, lebar)
16|     print "%s| %s" % (nomor, baris),
17| file.close()
```

Tentu saja Anda sudah tahu bagaimana isi programnya.

12.1.1 Awali Dengan Nol - `zfill()`

Fungsi `zfill()` akan memenuhi suatu angka atau string dengan huruf "0" (nol) di awalnya.

12.1.2 Penunjuk Pada File - seek()

Pemanggilan `readlines()` menyebabkan penunjuk (*pointer*) pada objek file `f` berada di akhir file. Sehingga pemanggilan keduanya hanya menghasilkan list hampa. Dengan fungsi `seek()` penunjuk tersebut dapat dialihkan ke posisi pertama lagi.

12.2 File Sebagai Tabel

Bagi Anda yang pernah membuat dokumen dalam spreadsheet mungkin pernah mengalami masalah dalam mencetak dokumen dengan printer dotmatrix. Masalah yang dimaksud adalah bahwa *dotmatrix* memiliki kecepatan yang rendah bila mencetak dalam modus grafis ketimbang mencetak dalam modus teks.

Oleh karena itu kita akan konversi dokumen tersebut ke dalam suatu file teks dimana setiap field dipisahkan dengan karakter titik koma (;).¹ File ini biasanya disebut berformat Text CSV. Contoh isinya sebagai berikut:

```
Nomor;Nama Barang;Harga
1;Duren;15000
2;Jeruk;7000
3;Rambutan;5000
4;Mangga;5500
5;Pisang;4750
6;Jambu;6500
7;Pepaya;2000
```

¹Sehingga bisa disebut sebagai *karakter pemisah*.

```
8;Nanas;1500
9;Bengkuang;2000
10;Belimbing;4000
```

Selanjutnya file ini dibaca, diolah, lalu dicetak, dengan algoritmanya adalah sebagai berikut:

1. File dibuka dan diasumsikan sebagai sebuah tabel dimana setiap baris (record) dipisahkan dengan karakter enter, dan setiap kolom dipisahkan oleh karakter pemisah.
2. Setiap kolom dianalisa: berapa jumlah karakter maksimumnya.
3. Kemudian setiap nilai dicetak ke layar yang akan “dipaskan” jumlah karakternya dengan karakter spasi hingga mencapai jumlah maksimum, tergantung posisi kolom elemen data tersebut.

Contoh hasilnya adalah sebagai berikut:

```
$ python csv.py barang.csv
```

Nomor	Nama Barang	Harga
1	Duren	15000
2	Jeruk	7000
3	Rambutan	5000
4	Mangga	5500
5	Pisang	4750
6	Jambu	6500

7	Pepaya	2000
8	Nanas	1500
9	Bengkuang	2000
10	Belimbing	4000

`csv.py` adalah program yang akan kita buat, sedangkan `buah.csv` adalah file hasil konversi dari aplikasi spreadsheet.

```
csv.py
-----
01| import string
02| import sys
03|
04| namafile = sys.argv[1]
05| pemisah = ";"
06| file = open(namafile,"r")
07| maks = {}
08| for baris in file.readlines():
09|     record = string.splitfields(baris, pemisah)
10|     kolom = -1
11|     for nilai in record:
12|         kolom = kolom + 1
13|         panjang = len(nilai)
14|         if not maks.has_key(kolom) or \
15|             panjang > maks[kolom]:
16|             maks.update( {kolom:panjang} )
17| file.seek(0)
18| for baris in file.readlines():
19|     record = string.splitfields(baris, pemisah)
```

```
20|     kolom = -1
21|     for nilai in record:
22|         kolom = kolom + 1
23|         s = string.strip(nilai)
24|         print string.ljust( s, maks[kolom] ),
25|         print
26| file.close()
```

Pemisah antar field bisa saja menggunakan karakter lain, titik koma (;) misalnya. Asalkan bisa dipastikan nilai di dalam field tidak mengandung karakter tersebut.

12.2.1 Membelah String - `splitfields()`

`splitfields()` merupakan fungsi untuk memecah suatu string menjadi list. Fungsi ini membutuhkan “string pemisah” untuk melakukan pemecahan.

12.2.2 Hapus Karakter Tak Tampak - `strip()`

`strip()` akan menghapus karakter yang tidak kelihatan seperti enter (“\n”), tab (“\t”), spasi (“ ”), dsb, dimana posisi karakter tersebut berada di paling kiri dan paling kanan suatu string. Untuk menghapus sisi kirinya saja gunakan `lstrip()`, atau sisi kanannya saja dengan `rstrip()`.

12.2.3 Rata Kiri dan Kanan - `ljust()` & `rjust()`

`ljust()` merupakan fungsi “rata kiri” yang akan memenuhi suatu string dengan karakter spasi di sebelah kanannya. Untuk

hal sebaliknya gunakan fungsi `rjust()`.

12.2.4 Kunci Pada Dictionary - `has_key()`

`maks` adalah variabel dictionary, dimana fungsi `has_key()` dipakai untuk mengetahui apakah suatu kunci terdapat pada dictionary tersebut.

Latihan Tampilkanlah angka dalam bentuk rata kanan dan terdapat pemisah ribuan. Gunakan `try` untuk mengetahui angka atau bukan.

Bagian III

Qt

Bab 13

Pendahuluan

Qt dibuat oleh Trolltech¹ dan merupakan *library*² untuk aplikasi yang menggunakan form³ dimana di dalamnya terdapat *button*, *radiobox*, *grid*⁴, dsb. Objek-objek tersebut tersedia dalam bentuk *class*⁵. Qt juga mendukung penggunaan *database server* untuk menyimpan data. Oleh karena itu library ini sangat cocok dalam pembuatan aplikasi bisnis.

Sebagai referensi tambahan, Qt juga digunakan dalam KDE - sebuah *window manager* terkemuka di lingkungan Linux.

¹www.trolltech.com

²Library: pustaka kumpulan class, konstanta, fungsi, dsb.

³Meski dengan Qt memungkinkan untuk membuat aplikasi console biasa, namun tujuan dibuatnya library ini memang untuk aplikasi berbasis form (window).

⁴Grid: bentuk tabel

⁵Class: tipe data objek

PyQt dibuat oleh Phil Thompson⁶ dan merupakan *library* Python untuk penggunaan Qt. Memungkinkannya penggunaan Qt pada Python diharapkan terbentuk aplikasi bisnis yang handal, menarik, mudah digunakan, serta memperkecil waktu pengembangan.

⁶phil@riverbankcomputing.co.uk

Bab 14

Aplikasi Pertama

Program pendek berikut akan menampilkan sebuah form seperti pada gambar 14.1. Kesederhanaannya ingin menunjukkan bagaimana membuat aplikasi Qt semudah mungkin.

```
hello.py
-----
01| from qt import *
02|
03| class FormHello(QWidget):
04|     def __init__(self):
05|         QWidget.__init__(self)
06|         self.setCaption("Hello World !")
07|         self.show()
08|
```

Gambar 14.1: Hello World !

```
09| app = QApplication([])
10| fm = FormHello()
11| app.setMainWidget(fm)
12| app.exec_loop()
```

QWidget merupakan class untuk membuat form. Form ini nantinya menjadi wadah bagi seluruh objek yang tampak (*visual class*). Sedangkan QApplication merupakan pengatur keseluruhan form yang ada dalam suatu aplikasi.¹ Cukup sebuah QApplication dalam suatu aplikasi yang akan “menunjuk” salah satu form sebagai form utama dengan fungsi `setMainWidget()`.

14.1 Berorientasi Objek

Qt merupakan library yang menggunakan teknik pemrograman berorientasi objek (*object oriented programming* disingkat OOP) yang memang menjadi salah satu fitur² Python. Sedikit mengulas tentang teknik ini, terutama seputar istilah-istilah yang digunakan.

Mirip dengan teknik pemrograman non-objek, *class* merupakan *tipe data*, sedangkan *instance* merupakan *variabel* yang sering disebut *objek* saja.

¹Sebuah aplikasi biasanya memiliki form lebih dari satu.

²Fitur: ciri, kemampuan, atau fasilitas. Dari kata: *feature*.

Pada contoh di atas, class `FormHello` merupakan keturunan (*inheritence*) dari class `QWidget`. Fungsi `__init__()` merupakan pembentuk (*constructor*) yang akan dijalankan ketika pertama kali suatu objek diciptakan, dan ini merupakan standar Python. Pemanggilan fungsi `QWidget.__init__()` berguna untuk menciptakan bentuk asli leluhur yang memang berupa sebuah *form*. Namun class `FormHello` melakukan perubahan sifat dengan memberi kalimat "Hello World !".

Secara umum manfaat dari teknik pemrograman berorientasi objek adalah menyederhanakan pembuatan program pada aplikasi yang kompleks (rumit) melalui proses penurunan sifat ini.³

14.2 Program Utama

Program utamanya sendiri dimulai saat penciptaan objek `app` yang ber-class `QApplication`, diikuti dengan penciptaan objek `fm` yang ber-class `FormHello` dimana `__init__()` mulai dijalankan, yang berarti pula objek `fm` dimuat di memori sekaligus ditampilkan di layar monitor melalui `show()`. Selanjutnya objek `app` selaku *form manager* menentukan form mana yang merupakan form utama. Tanpa penentuan ini aplikasi

³Pada teknik non objek, perubahan suatu sifat dapat dilakukan melalui parameter dalam suatu fungsi. Untuk algoritma yang sederhana hal ini masih bisa diterapkan. Tapi bila kompleksitas program sudah sedemikian besar seperti menyangkut pengaturan tampilan, mouse, keyboard, dsb, maka penerapan fungsi saja akan membuat program menjadi sangat besar dan pada akhirnya lebih menyulitkan pada saat penelusuran kesalahan (debugging).

tidak akan berhenti walaupun semua form sudah ditutup. Hal ini dimungkinkan karena `exec_loop()` memang akan menghentikan perulangan (*looping*) manakala form utama sudah ditutup.

Jadi di dalam `exec_loop()` memang terdapat semacam `while` yang akan terus dijalankan hingga form utama ditutup.

14.3 `self`

`self` sebagai parameter masukan `__init__()` mewakili `FormHello`. Setiap fungsi yang didefinisikan di dalam class setidaknya memiliki sebuah parameter yang mewakili class tersebut. Kata `self` memang merupakan perjanjian antar programmer saja yang sebenarnya bisa diganti dengan kata lainnya.

14.4 Fungsi Pada Objek

`self.setCaption()` berarti objek ber-class `FormRelasi` tersebut menggunakan fungsi `setCaption()` yang berasal dari leluhurnya. Fungsi ini memberikan nilai pada *caption*⁴ di suatu form.

Sedangkan `self.show()` berguna untuk menampilkannya. Tanpa ini, form tidak ditampilkan namun program tetap berjalan karena `exec_loop()` dijalankan. Coba saja Anda hapus `self.show()` dan jalankan kembali program tersebut. Kalau Anda kesulitan menghentikannya (tampak *hang*) lakukan saja perintah ini di console yang lain:

⁴Caption: judul form

```
$ killall python
```

`show()` dan `setCaption()` dipanggil pada saat penciptaan yang juga dapat dipanggil sesudahnya:

```
hello1.py
-----
01| from qt import *
02|
03| class FormHello(QWidget):
04|     def __init__(self):
05|         QWidget.__init__(self)
06|         self.setCaption("Hello World !")
07|
08| app = QApplication([])
09| fm = FormHello()
10| fm.show()
11| app.setMainWidget(fm)
12| app.exec_loop()
```


Bab 15

Visual Class

`QWidget` memang class untuk membuat form, form yang sederhana. Namun sebenarnya ia merupakan *leluhur* semua objek-tampak (*visual object*).¹

Dalam istilah Qt, visual object disebut sebagai *widget*. Buku ini juga menggunakan istilah yang sama untuk menyebut objek-tampak, yaitu `QWidget` dan keturunannya.

15.1 Buku Alamat

Berikut ini form yang “diniatkan” untuk pengisian buku alamat. Anda akan diperkenalkan dengan berbagai class untuk mema-

¹Lihat kembali halaman 84 tentang istilah object & class.

sukkan data (*editor*), menampilkan tulisan (*label*), maupun untuk melaksanakan perintah tertentu (*button*).

```
relasi1.py
-----
01| from qt import *
02|
03| class FormRelasi(QWidget):
04|     def __init__(self):
05|         QWidget.__init__(self)
06|         self.resize(268,194)
07|         self.setCaption("Relasi")
08|         labelNama = QLabel(self)
09|         labelNama.setGeometry(10,10,65,20)
10|         labelNama.setText("Nama")
11|         labelAlamat = QLabel(self)
12|         labelAlamat.setGeometry(10,40,65,20)
13|         labelAlamat.setText("Alamat")
14|         self.editNama = QLineEdit(self)
15|         self.editNama.setGeometry(90,10,160,20)
16|         self.editAlamat = QTextEdit(self)
17|         self.editAlamat.setGeometry(90,40,160,102)
18|         buttonSimpan = QPushButton(self)
19|         buttonSimpan.setGeometry(90,150,80,24)
20|         buttonSimpan.setText("Simpan")
21|         self.show()
22|
23| app = QApplication([])
24| fm = FormRelasi()
```



```
25| app.setMainWidget(fm)
26| app.exec_loop()
27| print "Nama", fm.editNama.text()
28| print "Alamat", fm.editAlamat.text()
```

Class yang dimaksud adalah:

QLabel untuk menampilkan tulisan. Sifatnya *read-only*, sehingga pengguna (*user*) tidak dapat berinteraksi langsung.

QLineEdit tempat menuliskan masukan (*input*) dari pengguna. Pada contoh berikut dipakai untuk menampung Nama.

QTextEdit mirip **QLineEdit**, tetapi dengan kemampuan lebih dari satu baris masukan (*multiline*), misalnya untuk menampung Alamat.

QPushButton tombol yang biasanya dipakai untuk perintah tertentu. Nanti dipakai untuk perintah menyimpan data.

Sedangkan fungsi berikut ini selain dimiliki **QWidget** juga terwarisi oleh keempat class tersebut.

resize() mengubah ukuran form yang membutuhkan masukan berupa dua bilangan bulat yang mewakili panjang dan lebar form dalam satuan *pixel*.

Gambar 15.1: Form Alamat

`setGeometry()` menentukan posisi dan ukuran widget. Dua angka pertama merupakan posisi yang ditentukan dari sudut kiri atas form. Sedangkan dua angka berikutnya merupakan ukuran (besarnya) objek label tersebut.

Pengisian Nama menggunakan class `QLineEdit` yang hanya mampu menampung masukan sebanyak satu baris saja, karena memang kebutuhannya seperti itu. Berbeda dengan Alamat yang menggunakan `QTextEdit` yang sanggup lebih dari satu baris, bahkan dengan kemampuan berganti baris secara otomatis (*autowrap*).

`setText()` pada `QLabel` dan `QPushButton` merupakan fungsi untuk mengisikan teks. Fungsi yang sama juga terdapat pada objek lain yang memiliki tempat untuk tulisan, seperti halnya pada `QLineEdit` dan `QTextEdit`.

Pencetakan Nama dan Alamat pada akhir program ingin menunjukkan bagaimana mendapatkan nilai yang sudah dimasukkan pemakai. Seperti juga `setText()`, `text()` dimiliki oleh objek lain yang memiliki tempat untuk tulisan, seperti `QLabel` dan `QPushButton`.

15.1.1 Parent dan Child

Kembali membahas istilah. *Parent* merupakan wadah² objek. Objek inilah yang disebut sebagai *child*. Pada contoh di atas dikatakan bahwa `labelNama` berwadah `FormRelasi`. Perhatikan baris:

```
labelNama = QLabel(self)
```

dimana `self` merupakan wadah bagi `QLabel` yang baru dibuat dengan nama variabel `labelNama`.

15.1.2 Parent dan Owner

Pembuatan `editNama` dan `editAlamat` menggunakan kata `self` di depannya. Ini artinya class `FormRelasi` memiliki dua objek tersebut. Atau dikatakan `FormRelasi` sebagai pemilik (*owner*) keduanya. Sedangkan `labelNama`, `labelAlamat`, dan `tombol-Simpan` bukan milik class `FormRelasi` tetapi milik fungsi `__init__()`,³ namun tetap ber-wadah pada `FormRelasi`. Jadi,

```
self.editNama = ...
```

berarti variabel `editNama` dimiliki `self`, sedangkan,

```
... = QLineEdit(self)
```

berarti `QLineEdit` tersebut berwadah pada `self`.

Sehingga bisa saja suatu objek dimiliki objek A namun berwadah objek B, meski hal ini jarang diterapkan.

²Lihat pembahasan mengenai wadah di halaman 123.

³Lihat Kepemilikan Variabel pada Bab Fungsi

15.1.3 Dengan Atau Tanpa self

Bagaimana kita menentukan suatu objek dimiliki oleh class atau cukup dimiliki oleh fungsi didalamnya? Kalau mengambil “jalan aman” sebaiknya semua dimiliki oleh class (dengan awalan `self`). Namun kalau Anda merasa repot karena terlalu banyak kata `self` yang ditulis gunakan pedoman berikut ini:

Gunakan awalan `self` apabila objek tersebut akan dipakai oleh blok lain diluar fungsinya yang membuatnya.

Kebanyakan objek bisa diterapkan dengan pedoman di atas, namun ada beberapa yang bisa menyebabkan *segmentation fault*.⁴

15.1.4 Modul qt

Modul `qt` berisi banyak class dan fungsi yang sering dipakai pada contoh-contoh program berikutnya. Oleh karena itu kita hindari penulisan

```
import qt
```

dan sebagai gantinya gunakan

```
from qt import *
```

agar tidak perlu menyebutkan nama modul didepan class yang digunakan.

⁴Segmentation fault: kesalahan program yang sering disebabkan masalah manajemen memori. Kesalahan ini cukup fatal karena tidak terdeteksi oleh programnya itu sendiri.

15.1.5 String Atau QString

QString digunakan untuk penanganan string, dan tergolong sebagai *non-visual class*⁵. Objek ini banyak terdapat pada berbagai properti objek seperti `caption()`, `text()`, dsb. Juga sebagai parameter masukan `setCaption()`, `setText()`, dst.

Kita bisa menuliskan baris berikut:

```
self.editNama.setText( QString("Nuryadi") )
```

sama hasilnya dengan:

```
self.editNama.setText( "Nuryadi" )
```

dimana editor bertipe `QLineEdit`, misalnya. Namun jangan harapkan hal yang sama ketika kita ingin mengolah `text()` dengan perintah standar Python lainnya mengenai string, misalnya penggunaan operator tambah (+) seperti:

```
s = "Nama " + self.editNama.text()
```

karena ini merupakan kesalahan dimana tipe string ditambahkan dengan `QString`. Untuk mengatasinya gunakan proses format string:

```
s = "Nama %s" % self.editNama.text()
```

atau bisa juga dengan

```
s = "Nama " + str(self.editNama.text())
```

⁵Non-visual class: bukan keturunan `QWidget`.

dan sampai di sini `s` tetap bertipe string dan bukan `QString`.

Bila Anda menggunakan `print` untuk mencetak `QString` maka hal ini tidak perlu dilakukan dimana penggabungan bisa dengan koma, seperti contoh di atas. Karena sifat `print` sudah seperti fungsi `str()` maupun *string formatting* %.

15.2 Sinyal

`FormRelasi` sudah dapat ditampilkan, namun tombol Simpannya belum berfungsi. Logikanya penyimpanan data dilakukan pada saat tombol tersebut di-klik. *Saat* inilah yang disebut sebagai *event*.⁶ Pada saat kejadiannya (tombol di-klik), tombol tersebut mengirimkan sinyal (*signal*). Sinyal ini dapat dimanfaatkan untuk menyisipkan fungsi penyimpanan data.

Berhubung sampai di sini belum dibahas tentang database, maka data disimpan dalam sebuah file bernama `relasi.txt` dengan bentuk sebagai berikut:

```
NAMA: <nama>
ALAMAT:
<alamat>
---
```

File *plaintext* ini merupakan file yang bisa dibaca dengan text editor biasa.

⁶Karena proses tersebut maka pemrograman berorientasi objek kerap disebut memiliki fitur event driven programming (pengendalian berdasarkan kejadian).

```
relasi2.py
-----
01| from qt import *
02| import os
03|
04| class FormRelasi(QWidget):
05|     def __init__(self):
06|         QWidget.__init__(self)
07|         self.resize(268,194)
08|         self.setCaption("Relasi")
09|         labelNama = QLabel(self)
10|         labelNama.setGeometry(10,10,65,20)
11|         labelNama.setText("Nama")
12|         labelAlamat = QLabel(self)
13|         labelAlamat.setGeometry(10,40,65,20)
14|         labelAlamat.setText("Alamat")
15|         self.editNama = QLineEdit(self)
16|         self.editNama.setGeometry(90,10,160,20)
17|         self.editAlamat = QTextEdit(self)
18|         self.editAlamat.setGeometry(90,40,160,102)
19|         buttonSimpan = QPushButton(self)
20|         buttonSimpan.setGeometry(90,150,80,24)
21|         buttonSimpan.setText("Simpan")
22|         self.show()
23|         self.connect( buttonSimpan,
24|             SIGNAL("clicked()"), self.tombolKlik )
25|
26|     def tombolKlik(self):
27|         rec = "NAMA: %s\nALAMAT: \n%s\n---\n" % \
```

```
28|         (self.editNama.text(),
29|          self.editAlamat.text())
30|         file = "relasi.txt"
31|         fileSementara = "~" + file
32|         os.system("touch " + file)
33|         os.system("touch " + fileSementara)
34|         f = open(file,"r")
35|         fSementara = open(fileSementara, "w")
36|         fSementara.write(f.read() + rec)
37|         fSementara.close()
38|         f.close()
39|         os.rename(fileSementara, file)
40|
41|     app = QApplication([])
42|     fm = FormRelasi()
43|     app.setMainWidget(fm)
44|     app.exec_loop()
```

Fungsi `connect()` di atas menyatakan bahwa:

Jalankan `tombolKlik()` pada saat `buttonSimpan` menjalankan `clicked()`

Karena proses itulah teknik ini disebut sebagai *penyisipan fungsi ke dalam slot yang telah disediakan*. Tanpa slot yang telah diberikan para pengembang Qt, maka Anda tidak dapat menyisipkan suatu fungsi ke dalam fungsi lain yang dimiliki suatu objek. Jika itu terjadi, berarti yang perlu dilakukan adalah mendefinisikan suatu class baru yang merupakan turunan dari class objek yang Anda kehendaki.

`connect()` sebenarnya milik class `QObject` yang merupakan leluhur semua class, baik yang tampak maupun tidak.

Latihan Pada saat `buttonSimpan` di-klik bukan hanya perintah penyimpanan yang dijalankan, melainkan diikuti dengan pengosongan `editNama` dan `editAlamat`, lalu secara otomatis kursor sudah berada pada `editNama`.

Petunjuk Gunakan fungsi `clear()` untuk mengosongkan, dan `setFocus()` untuk memindahkan kursor pada objek yang dimaksud. Tujuannya adalah memudahkan pengguna untuk mengisikan data berikutnya.

Modul os

`rename()` adalah fungsi untuk mengubah atau memindahkan file. Sedangkan `touch` yang dijalankan dengan fungsi `system()` merupakan perintah Linux yang berfungsi memeriksa keberadaan suatu file. Jika tidak ada ia akan membuatnya namun tidak berisi apa-apa.

15.2.1 Keterkaitan Dengan C++

Fungsi `SIGNAL()` pada program di atas merupakan penghubung antara Python dengan library Qt yang dibangun dengan bahasa C++. Python memang dikatakan sebagai antarmuka bahasa C⁷ (*C interface*). Hal ini bisa juga dikatakan library yang

⁷C dibangun sebelum C++. Namun C++ dapat mengenal dengan baik seluruh perintah C.

ditulis untuk C dapat diakses oleh Python. Praktis Python menawarkan kekayaan fitur C dan C++.

Anda dapat menggunakan dokumentasi Qt yang ditulis untuk C++ sebagai acuan untuk mempelajari penggunaan “Qt pada Python” (baca: PytQt). Tentunya Anda perlu menyesuaikannya dengan gaya bahasa Python pada bagian-bagian tertentu.

15.2.2 Sinyal atau Event

Sinyal (*signal*) berkaitan dengan suatu kejadian (*event*) dalam objek dimana sinyal yang muncul kemudian dihubungkan dengan suatu fungsi.

Istilah event dalam Qt sebenarnya juga berupa kejadian yang diwakili oleh suatu fungsi. Perbedaannya dengan sinyal adalah event merupakan fungsi yang bisa diganti (*override*⁸) sifatnya oleh turunan suatu objek. Contohnya adalah `__init__()` yang terjadi pada saat suatu objek diciptakan.

Berikut ini contoh lain penggunaan sinyal, yaitu turunan `QLineEdit` yang akan memperbesar huruf.

```
linecase.py
-----
01| from qt import *
02|
03| class LineCase(QLineEdit):
04|     Normal = 0
```

⁸Override:ditulis ulang atau dalam dokumentasi Qt disebut sebagai *reimplementation*.

```
05|     Upper = 1
06|     Lower = 2
07|
08|     def __init__(self, parent, case=0):
09|         QLineEdit.__init__(self, parent)
10|         self.case = case
11|         self._OnSet = 0
12|         self._panjang = 0
13|         self.teksBerubah = None
14|         self.connect( self,
15|             SIGNAL( "textChanged( const QString& )" ),
16|             self.berubah)
17|
18|     def berubah(self, t):
19|         if self._OnSet: return
20|         if self.case != self.Normal and \
21|             self._panjang < t.length():
22|             self._OnSet = 1
23|             if self.case == self.Upper:
24|                 self.setText( self.text().upper() )
25|             else:
26|                 self.setText( self.text().lower() )
27|             self._OnSet = 0
28|             self._panjang = t.length()
29|             if self.teksBerubah: self.teksBerubah( self )
30|
31| if __name__ == "__main__":
32|     app = QApplication([])
33|     fm = LineCase(None, LineCase.Upper)
```

```
34|     fm.show()
35|     app.setMainWidget(fm)
36|     app.exec_loop()
```

`linecase.py` dapat dijalankan sebagai modul maupun program utama. Hal ini dimungkinkan karena blok utama program akan memeriksa terlebih dahulu apakah ia merupakan program utama atau sebagai modul (lihat `__name__`)

15.2.3 Membuat Event

Sinyal `textChanged()` pada `LineCase` sudah digunakan, padahal Anda ingin menyisipkan fungsi lain pada saat teks berubah. Cara pertama bisa dengan menuliskan ulang (*reimplementation*) `LineCase` menjadi class baru dan mengubah source fungsi `berubah()`. Cara ini sudah dicontohkan oleh `__init__()`. Sedangkan cara kedua adalah dengan menyiapkan *variabel event* dan (kalau ada) menjalankannya di dalam fungsi `berubah()` tadi. Cara ini memang telah diterapkan `LineCase`, yaitu dengan adanya variabel event `teksBerubah` dimana baris

```
if self.teksBerubah: self.teksBerubah( self )
```

menunjukkan bahwa variabel event `teksBerubah` akan diperiksa apakah nilainya sudah diganti dengan yang lain. Jika ya, maka penggantinya harus berupa event dengan sebuah parameter masukan, dikarenakan ia dipanggil dengan sebuah parameter di dalamnya. Perhatikan bagian

```
self.teksBerubah( self )
```

Lebih jelasnya jalankan contoh berikut ini:

```
linecase1.py
-----
01| from qt import *
02| from linecase import LineCase
03|
04| class FormCase(QWidget):
05|     def __init__(self):
06|         QWidget.__init__(self)
07|         self.line = LineCase(self, LineCase.Lower)
08|         self.line.teksBerubah = self.ubahCaption
09|         self.show()
10|
11|     def ubahCaption(self, w):
12|         self.setCaption( w.text() )
13|
14| app = QApplication([])
15| fm = FormCase()
16| app.setMainWidget(fm)
17| app.exec_loop()
```

`ubahCaption()` harus memiliki parameter selain `self`⁹ yang jumlahnya sebanyak yang ditetapkan variabel event `teksBerubah` pada `LineCase`. Contoh di atas menunjukkan parameter tersebut bertipe `LineCase`, sehingga fungsi `text()` bisa digunakan.

⁹Ingat, `self` harus disertakan dalam setiap fungsi di dalam class. Bila fungsi didefinisikan di luar class, maka - tentu - `self` tidak perlu disertakan.

Pahami benar-benar tentang event ini, karena pada contoh-contoh selanjutnya dimanfaatkan seoptimal mungkin untuk meminimalkan kode program

15.3 Hiasan

Terlepas dari alur suatu sistem, ada baiknya kita mengetahui beberapa hal untuk memperbagus tampilan dengan tujuan aplikasi yang digunakan bisa lebih informatif.

15.3.1 Font - QFont

Font (QFont) merupakan atribut karakter, misalnya:

Jenis Arial, Helvetica, Courier

Italic *Miring*

Underline Garis bawah

Bold **Tebal**

`font.py` berikut akan menampilkan sebuah `QLabel` yang isinya bergantung dari hasil masukan dari `QLineEdit`.

```
font.py
-----
01| from qt import *
02|
03| class FormFont(QWidget):
```

```
04| def __init__(self):
05|     QWidget.__init__(self)
06|     self.setCaption("Font")
07|     self.label = QLabel(self)
08|     self.label.setAutoResize(1)
09|     self.label.setText( "Tulis saja" )
10|     font = QFont(self.label.font())
11|     font.setFamily("Courier [Adobe]")
12|     font.setPointSize(48)
13|     font.setBold(1)
14|     font.setItalic(1)
15|     font.setUnderline(1)
16|     self.label.setFont(font)
17|     self.teks = QLineEdit(self)
18|     self.teks.setGeometry( 10, 80, 106, 20 )
19|     self.show()
20|     self.connect( self.teks, SIGNAL( "textChanged( \
21|         const QString&)" ), self.ubahLabel)
22|
23|     def ubahLabel(self, teks):
24|         self.label.setText( teks )
25|
26| app = QApplication([])
27| fm = FormFont()
28| app.setMainWidget(fm)
29| app.exec_loop()
```

Pada dasarnya `setFont()` merupakan milik `QWidget` yang otomatis dapat digunakan juga oleh keturunannya seperti `QLabel` di

atas.

Penjelasan mengenai fungsi pada QFont di atas adalah:

font() mengembalikan QFont, berisi informasi font suatu widget.

setFamily() menentukan jenis font. Nama font yang tercantum berdasarkan font yang terpasang pada sistem. Untuk mengetahuinya bisa dengan **kwrite**. Pilih menu *Settings | Configure Editor | Fonts*.

setPointSize() menentukan ukuran. Angka di dalamnya juga perlu disesuaikan dengan yang terdaftar, tergantung dari jenisnya.

setBold() menentukan apakah font akan ditebalkan. Nilai masukannya hanyalah *boolean* (logika) 1 (ditebalkan) atau 0 (tidak ditebalkan).

setItalic() memiringkan font. Nilai masukannya juga boolean.

setUnderline() pemberian garis bawah. Nilai masukannya juga boolean.

Di luar masalah font, program di atas cukup menarik karena melibatkan **setAutoResize()** yang menentukan apakah ukuran **QLabel** akan mengikuti panjang tulisan. Serta penggunaan sinyal **textChanged()** yang terpicu saat teks pada **QLineEdit** berubah.

15.3.2 Warna - QColor

QColor merupakan “class warna” yang telah menyediakan nama-nama warna yang siap pakai, yaitu: black, white, darkGray, gray, lightGray, red, green, blue, cyan, magenta, yellow, darkRed, darkGreen, darkBlue, darkCyan, darkMagenta, dan darkYellow.

warna.py berikut melanjutkan font.py, dengan warna label yang telah diubah.

```
warna.py
-----
01| from qt import *
02|
03| class FormWarna(QWidget):
04|     def __init__(self):
05|         QWidget.__init__(self)
06|         self.setCaption("Warna")
07|         self.label = QLabel(self)
08|         self.label.setAutoResize(1)
09|         self.label.setPaletteForegroundColor(
10|             QColor("yellow"))
11|         self.label.setPaletteBackgroundColor(
12|             QColor("blue"))
13|         self.label.setText( "Tulis saja" )
14|         self.teks = QLineEdit(self)
15|         self.teks.setGeometry( 10, 20, 106, 20 )
16|         self.show()
17|         self.connect( self.teks, SIGNAL( "textChanged( \
18|             const QString&)" ), self.ubahLabel)
```

```

19|
20|     def ubahLabel(self, teks):
21|         self.label.setText( teks )
22|
23| app = QApplication([])
24| fm = FormWarna()
25| app.setMainWidget(fm)
26| app.exec_loop()

```

`setPaletteForegroundColor()` menentukan warna pada latar depan objek, sedangkan `setPaletteBackgroundColor()` pada latar belakangnya. Keduanya dimiliki `QWidget`.

`QColor` juga dapat menerima masukan berupa tiga bilangan bulat dengan nilai antara 0 hingga 255 yang mewakili komponen inti warna: merah (*red*), hijau (*green*), dan biru (*blue*) atau sering disebut sebagai RGB. Sehingga perintah seperti ini:

```
setPaletteForegroundColor(QColor("white"))
```

bisa juga ditulis demikian:

```
setPaletteForegroundColor(QColor(255,255,255))
```

15.3.3 Parent Berpengaruh

Font dan warna pada parent bisa mempengaruhi objek lain yang menjadi child-nya. Cobalah mengubah warna `FormWarna`: dengan mengganti

Gambar 15.2: Checkbox

```
self.label.setPaletteBackgroundColor(QColor("blue"))
```

menjadi

```
self.setPaletteBackgroundColor(QColor("blue"))
```

15.4 Ya Atau Tidak - QCheckBox

Checkbox (QCheckBox) merupakan objek yang dapat menerima masukan seperti QLineEdit dengan sifat khas berikut:

1. Bersifat *toggle* (ya atau tidak)
2. Memiliki sebuah kotak isian yang apabila di-klik bermakna ya, dan bila di-klik sekali lagi berarti tidak. Keduanya dibedakan dengan sebuah tanda *checked* (benar).
3. Memiliki label sebagai keterangan dari kotak isian tersebut.

`checkbox.py` menunjukkan bagaimana QCheckBox dapat digunakan untuk mengubah warna form.

```
checkbox.py
-----
01| from qt import *
02|
03| class FormCheckBox(QWidget):
04|     def __init__(self):
05|         QWidget.__init__(self)
06|         self.setCaption("Check Box")
07|         self.checkBox = QCheckBox( self )
08|         self.checkBox.setText( "Putihkan" )
09|         self.connect( self.checkBox, SIGNAL( "clicked()" ),
10|             self.putihkan )
11|         self.warnaAsli = self.paletteBackgroundColor()
12|         self.show()
13|
14|     def putihkan(self):
15|         if self.checkBox.isChecked():
16|             self.setPaletteBackgroundColor(QColor(
17|                 255,255,255))
18|         else:
19|             self.setPaletteBackgroundColor(self.warnaAsli)
20|
21| app = QApplication([])
22| fm = FormCheckBox()
23| app.setMainWidget(fm)
24| app.exec_loop()
```

isChecked() mengembalikan boolean dimana 1 berarti ya.

Pada sebuah form isian dalam suatu aplikasi, `QCheckBox` dapat dimanfaatkan untuk beberapa contoh berikut ini:

1. Dalam sistem kepegawaian, seorang pegawai diikutsertakan dalam asuransi kesehatan atau tidak.
2. Pada sistem KTP (Kartu Tanda Penduduk), seseorang merupakan WNI (Warga Negara Indonesia) atau WNA (Warga Negara Asing).
3. Dalam sebuah form pengaturan (*setting*), suatu proses akan dilakukan secara otomatis atau tidak.

Sebagai pedomanpraktis:

Gunakan checkbox apabila hanya ada dua pilihan.

15.5 Pilih Salah Satu - QRadioButton

Radiobutton (`QRadioButton`) merupakan *objek-pilihan* yang lazimnya terdiri dari beberapa objek yang tergabung dalam `QButtonGroup`. Meski di dalam `QButtonGroup` terdapat beberapa pilihan, namun hanya satu saja yang bisa dipilih.

Contoh berikut berisi daftar golongan darah yang bisa dipilih dimana hal yang ingin diungkapkan adalah:

1. Bagaimana mengetahui pilihan.
2. Membatalkan pilihan (*reset*).

```
radiobutton1.py
-----
01| from qt import *
02|
03| class FormRadioButton(QWidget):
04|     def __init__(self):
05|         QWidget.__init__(self)
06|         self.golDarah = QButtonGroup( self )
07|         self.golDarah.setTitle( "Gol. Darah" )
08|         self.golDarah.setGeometry( 20, 20, 100, 120 )
09|         self.golDarah.setPaletteBackgroundColor(
10|             QColor( "green" ) )
11|         self.golA = QRadioButton( self.golDarah )
12|         self.golA.setText( "A" )
13|         self.golA.setGeometry( 5, 20, 50, 20 )
14|         self.golB = QRadioButton( self.golDarah )
15|         self.golB.setText( "B" )
16|         self.golB.setGeometry( 5, 40, 50, 20 )
17|         self.golB = QRadioButton( self.golDarah )
18|         self.golB.setText( "AB" )
19|         self.golB.setGeometry( 5, 60, 50, 20 )
20|         self.golB = QRadioButton( self.golDarah )
21|         self.golB.setText( "0" )
22|         self.golB.setGeometry( 5, 80, 50, 20 )
23|         self.setCaption( "Ada %d golongan darah" %
24|             self.golDarah.count() )
25|         self.btReset = QPushButton( self )
26|         self.btReset.setText( "Reset" )
27|         self.btReset.move( 5, 150 )
```

```
28|     self.show()
29|     self.connect( self.golDarah,
30|         SIGNAL("clicked(int)", self.golDarahKlik )
31|     self.connect( self.btReset, SIGNAL("clicked()"),
32|         self.btResetKlik )
33|
34|     def golDarahKlik(self, index):
35|         self.setCaption( "Golongan %s (%d) dipilih" %
36|             ( self.golDarah.find(index).text(), index ) )
37|
38|     def btResetKlik(self):
39|         if self.golDarah.selected():
40|             self.golDarah.selected().setChecked( 0 )
41|             self.setCaption( "Tidak ada yang dipilih" )
42|
43| app = QApplication([])
44| fm = FormRadioButton()
45| app.setMainWidget(fm)
46| app.exec_loop()
```

Berikut ini penjelasan fungsi dan sinyal pada `QButtonGroup` di atas:

`selected()` mengembalikan radiobutton yang terpilih. Bila belum ada, ia mengembalikan `None`¹⁰.

`count()` mendapatkan jumlah radiobutton dalam `QButtonGroup`.

¹⁰None: objek hampa

`find()` mendapatkan radiobutton pada nomor index tertentu.

`clicked()` sinyal yang muncul saat radiobutton di-klik. Sinyal ini menyertakan nomor index radiobutton tersebut.

`QRadioButton` sebenarnya keturunan `QPushButton` - leluhur semua objek yang bersifat tombol seperti `QCheckBox`, `QPushButton`, `QToolButton`, dan `QRadioButton` sendiri. Fungsi seperti `setText()` dan `text()` berasal dari `QPushButton`. Kecuali `setChecked()` yang menentukan apakah radiobutton terpilih atau tidak, dimana nilai masukan 0 berarti tidak dipilih, sedangkan 1 berarti terpilih. Pada contoh di atas digunakan untuk me-*reset* `QButtonGroup`.

Anda juga bisa mengganti tombol Reset dengan sebuah pilihan baru pada radiobutton yang berisi “Tidak tahu” dimana pada saat pertama ditampilkan menjadi nilai default.¹¹

Meningkatkan Fleksibilitas

Mayoritas kita mungkin hanya tahu bahwa golongan darah itu A, B, AB, dan O saja. Padahal mungkin - dan bisa jadi - ada golongan darah lainnya di dunia ini. Bila itu terjadi, maka `radiobutton1.py` memiliki kelemahan dalam hal fleksibilitas, karena untuk menambah golongan darah harus mengubah source program yang cukup banyak, yaitu:

1. Mendefinisikan objek baru, misalnya `self.golX`.

¹¹Default: nilai standar atau nilai pertama

Gambar 15.3: Radiobutton

2. Menentukan lebar radiobutton, karena bila daftar pilihan bertambah maka lebarnya pun juga harus bertambah.
3. Menentukan koordinat setiap objek dalam radiobutton.

Berikut ini rencana perbaikannya:

1. Daftar golongan darah diletakkan dalam list (array).
2. Pembuatan objek golongan darah menggunakan perulangan (looping), sehingga koordinatnya ditentukan dengan suatu rumusan.
3. Tombol Reset diiadakan, dan sebagai gantinya radiobutton ditambah daftar baru yaitu “Tidak tahu” yang bermakna operator belum dapat mencatat golongan darah orang yang dimaksud.¹² “Tidak tahu” diletakkan di paling atas dan merupakan nilai default pada saat pertama kali program dijalankan.

```
radiobutton2.py
-----
01| from qt import *
02|
```

¹²Anggap saja radiobutton dalam program ini merupakan bagian dari suatu form pengisian identitas diri, nantinya.

```
03| class FormRadioButton(QWidget):
04|     def __init__(self):
05|         QWidget.__init__(self)
06|         daftarGol = ["Tidak tahu", "A", "B", "AB", "0"]
07|         # layout
08|         lebar = 20
09|         jmlGol = len(daftarGol)
10|         self.golDarah = QButtonGroup( self )
11|         self.golDarah.setTitle( "Gol. Darah" )
12|         self.golDarah.setGeometry(20,20,100,
13|             (jmlGol+2)*lebar)
14|         self.golDarah.setPaletteBackgroundColor(
15|             QColor("green"))
16|         i = 1
17|         for gol in daftarGol:
18|             rb = QRadioButton( self.golDarah )
19|             rb.setText( gol )
20|             rb.setGeometry( 5, i * lebar, 80, lebar )
21|             i = i + 1
22|         self.golDarah.setButton(0) # default
23|         self.setCaption( "Ada %d golongan darah" %
24|             (self.golDarah.count()-1) )
25|         self.show()
26|
27|         self.connect( self.golDarah,
28|             SIGNAL("clicked(int)"), self.golDarahKlik )
29|
30|     def golDarahKlik(self, index):
31|         self.setCaption( "Golongan %s (%d) dipilih" %
```

```
32|         ( self.golDarah.find(index).text(), index ) )
33|
34| app = QApplication([])
35| fm = FormRadioButton()
36| app.setMainWidget(fm)
37| app.exec_loop()
```

Jadi jumlah objek-pilihan tergantung dari jumlah datanya dimana penataannya menggunakan suatu rumus yang diletakkan dalam perulangan `for`. Penataan otomatis ini juga dapat dilakukan dengan cara lain, yaitu menggunakan *layouter*.¹³

`setButton()` menentukan radiobutton mana yang terpilih (*selected*). Pada contoh di atas fungsi ini dipakai untuk menentukan pilihan default .

Dari sifat-sifat radiobutton yang sudah dibahas sebelumnya, catatan berikut bisa Anda jadikan pedoman:

Gunakan radiobutton untuk daftar pilih berjumlah sedikit dan/atau tidak ada perubahan pada saat runtime.¹⁴

15.6 Daftar Fluktuatif - QComboBox

Sebagaimana radiobutton, combobox (`QComboBox`) juga merupakan daftar pilih. Perbedaannya adalah bahwa combobox lebih fleksibel dalam hal isi dimana daftar pilihannya dapat ditambah

¹³Lihat halaman 131.

¹⁴Runtime: saat program berjalan

Gambar 15.4: Combobox

atau dihapus tanpa perlu mengatur kembali tata letak dari setiap pilihan, karena ia merupakan kombinasi antara tombol dan *popup-list*.

Keuntungan lain penggunaan combobox adalah:

1. Daftar pilihannya dapat diurutkan secara *ascending* maupun *descending*.
2. Memiliki editor (`QLineEdit`).
3. Bila editor diaktifkan, maka fitur *autocompletion* dapat difungsikan, yaitu suatu fitur yang memudahkan pencarian. Pengetikan sebuah huruf saja - misal A - maka combobox akan melengkapinya sesuai dengan daftar pilih yang ada, misalnya Apel. Bila dilanjutkan dengan penekanan Enter maka combobox menganggap Apel telah dipilih yang mana hal ini sama saja dengan meng-klik pilihan Apel.

Lebih jelasnya jalankan `combobox.py` dan cobalah fitur tersebut.

```
combobox.py
-----
01| from qt import *
02|
03| class FormComboBox(QWidget):
```

```
04| def __init__(self):
05|     QWidget.__init__(self)
06|     self.setCaption("ComboBox")
07|     daftarBuah = ["Pisang", "Jeruk", "Apel",
08|                  "Mangga", "Pepaya", "Nanas", "Jambu"]
09|     self.buah = QComboBox(self)
10|     self.buah.insertStrList( daftarBuah )
11|     self.buah.setEditable(1)
12|     self.buah.setAutoCompletion(1)
13|     self.buah.listBox().sort()
14|     self.setCaption( "Ada %d pilihan buah" %
15|                     self.buah.count() )
16|     self.show()
17|     self.connect( self.buah, SIGNAL("activated(int)"),
18|                  self.buahKlik )
19|     self.connect( self.buah, SIGNAL("highlighted(int)"),
20|                  self.buahHighlighted )
21|
22| def buahKlik(self, index):
23|     self.setCaption( "Buah %s (%d) dipilih" %
24|                     ( self.buah.currentText(), index ) )
25|
26| def buahHighlighted(self, index):
27|     self.setCaption( "Buah %s (%d) tersorot" %
28|                     ( self.buah.text( index ), index ) )
29|
30| app = QApplication([])
31| fm = FormComboBox()
32| app.setMainWidget(fm)
```

```
33| app.exec_loop()
```

Adapun penjelasan mengenai fungsi `QComboBox` yang dipakai adalah:

`insertStrList()` menambah daftar pilih dengan nilai masukan berupa list.

`setEditable()` mengaktifkan editor.

`setAutoCompletion()` mengaktifkan fitur autocompletion.

`currentText()` mendapatkan teks yang sedang dipilih. Untuk mendapatkan nomor index-nya gunakan `currentItem()`.

`text()` mendapatkan teks pada nomor index tertentu.

`listBox()` mendapatkan listbox (`QListBox`) yang merupakan popup-list bagi combobox.

`count()` jumlah data dalam daftar.

Seperti biasa, sebagai pedoman penggunaan combobox:

Gunakan combobox untuk daftar pilih yang banyak dan/atau kerap berubah pada saat runtime.

15.7 Listbox

Listbox mirip dengan combobox yang dapat memuat banyak pilihan dengan kemampuan *scrolling*¹⁵. Perbedaannya ia tidak

¹⁵Scrolling: melihat isi yang ukurannya melebihi ukuran objek penampungnya.

Gambar 15.5: Listbox

memiliki editor, namun memungkinkan pemakai untuk memilih lebih dari satu pilihan. `listbox1.py` berikut akan menampilkan dua listbox kiri dan kanan yang apabila dilakukan klik ganda (*double-click*¹⁶) pada yang kiri maka pilihan tersebut akan berpindah ke yang kanan, begitu pula sebaliknya.

```
listbox1.py
-----
01| from qt import *
02|
03| class FormListBox(QWidget):
04|     def __init__(self):
05|         QWidget.__init__(self)
06|         self.setCaption("ListBox")
07|         daftarBuah = ["Pisang", "Jeruk", "Apel",
08|                       "Mangga", "Pepaya", "Nanas", "Jambu"]
09|         label = QLabel(self)
10|         label.move( 10, 10 )
11|         label.setText( "Buah-buahan" )
12|         self.buah = QListBox(self)
13|         self.buah.setGeometry( 10, 40, 140, 120 )
14|         self.buah.insertStrList( daftarBuah )
15|         self.buah.sort()
```

¹⁶Double-click: klik tombol kiri mouse sebanyak dua kali secara cepat.

```
16|     label = QLabel(self)
17|     label.move( 170, 10 )
18|     label.setText( "Terpilih" )
19|     self.terpilih = QListBox(self)
20|     self.terpilih.setGeometry( 170, 40, 140, 120 )
21|     label = QLabel(self)
22|     label.setGeometry( 10, 160, 200, 20 )
23|     label.setText("Lakukan klik ganda (double click)")
24|     self.show()
25|     self.connect(self.buah, SIGNAL("selected(int)"),
26|                 self.buahSelected)
27|     self.connect(self.terpilih, SIGNAL("selected(int)"),
28|                 self.terpilihSelected)
29|
30|     def buahSelected(self, index):
31|         self.terpilih.insertItem( self.buah.text(index))
32|         self.terpilih.sort()
33|         self.buah.removeItem( index )
34|
35|     def terpilihSelected(self, index):
36|         self.buah.insertItem( self.terpilih.text(index))
37|         self.buah.sort()
38|         self.terpilih.removeItem( index )
39|
40| app = QApplication([])
41| fm = FormListBox()
42| app.setMainWidget(fm)
43| app.exec_loop()
```


Berikut ini beberapa fungsi `QListBox` di atas yang perlu dibahas:

`sort()` mengurutkan daftar pilihan.

`insertItem()` menambah data.

`removeItem()` menghapus data.

`selected()` sinyal yang muncul saat listbox di-klik dua kali (*double-click*).

Pilihan Ganda - Multiselect

Listbox memang dirancang untuk aplikasi yang membutuhkan suatu masukan dengan pilihan lebih dari satu. `listbox2.py` berikut memberikan contoh bagaimana kita dapat memilih hanya dengan satu kali klik pada setiap pilihan.

```
listbox2.py
-----
01| from qt import *
02|
03| class FormListBox(QWidget):
04|     def __init__(self):
05|         QWidget.__init__(self)
06|         self.setCaption("ListBox")
07|         daftarBuah = ["Pisang", "Jeruk", "Apel",
08|                       "Mangga", "Pepaya", "Nanas", "Jambu"]
09|         self.buah = QListBox(self)
```

```
10| self.buah.setGeometry( 10, 40, 140, 120 )
11| self.buah.insertStrList( daftarBuah )
12| self.buah.sort()
13| self.buah.setSelectionMode( QListBox.Multi )
14| self.terpilih = QListBox(self)
15| self.terpilih.setGeometry( 210, 40, 140, 120 )
16| self.terpilih.setSelectionMode( QListBox.Multi )
17| btKanan = QPushButton(self)
18| btKanan.setGeometry( 160, 80, 40, 24 )
19| btKanan.setText( ">" )
20| btKiri = QPushButton(self)
21| btKiri.setGeometry( 160, 110, 40, 24 )
22| btKiri.setText( "<" )
23| self.show()
24| self.connect( btKanan, SIGNAL( "clicked()" ),
25|             self.btKananKlik )
26| self.connect( btKiri, SIGNAL( "clicked()" ),
27|             self.btKiriKlik )
28|
29| def btKananKlik(self):
30|     for i in range( self.buah.count() ):
31|         if self.buah.isSelected( i ):
32|             self.terpilih.insertItem(self.buah.text(i))
33|     self.terpilih.sort()
34|     i = 0
35|     while i < self.buah.count():
36|         if self.buah.isSelected( i ):
37|             self.buah.removeItem( i )
38|         else:
```

```
39|         i = i + 1
40|
41|     def btKiriKlik(self):
42|         for i in range( self.terpilih.count() ):
43|             if self.terpilih.isSelected( i ):
44|                 self.buah.insertItem(self.terpilih.text(i))
45|             self.buah.sort()
46|             i = 0
47|         while i < self.terpilih.count():
48|             if self.terpilih.isSelected( i ):
49|                 self.terpilih.removeItem( i )
50|             else:
51|                 i = i + 1
52|
53| app = QApplication([])
54| fm = FormListBox()
55| app.setMainWidget(fm)
56| app.exec_loop()
```

`setSelectionMode()` menentukan metode memilih. Ia membutuhkan masukan berupa:

`QListBox.Single` hanya satu saja yang bisa dipilih.

`QListBox.Multi` bisa memilih lebih dari satu. Pilihan bersifat *toggle*, artinya bila klik pada suatu pilihan maka pilihan tersebut menjadi tidak terpilih lagi.

`QListBox.Extended` bisa memilih lebih dari satu dengan terlebih dahulu menekan `Ctrl` atau `Shift` sebelum klik

Gambar 15.6: Disable Widget

pada pilihan yang dimaksud. Bila klik saja berarti hanya satu pilihan saja. Metode ini biasanya untuk memilih sekumpulan pilihan pada rangkaian (*range*) tertentu, atau hanya sebuah pilihan saja.

`QListBox.NoSelection` tidak ada yang bisa dipilih. Atau dengan kata lain listbox hanya bisa dilihat saja (*read-only*).

15.8 Widget Aktif - Enable

`QWidget` beserta turunannya dapat dinonaktifkan (*disable*), sehingga meski tampak namun “tidak berfungsi”, dengan tanda-tanda seperti:

1. Tidak dapat diklik
2. Tidak bisa difokuskan.¹⁷ Kalau objek itu `QLineEdit` maka kursor tidak dapat berada di sana, dan Anda tidak dapat memasukkan teks apapun, kecuali dari dalam program.

Objek yang dinonaktifkan akan berubah warnanya untuk membedakan dengan objek yang aktif (*enable*). `enable.py`

¹⁷Dalam sebuah form hanya ada satu objek yang fokus (*focused*), artinya objek tersebut sedang berinteraksi dengan pemakai.

berikut berisi sebuah QLineEdit dan sebuah QPushButton yang hanya dapat diklik apabila QLineEdit terisi.

```
enable.py
-----
01| from qt import *
02|
03| class FormNama(QWidget):
04|     def __init__(self):
05|         QWidget.__init__(self)
06|         self.setCaption("Masukkan nama")
07|         self.resize(200,80)
08|         self.nama = QLineEdit( self )
09|         self.nama.setGeometry(5,5, 100,20)
10|         self.tombol = QPushButton( self )
11|         self.tombol.setText("&OK")
12|         self.tombol.setGeometry(5,30, 60,30)
13|         self.tombol.setEnabled(0)
14|         self.show()
15|         self.connect(self.nama,
16|             SIGNAL("textChanged( const QString& )"),
17|             self.berubah)
18|         self.connect(self.tombol, SIGNAL("clicked()"),
19|             self.selesai)
20|
21|     def berubah(self, teks):
22|         if teks.isEmpty():
23|             self.tombol.setEnabled(0)
24|         else:
```

```

25|         self.tombol.setEnabled(1)
26|
27|     def selesai(self):
28|         self.close()
29|
30| app = QApplication([])
31| fm = FormNama()
32| app.setMainWidget(fm)
33| app.exec_loop()

```

`setEnabled()` memerlukan masukan berupa nilai logika 1 atau 0 yang menandakan suatu objek diaktifkan atau tidak. Fungsi ini berasal dari `QWidget`.

Sinyal `textChanged()` muncul pada saat teks pada `QLineEdit` berubah. Sinyal ini menyertakan isi teksnya yang bertipe `QString` dimana `isEmpty()` menghasilkan nilai logika 1 apabila teks kosong, dan 0 bila sebaliknya. Dengan demikian fungsi `berubah()` sebenarnya bisa ditulis cukup seperti ini:

```

def berubah(self, teks):
    self.tombol.setEnabled( not teks.isEmpty() )

```

15.9 LCD

`QLCDNumber` merupakan label dengan tampilan seperti layar kalkulator sederhana. Contoh berikut ini menampilkan jam digital.

```

jamdigital.py
-----

```

Gambar 15.7: LCD

```
01| from qt import *
02|
03| class JamDigital( QLCDNumber ):
04|     def __init__( self, parent):
05|         QLCDNumber.__init__(self, parent)
06|         self.setSegmentStyle( QLCDNumber.Flat )
07|         self.setNumDigits(8)
08|         self.tampilkan()
09|         self.startTimer( 500 )
10|
11|     def timerEvent( self, e ):
12|         self.tampilkan()
13|
14|     def tampilkan( self ):
15|         t = str(QTime.currentTime().toString("hh:mm:ss"))
16|         self.display( t )
17|
18|
19| if __name__ == "__main__":
20|     app = QApplication([])
21|     fm = JamDigital(None)
22|     fm.show()
23|     app.setMainWidget(fm)
24|     app.exec_loop()
```

Fungsi yang digunakan pada contoh di atas adalah:

`display()` menampilkan karakter tertentu, yaitu: 0/O, 1, 2, 3, 4, 5/S, 6, 7, 8, 9/g, minus (-), titik desimal (.), A, B, C, D, E, F, h, H, L, o, P, r, u, U, Y, titik dua (:), derajat (kutip tunggal pada string-nya) dan spasi. Karakter lainnya akan ditampilkan sebagai spasi.

`setNumDigits()` menentukan jumlah karakter yang boleh tampil.

`setSegmentStyle()` menentukan model tampilan. Nilai yang mungkin adalah `Flat`, `Filled`, dan `Outline`.

Sedangkan pembahasan yang berkaitan dengan waktu dapat dilihat pada halaman 139.

15.10 Hanya Keyboard

Berikut ini pembahasan seputar penggunaan keyboard secara intensif.

15.10.1 Tanpa Mouse

Meski aplikasi window (termasuk KDE) kerap menggunakan mous dalam pengoperasiannya, namun sebenarnya tanpa mouse pun dapat dilakukan:

- Untuk berpindah dari satu objek ke objek lainnya bisa menggunakan tombol Tab atau Shift-Tab.

- Untuk menampilkan daftar pilih pada combobox tekan Alt-Down.¹⁸ lalu gunakan Up atau Down untuk meny-
erot pilihan, dan tekan Enter untuk menentukan pilihan.
- Tekan Spacebar untuk meng-klik tombol.
- Tekan Alt-F4 untuk menutup form.
- Tekan Esc untuk menutup form dialog.

Cara di atas merupakan fasilitas standar yang tidak memerlukan sentuhan programming. Cara lain adalah menggunakan tombol pintas (*shortcut-key*) yang telah didefinisikan melalui QLabel seperti pada contoh berikut:

```
alamat.py
-----
01| from qt import *
02|
03| class FormAlamat(QWidget):
04|     def __init__(self):
05|         QWidget.__init__(self)
06|         self.setCaption("Alamat")
07|         self.resize( 270,130 )
08|
09|         labelNama    = QLabel      ( self )
10|         self.nama     = QLineEdit  ( self )
11|         labelAlamat  = QLabel      ( self )
12|         self.alamat  = QLineEdit  ( self )
```

¹⁸Down: panah bawah

```
13|         btOk          = QPushButton( self )
14|
15|         labelNama.setBuddy ( self.nama )
16|         labelAlamat.setBuddy( self.alamat )
17|
18|         labelNama.setGeometry ( 20,20, 40,20 )
19|         self.nama.setGeometry ( 70,20,180,20 )
20|         labelAlamat.setGeometry ( 20,50, 50,20 )
21|         self.alamat.setGeometry ( 70,50,180,20 )
22|         btOk.setGeometry      ( 90,90, 80,24 )
23|
24|         labelNama.setText ( "&Nama" )
25|         labelAlamat.setText( "&Alamat" )
26|         btOk.setText      ( "&OK" )
27|         self.show()
28|         self.connect(btOk, SIGNAL("clicked()"),
29|             self.selesai)
30|
31|     def selesai(self):
32|         self.close()
33|
34| app = QApplication([])
35| fm = FormAlamat()
36| app.setMainWidget(fm)
37| app.exec_loop()
38| print fm.nama.text(), fm.alamat.text()
```

Perhatikan teks pada `labelNama` dan `labelAlamat` yang telah diberi karakter `&`, dan dengan menggunakan fungsi `setBuddy()`,

Gambar 15.8: Tombol pintas (shortcut-key)

keduanya dikaitkan dengan **nama** dan **alamat**. Ini artinya bila kita menekan Alt-N, maka nama menjadi fokus (kursor berada di nama). Sedangkan bila ditekan Alt-A maka alamat yang terfokus. Begitu pula dengan tombol **btOk** yang telah diberi &. Penekanan Alt-O sama artinya dengan meng-klik mouse pada tombol tersebut.

Pada tampilannya, pemberian karakter & ini memberikan garis bawah (underline) pada karakter setelahnya yang berguna sebagai informasi kepada pemakai. Lebih jelasnya lihat gambar 15.8.

15.10.2 Tombol Keyboard

Program kalkulator berikut ini hanya menampilkan layar angka saja dengan fitur sebagai berikut:

1. Operasi matematika yang dibolehkan adalah penjumlahan (+), pengurangan (-), perkalian (*), dan pembagian (/).
2. Pemisah desimal menggunakan titik (.)
3. Tombol Enter (pada *keypad*¹⁹) berarti *samadengan* (=), yaitu untuk menghitung hasil.

¹⁹Keypad: kelompok tombol angka dan operasi matematika dasar seperti +, -, *, dan /. Biasanya terletak di paling kanan keyboard. Pastikan lampu indikator *numlock* menyala untuk mengaktifkannya (tekan tombol

4. Tombol Return²⁰ berfungsi sama dengan tombol Enter, namun sekaligus menutup form. Berguna bagi form lain yang ingin menggunakan form kalkulator untuk mendapatkan hasil perhitungan.
5. Tombol N untuk menegatifkan angka.
6. Tombol Escape untuk me-*reset*, mengembalikan kondisi seperti baru pertama kali dijalankan.

QLCDNumber akan dipakai untuk memberi kesan kalkulator sebenarnya.

```
kalkulator.py
-----
01| from qt import *
02|
03| class FormKalkulator(QDialog):
04|     def __init__(self):
05|         QDialog.__init__(self)
06|         self.setCaption("Kalkulator")
07|         self.resize(640,120)
08|         self.lcd = QLCDNumber(self)
09|         self.lcd.resize( self.size() )
10|         self.lcd.setSegmentStyle(QLCDNumber.Flat)
11|         self.lcd.setNumDigits(10)
12|         self.reset()
```

NumLock). Pada laptop, keypad biasanya diaktifkan dengan menggunakan tombol Fn dan tombol lainnya. Bacalah petunjuk penggunaannya.

²⁰Posisi tombol Return menjadi satu bagian dengan tombol huruf.

```
13|     self.show()
14|
15|     def reset(self, teks=0):
16|         self.nilai = 0
17|         self.operator = ""
18|         self.teks = ""
19|         self.lcd.display(teks)
20|
21|     def angka(self):
22|         if self.lcd.intValue() == self.lcd.value():
23|             return self.lcd.intValue()
24|         else:
25|             return self.lcd.value()
26|
27|     def setAngka(self, a):
28|         if not self.teks and a == ".":
29|             self.teks = "0"
30|             x = "%s%s" % (self.teks, a)
31|             try: n = float(x)
32|             except ValueError: return
33|             self.teks = x
34|             self.lcd.display(x)
35|
36|     def negatif(self):
37|         self.lcd.display( - self.angka() )
38|
39|     def setOperator(self, op):
40|         self.hitung()
41|         self.operator = op
```

```
42|
43| def hitung(self):
44|     if self.operator:
45|         s = "self.nilai = self.nilai %s %s" % \
46|             (self.operator, self.lcd.value() )
47|         try:
48|             exec(s)
49|         except ZeroDivisionError:
50|             self.reset("salah")
51|         return
52|     else:
53|         self.nilai = self.angka()
54|         self.lcd.display( self.nilai )
55|         self.teks = ""
56|         self.operator = ""
57|
58| def selesai(self):
59|     self.hitung()
60|     self.close()
61|
62| def keyPressEvent(self, e):
63|     ch = "%s" % e.text()
64|     if ch in ["+", "-", "*", "/"]: self.setOperator(ch)
65|     elif e.key() == Qt.Key_N: self.negatif()
66|     elif e.key() == Qt.Key_Escape: self.reset()
67|     elif e.key() == Qt.Key_Enter: self.hitung()
68|     elif e.key() == Qt.Key_Return: self.selesai()
69|     else: self.setAngka( ch )
70|
```

```
71| if __name__ == "__main__":  
72|     app = QApplication([])  
73|     fm = FormKalkulator()  
74|     app.setMainWidget(fm)  
75|     app.exec_loop()  
76|     print fm.angka()
```

Saat tombol keyboard ditekan, event²¹ `keyPressEvent()` dipanggil. Event ini menyertakan parameter `e` bertipe `QKeyEvent` yang memiliki fungsi:

- `key()` mengembalikan nomor (integer) tombol keyboard yang ditekan. `Qt.Key_Escape`, `Qt.Key_Enter`, dan `Qt.Key_Return` adalah contoh konstanta yang sudah disiapkan untuk nilai fungsi ini. Untuk tombol lainnya dapat diketahui dengan perintah `print e.key()` pada event tersebut, lalu tekan tombol keyboard yang Anda maksud agar tampil nomornya.
- `text()` mengembalikan karakter atau tulisan sesuai dengan tombol yang ditekan.
- `state()` mengembalikan nilai `Qt.ShiftButton`, `Qt.ControlButton`, atau `Qt.AltButton`. Ketiganya mewakili tombol Shift, Control (Ctrl), dan Alt. Contoh penggunaannya ada di halaman 158.
- `value()` mencoba mengkonversi nilai yang tampil menjadi float, sedangkan `intValue()` menjadi integer.

²¹Event: fungsi yang berkaitan dengan suatu kejadian.

15.10.3 NumLock

Secara default, lampu indikator NumLock tidak menyala ketika seseorang login. Hal ini berakibat pada fungsi keypad sebagai tombol navigasi dan bukan tombol angka. Untuk memastikan keypad berfungsi sebagai tombol angka, buatlah sebuah file `.Xmodmap` pada *home directory*²² yang berisi translasi tombol keypad berikut ini:

```
keycode 79 = 7
keycode 80 = 8
keycode 81 = 9

keycode 83 = 4
keycode 84 = 5
keycode 85 = 6

keycode 87 = 1
keycode 88 = 2
keycode 89 = 3

keycode 90 = 0
keycode 91 = period

keycode 77 =
```

Bila Anda *superuser* root, simpan script di atas pada `/etc/X11/Xmodmap` agar user lainnya bisa langsung menggunakannya.

²²Home directory: direktori default milik user tertentu. Misalkan usernya bernama toni maka - biasanya - home directory baginya adalah `/home/toni`.

Bab 16

Kasir I

Kasir I adalah program kasir sederhana namun sudah layak dipakai untuk mencatat transaksi penjualan. Ide pembuatannya adalah:

1. Pemakai sudah mengetahui harga jual produknya.
2. Tidak memerlukan pencatatan nama barang, yang penting nilainya tercetak.
3. Kebebasan mencatat transaksi seperti pemberian discount, perubahan harga jual, seketika, dsb. Operasi perhitungan dengan operator tambah, kurang, kali, dan bagi dimungkinkan.
4. Struk cukup informatif dimana tercetak rincian nilai, total penjualan, uang yang dibayarkan serta nilai kembaliannya. Juga tercetak tanggal dan jam transaksi.

5. Cukup cepat karena nilai langsung tercetak.
6. Mudah digunakan karena kerap menggunakan keypad.
7. Mudah perawatan karena transaksi tidak tersimpan sehingga tidak khawatir ruang harddisk menjadi penuh. Bila ada gangguan printer, pencatatan bisa dialihkan ke file biasa.

Program ini beranjak dari `kalkulator.py` pada contoh sebelumnya¹ dimana keturunan `FormKalkulator` yang bernama `FormKasir` nanti akan mengubah sifat leluhurnya agar sesuai dengan kriteria di atas. Adapun fitur dari form ini adalah:

1. Tombol Enter berarti mencetak harga barang namun yang tampil di layar LCD adalah totalnya (akumulasi).
2. Tombol Return berarti mengasumsikan nilai yang tampil di layar LCD adalah pembayaran. Selanjutnya nilai tersebut dikurangi total penjualan dan ditampilkan kembalinya. Informasi yang tercetak adalah total penjualan, nilai pembayaran serta kembaliannya, dan terakhir waktu transaksi.

Berikut ini contoh tampilan pada struk:

```
2.500
3.000
7.500
```

¹Lihat halaman 115.

```
13.000 T
20.000 B
 7.000 K
290103 16:46
```

Makna T, B, dan K berturut-turut adalah Total, Bayar, dan Kembali. Baris terakhir merupakan tanggal, bulan, tahun dan jam pencetakan.

```
kasir1.py
-----
01| from qt import *
02| from string import rjust
03| from locale import setlocale, LC_ALL, format
04| from os import system
05| import sys
06| from kalkulator import FormKalkulator
07|
08| setlocale(LC_ALL, "")
09|
10| class FormKasir(FormKalkulator):
11|     def __init__(self):
12|         FormKalkulator.__init__(self)
13|         self.setCaption("Kasir I")
14|         if sys.argv[1:]: filename = sys.argv[1]
15|         else: filename = "/dev/lp0"
16|         self.file = open(filename, "w")
17|         self.total = 0
18|
```

```
19| def simpan(self, s):
20|     self.file.write("%s" % s)
21|     self.file.flush()
22|
23| def cetak(self, angka, kode=""):
24|     n = format("%2.f", angka, 1)
25|     _angka = rjust(n,10)
26|     _kode = rjust(kode,2)
27|     s = "%s%s\n" % (_angka, _kode)
28|     self.simpan(s)
29|
30| def keyPressEvent(self, e):
31|     if e.key() == Qt.Key_Enter:
32|         self.hitung()
33|         self.total = self.total + self.nilai
34|         self.lcd.display( self.total )
35|         self.cetak( self.nilai )
36|     elif e.key() == Qt.Key_Return:
37|         bayar = self.angka()
38|         kembali = bayar - self.total
39|         self.cetak( self.total, "T" )
40|         self.cetak( bayar, "B" )
41|         self.cetak( kembali, "K" )
42|         w = QDateTime.currentDateTime().toString(
43|             "ddMMyy hh:mm")
44|         self.simpan( w )
45|         self.simpan("\n" * 5) # mudah merobeknya
46|         self.reset()
47|         self.total = 0
```

```

48|         self.lcd.display( kembali )
49|     else: FormKalkulator.keyPressEvent(self, e)
50|
51| app = QApplication([])
52| fm = FormKasir()
53| app.setMainWidget(fm)
54| app.exec_loop()

```

Fungsi `flush()` digunakan untuk mengirimkan string yang sudah di-`write()`. Fungsi ini mirip dengan `close()` hanya saja file tidak ditutup. Sedangkan `QDateTime` adalah class yang berkenaan dengan informasi waktu (tanggal dan jam). `currentDateTime()`-nya digunakan untuk mendapatkan waktu saat ini. Fungsi ini mengembalikan nilai bertipe `QDateTime` juga. `toString()` digunakan untuk mencetak waktu sesuai dengan bentuk yang sudah ditentukan.

Bila ada masalah pada printer maka nama file alternatif bisa disertakan:

```
$ python kasir1.py /tmp/kasir.txt
```

Perlu diketahui pula bahwa file tersebut akan dikosongkan terlebih dahulu manakala program dijalankan kembali. Jadi penyimpanannya memang tidak permanen. Bila tidak ingin disimpan kemana-mana bisa menggunakan `/dev/null`

```
$ python kasir1.py /dev/null
```


Bab 17

Wadah - Container

Wadah atau *container* merupakan objek visual yang dapat menampung objek visual lainnya. Form bisa disebut merupakan wadah utama. Wadah lainnya adalah `QTabWidget`, `QFrame`, `QButtonGroup`, dan `QTabWidget`. Masing-masing memiliki ciri khas yang perlu diketahui agar dipergunakan secara tepat.

Wadah non-form tersebut bertujuan untuk mengumpulkan beberapa widget yang memiliki “kesamaan” sehingga perlu dikelompokkan (*grouping*). Pengelompokan ini mempermudah penataan, karena posisi objek-objek tersebut relatif terhadap wadahnya. Dengan kata lain untuk mengubah posisi sekelompok objek, cukup mengubah posisi wadahnya saja.

17.1 Widget

Meski selama ini `QWidget` telah digunakan sebagai form, namun sebenarnya ia juga dapat digunakan sebagai wadah di dalam form.¹

17.2 Panel

Frame (`QFrame`) - bisa juga disebut *panel* - merupakan wadah yang bisa diatur model bingkainya. Contoh penggunaannya dapat dilihat dalam `suhu.py` di halaman 124.

17.3 Groupbox

GroupBox (`QButtonGroup`) pada bahasan sebelumnya juga dikategorikan sebagai wadah. Ia dilengkapi label di atasnya sebagai keterangan perihal daftar objek yang ada di dalamnya. Class ini biasanya dipakai untuk objek sejenis.

Mari membuat simulasi pengendalian suhu yang dapat dilakukan melalui dua cara:

1. Langsung ditentukan suhunya dengan nilai antara 0 hingga 99 derajat Celcius. Objek yang digunakan adalah *spinbox*² (`QSpinBox`).

¹Lihat `tab.py` halaman 127.

²Spinbox: editor yang menerima masukan berupa angka. Memiliki dua tombol untuk menambah atau mengurangi nilai.

2. Melalui pilihan yang mewakili suhu tertentu, yaitu: dingin, hangat, dan panas. Objek yang digunakan adalah radiobutton (`QButtonGroup`).

Dua pengendali ini akan saling mempengaruhi satu sama lain. Bila Anda mengubahnya melalui spinbox, maka radiobutton akan berubah pula sesuai dengan nilai-antara (*range*) berikut ini:

- 0-25 derajat Celcius tergolong dingin.
- 26-40 derajat Celcius tergolong hangat.
- 41-99 derajat Celcius tergolong panas.

Begitu pula sebaliknya, bila pengendalian melalui radiobutton maka spinbox akan menunjukkan nilai berikut:

- dingin berarti 25 derajat Celcius.
- hangat berarti 40 derajat Celcius.
- panas berarti 60 derajat Celcius.

Hati-hati, dalam keadaan yang sebenarnya mungkin Anda perlu memastikan bahwa nilai-antara di atas memang sesuai dengan kondisi lingkungan.

```
suhu.py
-----
01| from qt import *
```

```
02|
03| class FormSuhu(QWidget):
04|     Dingin = 25
05|     Hangat = 40
06|     Panas  = 60
07|
08|     def __init__(self):
09|         QWidget.__init__(self)
10|         self.setCaption( "Temperatur" )
11|         self.resize( 260, 180 )
12|
13|         panelMeter = QFrame(self)
14|         panelMeter.setGeometry(10,10,120,160)
15|         panelMeter.setFrameShape( QFrame.StyledPanel )
16|         panelMeter.setFrameShadow( QFrame.Raised )
17|
18|         self.LCD = QLCDNumber(panelMeter)
19|         self.LCD.setGeometry(30,10,60,23)
20|
21|         self.Dial = QDial(panelMeter)
22|         self.Dial.setGeometry(10,50,100,100)
23|
24|         panelSuhu = QFrame(self)
25|         panelSuhu.setGeometry(140,10,110,50)
26|         panelSuhu.setFrameShape( QFrame.WinPanel )
27|         panelSuhu.setFrameShadow ( QFrame.Sunken )
28|
29|         labelSuhu = QLabel( panelSuhu )
30|         labelSuhu.setGeometry(10,10,50,30)
```

```
31|     labelSuhu.setText("Suhu (Celcius)")
32|     labelSuhu.setAlignment( QLabel.WordBreak |
33|         QLabel.AlignVCenter)
34|
35|     self.suhu = QSpinBox( panelSuhu )
36|     self.suhu.setGeometry(65,10,40,20)
37|
38|     grupSuhu = QButtonGroup(self)
39|     grupSuhu.setGeometry(140,80,110,90)
40|     grupSuhu.setTitle("Bagi Manusia")
41|
42|     self.rbDingin = QRadioButton(grupSuhu)
43|     self.rbDingin.setGeometry(10,20,96,20)
44|     self.rbDingin.setText("Dingin")
45|
46|     self.rbHangat = QRadioButton(grupSuhu)
47|     self.rbHangat.setGeometry(10,40,96,20)
48|     self.rbHangat.setText("Hangat")
49|
50|     self.rbPanas = QRadioButton(grupSuhu)
51|     self.rbPanas.setGeometry(10,60,96,20)
52|     self.rbPanas.setText("Panas")
53|
54|     self.connect(self.suhu, SIGNAL(
55|         "valueChanged(int)" ), self.suhuBerubah)
56|     self.connect( grupSuhu, SIGNAL(
57|         "clicked(int)"), self.grupSuhuBerubah )
58|
59|     self.ubah = 1
```

```
60|         self.suhu.setValue( self.Dingin ) # default
61|         self.show()
62|
63|     def suhuBerubah(self, nilai):
64|         self.LCD.display( nilai )
65|         self.Dial.setValue( nilai )
66|         if self.ubah:
67|             self.ubah = 0
68|             if self.suhu.value() in range(self.Dingin+1):
69|                 self.rbDingin.setChecked(1)
70|             elif self.suhu.value() in range(self.Dingin+1,
71|                 self.Panas):
72|                 self.rbHangat.setChecked(1)
73|             else:
74|                 self.rbPanas.setChecked(1)
75|             self.ubah = 1
76|
77|     def grupSuhuBerubah(self, index):
78|         if self.ubah:
79|             self.ubah = 0
80|             if index == 0:
81|                 self.suhu.setValue( self.Dingin )
82|             elif index == 1:
83|                 self.suhu.setValue( self.Hangat )
84|             else:
85|                 self.suhu.setValue( self.Panas )
86|             self.ubah = 1
87|
88| app = QApplication([])
```

```
89| fm = FormSuhu()
90| app.setMainWidget(fm)
91| app.exec_loop()
```

`QSpinBox` memiliki nilai-antara 0 hingga 99 (default) dimana nilainya berupa integer yang dapat diubah menggunakan fungsi `setValue()`. Class ini memiliki sinyal `valueChanged()` yang timbul pada saat nilainya berubah.

Sedangkan `QDial` memiliki tampilan seperti *speedometer* yang juga memiliki nilai-antara 0 hingga 99. Fungsi `setValue()`-nya akan mengubah posisi jarum.

Sinyal yang Saling Terkait

Dari sudut teknik pemrograman, daya tarik `suhu.py` adalah adanya saling mempengaruhi objek satu dengan lainnya.

`suhuBerubah()` dipanggil pada saat `QSpinBox` nilainya berubah, baik oleh pemakai secara langsung atau melalui `QButtonGroup`. Hal serupa terjadi dengan `grupSuhuBerubah()` yang berkaitan dengan perubahan pilihan pada `QButtonGroup`.

Karena bisa menimbulkan perulangan yang tak berkesudahan³, maka diperlukan pembeda pada saat suatu nilai berubah. Masing-masing fungsi harus bisa membedakan apakah ia dipanggil melalui cara pertama (oleh pemakai langsung) atau melalui cara kedua (melalui fungsi). Variabel `self.ubah` merupakan variabel logika untuk mengatasi permasalahan tersebut, dan dengan demikian rekursif tak berkesudahan dapat dihindari.

³Atau lebih tepat dikatakan terjadi rekursif tak berkesudahan. Rekursif: fungsi yang memanggil dirinya sendiri.

17.4 Multigroup

Objek lain yang mirip groupbox namun dapat memiliki wadah lebih dari satu adalah `QTabWidget`. `tab.py` berikut menunjukkan dua kelompok identitas seseorang yaitu: alamat rumah dan alamat internet, dimana keduanya disatukan berada dalam wadah yang berbeda namun disatukan dalam `QTabWidget`.

```
tab.py
-----
01| from qt import *
02|
03| class FormIdentitas(QWidget):
04|     def __init__(self):
05|         QWidget.__init__(self)
06|         self.setCaption( "Identitas" )
07|
08|         self.TabWidget = QTabWidget(self)
09|         self.TabWidget.setGeometry(20,20,290,210)
10|         self.tabAlamat = QWidget(self.TabWidget)
11|         self.tabInternet = QWidget(self.TabWidget)
12|         self.TabWidget.insertTab(self.tabAlamat, "Alamat")
13|         self.TabWidget.insertTab(self.tabInternet,"Internet")
14|
15|         # Isi tabAlamat
16|         labelJalan = QLabel(self.tabAlamat)
17|         labelJalan.setGeometry(11,11,66,20)
18|         labelJalan.setText("Jalan")
19|         self.jalan = QLineEdit(self.tabAlamat)
```

```
20|         self.jalan.setGeometry(83,11,196,20)
21|
22|         labelKel = QLabel(self.tabAlamat)
23|         labelKel.setGeometry(11,37,66,20)
24|         labelKel.setText("Kelurahan")
25|         self.kelurahan = QLineEdit(self.tabAlamat)
26|         self.kelurahan.setGeometry(83,37,196,20)
27|
28|         labelKec = QLabel(self.tabAlamat)
29|         labelKec.setGeometry(11,63,66,20)
30|         labelKec.setText("Kecamatan")
31|         self.kecamatan = QLineEdit(self.tabAlamat)
32|         self.kecamatan.setGeometry(83,63,196,20)
33|
34|         labelPropinsi = QLabel(self.tabAlamat)
35|         labelPropinsi.setGeometry(11,89,66,20)
36|         labelPropinsi.setText("Propinsi")
37|         self.propinsi = QLineEdit(self.tabAlamat)
38|         self.propinsi.setGeometry(83,89,196,20)
39|
40|         labelKodePos = QLabel(self.tabAlamat)
41|         labelKodePos.setGeometry(11,115,66,20)
42|         labelKodePos.setText("Kode Pos")
43|         self.kodePos = QLineEdit(self.tabAlamat, "kodePos")
44|         self.kodePos.setGeometry(83,115,70,20)
45|
46|         labelTelp = QLabel(self.tabAlamat)
47|         labelTelp.setGeometry(11,141,66,20)
48|         labelTelp.setText("Telp")
```

```
49| self.telp = QLineEdit(self.tabAlamat)
50| self.telp.setGeometry(83,141,196,20)
51|
52| # Isi tabInternet
53| labelEmail = QLabel(self.tabInternet)
54| labelEmail.setGeometry(11,11,41,20)
55| labelEmail.setText("Email")
56| self.email = QLineEdit(self.tabInternet)
57| self.email.setGeometry(58,11,211,20)
58|
59| labelWeb = QLabel(self.tabInternet)
60| labelWeb.setGeometry(11,37,41,20)
61| labelWeb.setText("Web")
62| self.web = QLineEdit(self.tabInternet)
63| self.web.setGeometry(58,37,211,20)
64|
65| labelYahoo = QLabel(self.tabInternet)
66| labelYahoo.setGeometry(11,63,41,20)
67| labelYahoo.setText("Yahoo")
68| self.yahoo = QLineEdit(self.tabInternet)
69| self.yahoo.setGeometry(58,63,211,20)
70|
71| labelIcq = QLabel(self.tabInternet)
72| labelIcq.setGeometry(11,89,41,20)
73| labelIcq.setText("ICQ")
74| self.icq = QLineEdit(self.tabInternet)
75| self.icq.setGeometry(58,89,211,20)
76|
77| self.show()
```



```
78|  
79| app = QApplication([])  
80| fm = FormIdentitas()  
81| app.setMainWidget(fm)  
82| app.exec_loop()
```

Fungsi `insertTab()` digunakan untuk menambah kelompok baru. Masukan pertamanya adalah wadah, sedangkan yang kedua berupa string sebagai nama kelompok.

Bab 18

Penataan

Menentukan posisi dan ukuran widget memang merupakan pekerjaan tersendiri. Untunglah Qt menyediakan *layouter* (`QLayout` beserta turunannya) untuk menata widget secara otomatis, termasuk ukurannya.

18.1 Fleksibilitas Ukuran

Anda pernah melihat sebuah text editor seperti `kwrite` ? Bila melihat contoh sebelumnya, `kwrite` tampak dibangun dengan `QTextEdit`. Cobalah menjalankan `kwrite` dan perhatikan ukuran `QTextEdit`-nya saat Anda mengubah-ubah ukuran form. Ya, ternyata ukurannya mengikuti besarnya form. Lalu bagaimana hal itu dilakukan ?

Qt menyediakan sebuah layouter `QVBoxLayout` dan `QHBoxLayout` yang bertugas mengatur ukuran dan posisi widget secara vertikal dan horizontal. Bila hanya ada satu widget, maka salah satunya dapat digunakan dengan hasil yang sama.

```
texteditor1.py
-----
01| from qt import *
02|
03| class FormEditor(QWidget):
04|     def __init__(self):
05|         QWidget.__init__(self)
06|         self.setCaption("Text Editor")
07|         self.editor = QTextEdit(self)
08|         layout = QVBoxLayout(self)
09|         layout.addWidget( self.editor )
10|         self.show()
11|
12| app = QApplication([])
13| fm = FormEditor()
14| app.setMainWidget(fm)
15| app.exec_loop()
```

`addWidget()` digunakan untuk menambahkan widget ke dalam "daftar penataan". Cobalah mengubah-ubah ukuran form pada saat runtime. Sebagaimana `kwrite`, ukuran `QTextEdit` akan mengikuti ukuran form.

Selanjutnya akan kita tambahkan sebuah panel (`QFrame`) di atas editor (`QTextEdit`). Panel ini hanya mengikuti panjang

form saja, sedangkan lebarnya tetap. Fungsinya nanti sebagai tempat tombol-tombol menu pada contoh berikutnya.

```
texteditor2.py
-----
01| from qt import *
02|
03| class FormEditor(QWidget):
04|     def __init__(self):
05|         QWidget.__init__(self)
06|         self.setCaption("Text Editor")
07|         self.editor = QTextEdit(self)
08|         panel      = QFrame (self)
09|         panel.setMinimumHeight(40)
10|         layout = QVBoxLayout(self)
11|         layout.addWidget( panel )
12|         layout.addWidget( self.editor )
13|         self.show()
14|
15| app = QApplication([])
16| fm = FormEditor()
17| app.setMainWidget(fm)
18| app.exec_loop()
```

`setMinimumHeight()` digunakan untuk menentukan lebar minimum widget. Tanpanya panel tidak akan terlihat.

Latihan Gantilah `QVBoxLayout` menjadi `QHBoxLayout`, dan amati perbedaannya. Bila Anda kehilangan panel, coba ganti `setMinimumHeight()` menjadi `setMinimumWidth()`.

`addWidget()` bisa diganti dengan fungsi `setAutoAdd()` yang memasukkan widget ke dalam daftar penataan secara otomatis.

```
texteditor2a.py
-----
01| from qt import *
02|
03| class FormEditor(QWidget):
04|     def __init__(self):
05|         QWidget.__init__(self)
06|         self.setCaption("Text Editor")
07|         layout = QVBoxLayout(self)
08|         layout.setAutoAdd(1)
09|         panel = QFrame(self)
10|         panel.setMinimumHeight(40)
11|         self.editor = QTextEdit(self)
12|         self.show()
13|
14| app = QApplication([])
15| fm = FormEditor()
16| app.setMainWidget(fm)
17| app.exec_loop()
```

18.2 Fleksibilitas Posisi

Kini tombol menu akan diletakkan di dalam panel, yaitu:

Baru Mengosongkan editor untuk membuat file baru

Buka Membuka file dan menampilkannya pada editor

Simpan Menyimpan file

Simpan_Sebagai Menyimpan dengan nama file lain

```
texteditor3.py
-----
01| from qt import *
02|
03| class FormEditor(QWidget):
04|     def __init__(self):
05|         QWidget.__init__(self)
06|         self.setCaption("Text Editor")
07|         layout      = QVBoxLayout( self )
08|         layout.setAutoAdd(1)
09|         panel       = QFrame      ( self )
10|         self.editor = QTextEdit  ( self )
11|
12|         layout      = QHBoxLayout( panel )
13|         layout.setAutoAdd(1)
14|         btBaru      = QPushButton( panel )
15|         btBuka      = QPushButton( panel )
16|         btSimpan    = QPushButton( panel )
17|         btSimpanSbg = QPushButton( panel )
18|
19|         btBaru.setText      ( "Baru" )
20|         btBuka.setText      ( "Buka" )
21|         btSimpan.setText    ( "Simpan" )
22|         btSimpanSbg.setText( "Simpan Sebagai" )
```

```

23|
24|     self.showMaximized()
25|
26| app = QApplication([])
27| fm = FormEditor()
28| app.setMainWidget(fm)
29| app.exec_loop()

```

Perhatikan, kini `setMinimumHeight()` sudah tidak diperlukan lagi. Kelihatannya `QPushButton` memiliki ukuran minimum yang bisa mempengaruhi ukuran minimum parent-nya.

Sedangkan `showMaximized()` berguna untuk memperbesar form hingga memenuhi layar desktop.

Semua tombol sudah tertata sesuai urutannya, namun ketika form diperbesar (*maximized*), ukuran tombol juga ikut membesar. Tentu saja tampilannya tidak lagi nyaman dipandang. Sepantasnya ukuran tombol tidak berubah. Oleh karena itu dibutuhkan “pengganjal” yang akan memenuhi sisa ruang, dimana kita bisa menggunakan fungsi `addStretch()`. Fungsi ini milik `QBoxLayout`, leluhur `QHBoxLayout` dan `QVBoxLayout`.

```

texteditor3a.py
-----
01| from qt import *
02|
03| class FormEditor(QWidget):
04|     def __init__(self):
05|         QWidget.__init__(self)
06|         self.setCaption("Text Editor")

```



```
07|     layout      = QVBoxLayout( self )
08|     layout.setAutoAdd(1)
09|     panel       = QFrame      ( self )
10|     self.editor = QTextEdit  ( self )
11|
12|     layout      = QHBoxLayout( panel )
13|     layout.setAutoAdd(1)
14|     layout.addStretch()
15|     btBaru      = QPushButton( panel )
16|     btBuka      = QPushButton( panel )
17|     btSimpan    = QPushButton( panel )
18|     btSimpanSbg = QPushButton( panel )
19|
20|     btBaru.setText      ( "Baru" )
21|     btBuka.setText      ( "Buka" )
22|     btSimpan.setText    ( "Simpan" )
23|     btSimpanSbg.setText( "Simpan Sebagai" )
24|
25|     self.showMaximized()
26|
27| app = QApplication([])
28| fm = FormEditor()
29| app.setMainWidget(fm)
30| app.exec_loop()
```

Ya, kini ukuran tombol lebih nyaman dilihat. Namun mungkin masih ada yang kurang pas, karena posisinya tidak berada di sisi kiri form melainkan di sisi kanan. Perlu sedikit perubahan agar sesuai dengan yang dimaksud.

```
texteditor3b.py
-----
01| from qt import *
02|
03| class FormEditor(QWidget):
04|     def __init__(self):
05|         QWidget.__init__(self)
06|         self.setCaption("Text Editor")
07|         layout      = QVBoxLayout(self)
08|         layout.setAutoAdd(1)
09|         panel       = QFrame(self)
10|         self.editor = QTextEdit(self)
11|
12|         layout = QHBoxLayout( panel )
13|         layout.setAutoAdd(1)
14|         layout.addStretch()
15|         layout.setDirection( QVBoxLayout.RightToLeft )
16|
17|         btSimpanSbg = QPushButton( panel )
18|         btSimpan    = QPushButton( panel )
19|         btBuka      = QPushButton( panel )
20|         btBaru      = QPushButton( panel )
21|
22|         btBaru.setText      ( "Baru" )
23|         btBuka.setText      ( "Buka" )
24|         btSimpan.setText    ( "Simpan" )
25|         btSimpanSbg.setText( "Simpan Sebagai" )
26|
27|         self.showMaximized()
```

```
28|  
29| app = QApplication([])  
30| fm = FormEditor()  
31| app.setMainWidget(fm)  
32| app.exec_loop()
```

`setDirection()` digunakan untuk menentukan arah penataan. Fungsi ini dimiliki oleh `QBoxLayout` dan dapat diisi dengan konstanta berikut ini:

`LeftToRight` dari kiri ke kanan (horizontal)

`RightToLeft` dari kanan ke kiri (horizontal)

`Down` dari atas ke bawah (vertikal)

`Up` dari bawah ke atas (vertikal)

Dari beberapa contoh di atas tentang penggunaan layouter, dapat diambil beberapa kesimpulan:

1. Layouter hanya berfungsi sebagai penata widget - baik ukuran maupun posisi - bukan suatu wadah.
2. Layouter akan memenuhi wadah (parent) dengan widget (child) yang diaturnya.

18.3 Layout Dengan Metode Grid

Dengan `QHBoxLayout` dan `QVBoxLayout` dapat menata objek dengan arah horizontal maupun vertikal. Namun untuk penataan

widget seperti pada `tab.py` di halaman 127 tentu cukup merepotkan - meskipun bisa dengan dua class tersebut. Qt memiliki `QGridLayout` yang menata objek dengan sistem tabel (*grid*), yaitu menggunakan koordinat berdasarkan baris dan kolom.

```
alamat1.py
-----
01| from qt import *
02|
03| class FormAlamat(QWidget):
04|     def __init__(self):
05|         QWidget.__init__(self)
06|         self.setCaption("Alamat")
07|         layout = QGridLayout( self )
08|         layout.setMargin(5)
09|         labelNama    = QLabel    ( self )
10|         labelAlamat  = QLabel    ( self )
11|         self.nama    = QLineEdit( self )
12|         self.alamat  = QLineEdit( self )
13|         labelNama.setText      ("Nama")
14|         labelAlamat.setText    ("Alamat")
15|         layout.addWidget(labelNama,  0,0)
16|         layout.addWidget(labelAlamat, 1,0)
17|         layout.addWidget(self.nama,  0,1)
18|         layout.addWidget(self.alamat, 1,1)
19|         self.show()
20|
21| app = QApplication([])
22| fm = FormAlamat()
```

```
23| app.setMainWidget(fm)
24| app.exec_loop()
```

`setMargin()` berfungsi untuk memberi jarak antara widget dengan wadahnya agar tampilan tampak lebih baik. Fungsi ini dimiliki oleh `QLayout`, leluhur `QGridLayout` dan `QBoxLayout`. Fungsi lain yang mirip adalah `setSpacing()` yang memberi jarak antar widget.

`addWidget()` membutuhkan informasi tambahan berupa baris dan kolom widget yang akan ditata. `setAutoAdd()` juga bisa dipakai dimana urutan penataan dimulai dari baris pertama kolom pertama dilanjutkan ke kolom berikutnya pada baris yang sama. Jika pada baris tersebut sudah penuh, dilanjutkan ke baris kedua, begitu seterusnya. Karena itu jumlah baris dan kolom perlu ditentukan pada saat pembuatan `QGridLayout`.

```
alamat2.py
-----
01| from qt import *
02|
03| class FormAlamat(QWidget):
04|     def __init__(self):
05|         QWidget.__init__(self)
06|         self.setCaption("Alamat")
07|         layout = QGridLayout( self, 2,2 )
08|         layout.setAutoAdd(1)
09|         layout.setMargin(5)
10|         labelNama = QLabel ( self )
11|         self.nama = QLineEdit( self )
```

```
12|     labelAlamat = QLabel    ( self )
13|     self.alamat = QLineEdit( self )
14|     labelNama.setText  ("Nama")
15|     labelAlamat.setText("Alamat")
16|     self.show()
17|
18| app = QApplication([])
19| fm = FormAlamat()
20| app.setMainWidget(fm)
21| app.exec_loop()
```

Bab 19

Waktu

Qt memiliki class khusus berkaitan dengan waktu, yaitu `QTime` (jam), `QDate` (tanggal), dan `QDateTime` (tanggal dan jam).

19.1 Jam

`QTime` adalah class yang memuat jam, menit, detik, hingga milidetik. Baris berikut contoh untuk “membuat” jam:

```
j = QTime( 23, 45, 55, 0 )
```

Nilai masukan di atas berturut-turut adalah: jam, menit, detik, dan milidetik. Berikut ini fungsi yang sering dipakai:

`currentTime()` mengembalikan waktu saat ini yang juga bertipe `QTime`.

<code>hour()</code>	jam
<code>minute()</code>	menit
<code>second()</code>	detik
<code>msec()</code>	milidetik
<code>addSecs(<i>n</i>)</code>	tambah <i>n</i> detik dimana <i>n</i> integer, mengembalikan <code>QTime</code> .
<code>secsTo(<i>t</i>)</code>	selisih detik (integer) dengan <i>t</i> (<code>QTime</code>).
<code>toString()</code>	menampilkan jam sesuai dengan format yang dikehendaki. Adapun pola yang bisa digunakan dalam fungsi ini adalah:
<code>h</code>	jam tanpa nol di depan (0..23 atau 1..12 jika ditampilkan dengan AM/PM)
<code>hh</code>	jam dengan nol di depan (00..23 atau 01..12 jika ditampilkan dengan AM/PM)
<code>m</code>	menit tanpa nol di depan (0..59)
<code>mm</code>	menit dengan nol di depan (00..59)
<code>s</code>	detik tanpa nol di depan (0..59)
<code>ss</code>	detik dengan nol di depan (00..59)
<code>z</code>	milidetik tanpa nol di depan (0..999)
<code>zzz</code>	milidetik dengan nol di depan (000..999)

AP menampilkan AM/PM.

ap menampilkan am/pm.

Contoh penggunaannya ada pada halaman 111.

19.2 Tanggal - QDate

QDate memuat tanggal, bulan, dan tahun. Contoh:

```
t = QDate( 2002, 12, 31 )
```

dan tentunya ada sudah tahu makna urutannya. Berikut ini fungsi yang sering digunakan:

`currentDate()` tanggal saat ini, juga mengembalikan QDate.

`year()` tahun.

`month()` bulan.

`day()` tanggal.

`dayOfWeek()` urutan hari dalam minggu dimana Senin = 1 dan Minggu = 7.

`dayOfYear()` urutan hari dalam tahun.

`daysInMonth()` jumlah hari di bulan `month()`

`daysInYear()` jumlah hari di tahun `year()`

- `addDays(n)` menambah *n* hari, mengembalikan `QDate`.
- `addMonths(n)` menambah *n* bulan, mengembalikan `QDate`.
- `addYears()` menambah *n* tahun, mengembalikan `QDate`.
- `daysTo(d)` selisih hari dengan *d* (`QDate`), mengembalikan integer.
- `toString()` menampilkan tanggal sesuai dengan format yang dikehendaki. Format yang bisa digunakan adalah:
- | | |
|-------------------|---|
| <code>d</code> | tanggal (day) tanpa awalan nol (1-31) |
| <code>dd</code> | tanggal (day) dengan awalan nol (01-31) |
| <code>ddd</code> | nama hari yang pendek dalam bahasa Inggris (Mon - Sun) |
| <code>dddd</code> | nama hari yang panjang dalam bahasa Inggris (Monday - Sunday) |
| <code>M</code> | bulan tanpa awaln nol (1-12) |
| <code>MM</code> | bulan dengan awalan nol (01-12) |
| <code>MMM</code> | nama pendek bulan dalam bahasa Inggris (Jan - Dec) |
| <code>MMMM</code> | nama panjang bulan (January - December) |
| <code>yy</code> | dua angka tahun (00-99) |
| <code>yyyy</code> | empat angka tahun (0000-9999) |

Class ini bisa kita definisi ulang (reimplementation) agar berbahasa Indonesia. Format yang ditambahkan adalah:

hari nama hari (Senin - Minggu)
bulan nama bulan (Januari - Desember)
pasaran nama pasaran (Pon - Pahing)

Class-nya dinamakan `Tanggal` dan termuat dalam modul `tanggal.py` agar dapat digunakan program lainnya:

```
tanggal.py
-----
01| from qt import *
02| import string
03|
04| class Tanggal(QDate):
05|     Hari = ["Senin", "Selasa", "Rabu", "Kamis",
06|            "Jumat", "Sabtu", "Minggu"]
07|     Bulan = ["Januari", "Februari", "Maret", "April", "Mei",
08|             "Juni", "Juli", "Agustus", "September", "Oktober",
09|             "November", "Desember"]
10|     Pasaran = ["Pon", "Wage", "Kliwon", "Legi", "Pahing"]
11|
12|     # Konstanta: 1-2-2003 Pon
13|     TGL = QDate(2003, 2, 1)
14|
15|     def __init__(self, y, m, d):
16|         QDate.__init__(self, y, m, d)
```

```
17|
18| def toString(self,
19|     format="hari pasaran, dd bulan yyyy"):
20|     s = str(QDate.toString(self, format))
21|     if string.find(s, "hari") > -1:
22|         hari = self.Hari[self.dayOfWeek()-1]
23|         s = string.replace(s, "hari", hari)
24|     if string.find(s, "bulan") > -1:
25|         bln = self.Bulan[self.month()-1]
26|         s = string.replace(s, "bulan", bln)
27|     if string.find(s, "pasaran"):
28|         jml = self.TGL.daysTo(self)
29|         sisa = jml % 5
30|         psr = self.Pasaran[sisa]
31|         s = string.replace(s, "pasaran", psr)
32|     return QString(s)
33|
34|
35| if __name__ == "__main__":
36|     n = QDate.currentDate()
37|     t = Tanggal(n.year(), n.month(), n.day())
38|     print t.toString()
```

19.3 Tanggal dan Jam

`QDateTime` adalah gabungan `QDate` dan `QTime`. Inisialisasinya juga melibatkan dua class tersebut:

```
t = QDate.currentDate()
```

```
j = QTime.currentTime()
w = QDateTime( t, j )
```

Adapun fungsi yang sering digunakan adalah:

`currentDateTime()` waktu saat ini, mengembalikan `QDateTime`.

`date()` mengembalikan porsi tanggal (`QDate`).

`time()` mengembalikan porsi jam (`QTime`).

`toString()` menampilkan waktu sesuai format dimana format tersebut sama dengan yang berlaku pada `QDate` dan `QTime`.

Fungsi lainnya mirip dengan yang terdapat pada `QDate` maupun `QTime` seperti `addDays()`, `addMonths()`, `addYears()`, `addSecs()`, `daysTo()`, dan `secsTo()`.

19.4 Timer

Pewaktu atau *timer* adalah “mesin” yang dapat menjalankan suatu **proses di waktu tertentu**, misalnya setiap 1 detik. Fitur ini dimiliki oleh `QObject` yang merupakan leluhur semua class pada Qt.

`startTimer()` berfungsi untuk menjalankan timer dan pada contoh di halaman 111 ia melaksanakan event `timerEvent()` setiap 500 milidetik (setengah detik).

Class `WaktuDigital` di bawah ini juga merupakan jam digital namun dengan informasi yang lebih lengkap.

```
waktudigital.py
-----
01| from qt import *
02| from tanggal import Tanggal
03|
04| class WaktuDigital(QLabel):
05|     def __init__(self, parent):
06|         QLabel.__init__(self, parent)
07|         self.startTimer( 500 )
08|
09|     def timerEvent(self, e):
10|         w = QDateTime.currentDateTime()
11|         t = w.date()
12|         tgl = Tanggal( t.year(), t.month(), t.day() )
13|         _tgl = tgl.toString("hari pasaran d bulan yyyy")
14|         _jam = w.time().toString("hh:mm:ss")
15|         s = "%s %s" % (_tgl, _jam)
16|         self.setText( s )
17|
18|
19| if __name__ == "__main__":
20|     app = QApplication([])
21|     fm = WaktuDigital(None)
22|     fm.resize(300,50)
23|     fm.show()
24|     app.setMainWidget(fm)
25|     app.exec_loop()
```

Bab 20

Form Dialog

Form dialog (`QDialog`) dibangun dengan konsep untuk melaksanakan suatu fungsi form yang singkat, misalnya menampilkan pesan, pengisian username dan password untuk login, dsb. `QDialog` biasanya dipanggil dari form lain yang menjadi parent-nya. Beberapa fiturnya antar lain:

1. Fungsi `accept()` untuk menutup form sehingga `result()` menghasilkan `QDialog.Accepted`. Biasanya untuk "meng-iyakan" input.
2. Fungsi `reject()` juga menutup form, namun `result()` bernilai `QDialog.Rejected` yang berarti membatalkan input.
3. Bila Enter ditekan sama artinya dengan meng-klik tombol

default. Default tidaknya suatu tombol dapat diset dengan fungsi `QPushButton.setDefault()`.

4. Bila Escape ditekan sama artinya dengan pemanggilan `reject()`.

dialog.py

```

01| from qt import *
02|
03| class FormNama(QDialog):
04|     def __init__(self, parent):
05|         QDialog.__init__(self, parent)
06|         self.setCaption("Nama")
07|         layout = QVBoxLayout( self )
08|         layout.setAutoAdd(1)
09|
10|         self.nama = QLineEdit( self )
11|         wadah      = QWidget  ( self )
12|
13|         layout      = QHBoxLayout( wadah )
14|         layout.setAutoAdd(1)
15|         self.tombolOK      = QPushButton( wadah )
16|         self.tombolBatal   = QPushButton( wadah )
17|
18|         self.tombolOK.setText      ("%OK")
19|         self.tombolBatal.setText ("%Batal")
20|         self.tombolOK.setDefault(1)
21|         self.connect(self.tombolOK, SIGNAL("clicked()"),
22|             self.ok)

```



```
23|     self.connect(self.tombolBatal, SIGNAL("clicked()"),
24|                 self.batal)
25|
26|     def ok(self):
27|         self.accept()
28|
29|     def batal(self):
30|         self.reject()
31|
32| app = QApplication([])
33| fm = FormNama(None)
34| fm.show()
35| fm.exec_loop()
36| if fm.result() == QDialog.Accepted:
37|     print "Inputnya:", fm.nama.text()
38| else:
39|     print "Input dibatalkan"
```

Pemanggilan `exec_loop()` pada form menyebabkan alur program “terhenti” hingga form tersebut ditutup. Kondisi ini biasa disebut sebagai *showmodal* yang kerap dijumpai pada `QDialog`.

Class ini merupakan basis bagi dialog lainnya seperti yang sering ditemui dalam berbagai aplikasi, antara lain: `QFileDialog`, `QMessageBox`, atau `QInputDialog`.

20.1 File Dialog

File dialog (`QFileDialog`) digunakan untuk hal yang berkaitan dengan file. Form ini berisi daftar nama file dan direktori ser-

Gambar 20.1: File Dialog

ta memiliki fungsi lainnya seperti pindah direktori, membuat direktori, dsb.

Tombol-tombol text editor pada contoh sebelumnya belum berfungsi sebagaimana mestinya. `texteditor4.py` berikut ini akan melengkapi fungsi tersebut.

```
texteditor4.py
-----
01| from qt import *
02|
03| class FormEditor(QWidget):
04|     def __init__(self):
05|         QWidget.__init__(self)
06|         self.setCaption("Text Editor")
07|         layout = QVBoxLayout( self )
08|         layout.setAutoAdd(1)
09|         panel      = QFrame    ( self )
10|         self.editor = QTextEdit( self )
11|         font       = self.editor.font()
12|         font.setFamily("Courier")
13|         self.editor.setFont( font )
14|
15|         layout = QHBoxLayout( panel )
16|         layout.setAutoAdd(1)
17|         layout.addStretch()
```

```
18|     layout.setDirection( QVBoxLayout.RightToLeft )
19|
20|     btSimpanSbg = QPushButton( panel )
21|     btSimpan    = QPushButton( panel )
22|     btBuka     = QPushButton( panel )
23|     btBaru     = QPushButton( panel )
24|
25|     btBaru.setText    ( "Baru" )
26|     btBuka.setText    ( "Buka" )
27|     btSimpan.setText  ( "Simpan" )
28|     btSimpanSbg.setText( "Simpan Sebagai" )
29|
30|     self.setFilename( None )
31|     self.showMaximized()
32|
33|     self.connect(btBaru, SIGNAL("clicked()"), self.baru)
34|     self.connect(btBuka, SIGNAL("clicked()"), self.buka)
35|     self.connect(btSimpan, SIGNAL("clicked()"),
36|                 self.simpan)
37|     self.connect(btSimpanSbg, SIGNAL("clicked()"),
38|                 self.simpanSbg)
39|
40|     def _simpan(self, filename):
41|         f = open( filename, "w" )
42|         f.write( "%s" % self.editor.text() )
43|         f.close()
44|
45|     def setFilename(self, filename):
46|         self.filename = filename
```

```
47|         self.setCaption("Text Editor - %s" % self.filename)
48|
49|     def baru(self):
50|         self.editor.clear()
51|         self.setFilename( None )
52|
53|     def buka(self):
54|         filename = QFileDialog(self).getOpenFileName()
55|         if not filename.isEmpty():
56|             _filename = "%s" % filename
57|             f = open( _filename )
58|             self.editor.setText( "%s" % f.read() )
59|             f.close()
60|             self.setFilename( _filename )
61|
62|     def simpan(self):
63|         if self.filename:
64|             self._simpan( "%s" % self.filename )
65|         else:
66|             self.simpanSbg()
67|
68|     def simpanSbg(self):
69|         filename = QFileDialog(self).getSaveFileName()
70|         if not filename.isEmpty():
71|             _filename = "%s" % filename
72|             self._simpan( _filename )
73|             self.setFilename( _filename )
74|
75| app = QApplication([])
```

```
76| fm = FormEditor()
77| app.setMainWidget(fm)
78| app.exec_loop()
```

`getOpenFileName()` dan `getSaveFileName()` merupakan fungsi siap pakai untuk menampilkan file dialog. Keduanya mengembalikan nama file (`QString`) apabila tombol OK diklik. Jika tidak, nama filenya dikosongkan.

`texteditor4.py` juga menyertakan perubahan font, yaitu `Courier` yang bersifat *fixed*, artinya ukuran huruf “i” dan huruf “M” sama besar, hal yang biasa kita perlukan untuk mengedit suatu file teks.

Latihan

`texteditor4.py` sudah memenuhi syarat untuk sebuah aplikasi text editor, namun perlu ditingkatkan lagi faktor kenyamanan penggunaannya, terutama berkaitan dengan tombol:

1. Saat pertama kali program dijalankan, tombol yang aktif hanyalah “Buka”.
2. Adanya “status perubahan” (*modified*) yang menunjukkan apakah ada perubahan pada editor sejak ia terakhir disimpan. Bila ya maka ke empat tombol aktif. Sebaliknya bila tidak terjadi perubahan - misalnya setelah tombol “Simpan” di-klik - maka tombol yang aktif adalah “Baru” dan “Simpan Sebagai”.

Petunjuk Gunakan `setEnabled()`.

Gambar 20.2: Text Editor

20.2 Pesan & Konfirmasi

Penyempurnaan berlanjut terus. Kali ini melibatkan faktor pengamanan data, yaitu adanya kemungkinan pemakai melakukan hal-hal yang menyebabkan hilangnya data pada saat seperti berikut ini:

1. Menutup aplikasi
2. Membuka file
3. Memulai file baru
4. Menyimpan file dengan nama file yang sudah ada

Oleh karena itu diperlukan suatu konfirmasi terlebih dahulu sebelum hal-hal yang tidak diinginkan terjadi. Konfirmasi ini berupa sebuah form yang berisi pertanyaan yang perlu dijawab. Qt telah menyediakan `QMessageBox` untuk kebutuhan tersebut dimana ia juga telah dilengkapi dengan gambar (*icon*) sehubungan dengan pesan yang berada di dalamnya.

```
texteditor5.py
-----
001| from qt import *
002|
```

```
003| class FormEditor(QWidget):
004|     def __init__(self):
005|         QWidget.__init__(self)
006|         self.setCaption("Text Editor")
007|         layout = QVBoxLayout( self )
008|         layout.setAutoAdd(1)
009|         panel      = QFrame    ( self )
010|         self.editor = QTextEdit( self )
011|         font        = self.editor.font()
012|         font.setFamily("Courier")
013|         self.editor.setFont( font )
014|
015|         layout = QHBoxLayout( panel )
016|         layout.setAutoAdd(1)
017|         layout.addStretch()
018|         layout.setDirection( QVBoxLayout.RightToLeft )
019|
020|         btSimpanSbg = QPushButton( panel )
021|         btSimpan     = QPushButton( panel )
022|         btBuka       = QPushButton( panel )
023|         btBaru       = QPushButton( panel )
024|
025|         btBaru.setText      ( "Baru" )
026|         btBuka.setText      ( "Buka" )
027|         btSimpan.setText    ( "Simpan" )
028|         btSimpanSbg.setText( "Simpan Sebagai" )
029|
030|         self.setFilename( None )
031|         self.showMaximized()
```

```
032|
033|     self.connect(btBaru, SIGNAL("clicked()"), self.baru)
034|     self.connect(btBuka, SIGNAL("clicked()"), self.buka)
035|     self.connect(btSimpan, SIGNAL("clicked()"),
036|                 self.simpan)
037|     self.connect(btSimpanSbg, SIGNAL("clicked()"),
038|                 self.simpanSbg)
039|     self.connect( self.editor, SIGNAL("textChanged()")
040|                 self.teksBerubah)
041|
042|     def teksBerubah(self):
043|         self.berubah = 1
044|
045|     def lanjutkan(self):
046|         if not self.berubah: return 1
047|         id = QMessageBox.warning( self,
048|                                   "Perhatian",
049|                                   "Simpan perubahan terlebih dahulu ?",
050|                                   "Ya", "Tidak perlu", "Kembali")
051|         if id == 0: return self.simpan()
052|         return id == 1
053|
054|     def _simpan(self, filename=None):
055|         if filename: _filename = filename
056|         else         : _filename = self.filename
057|         f = open( _filename, "w" )
058|         f.write( "%s" % self.editor.text() )
059|         f.close()
060|
```



```
061|     def setFilename(self, filename):
062|         self.filename = filename
063|         self.setCaption("Text Editor - %s" % self.filename)
064|         self.berubah = 0
065|
066|     def baru(self):
067|         if not self.lanjutkan(): return
068|         self.editor.clear()
069|         self.setFilename( None )
070|
071|     def buka(self):
072|         if not self.lanjutkan(): return
073|         filename = QFileDialog(self).getOpenFileName()
074|         if not filename.isEmpty():
075|             _filename = "%s" % filename
076|             f = open( _filename )
077|             self.editor.setText( "%s" % f.read() )
078|             f.close()
079|             self.setFilename( _filename )
080|
081|     def simpan(self):
082|         if not self.berubah: return
083|         if self.filename:
084|             self._simpan()
085|             self.berubah = 0
086|             return 1
087|         else:
088|             return self.simpanSbg()
089|
```

```
090| def simpanSbg(self):
091|     if not self.berubah: return
092|     tanya = 1
093|     while tanya:
094|         filename = QFileDialog(self).getSaveFileName()
095|         if filename.isEmpty(): return
096|         if QFile.exists( filename ):
097|             pesan="File %s sudah ada. Timpa ?" % filename
098|             id = QMessageBox.warning( self, "Perhatian",
099|                 pesan, "Ya", "Jangan", "Kembali")
100|             if id == 2: return
101|             if id == 0: tanya = 0
102|             else:
103|                 tanya = 0
104|             _filename = "%s" % filename
105|             self._simpan( _filename )
106|             self.setFilename( _filename )
107|             return 1
108|
109|     def closeEvent(self, e):
110|         if self.lanjutkan(): e.accept()
111|         else : e.ignore()
112|
113| app = QApplication([])
114| fm = FormEditor()
115| app.setMainWidget(fm)
116| app.exec_loop()
```

`QMessageBox` menyediakan fungsi `warning()` yang mengembalikan nilai berupa nomor index tombol yang di-klik.

Fungsi `closeEvent()` (milik `QWidget`) akan dipanggil pada saat form akan ditutup. Fungsi ini menyertakan parameter `e` bertipe `QCloseEvent` yang memiliki fungsi `accept()` bermakna form ditutup dan `ignore()` yang berarti tidak jadi ditutup.

Latihan Tambahkan fasilitas untuk mencetak. Anda dapat menggunakan perintah Linux `cat namafile > /dev/lp0`.

20.3 Input

Bila dalam program Anda ada sebuah proses yang membutuhkan sebuah masukan saja, maka `QInputDialog` adalah jawaban yang tepat. Berikut ini program kasir “layak pakai” dimana nama barang, jumlah, serta harga satuannya dimasukkan menggunakan `QInputDialog`. Tidak ketinggalan nilai pembayarannya juga melalui class yang sama.

```
input.py
-----
01| from qt import *
02| from string import rjust, ljust, upper
03| from locale import setlocale, LC_ALL, format
04| import sys
05|
06| def cetak(s):
07|     if sys.argv[1:]: output = sys.argv[1]
08|     else:             output = "/dev/lp0"
```

```
09|     file = open(output,"w")
10|     c = chr(15) + s + "\n"
11|     file.write(c)
12|     file.close()
13|
14| def cetakBrg(nama, jml, hrg):
15|     j = format("%.2f", jml, 1)
16|     h = format("%d", hrg, 1)
17|     a = ljust(nama[:20], 20)
18|     b = rjust(j, 7)
19|     c = rjust(h, 8)
20|     s = "%s%s%s" % (a,b,c)
21|     cetak(s)
22|
23| def cetakAkhir(nama, nilai):
24|     n = format("%d", nilai, 1)
25|     a = ljust(nama, 27)
26|     b = rjust(n, 8)
27|     s = "%s%s" % (a,b)
28|     cetak(s)
29|
30|
31| setlocale(LC_ALL, "")
32| total = 0
33| app = QApplication([])
34| while 1:
35|     nama, ok = QDialog.getText("Kasir",
36|         "Nama barang (Enter: Total, Esc: Selesai)",
37|         QLineEdit.Normal, "")
```

```
38|     if not ok: break
39|     if nama.isEmpty():
40|         t = format("%d", total, 1)
41|         bayar, ok = QInputDialog.getInteger("Kasir",
42|             "Pembayaran (Total Rp %s)" % t, total)
43|         if ok:
44|             kembali = bayar - total
45|             cetakAkhir("TOTAL", total)
46|             cetakAkhir("BAYAR", bayar)
47|             cetakAkhir("KEMBALI", kembali)
48|             c = "\n" * 5
49|             cetak(c)
50|             total = 0
51|     else:
52|         nama = upper("%s" % nama)
53|         jml, ok = QInputDialog.getDouble(nama,
54|             "Jumlah barang", 1, 0, 9999999, 2)
55|         if ok:
56|             hrg, ok = QInputDialog.getInteger(nama,
57|                 "Harga satuan")
58|             if ok:
59|                 subtotal = round(hrg * jml)
60|                 cetakBrg(nama, jml, subtotal)
61|                 total = total + subtotal
```

Siapkan printer, dan jalankan program ini:

```
$ python input.py
```

namun bila printer tidak ada, masukkan nama file pengganti:

```
$ python input.py /tmp/kasir.txt
```

Pada contoh di atas `QInputDialog` memiliki tiga fungsi untuk memperoleh nilai string, integer, dan float:

```
getText(caption, label, mode, nilai="")  
getInteger(caption, label, nilai=0)  
getDouble(caption, label, nilai=0, terkecil=-214748364, terbesar=214748364, desimal=1)
```

Bab 21

Tabel

Tabel merupakan daftar yang terdiri dari baris dan kolom. Qt telah menyediakan class `QTable` yang terdapat pada modul `qtable`. Sifat `QTable` sangat mirip dengan *sheet* yang bisa kita jumpai pada aplikasi *spreadsheet*¹.

`qtable1.py` memberikan contoh sederhana penggunaan `QTable`. Adapun penjelasan yang ingin disampaikan adalah:

1. Sumber data berupa list dua dimensi sebagai perwujudan dari struktur tabel.
2. Jumlah baris ditentukan dari sumber data.
3. Jumlah kolom ditentukan dari daftar judul kolom.

¹Spreadsheet merupakan aplikasi yang sering digunakan dalam aplikasi office. Contohnya: StarOffice, OpenOffice, KOffice, dst.

Gambar 21.1: QTable

4. Bagaimana mengisi nilai pada *cell*.²
5. Bagaimana mengambil nilai dari *cell*.

```

qtable1.py
-----
01| from qt import *
02| from qtable import QTable
03|
04| class FormTable(QWidget):
05|     def __init__(self):
06|         QWidget.__init__(self)
07|         self.setCaption("Table")
08|         kolom = [ "Nama", "Alamat" ]
09|         isi = [
10|             [ "Pribadi Endro", "Kemayoran" ],
11|             [ "Ahmad", "Purwokerto" ],
12|             [ "Putera Sinaga", "Bogor" ]
13|         ]
14|         self.t = QTable(self)
15|         self.t.setNumCols( len(kolom) )
16|         self.t.setNumRows( len(isi) )
17|         i = -1

```

²Cell: elemen pada QTable


```

18|     for judul in kolom:
19|         i = i + 1
20|         self.t.horizontalHeader().setLabel( i, judul )
21|     b = -1
22|     for brs in isi:
23|         b = b + 1
24|         k = -1
25|         for nilai in brs:
26|             k = k + 1
27|             self.t.setText( b,k, nilai )
28|
29|     self.show()
30|     self.connect( self.t,
31|         SIGNAL("currentChanged(int,int)"),
32|         self.posisiBerubah )
33|
34|     def posisiBerubah(self, row, col):
35|         self.setCaption( self.t.text( row, col ) )
36|
37| app = QApplication([])
38| fm = FormTable()
39| app.setMainWidget(fm)
40| app.exec_loop()

```

Penjelasan mengenai fungsi `QTable` yang digunakan adalah:

`setNumCols()` menentukan jumlah kolom.

`setNumRows()` menentukan jumlah baris.

`horizontalHeader()` mendapatkan objek header (judul kolom) yang bertipe `QHeader`. Fungsi `setLabel()`-nya digunakan untuk memberikan judul pada kolom tertentu.

`setText()` mengisi teks pada elemen (*cell*) tertentu.

`text()` mendapatkan teks pada cell tertentu.

`currentChanged()` sinyal yang muncul pada saat *current cell*³ berubah.

Secara default `QTable` mengizinkan pemakai untuk mengubah isi, lebar baris, dan lebar kolom pada saat runtime.

Beberapa tombol keyboard berikut ini juga dapat Anda gunakan:

F2 isi cell siap diubah. Pada kasus isi cell terlalu panjang namun Anda hanya ingin mengubah sedikit saja maka tombol ini sangat berguna.

Escape membatalkan perubahan

21.1 Mengubah Sifat

Untuk berbagai keperluan kita perlu mengubah sifat `QTable` dengan membuat class baru sebagai keturunannya. Untunglah `QTable` memiliki banyak fungsi yang memungkinkan perubahan

³Current cell: cell yang sedang fokus.

sifat ini dilakukan dengan mudah. Class tersebut kita namakan saja **Grid**. Salah satu perubahan yang dimaksud adalah pada penggunaan tombol keyboard berikut ini:

Insert	menyisipkan baris
Down	menambah baris bila baris aktif (<i>current row</i>) berada di baris terakhir.
Delete	menghapus isi cell
Ctrl-Delete	menghapus baris
Home	menuju kolom paling kiri yang tampak ⁴ pada baris aktif (<i>current row</i>)
End	menuju kolom paling kanan yang tampak pada baris aktif
Ctrl-Home	menuju baris pertama dan kolom paling kiri yang tampak (kiri atas)
Ctrl-End	menuju baris pertama dan kolom paling kanan yang tampak (kanan bawah)

Selain itu ada beberapa sifat lainnya yang ditambahkan:

⁴Kolom bisa disembunyikan dengan perintah `setHide()`. Kolom seperti ini dihindari sebagai kolom aktif (*current column*). Sehingga dikatakan bahwa “kolom paling kiri” belum tentu merupakan “kolom pertama”.

1. Pada saat tidak ada baris, `QTable` tidak menunjukkan bahwa dirinya merupakan *focus widget*.⁵ Pada form yang memiliki widget lebih dari satu tentu saja bisa membingungkan pemakai. Pada `Grid` minimal terdapat satu baris agar tampak *focus*-nya.
2. `QTable` tidak memiliki sarana untuk menambah baris secara langsung. Pada `Grid` penekanan tombol Insert akan menyisipkan baris. Bahkan penekanan tombol Down juga akan menambah baris.
3. Penekanan tombol Enter setelah mengubah nilai cell akan mengarahkan kursor ke kanan.
4. Penambahan baris baru mengarahkan kursor ke kolom paling kiri agar memudahkan pengisian.

grid.py

```
01| from qt import *
02| from qtable import QTable
03|
04| class Grid(QTable):
05|     def __init__(self, parent):
06|         QTable.__init__(self, parent)
07|         self.setNumRows(1)
08|
09|     def activateNextCell(self):
10|         if self.currentColumn() < self.numCols()-1:
```

⁵focus widget: objek aktif yang siap menerima input atau aksi lainnya.

```
11|         self.setCurrentCell(self.currentRow(),
12|             self.currentColumn()+1)
13|
14|     def setNumRows(self, count):
15|         if count > 1: c = count
16|         else:         c = 1
17|         QTable.setNumRows(self, c)
18|
19|     def insertRows(self, row, count=1):
20|         QTable.insertRows(self, row, count)
21|         self.setCurrentCell( row, self.kolomKiri() )
22|
23|     def removeRow(self, row):
24|         r = self.currentRow()
25|         if self.numRows() > 1: QTable.removeRow(self, row)
26|         else:
27|             for col in range( self.numCols() ):
28|                 self.clearCell( row, col )
29|         if r == self.numRows(): r = self.numRows() - 1
30|         self.setCurrentCell( r, self.currentColumn() )
31|
32|     def sisip(self):
33|         self.insertRows( self.currentRow() )
34|
35|     def turun(self):
36|         if self.currentRow() == self.numRows()-1:
37|             self.insertRows( self.numRows() )
38|         else:
39|             self.setCurrentCell( self.currentRow()+1,
```

```
40|         self.currentColumn() )
41|
42| def hapusCell(self):
43|     self.clearCell( self.currentRow(),
44|         self.currentColumn() )
45|
46| def hapusBaris(self):
47|     self.removeRow( self.currentRow() )
48|
49| def kolomKiri(self):
50|     col = 0
51|     while self.columnWidth(col) == 0:
52|         if col == self.numCols()-1: return col
53|         col = col + 1
54|     return col
55|
56| def kolomKanan(self):
57|     col = self.numCols()-1
58|     while self.columnWidth(col) == 0:
59|         if col == 0: return col
60|         col = col - 1
61|     return col
62|
63| def kiriAtas(self):
64|     self.setCurrentCell(0, self.kolomKiri() )
65|
66| def palingKiri(self):
67|     self.setCurrentCell( self.currentRow(),
68|         self.kolomKiri() )
```

```
69|
70| def kananBawah(self):
71|     self.setCurrentCell( self.numRows()-1,
72|         self.kolomKanan() )
73|
74| def palingKanan(self):
75|     self.setCurrentCell( self.currentRow(),
76|         self.kolomKanan() )
77|
78| def keyPressEvent(self, e):
79|     if self.numCols() == 0: return
80|     elif e.key() == Qt.Key_Insert: self.sisip()
81|     elif e.key() == Qt.Key_Down: self.turun()
82|     elif e.key() == Qt.Key_Delete:
83|         if e.state() == Qt.ControlButton: self.hapusBaris()
84|         else: self.hapusCell()
85|     elif e.key() == Qt.Key_Home:
86|         if e.state() == Qt.ControlButton: self.kiriAtas()
87|         else: self.palingKiri()
88|     elif e.key() == Qt.Key_End:
89|         if e.state() == Qt.ControlButton: self.kananBawah()
90|         else: self.palingKanan()
91|     else: QTable.keyPressEvent(self, e)
92|
93| if __name__ == "__main__":
94|     app = QApplication([])
95|     fm = Grid(None)
96|     fm.setNumCols(5)
97|     fm.show()
```

```

98|   app.setMainWidget(fm)
99|   app.exec_loop()

```

Berikut ini fungsi `QTable` yang dituliskan (*reimplementation*):

`__init__()` memastikan `Grid` memiliki satu baris.

`activateNextCell()` dipanggil saat pemakai menekan Enter setelah selesai mengubah isi cell (*edit-mode*⁶ berakhir)

`setNumRows()` diubah sifatnya agar `Grid` - setidaknya - memiliki satu baris.

`insertRows()` menyisipkan baris dan menempatkan kursor di kolom paling kiri.

`removeRow()` menghapus baris namun tetap memastikan `Grid` memiliki setidaknya satu baris.

`clearCell()` menghapus isi cell.

`setCurrentCell()` mengarahkan kursor ke cell tertentu. Fungsi ini berpengaruh pada nilai `currentRow()` dan `currentColumn()`.

`keyPressEvent()` dipanggil saat tombol pada keyboard ditekan dimana cell tidak sedang di-edit (tidak berlaku pada saat *edit-mode*).

⁶*Edit-mode*: saat suatu cell sedang diubah isinya. Saat ini ditandai dengan adanya editor (`QLineEdit`) pada cell.

Gambar 21.2: ValueGrid

21.2 Bentuk Tampilan

Berkaitan dengan angka biasanya kita dihadapkan pada masalah tampilan seperti rata kanan, pemisah ribuan, atau jumlah angka dibelakang koma pada bilangan pecahan. `QTable` memiliki fungsi `paintCell()` yang memang bertugas untuk menampilkan tulisan. `paintCell()` perlu ditulis ulang agar data ditampilkan dengan bentuk yang diinginkan.

Kita akan membuat sebuah class baru bernama `ValueGrid` yang merupakan keturunan class `Grid` sebelumnya dengan ciri sebagai berikut:

1. Struktur data disimpan dalam list dua dimensi yang mencerminkan bentuk tabel.
2. Tipe data elemennya bisa string, integer, atau float.
3. Integer dan float ditampilkan rata kanan dan disertai pemisah ribuan.
4. Pemisah ribuan menggunakan karakter titik, contoh: 2.340.
5. Float ditampilkan dengan dua angka pecahan, contoh: 9.234,25.
6. Pemisah pecahan menggunakan karakter koma.

7. Input untuk bilangan pecahan bisa menggunakan pemisah koma maupun titik.
8. Meniru spreadsheet, `ValueGrid` juga mengenal bentuk rumus matematika, yang penulisannya perlu diawali dengan karakter samadengan (=). Sehingga string “=80*5” akan ditampilkan angka 40. Namun yang tersimpan dalam struktur data tetap string tersebut, bukan hasilnya. Fungsi `exec()` digunakan untuk menerjemahkannya.
9. Baris aktif (*current row*) diterangi (*highlight*).

Sebelumnya akan kita buat dulu dua buah fungsi pendukung yaitu `angka()` dan `ribu()`. `angka()` menerima masukan bertipe apa saja yang akan menerjemahkannya menjadi bilangan. Sedangkan `ribu()` menerima masukan berupa bilangan integer atau float dan mengembalikan string dari bilangan tersebut dalam bentuk sebagaimana pembahasan sebelumnya. Karena ini bisa dianggap sebagai fungsi umum, maka sebaiknya kita letakkan keduanya dalam suatu modul.

```
fungsi.py
-----
01| import string
02| import locale
03| from math import *
04|
05| locale.setlocale(locale.LC_ALL, "")
06|
07|
```



```
37|         return
```

Modul ini digunakan dalam `valuegrid.py` yang selain dapat sebagai modul juga dapat dijalankan sebagai program utama untuk mencoba.

```
valuegrid.py
-----
001| from qt import *
002| from qtable import QTable
003| from grid import Grid
004| from fungsi import angka, ribu
005|
006| class ValueGrid(Grid):
007|     def __init__(self, parent):
008|         Grid.__init__(self, parent)
009|         self.penBox = QPen()
010|         self.penBox.setColor( QColor("lightgray") )
011|         self.penText = QPen()
012|         self.penText.setColor( QColor("black") )
013|         self.brushCurrent = QBrush()
014|         self.brushCurrent.setColor( QColor("yellow") )
015|         self.brushCurrent.setStyle( QBrush.SolidPattern )
016|         self.data = []
017|         self._row = 0 # baris terakhir sebelum currentRow
018|         self.beforeDelete = None # sebelum data dihapus
019|         self.connect(self, SIGNAL("currentChanged(int,int)"),
020|             self.saatPindah)
021|
```

```
022|     def resizeData(self, i):
023|         pass # hemat memory
024|
025|     def setData(self, data):
026|         if data:
027|             self.data = data
028|             self.setNumRows( len(data) )
029|         else:
030|             self.data = []
031|             self.setNumRows(1)
032|             self.setCurrentCell( 0, self.currentColumn() )
033|             self.repaintContents()
034|
035|     def setNumCols(self, count):
036|         self.barisKosong = []
037|         for i in range( count ):
038|             self.barisKosong.append("")
039|         QTableWidgetItem.setNumCols(self, count)
040|
041|     def insertRows(self, row, count=1):
042|         if not self.data:
043|             self.data.append( self.barisKosong )
044|         for i in range(count):
045|             self.data.insert( (row, list(self.barisKosong)) )
046|         Grid.insertRows(self, row, count)
047|
048|     def removeRow(self, row):
049|         if self.data:
050|             if self.beforeDelete: self.beforeDelete(row)
```

```
051|         del self.data[ row ]
052|         Grid.removeRow(self, row)
053|
054|     def createEditor(self, row, col, initFromCell):
055|         e = QLineEdit(self)
056|         if self.data and initFromCell:
057|             e.setText("%s" % self.data[row][col])
058|         return e
059|
060|     def setCellContentFromEditor(self, row, col):
061|         s = "%s" % self.cellWidget( row, col ).text()
062|         if s and s[0] != "=":
063|             a = angka(s)
064|             if a: s = a
065|         if not self.data:
066|             self.data.append( list(self.barisKosong) )
067|         self.data[row][col] = s
068|
069|     def clearCell(self, row, col):
070|         if self.data:
071|             self.data[row][col] = ""
072|         self.updateCell(row, col)
073|
074|     def paintCell(self, painter, row, col, cr, selected):
075|         align = QPainter.RTL
076|         if self.data:
077|             d = self.data[row][col]
078|             a = angka(d)
079|             if a:
```

```
080|         s = ribu(a)
081|         align = QPainter.LTR
082|     else:
083|         s = d
084|     else:
085|         s = ""
086|     if row == self.currentRow() and self.data:
087|         painter.fillRect( 0,0, cr.width(), cr.height(),
088|             self.brushCurrent )
089|     else:
090|         painter.eraseRect( 0,0, cr.width(), cr.height() )
091|     painter.setPen( self.penBox )
092|     painter.drawLine( 0, cr.height()-1, cr.width(),
093|         cr.height()-1 )
094|     painter.drawLine( cr.width()-1, cr.height()-1,
095|         cr.width()-1, 0 )
096|     painter.setPen( self.penText )
097|     painter.drawText(2,2, cr.width()-4, cr.height()-4,
098|         align, s)
099|
100| def saatPindah(self, row, col):
101|     if row != self._row:
102|         r = self.cellGeometry( self._row, 0 )
103|         r.setRight( self.width() )
104|         self.repaintContents(r)
105|         r = self.cellGeometry( row, 0 )
106|         r.setRight( self.width() )
107|         self.repaintContents(r)
108|     self._row = row
```

```
109|
110|
111| if __name__ == "__main__":
112|     app = QApplication([])
113|     fm = ValueGrid(None)
114|     data = [ [1001,"Belimbing",3000.0,"Kg"],
115|              [1287,"Jeruk", "=7000+500.0", "Kg" ] ]
116|     fm.setNumCols(4)
117|     fm.setData(data)
118|     fm.show()
119|     app.setMainWidget(fm)
120|     app.exec_loop()
```

Adapun fungsi yang ditulisulang adalah:

`__init__()` menyiapkan struktur data tabel berupa list dua dimensi (`data`), juga `QPen` untuk menulis teks dan menggambar kotak.

`resizeData()` menyiapkan struktur data internal `QTable`. Karena `ValueGrid` memiliki struktur data sendiri maka fungsi ini tidak diperlukan guna menghemat memori. `pass` berarti tidak melakukan apa-apa.

`setNumCols()` penyiapan list berupa baris kosong untuk mempercepat penambahan baris (`record`).

`insertRows()` memastikan jumlah baris data sesuai dengan jumlah baris yang tampak.

`removeRow()` alasannya sama dengan `insertRows()`.

`createEditor()` dipanggil menjelang edit-mode. Fungsi ini harus mengembalikan (`return`) widget yang akan digunakan untuk memasukkan data. Standarnya adalah `QLineEdit`, dan tentunya Anda dapat menggunakan widget lainnya sesuai dengan kebutuhan. Sesuai namanya fungsi ini bisa juga dikatakan sebagai pembuat widget. `initFromCell` bernilai logika yang apabila `true` berarti tombol F2 ditekan.⁷

`setCellContentFromEditor()` dipanggil usai edit-mode. Sampai disini widget yang dibuat `createEditor()` belum dihapus dari memori, sehingga ini saat yang tepat untuk mengisi data sesuai dengan isi widget tersebut. Fungsi `cellWidget()` digunakan untuk mendapatkan widget yang dimaksud.

`clearCell()` mengganti elemen data dengan string hampa sebagai perwujudan proses penghapusan data.

`updateCell()` memanggil `paintCell()` untuk memperbaharui tampilan pada cell tertentu.

`paintCell()` merupakan event yang terjadi saat “ada perintah” untuk menggambar cell tertentu. Misalnya saat form di-*minimize* lalu di-*maximized*. *Reimplementation* ini perlu dilakukan karena `ValueGrid` memiliki struktur data sendiri yang tidak dikenal oleh fungsi `paintCell()` leluhurnya. Fungsi ini menyertakan beberapa variabel yang berguna dalam pros-

⁷Lihat halaman 157 mengenai penggunaan tombol ini.

es penggambaran. `painter` bertipe `QPainter` yang bertugas menggambar cell yang terletak pada `cr` (`QRect` ⁸). `selected` bernilai logika yang apabila `true` berarti kursor berada pada cell di baris `row` kolom `col` (sedang fokus).

Pada saat kursor pindah ke cell lain di baris yang berbeda, `paintCell()` tidak dipanggil untuk menghilangkan efek terang (*highlight*) pada baris sebelumnya. Oleh karena itu digunakanlah fungsi `repaintContents()` yang otomatis akan memanggil `paintCell()` sesuai dengan *area* yang sudah ditentukan. Area yang dimaksud adalah *baris* sebelum dan sesudah perpindahan kursor. Untuk mendapatkan area tersebut berdasarkan nomor baris, `QTable` memiliki fungsi `cellGeometry()` yang akan mengembalikan nilai bertipe `QRect`.

`repaintContents()` tanpa nilai masukan berarti menggambar ulang seluruh area `QTable`. Fungsi ini sebenarnya milik `QScrollView`, leluhur `QTable`.

QPen

Merupakan pena gambar yang dipakai `QPainter`. Fungsi `setColor()` pada contoh di atas digunakan untuk menentukan warnanya.

QBrush

Merupakan “cat” yang dipakai `QPainter` untuk memberi corak pada latar belakang (*background*). Fungsi `setColor()`-nya un-

⁸Lihat halaman 168 tentang `QRect`.

tuk menentukan warna cat, sedangkan `setStyle()` untuk menentukan pola dalam pengecatan. Berikut ini beberapa pola yang bisa diterapkan:

`NoBrush` tidak menggunakan cat (default).

`SolidPattern` kepadatan 100%.

`Dense1Pattern` kepadatan 94%.

`Dense2Pattern` kepadatan 88%.

`Dense3Pattern` kepadatan 63%.

`Dense4Pattern` kepadatan 50%.

`Dense5Pattern` kepadatan 37%.

`Dense6Pattern` kepadatan 12%.

`Dense7Pattern` kepadatan 6%.

`HorPattern` garis horizontal.

`VerPattern` garis vertikal.

`CrossPattern` kotak-kotak.

`BDiagPattern` garis miring ke kanan (/).

`FDiagPattern` garis miring ke kiri (\).

`DiagCrossPattern` belah ketupat.

QPainter

Merupakan alat gambar pada widget. Fungsi yang digunakan pada contoh di atas adalah:

`fillRect()` mengecat bidang tertentu.

`eraseRect()` menghapus bidang tertentu.

`setPen()` menentukan pena (`QPen`) untuk menggambar.

`drawLine()` membuat garis.

`drawText()` membuat tulisan. Fungsi ini memiliki parameter masukan `align` yang menentukan apakah tulisan ditampilkan rata kiri (RTL) atau rata kanan (LTR).

QRect

Merupakan definisi suatu kotak dalam “bidang gambar” (*shape*). Definisi yang dimaksud seperti posisi dan ukuran berikut ini:

`x()` koordinat dari kiri. Untuk mengubahnya gunakan `setX()`.

`y()` koordinat dari atas. Untuk mengubahnya gunakan `setY()`.

`height()` lebar kotak. Untuk mengubahnya gunakan `setHeight()`.

`width()` panjang kotak. Untuk mengubahnya gunakan `setWidth()`.

- `left()` koordinat kiri, sama dengan `x()`. Untuk mengubahnya gunakan `setLeft()`.
- `right()` koordinat kanan, sama dengan `left() + width()`. Untuk mengubahnya gunakan `setRight()`.
- `top()` koordinat atas, sama dengan `y()`. Untuk mengubahnya gunakan `setTop()`.
- `bottom()` koordinat bawah, sama dengan `top() + height()`. Untuk mengubahnya gunakan `setBottom()`.

21.3 Form Pencarian

Form Pencarian adalah form yang digunakan sebagai alat untuk mencari kata tertentu dalam suatu kolom tabel. Kita ambil contoh sebuah tabel barang yang memuat nama barang dan harganya. Form Pencarian akan menggunakan kolom nama sebagai tempat pencarian. Form ini memiliki fitur sebagai berikut:

1. Terdapat dua widget utama: `QLineEdit` untuk menuliskan teks yang dicari dan `ValueGrid` sebagai tabelnya.
2. Pencarian berdasarkan nama dan tidak membedakan huruf kecil maupun besar. Misalkan kata KAPAL dimasukkan, maka kursor akan mengarah pada baris dimana terdapat nama barang yang diawali KAPAL atau misalnya Kapal (sama saja). Namun apabila tidak ditemukan, maka akan dicari nama barang yang mengandung kata KAPAL. Sampai di sini pencarian bisa dilanjutkan dengan

menekan tombol Ctrl-PageDown. Apabila tidak ditemukan juga, maka akan ditampilkan pesan.

3. Navigasi untuk perpindahan kursor menggunakan tombol panah atas atau panah bawah.
4. Tombol Enter digunakan untuk menutup form sekaligus menyatakan bahwa barang telah dipilih sesuai dengan yang ditunjuk kursor. Bermanfaat manakala form ini dipakai oleh form lain sebagai alat bantu pencarian barang.
5. Penekanan tombol Esc akan menghapus kata yang dicari, dan bisa juga berfungsi untuk menutup form apabila memang tidak ada yang dihapus.

Sebagai sumber data akan kita gunakan file teks dengan contoh seperti di bawah ini:

```
Nama;Harga
mangga harum manis;9000
mangga indramayu;7000
jeruk medan;6500
semangka tanpa biji;3000
semangka kuning;5000
duku;5500
rambutan binjai;3000
rambutan rapih;6000
durian monthong;10000
alpukat;4000
anggur merah;20000
```

```
anggur hijau;15000
jeruk lokam;8000
jeruk pakistan;10000
mangga golek;5000
mangga gedong;15000
lengkeng impor;10000
lengkeng lokal;7000
pir australia;12000
pir shandong;9000
```

Seperti terlihat pada contoh di atas, pemisah antar field menggunakan karakter titik koma (;). Baris pertama merupakan judul kolom dan baris berikutnya merupakan datanya. Untuk menangani file ini akan dibuatkan sebuah class bernama `TableFile`.

Data boleh ditulis dengan huruf kecil maupun besar karena nanti semuanya akan ditampilkan dengan huruf besar secaraurut berdasarkan kolom pertama. Program ini menggunakan modul `linecase`⁹ untuk memastikan kata yang dicari ditulis dengan huruf besar.

```
cari.py
-----
001| from qt import *
002| from valuegrid import ValueGrid
003| from string import splitfields, strip, upper
004| from linecase import LineCase
005|
```

⁹Lihat halaman 94.

```
006| class FormCari(QDialog):
007|     def __init__(self, parent, data, labels=[],
008|         kolomCari=0):
009|         QDialog.__init__(self, parent)
010|         layout = QVBoxLayout(self)
011|         self.teks = LineCase(self, LineCase.Upper)
012|         self.tabel = ValueGrid(self)
013|         layout.addWidget(self.teks)
014|         layout.addWidget(self.tabel)
015|         self.tabel.setNumCols( len(labels) )
016|         i = -1
017|         for label in labels:
018|             i = i + 1
019|             self.tabel.horizontalHeader().setLabel(i,label)
020|         self.tabel.setData( data )
021|         self.kolomCari = kolomCari
022|         self.teks.teksBerubah = self.teksBerubah
023|
024|     def keyPressEvent(self, e):
025|         if e.key() == Qt.Key_Escape: self.escape()
026|         elif e.key() in [Qt.Key_Return, Qt.Key_Enter]:
027|             self.accept()
028|         elif e.state() == Qt.ControlButton and \
029|             e.key() == Qt.Key_PageDown: self.lanjut()
030|         else: self.tabel.keyPressEvent( e )
031|
032|     def escape(self):
033|         if self.teks.text().isEmpty(): self.reject()
034|         else: self.teks.clear()
```



```
035|
036| def teksBerubah(self, w):
037|     if self.cari() < 0 and self.cari(0) < 0:
038|         QMessageBox.warning(self, "Perhatian",
039|             "%s tidak ditemukan" % w.text())
040|
041| def cari(self, awalan=1, mulaiBaris=-1):
042|     s = "%s" % self.teks.text().upper()
043|     brs = mulaiBaris
044|     ketemu = 0
045|     while brs < self.tabel.numRows()-1:
046|         brs = brs + 1
047|         data = self.tabel.data[brs][self.kolomCari]
048|         t = QString(data).upper()
049|         if awalan: ketemu = t.find(QRegExp("^"+s)) > -1
050|         else: ketemu = t.find(s) > -1
051|         if ketemu:
052|             if self.tabel.currentRow() != brs:
053|                 self.tabel.setCurrentCell(brs, self.kolomCari)
054|             return brs
055|     return -1
056|
057| def lanjut(self):
058|     if self.teks.text().isEmpty(): return
059|     if self.cari(0, self.tabel.currentRow() ) < 0:
060|         self.cari(0)
061|
062|
063| """ TableFile
```

```
064| """
065| class TableFile:
066|     def __init__(self, filename, pemisah=";",
067|                 uppercase=1, sort=1):
068|         self.filename = filename
069|         self.pemisah = pemisah
070|         self.updateLines()
071|         self.updateStructure()
072|         self.updateRecords(uppercase, sort)
073|
074|     def updateLines(self):
075|         f = open(self.filename,"r")
076|         self.lines = f.readlines()
077|         f.close()
078|
079|     def updateStructure(self):
080|         # Baris pertama judul kolom
081|         labels = self.lines[0]
082|         self.labels = splitfields(strip(labels),
083|                                   self.pemisah)
084|         # Baris berikutnya data
085|         self.datalines = self.lines[1:]
086|
087|     def updateRecords(self, uppercase=0, sort=0):
088|         if sort: self.datalines.sort()
089|         self.records = []
090|         for line in self.datalines:
091|             fields = splitfields(line, self.pemisah)
092|             rec = []
```

```
093|         for field in fields:
094|             n = strip(field)
095|             if uppercase: n = upper(n)
096|             try:
097|                 n = float(n)
098|                 if n == int(n): n = int(n)
099|             except ValueError:
100|                 pass
101|             rec.append(n)
102|         self.records.append( rec )
103|
104|
105| if __name__ == "__main__":
106|     import sys
107|     filename = sys.argv[1]
108|     t = TableFile( filename )
109|
110|     app = QApplication([])
111|     fm = FormCari(None, t.records, t.labels)
112|     fm.tabel.setColumnWidth(0,200)
113|     fm.showMaximized()
114|     fm.exec_loop()
115|     if fm.result() == QDialog.Accepted:
116|         row = fm.tabel.currentRow()
117|         col = fm.tabel.currentColumn()
118|         print fm.tabel.data[row][col]
```

Misalkan sumber data disimpan dalam file `barang.csv` maka perintah untuk menjalankan program ini adalah:

```
$ python cari.py barang.csv
```

Terlepas dari sumber datanya, `FormCari` sebenarnya membebaskan bentuk penyimpanan data. Yang penting data tersebut dikonversi ke bentuk list dua dimensi (struktur tabel).

Untuk mencari kata digunakanlah fungsi `find()` pada `QString`. `find()` dapat menerima masukan berupa *regular expression*¹⁰ (`QRegExp`). Pada program di atas digunakan untuk mencari awalan kata tertentu. Perhatikan karakter pangkat (^) di awal string.

`setColumnWidth()` digunakan untuk menentukan lebar kolom tertentu pada `QTable`. Masukan pertamanya adalah nomor index kolom, dan yang kedua merupakan lebar kolom dalam *pixel*.

Form ini mengoptimalkan penggunaan event. Perhatikan event `keyPressEvent()` yang didalamnya terdapat baris yang memanggil event serupa milik `tabel` (`ValueGrid`). Ini artinya meski kursor berada pada `teks` (`LineCase`) namun dapat menerapkan tombol-tombol fungsi yang dimiliki `tabel`, misalnya navigasi perpindahan current record. Praktis lebih menghemat waktu karena tidak harus menekan tombol `Tab` untuk memindahkan fokus ke `tabel`.

¹⁰Regular expression: ekspresi untuk pencocokan pola.

Bab 22

Kasir II

Kasir II merupakan aplikasi kasir yang memiliki fitur lebih ketimbang pendahulunya, Kasir I¹. Fitur yang dimaksud adalah adanya daftar harga barang. Berikut ini fitur dari Kasir II:

1. Form-nya merupakan turunan dari `FormCari` pada `cari.py`.²
2. Mudah perawatan dimana daftar barang disimpan dalam file teks biasa sebagaimana yang digunakan `cari.py`.
3. Metode pencarian dan tombol navigasi masih sama dengan leluhurnya.
4. Bila teks yang dimasukkan berupa angka maka tidak dilakukan pencarian.

¹Lihat halaman 119.

²Lihat halaman 168.

5. Tombol Enter atau Return digunakan untuk pembayaran. Nilainya diambil dari teks pada poin 4.
6. Terdapat label yang menampilkan total transaksi.
7. Sifat pencatatan transaksi masih mirip Kasir I yaitu langsung ke printer atau bisa juga ke sebuah file.

Adapun bentukstruknya hampir mirip dengan yang dihasilkan Kasir I, tentunya dengan informasi yang lebih lengkap:

```

RAMBUTAN BINJAI      50,25 150.750
ANGGUR MERAH       5,65 113.000
TOTAL                263.750
BAYAR                300.000
KEMBALI              36.250
30-01-03 00:42

```

Jadi secara umum Kasir II masih menyimpan sifat kesederhanaan Kasir I yaitu kemudahan instalasi dan perawatan.

```

kasir2.py
-----
01| from qt import *
02| from cari import FormCari
03| from fungsi import ribu
04| from string import rjust, ljust
05|
06| class FormKasir(FormCari):
07|     def __init__(self, parent, t, output="/dev/lp0"):

```

```
08|     FormCari.__init__(self, parent, t.records,
09|         t.labels, 0)
10|     self.setCaption("Kasir II")
11|     self.lcd = QLCDNumber(self)
12|     self.lcd.setSegmentStyle( QLCDNumber.Flat )
13|     self.lcd.setMinimumHeight(150)
14|     self.lcd.setNumDigits(10)
15|     self.layout().insertWidget(0, self.lcd)
16|     self.tabel.setColumnWidth(0,300)
17|     self.total = 0
18|     self.file = open(output,"w")
19|
20|     def keyPressEvent(self, e):
21|         if e.key() in [Qt.Key_Return, Qt.Key_Enter]:
22|             if not self.bayar(): self.beli()
23|             else: FormCari.keyPressEvent(self, e)
24|
25|     def escape(self):
26|         self.teks.clear()
27|
28|     def teksBerubah(self, w):
29|         s = "%s" % w.text()
30|         try:
31|             n = int(s)
32|         except ValueError:
33|             FormCari.teksBerubah(self, w)
34|
35|     def beli(self):
36|         row = self.tabel.currentRow()
```

```
37|     nama = self.tabel.data[row][0]
38|     harga = self.tabel.data[row][1]
39|     jml, ok = QInputDialog.getDouble(nama,
40|         "Sebanyak", 1, 0, 99999, 2)
41|     if ok:
42|         subtotal = int(jml * harga)
43|         self.total = self.total + subtotal
44|         self.lcd.display( ribu(self.total) )
45|         self.cetakBarang(nama, jml, subtotal)
46|         self.teks.selectAll()
47|
48|     def bayar(self):
49|         s = "%s" % self.teks.text()
50|         try: bayar = int(s)
51|         except ValueError: return
52|         kembali = bayar - self.total
53|         self.lcd.display( ribu(kembali) )
54|         self.cetakAkhir("TOTAL", self.total)
55|         self.cetakAkhir("BAYAR", bayar)
56|         self.cetakAkhir("KEMBALI", kembali)
57|         w = QDateTime.currentDateTime().toString(
58|             "dd-MM-yy hh:mm")
59|         s = "\n" * 5
60|         self.cetak( "%s%s" % (w,s) )
61|         self.teks.clear()
62|         self.total = 0
63|         return 1
64|
65|     def cetak(self, s):
```



```
66|         self.file.write(chr(15)+s)
67|         self.file.flush()
68|
69|     def cetakBarang(self, nama, jml, subtotal):
70|         _nama = ljust(nama[:20], 20)
71|         _jml = rjust(ribu(jml), 7)
72|         _subtotal = rjust(ribu(subtotal),8)
73|         s = "%s%s%s\n" % (_nama, _jml, _subtotal)
74|         self.cetak(s)
75|
76|     def cetakAkhir(self, nama, total):
77|         _nama = ljust(nama,10)
78|         _total = rjust(ribu(total),25)
79|         s = "%s%s\n" % (_nama, _total)
80|         self.cetak(s)
81|
82|
83| if __name__ == "__main__":
84|     import sys
85|     from cari import TableFile
86|
87|     filename = sys.argv[1]
88|     t = TableFile( filename )
89|     if sys.argv[2:]: output = sys.argv[2]
90|     else:             output = "/dev/lp0"
91|     app = QApplication([])
92|     fm = FormKasir(None, t, output)
93|     fm.showMaximized()
94|     fm.exec_loop()
```

Cara menggunakannya mirip dengan `cari.py`:

```
$ python kasir2.py barang.csv
```

dimana `barang.csv` adalah sumber data harga barang. Jangan lupa, pastikan status printer dalam keadaan siap (*ready*).

Apabila Anda ingin menyimpan transaksi ke sebuah file maka tambahkan nama filenya:

```
$ python kasir2.py barang.csv /tmp/kasir.txt
```

Perlu diketahui pula bahwa file tersebut akan dikosongkan terlebih dahulu manakala program dijalankan kembali. Jadi penyimpanannya tidak permanen. Anda juga bisa menggunakan `/dev/null` bila transaksi tidak ingin disimpan:

```
$ python kasir2.py barang.csv /dev/null
```

Bab 23

Database

Database merupakan penyimpan data yang terpisah dari program. File `barang.csv` sebelumnya juga bisa dikatakan sebagai database, atau lebih tepatnya *database file*. Sedangkan pada bab ini database yang dimaksud adalah menggunakan *database server*, yaitu suatu program yang dirancang khusus untuk menyimpan dan mengolah data.

Untuk menggunakan produk database tertentu dalam program, dibutuhkan suatu *driver* sebagai penghubungnya. Qt memiliki beberapa driver untuk PostgreSQL, MySQL, Oracle, Sybase, MS-SQL, ODBC, dsb.¹ Pada Qt kumpulan class yang berkaitan dengan database ini berada dalam modul `qtsql`.

Adapun tahapan yang biasa terjadi dalam pemrograman ap-

¹Beberapa diantaranya bersifat komersil.

likasi database adalah melalui tahapan berikut:

1. Login ke database server untuk mendapatkan apa yang disebut sebagai *connection ID*
2. Query² dengan menggunakan *connection ID* tersebut
3. Menampilkan hasil query (kalau ada)

Proses login setidaknya membutuhkan informasi nama database, username, password, dan hostname (bisa juga nomor IP³). Pada Qt informasi lain yang dibutuhkan adalah nama driver yang akan digunakan. Driver ini merupakan penghubung aplikasi dengan database bersangkutan. Qt memiliki beberapa driver untuk login ke beberapa produk database.

Sebelum Anda mencoba contoh program berikutnya, pastikan database PostgreSQL atau MySQL terpasang dengan baik.

23.1 Membuat Database

Sebelum membuat tabel, tentu kita perlu membuat databasenya terlebih dahulu, dimana pembahasan menyangkut dua produk database terkemuka yang ada di Linux, yaitu PostgreSQL dan MySQL. User yang digunakan untuk login adalah *superuser* dari masing-masing produk, dimana PostgreSQL dengan user

²Perintah yang dikirimkan ke database server.

³IP: Internet Protocol

postgres-nya, dan MySQL dengan **root**-nya.⁴ Sedangkan nama database yang akan dibuat adalah **latihan**.

23.1.1 PostgreSQL

Pertama kali sistem PostgreSQL terpasang, user yang sudah dibuat adalah postgres dengan database template1. Berbekal keduanya, kita akan membuat database yang baru dimana Anda bisa sebagai user Linux apa saja untuk login ke PostgreSQL.

Anda tidak perlu membuat program khusus untuk hal tersebut. PostgreSQL memiliki `psql` yang dapat digunakan untuk query.

```
$ psql -U postgres template1
Welcome to psql, the PostgreSQL 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

template1=# CREATE DATABASE latihan;
CREATE DATABASE
template1=# \q
$
```

⁴Linux, PostgreSQL, dan MySQL memiliki manajemen user yang terpisah.

Kemudian login ke database latihan:

```
$ psql -U postgres latihan
Welcome to psql, the PostgreSQL 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

latihan=#
```

Sampai di sini Anda siap untuk membuat tabel.

23.1.2 MySQL

MySQL juga memiliki program serupa, yaitu `mysql`. Langkahnya pun hampir sama:⁵

```
$ mysql -u root
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 1 to server version: 3.23.49

Type 'help;' or '\h' for help. Type '\c' to clear the buffer.

mysql> CREATE DATABASE latihan;
Query OK, 1 row affected (0.14 sec)
```

⁵Perhatikan, option `-u` menggunakan huruf kecil.

Gambar 23.1: Form Login

```
mysql> \q
Bye
$
```

Kemudian lanjutkan dengan login ke database latihan:

```
$ mysql -u root latihan
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 2 to server version: 3.23.49

Type 'help;' or '\h' for help. Type '\c' to clear the buffer.

mysql>
```

Kini siap untuk membuat tabel.

23.2 Form Login

Program database biasanya digunakan oleh lebih dari satu orang pengguna (*multiuser*). Oleh karena itu kita akan membuat sebuah form untuk proses login dimana pemakai hanya ditanyai username dan password saja. Sedangkan informasi lainnya ter-tanam dalam program.

`login.py` berisi Form Login yang dapat dipanggil dari program utama untuk proses login. Fitur yang ada dalam form ini adalah:

1. Pengguna cukup mengisikan username dan password, karena data mengenai driver dan hostname sudah didefinisikan (hardcode).
2. Untuk login cukup klik tombol OK atau tekan Enter.

Meski ada pertanyaan password, Anda bisa mengosongkannya apabila memang instalasi server database masih default.

```
login.py
-----
01| from qt import *
02| from qtsql import QSqlDatabase
03|
04| """ Driver
05|     QPSQL7 : PostgreSQL
06|     QMYSQL3 : MySQL
07|     QODBC3  : ODBC
08|     QOCI8   : Oracle
09|     QTDS7   : Sybase / MS SQL Server
10| """
11|
12| DB_DRIVER = "QPSQL7"
13| DB_USER   = "postgres"
14|
15| #DB_DRIVER = "QMYSQL3"
```



```
16| #DB_USER    = "root"
17|
18| #DB_DRIVER  = "QOCI8"
19| #DB_USER    = "scott"
20|
21| DB_NAME     = "latihan"
22| DB_HOST     = "localhost"
23|
24| class FormLogin(QDialog):
25|     def __init__(self, parent):
26|         QDialog.__init__(self, parent)
27|         self.resize(150,100)
28|         self.setCaption("Login")
29|         layout = QVBoxLayout(self)
30|         layout.setAutoAdd(1)
31|         layout.setMargin(5)
32|
33|         wadahInput = QWidget(self)
34|         layout = QGridLayout(wadahInput, 2,2)
35|         layout.setAutoAdd(1)
36|         layout.setSpacing(5)
37|         labelUsername = QLabel ( wadahInput )
38|         self.username = QLineEdit( wadahInput )
39|         labelPassword = QLabel ( wadahInput )
40|         self.password = QLineEdit( wadahInput )
41|         labelUsername.setText( "Username" )
42|         labelPassword.setText( "Password" )
43|         self.username.setText( DB_USER )
44|         self.password.setEchoMode( QLineEdit.Password )
```



```
74|         self.reject()
75|     else:
76|         self.username.clear()
77|         self.password.clear()
78|         self.username.setFocus()
79|
80|     def batal(self):
81|         self.reject()
82|
83|
84| if __name__ == "__main__":
85|     app = QApplication([])
86|     if FormLogin(None).db.isOpen():
87|         print "Login berhasil"
88|     else:
89|         print "Login dibatalkan"
```

Bagi pengguna MySQL hapuslah remark (#) pada baris berikut:

```
#DB_DRIVER = "QMYSQL3"
#DB_USER   = "root"
```

Form ini dapat digunakan sebagai otorisasi penggunaan program dimana aplikasi tidak perlu dilanjutkan manakala proses login gagal. Perhatikan baris yang berisi pesan keberhasilan proses login. Pada blok itulah Anda dapat meletakkan pemanggilan form utama.

`QSqlDatabase` di atas adalah class untuk login ke server database. Berikut ini penjelasan fungsinya:

`addDatabase()` mengembalikan `QSqlDatabase` juga. Fungsi ini membutuhkan nama driver yang akan digunakan untuk menghubungi server database.

`setHostName()` menentukan nama atau nomor IP komputer dimana server database berada.

`setDatabaseName()` menentukan nama database yang digunakan.

`setUsername()` menentukan user yang digunakan untuk login.

`setPassword()` menentukan password.

`open()` login ke server database.

`isOpen()` mengembalikan logika true bila login berhasil.

`lastError()` mengembalikan nilai bertipe `QSqlError` yang memuat berbagai informasi mengenai kesalahan login atau query. Fungsi `databaseText()`-nya berisi pesan kesalahan.

23.3 Membuat Tabel

Database latihan yang sudah dibuat akan kita isi dengan tabel. Perintah `CREATE TABLE` digunakan untuk membuat struktur serta “aturan main” suatu tabel.

Melanjutkan program buku alamat yang lalu dimana nama dan alamat disimpan dalam sebuah file, kini akan disimpan dalam tabel relasi.

```
latihan=# CREATE TABLE relasi(
latihan(# nama VARCHAR(30) NOT NULL PRIMARY KEY,
latihan(# alamat TEXT);
latihan=#
```

`nama` dan `alamat` disebut sebagai field, sedangkan `VARCHAR` dan `TEXT` adalah tipe datanya.

`VARCHAR(30)` adalah tipe data string dengan jumlah karakter maksimum 30 banyaknya. `NOT NULL` menunjukkan suatu field tidak boleh hampa.⁶ `PRIMARY KEY` menunjukkan field tersebut merupakan identitas record. Dengan kata lain tidak boleh ada nama yang sama dalam tabel relasi.

`TEXT` juga string namun dengan jumlah karakter yang tidak dibatasi. Ketiadaan `NOT NULL` menunjukkan alamat boleh diisi boleh tidak.

23.4 Query

Berbicara mengenai database - terutama yang berbasis SQL⁷ - tidak terlepas dari `SELECT`, `INSERT`, `UPDATE`, dan `DELETE`.⁸ Keempatnya merupakan perintah SQL standar yang berlaku di berbagai produk database berbasis SQL, berfungsi untuk menampilkan, memasukkan data, serta mengubah maupun menghapusnya.

⁶Kehampaan dalam field adalah `NULL`. Perlu diketahui juga bahwa string hampa ("") yang selama ini kita kenal sebagai kehampaan bagi string bukan kehampaan bagi field.

⁷Structured Query Language

⁸Disebut juga sebagai perintah query

Kini akan kita buat form yang memiliki keempat fungsi tersebut sesuai dengan struktur tabel di atas.. Fitur lainnya adalah:

1. Nama ditampilkan dalam sebuah combobox yang dapat di-edit. Sedangkan alamat tetap ditampilkan dalam textedit.
2. Alamat yang tampil akan disesuaikan dengan nama yang tampak pada combobox.
3. Ada dua tombol untuk menyimpan dan menghapus.

Tombol simpan dihadapkan pada dua kemungkinan dalam menyimpan data:

1. Apabila data baru maka ia menggunakan INSERT.
2. Apabila data merupakan hasil perubahan maka ia menggunakan UPDATE.

Agar bisa membedakannya, di dalam daftar combobox ditambahkan satu data buatan (*virtual*) yang berisi string “<baru>” dan diletakkan paling atas pada daftar. Sehingga cukup klik pada string tersebut untuk menambah data.

```
relasi3.py
-----
001| from qt import *
002| from qtsql import QSqlQuery
003| from string import replace
004| from login import FormLogin
```

```
005|
006| def strSql(s):
007|     return replace("%s" % s, "'", "")
008|
009| def salahQuery( parent, query ):
010|     s = "%s\n%s" % ( query.lastError().databaseText(),
011|         query.lastQuery() )
012|     QMessageBox.warning(parent, "Perhatian", s )
013|
014|
015| class FormRelasi(QWidget):
016|     def __init__(self):
017|         QWidget.__init__(self)
018|         self.setCaption("Relasi")
019|         layout = QGridLayout( self, 3,2)
020|         layout.setAutoAdd(1)
021|         layout.setSpacing(5)
022|         layout.setMargin(5)
023|
024|         labelNama      = QLabel      (self)
025|         self.nama      = QComboBox  (self)
026|         labelAlamat    = QLabel      (self)
027|         self.alamat    = QTextEdit  (self)
028|         tombolSimpan   = QPushButton(self)
029|         tombolHapus    = QPushButton(self)
030|         labelNama.setText ("Nama")
031|         labelAlamat.setText("Alamat")
032|         self.nama.setEditable(1)
033|         self.nama.insertItem("<baru>")
```

```
034|         self.nama.clearEdit()
035|         tombolSimpan.setText("&Simpan")
036|         tombolHapus.setText("&Hapus")
037|
038|         self.query = QSqlQuery( "SELECT nama FROM relasi"
039|         while self.query.next():
040|             self.nama.insertItem(
041|                 self.query.value(0).toString() )
042|
043|         self.nama.setFocus()
044|         self.show()
045|         self.connect( self.nama,
046|             SIGNAL("activated(int)"), self.pilihNama )
047|         self.connect( tombolSimpan,
048|             SIGNAL("clicked()"), self.simpan )
049|         self.connect( tombolHapus,
050|             SIGNAL("clicked()"), self.hapus )
051|
052|     def pilihNama(self, index):
053|         if index == 0:
054|             self.nama.clearEdit()
055|             self.alamat.clear()
056|         else:
057|             nama = strSql( self.nama.text(index) )
058|             sql = "SELECT alamat FROM relasi WHERE \
059|                 nama='%s'" % nama
060|             self.query.executeQuery( sql )
061|             self.query.next()
062|             self.alamat.setText(
```



```
063|         self.query.value(0).toString() )
064|
065| def simpan(self):
066|     index      = self.nama.currentItem()
067|     insert     = index == 0
068|     nama       = strSql( self.nama.currentText() )
069|     alamat     = strSql( self.alamat.text() )
070|     if insert:
071|         sql = "INSERT INTO relasi ( nama, alamat ) " + \
072|             "VALUES( '%s', '%s' )" % ( nama, alamat )
073|     else:
074|         _nama = strSql( self.nama.text(index) )
075|         sql = "UPDATE relasi SET " + \
076|             "nama='%s', alamat='%s' WHERE nama='%s'" % (
077|             nama, alamat, _nama )
078|
079|     if self.query.execQuery( sql ):
080|         if insert:
081|             index = index + 1
082|             self.nama.insertItem( self.nama.currentText(),
083|                 index )
084|             self.nama.setCurrentItem( index )
085|         else:
086|             self.nama.changeItem( self.nama.currentText(),
087|                 index )
088|     else:
089|         salahQuery(self, self.query)
090|
091| def hapus(self):
```

```

092|         if self.nama.currentItem() <= 0: return
093|         index = self.nama.currentItem()
094|         nama = strSql( self.nama.text(index) )
095|         sql = "DELETE FROM relasi WHERE nama='%s'" % nama
096|         if self.query.execQuery( sql ):
097|             self.nama.removeItem( index )
098|             if index + 1 > self.nama.count():
099|                 index = index - 1
100|                 self.nama.setCurrentItem( index )
101|                 self.pilihNama( index )
102|         else:
103|             salahQuery(self, self.query)
104|
105| app = QApplication([])
106| if FormLogin(None).db.isOpen():
107|     fm = FormRelasi()
108|     app.setMainWidget(fm)
109|     app.exec_loop()

```

QSqlQuery merupakan objek yang digunakan untuk perintah query. Pada saat penciptaannya (`__init__()`), class ini dapat langsung menerima perintah tersebut atau tidak, dimana baris

```
self.query = QSqlQuery( "SELECT nama FROM relasi" )
```

dapat ditulis

```
self.query = QSqlQuery()
self.query.execQuery( "SELECT nama FROM relasi" )
```

`execQuery()` digunakan untuk menjalankan perintah query. Bila query berhasil ia mengembalikan logika true. Pada contoh di atas bila query gagal akan ditampilkan pesan kesalahannya melalui fungsi `salahQuery()`.

`WHERE` pada query melibatkan field nama dikarenakan field ini merupakan *primary key* tabel `relasi`. Lihat pembahasan sebelumnya mengenai primary key.

Untuk mendapatkan *record* dari hasil query digunakanlah fungsi `next()`. Lebih lanjut mengenai fungsi ini lihat penjelasan mengenai *current record* di bawah. Sedangkan untuk mendapatkan *nilai dari setiap field pada record* tersebut digunakanlah fungsi `value()`. Fungsi ini membutuhkan masukan berupa nomor index field dimana 0 berarti field pertama. `value()` mengembalikan nilai `QVariant` berisi nilai fieldnya. Lebih lanjut mengenai `QVariant` lihat halaman 190.

Fungsi `strSql()` dibuat untuk mengantisipasi adanya karakter kutip tunggal (') pada nilai field. Menggantinya dengan kutip tunggal dua kali berarti memberitahu server database bahwa itu kutip tunggal.

`QComboBox` memiliki fungsi ganda, selain dipakai untuk menampilkan daftar nama juga untuk menambahkan data baru dan juga mengubah data yang lama. Berikut ini daftar fungsi yang belum dibahas sebelumnya:

`insertItem()` menambah data. Masukan pertamanya berupa string yang akan ditambahkan dalam daftar. Sedangkan masukan kedua (*optional*⁹) untuk menentukan

⁹Optional: boleh disertakan boleh tidak.

posisi string tersebut disisipkan. Bila tidak disebutkan maka string akan diletakkan di paling bawah.

`clearEdit()` menghapus isi editor.

`setCurrentItem()` menentukan nomor index yang aktif (terpilih). Fungsi ini mempengaruhi `currentItem()`.

`removeItem()` menghapus daftar pada nomor index tertentu.

23.4.1 Current Record

`QSqlQuery` bekerja dengan sistem current record, artinya hanya ada satu record yang aktif dalam satu saat. Sehingga untuk mendapatkan nilai field di record tertentu, Anda perlu memindahkan penunjuk record. `QSqlQuery` memiliki beberapa fungsi navigasi untuk memindahkan penunjuk record:

`next()` ke record berikutnya.

`prev()` ke record sebelumnya.

`first()` ke record pertama.

`last()` ke record terakhir.

`seek(n)` record ke *n*.

`at()` Nomor index current record. Record pertama berarti `at() = 0`.

`next()`, `prev()`, `first()`, dan `last()` mengembalikan logika true apabila berhasil ke record yang dimaksud.

Untuk lebih memahami bagaimana fungsi navigasi pada `QSqlQuery` bekerja, isilah tabel relasi dengan beberapa record, lalu cobalah menjalankan contoh berikut:

```
query1.py
-----
01| from qt import *
02| from qtsql import QSqlQuery
03| from login import FormLogin
04|
05| app = QApplication([])
06| if FormLogin(None).db.isOpen():
07|     q = QSqlQuery("SELECT nama, alamat FROM relasi")
08|     while q.next():
09|         print q.value(0).toString(), q.value(1).toString()
10|         print
11|     while q.prev():
12|         print q.value(0).toString(), q.value(1).toString()
```

23.4.2 Variant

Seperti dibahas sebelumnya bahwa `QSqlQuery.value()` mengembalikan `QVariant`. `Variant` atau `QVariant` merupakan tipe data “apa saja”. `QVariant` memiliki banyak fungsi konversi ke berbagai bentuk tipe data seperti:

```
toString()QString
```

```
toDateTime() QDateTime
```

```
toDouble() float
```

```
toInt() integer
```

Untuk mengetahui tipe data suatu variant dapat menggunakan fungsi `QVariant.type()` yang akan mengembalikan konstanta integer berupa nomor tipe yang dimaksud. Beberapa diantaranya adalah:

```
QVariant.String QString
```

```
QVariant.Double float
```

```
QVariant.Int integer
```

```
QVariant.Date QDate
```

```
QVariant.Time QTime
```

```
QVariant.DateTime QDateTime
```

23.5 Cara Lain Menangani Tabel

Dengan `QSqlQuery` manipulasi data tabel sudah dapat dilakukan. Qt memiliki class lainnya sebagai alternatif yang “lebih mudah”, yaitu `QSqlCursor`. Class ini merupakan turunan dari `QSqlQuery` juga dan `QSqlRecord`.¹⁰

¹⁰Sebagaimana C++, Python membolehkan suatu class merupakan turunan beberapa class.

```

cursor1.py
-----
01| from qt import *
02| from qtsql import QSqlCursor
03| from login import FormLogin
04|
05| app = QApplication([])
06| if FormLogin(None).db.isOpen():
07|     c = QSqlCursor( "relasi" )
08|     if c.select():
09|         while c.next():
10|             print c.value("nama").toString(), \
11|                 c.value("alamat").toString()
12|     else:
13|         print c.lastError().databaseText()

```

`select()` digunakan untuk menjalankan query `SELECT`. Ia mengembalikan logika `true` apabila query berhasil.¹¹ Baris

```
c.select()
```

sama artinya dengan query

```
SELECT * FROM relasi;
```

Bila ingin ditambahkan suatu kondisi, bisa langsung disertakan pada `select()`

¹¹Contoh kegagalan query pada contoh di atas adalah ketiadaan hak akses (`GRANT`) bagi user yang sedang login terhadap tabel `relasi`.

```
c.select("nama='Budi'")
```

Nilai masukan bagi `value()` pada `QSqlCursor` selain dapat berupa nomor urut field juga dapat berupa nama fieldnya.

Praktis `QSqlCursor` menawarkan gaya Qt dalam memprogram database dan - dengan alasan tersebut - diharapkan lebih memudahkan programmer karena tidak perlu menuliskan query. Namun dalam beberapa hal dengan `QSqlQuery` proses query jauh lebih cepat. Jadi yang paling penting keduanya akan kita gunakan pada kasus yang sesuai dengan fitur masing-masing.

23.5.1 Browsing

`QDataTable` merupakan `QTable` yang dapat menampilkan isi record `QSqlCursor` secara otomatis. Cobalah program berikut ini:

```
datatable1.py
-----
01| from qt import *
02| from qtsql import QSqlCursor, QDataTable
03| from login import FormLogin
04|
05| class FormRelasi(QWidget):
06|     def __init__(self):
07|         QWidget.__init__(self)
08|         self.setCaption("Tabel relasi")
```



```

09|         layout = QHBoxLayout(self)
10|         layout.setAutoAdd(1)
11|         self.cursor = QSqlCursor( "relasi" )
12|         self.table = QDataTable(self)
13|         self.table.setSqlCursor(self.cursor, 1)
14|         self.table.refresh()
15|         self.show()
16|
17| app = QApplication([])
18| if FormLogin(None).db.isOpen():
19|     fm = FormRelasi()
20|     app.setMainWidget(fm)
21|     app.exec_loop()

```

`QSqlCursor` dipakai oleh `QDataTable` sebagai sumber data melalui fungsi `setSqlCursor()`. Angka 1 (true) pada masukan keduanya (optional) berarti menampilkan kolom sebanyak jumlah field pada tabel bersangkutan. Bila Anda hanya ingin menampilkan field tertentu saja maka jangan sertakan angka 1 tersebut. Sebagai gantinya untuk menampilkan kolom tertentu gunakan `addColumn()`.

```

self.table.setSqlCursor(self.cursor)
self.table.addColumn("nama", "Nama Relasi")

```

`QDataTable` memiliki menu *popup* yang dapat ditampilkan dengan cara klik-kanan¹² pada `QDataTable`. Menu ini berisi Insert, Update, dan Delete yang mewakili perintah SQL untuk memanipulasi record.

¹²Klik-kanan: klik tombol mouse paling kanan.

23.5.2 Bentuk Tampilan

Tampilan nilai field pada `QDataTable` masih perlu diuji dengan beberapa tipe data yang sering digunakan, misalnya bilangan pecahan dan yang berkaitan dengan waktu. Oleh karena itu, buatlah struktur tabel pegawai berikut ini, bisa melalui program `psql` atau `mysql`:

```
CREATE TABLE pegawai(  
    id INTEGER NOT NULL,  
    nama VARCHAR(30) NOT NULL,  
    tgl_lahir DATE NOT NULL,  
    gaji FLOAT NOT NULL,  
    jadual_masuk TIME,  
    waktu_input DATETIME NOT NULL,  
    PRIMARY KEY (id)  
);
```

lalu isi dengan query ini:

```
INSERT INTO pegawai  
    (id,nama,tgl_lahir,gaji,jadual_masuk,waktu_input)  
VALUES  
    (1,'Prakoso','1973-12-20',2500000,NULL,  
    '2003-01-17 12:59:48');
```

```
INSERT INTO pegawai  
    (id,nama,tgl_lahir,gaji,jadual_masuk,waktu_input)  
VALUES  
    (2,'Anom','1974-01-03',1500000,'07:30',
```

Gambar 23.3: QDataTable

```
'2003-01-17 13:09:34');  
  
INSERT INTO pegawai  
  (id,nama,tgl_lahir,gaji,jadual_masuk,waktu_input)  
VALUES  
  (3,'Budi Respati','1973-12-20',1750000,'07:30',  
  '2003-01-17 13:14:56');
```

kemudian ubahlah `datatable1.py` dimana tabel `relasi` diganti dengan `pegawai`. Lalu lihat hasilnya seperti pada gambar 23.3.

Dari hasil tersebut, ada beberapa hal yang perlu "dikoreksi" dari `QDataTable` ini:

1. Field `NULL` tidak perlu ditampilkan apa-apa.
2. Bilangan - baik integer maupun float - ditampilkan rata kanan.
3. Format bilangan pecahan (float) perlu ditampilkan secara utuh, memiliki pemisah ribuan, rata kanan, dan hanya ada dua angka di belakang koma.
4. Format waktu sesuai gaya Indonesia. Tahun juga perlu ditampilkan secara utuh untuk membedakan 19xx dengan 20xx.

Oleh karena itu perlu dibuat class baru turunan `QDataTable` untuk mewujudkan fitur tersebut. Kita namakan saja `CursorTable`.

```
cursorstable.py
-----
01| from qt import *
02| from qtsql import QDataTable
03| import locale
04|
05| locale.setlocale(locale.LC_ALL, "")
06|
07| class CursorTable(QDataTable):
08|     Tanggal = "dd-MM-yyyy"
09|     Jam      = "hh:mm:ss"
10|     Waktu   = "%s %s" % (Tanggal, Jam)
11|
12|     def __init__(self, parent):
13|         QDataTable.__init__(self, parent)
14|
15|     def paintField(self, painter, field, cr, selected):
16|         if field.type() in [QVariant.Int, QVariant.Double]:
17|             align = QPainter.LTR
18|         else:
19|             align = QPainter.RTL
20|         if field.isNull():
21|             teks = ""
22|         elif field.type() == QVariant.Double:
23|             teks = locale.format( "%.2f",
24|                 field.value().toDouble(), 1 )
```

```
25|     elif field.type() == QVariant.Date:
26|         teks = field.value().toDate().toString(
27|             self.Tanggal)
28|     elif field.type() == QVariant.DateTime:
29|         teks = field.value().toDateTime().toString(
30|             self.Waktu)
31|     elif field.type() == QVariant.Time:
32|         teks = field.value().toTime().toString(self.Jam)
33|     else:
34|         teks = field.value().toString()
35|     painter.drawText(2,2, cr.width()-4, cr.height()-4,
36|         align, teks )
37|
38|
39| if __name__ == "__main__":
40|     from login import FormLogin
41|     from qtsql import QSqlCursor
42|
43|     app = QApplication([])
44|     if FormLogin(None).db.isOpen():
45|         cursor = QSqlCursor( "pegawai" )
46|         fm = CursorTable(None)
47|         fm.setSqlCursor( cursor, 1 )
48|         fm.refresh()
49|         fm.show()
50|         app.setMainWidget(fm)
51|         app.exec_loop()
```

Gambar 23.4: CursorTable

Jalankan program di atas, dan contoh tampilannya dapat dilihat pada gambar 23.4.

23.5.3 Pengganti Perintah SQL

`QSqlCursor` dibuat sebagai “jalan lain” untuk melakukan `INSERT`, `UPDATE`, maupun `DELETE`. Contoh berikut ini menampilkan fungsi ketiganya dengan skenario sebagai berikut:

1. Tabel relasi dikosongkan.
2. Record ditambahkan, dan ditampilkan.
3. Record diubah dan ditampilkan kembali hasil perubahannya.

```
cursor2.py
```

```
-----
```

```
01| from qt import *
02| from qtsql import QSqlCursor
03| from login import FormLogin
04|
05| app = QApplication([])
06| if FormLogin(None).db.isOpen():
07|     c = QSqlCursor( "relasi" )
08|     c.select()
```

```
09| while c.next():
10|     c.primeDelete()
11|     c.delRecords()
12|     c.select()
13|
14| b = c.primeInsert()
15| b.setValue("nama", QVariant("Udin") )
16| b.setValue("alamat", QVariant("Jakarta") )
17| c.insert()
18|
19| c.select()
20| while c.next():
21|     print c.value("nama").toString(), \
22|         c.value("alamat").toString()
23|
24| b = c.primeUpdate()
25| b.setValue("alamat", QVariant("Bogor") )
26| c.update()
27|
28| c.select()
29| while c.next():
30|     print c.value("nama").toString(), \
31|         c.value("alamat").toString()
```

QSqlCursor bekerja dengan sistem *buffer*¹³ untuk memanipulasi data. `primeInsert()`, `primeUpdate()`, maupun `primeDelete()` digunakan untuk mempersiapkan buffer ini. Ketiga fungsi ini mengembalikan QSqlRecord. Jalan lain untuk mendapatkan

¹³Buffer: penyimpanan sementara.

buffer ini adalah melalui fungsi `editBuffer()` pada `QSqlCursor`. Jadi baris

```
b = c.primeUpdate()  
b.setValue("alamat", QVariant("Bogor") )  
c.update()
```

bisa ditulis

```
c.primeUpdate()  
c.editBuffer().setValue("alamat", QVariant("Bogor") )  
c.update()
```

Primary Key

Telah dibahas pada contoh sebelumnya bahwa `UPDATE` dan `DELETE` membutuhkan kondisi (`WHERE`) untuk mengubah suatu record. Kondisi yang dimaksud untuk mencari identitas suatu record yang tercermin dalam primary key pada struktur tabel. Tanpa-panya, `UPDATE` dan `DELETE` tidak dapat menentukan record yang dimaksud. Sama halnya dengan perintah `update()` dan `delRecords()` pada `QSqlCursor`. Jika tabelnya tidak memiliki primary key maka kedua perintah tersebut tidak melakukan apa-apa dan mengembalikan `FALSE`.

23.5.4 NULL

`NULL` merupakan kehampaan bagi field dalam SQL. Fungsi `field()` dapat digunakan untuk mendapatkan objek field (`QSqlField`)

pada `QSqlCursor`. Dengan objek field ini kita dapat menghapus nilai field (men-NULL-kan) dengan fungsi `clear()`.

Contoh berikut ini akan menghapus field `alamat` pada nama Udin.

```
cursor3.py
-----
01| from qt import *
02| from qtsql import QSqlCursor
03| from login import FormLogin
04|
05| app = QApplication([])
06| if FormLogin(None).db.isOpen():
07|     c = QSqlCursor( "relasi" )
08|     c.select("nama='Udin'")
09|     if c.next():
10|         b = c.primeUpdate()
11|         b.field("alamat").clear()
12|         c.update()
13|         c.select()
14|         while c.next():
15|             print c.value("nama").toString(), \
16|                   c.value("alamat").toString()
17|     else:
18|         print "Udin belum terdaftar"
```

Objek field ini sebenarnya milik leluhur `QSqlCursor`, yaitu `QSqlRecord`.

23.5.5 Pengisian Data yang Lebih Nyaman

Kebanyakan orang terbiasa dengan aplikasi spreadsheet untuk memasukkan data. Penggunaan menu popup pada `QDataTable` - sedikit banyak - menambah waktu proses input karena melibatkan dua alat yaitu keyboard dan mouse.

Untuk memperbaikinya, kita akan membuat class baru keturunan `Grid` ¹⁴ dengan tambahan fitur sebagai berikut:

1. Sama seperti `QDataTable`, sumberdatanya `QSqlCursor`.
2. Current record memiliki tiga status: Browse (tersimpan), Update (sedang diubah untuk di-UPDATE), dan Insert (sedang diubah untuk di-INSERT).
3. Bentuk tampilan untuk field angka dan waktu disesuaikan dengan ciri Indonesia.
4. Penyesuaian bentuk tampilan, misalkan angka yang tampil 1.234,56 namun pada saat memasukkan data cukup ditulis 1234,56 atau 1234.56 (desimal boleh dengan koma atau titik). Dengan fungsi `angka()` ¹⁵ maka masukan bisa berupa rumus matematika.
5. Waktu bisa ditulis `now` yang berarti saat ini.
6. Masukan tanggal yang tidak lengkap akan dilengkapi dengan waktu saat ini. Misalkan sekarang tanggal 17-2-2003,

¹⁴Lihat halaman 157 mengenai class ini.

¹⁵Fungsi `angka()` berada pada file `fungsi.py`. Lihat halaman 161 tentang fungsi ini.

Gambar 23.5: DBGrid

bila hanya angka 10 yang dimasukkan maka menjadi 10-2-2003. Bila masukannya 31-1 maka menjadi 31-1-2003.

7. Masukan jam juga bisa ditulis “depannya” saja. Misalkan hanya dimasukkan angka 10 maka menjadi 10:00:00.
8. Untuk tipe DATETIME tanggal dan jam dipisahkan spasi, masing-masing dengan aturan penulisan seperti di atas.
9. Bila terjadi kesalahan input, nilai dikembalikan seperti semula.

Class baru ini dinamakan `DBGrid`, merupakan nama objek pada produk Borland Delphi yang berfungsi untuk menampilkan tabel dari database. Konsep `DBGrid` di sini memang berasal dari produk tersebut.

Modul `fungsi.py` yang pernah dibuat sebelumnya akan dilengkapi dengan tiga fungsi baru yaitu: `tanggal()`, `jam()`, dan `waktu()`. Ketiganya menerima masukan string dan mengubahnya menjadi `QDate`, `QTime`, serta `QDateTime`.

```
fungsi.py
-----
001| from qt import *
002| import string
003| import locale
```

```
004|
005| locale.setlocale(locale.LC_ALL, "")
006|
007|
008| """ Mengubah angka menjadi string dengan format ribuan
009| """
010| def ribu(n):
011|     if type(n) == type(0):
012|         return locale.format( "%d", n, 1 )
013|     elif type(n) == type(0.0):
014|         return locale.format( "%.2f", n, 1 )
015|     else:
016|         return ""
017|
018|
019| """ Mengubah string menjadi angka.
020|     Membolehkan pemisah pecahan dengan koma.
021|     Membolehkan rumus matematika.
022| """
023| def angka(s):
024|     a = string.strip("%s" % s)
025|     a = string.replace(a,",",".")
026|     try:
027|         return int(a)
028|     except ValueError:
029|         try:
030|             return float(a)
031|         except ValueError:
032|             if a and a[0] == "=":
```

```
033|         try:
034|             exec("a"+a)
035|             return a
036|         except:
037|             return
038|
039|
040| """ Mengubah string tanggal format dd-MM-yyyy menjadi
041|     QDate. Bila hanya dd maka MM dan yyyy diisi bulan
042|     dan tahun sekarang. Bila hanya dd-MM maka yyyy diisi
043|     tahun ini. Bila 'NOW' berarti tanggal sekarang.
044| """
045| def tanggal(s):
046|     w = string.strip("%s" % s)
047|     if string.upper(w) == "NOW":
048|         return QDate.currentDate()
049|
050|     if string.find(w, "-") > -1: p = "-"
051|     else: p = "/"
052|     t = string.splitfields(w,p)
053|
054|     try: d = int(t[0])
055|     except ValueError: return
056|
057|     # Untuk bulan dan tahun default
058|     now = QDate.currentDate()
059|
060|     if t[1:]:
061|         try: m = int(t[1])
```

```
062|         except ValueError: return
063|         if t[2:]:
064|             try: y = int(t[2])
065|                 except ValueError: return
066|             else:
067|                 y = now.year()
068|         else:
069|             y, m = now.year(), now.month()
070|
071|         tgl = QDate( y, m, d )
072|         if tgl.isValid() and not tgl.isNull(): return tgl
073|
074|
075| """ Mengubah string jam bentuk hh:mm:ss.zzz menjadi
076|     QTime. Bila hanya hh maka lainnya dianggap nol.
077|     Begitu seterusnya.
078| """
079| def jam(s):
080|     w = string.strip("%s" % s)
081|     if string.upper(w) == "NOW":
082|         return QTime.currentTime()
083|     t = string.splitfields(w,":")
084|
085|     try: h = int(t[0])
086|         except ValueError: return
087|
088|     if t[1:]:
089|         try: m = int(t[1])
090|             except ValueError: return
```

```
091|     if t[2:]:
092|         x = t[2]
093|         x = string.replace(x,"",".")
094|         x = string.splitfields(x,".")
095|         try: s = int(x[0])
096|         except ValueError: return
097|         if x[1:]:
098|             y = "0." + x[1]
099|             try: ms = float(y) * 1000
100|             except ValueError: return
101|         else:
102|             ms = 0
103|     else:
104|         s, ms = 0, 0
105|     else:
106|         m, s, ms = 0, 0, 0
107|         j = QTime(h,m,s,ms)
108|         if j.isValid() and not j.isNull(): return j
109|
110|
111| """ Mengubah string waktu bentuk dd-MM-yyyy hh:mm:ss.zzz
112|     menjadi QDateTime
113| """
114| def waktu(s):
115|     w = string.strip("%s" % s)
116|     if string.upper(w) == "NOW":
117|         return QDateTime.currentDateTime()
118|
119|     w = string.splitfields("%s" % w)
```

```
120|   if w: d = tanggal(w[0])
121|   else: return
122|
123|   if w[1:]: t = jam(w[1])
124|   else: t = QTime(0,0,0,0)
125|
126|   return QDateTime( d, t )
```

fungsi.py di atas akan digunakan dalam dbgrid.py berikut ini.

```
dbgrid.py
-----
001| from qt import *
002| from qtable import QTable
003| from qsql import QSqlIndex
004| import string
005| from grid import Grid
006| from fungsi import *
007| from linecase import LineCase
008|
009| class DBGrid(Grid):
010|     Tgl = "dd-MM-yyyy"
011|     Jam = "hh:mm:ss"
012|     Wkt = "%s %s" % (Tgl, Jam)
013|
014|     # status current record
015|     Browse = 0
016|     Insert = 1
```



```
017|     Update = 2
018|
019|     def __init__(self, parent):
020|         Grid.__init__(self, parent)
021|         self.penBox = QPen()
022|         self.penBox.setColor( QColor("lightGray") )
023|         self.penText = QPen()
024|         self.penText.setColor( QColor("black") )
025|         self.brushBrowse = QBrush()
026|         self.brushBrowse.setColor( QColor("yellow") )
027|         self.brushBrowse.setStyle( QBrush.SolidPattern )
028|         self.brushInsert = QBrush()
029|         self.brushInsert.setColor( QColor("cyan") )
030|         self.brushInsert.setStyle( QBrush.SolidPattern )
031|         self.brushUpdate = QBrush()
032|         self.brushUpdate.setColor( QColor("green") )
033|         self.brushUpdate.setStyle( QBrush.SolidPattern )
034|         self._cursor = None
035|         self.status = self.Browse
036|         self._row = -1 # sebelum currentRow()
037|         self._col = -1 # sebelum currentColumn()
038|         self.editRow = -1 # baris yang sedang diedit
039|         self._maxLength = {} # fieldname:length
040|         self._case = {} # fieldname:Normal|Upper|Lower
041|         self.intValidator = QIntValidator(self)
042|         self.doubleValidator = QDoubleValidator(self)
043|         self.afterInsert = None # event setelah tambah()
044|         self.beforePost = None # event sebelum UPDATE/INSERT
045|         self.afterPost = None # event sesudah UPDATE/INSERT
```

```
046|         self.connect(self,
047|             SIGNAL("currentChanged(int,int)"),
048|             self.saatPindah)
049|
050|     def resizeData(self, i):
051|         pass # hemat memory
052|
053|     def beginEdit(self, row, col, replace):
054|         if self.ubah():
055|             return QTable.beginEdit(self, row, col, replace)
056|
057|     def createEditor(self, row, col, initFromCell):
058|         field = self._cursor.editBuffer().field(col)
059|         fieldname = str(field.name())
060|         if self._case.has_key(fieldname):
061|             case = self._case[fieldname]
062|         else: case = LineCase.Normal
063|         e = LineCase(self, case)
064|         if field.type() == QVariant.String:
065|             if self._maxLength.has_key(fieldname):
066|                 e.setMaxLength(self._maxLength[fieldname])
067|         elif field.type() == QVariant.Int:
068|             e.setValidator( self.intValidator )
069|         elif field.type() == QVariant.Double:
070|             e.setValidator( self.doubleValidator )
071|
072|     if initFromCell:
073|         if field.isNull(): s = ""
074|         elif field.type() == QVariant.Date:
```

```
075|         s = field.value().toDate().toString(self.Tgl)
076|     elif field.type() == QVariant.DateTime:
077|         s = field.value().toDateTime().toString(self.Wkt)
078|     elif field.type() == QVariant.Time:
079|         s = field.value().toTime().toString(self.Jam)
080|     elif field.type() == QVariant.Double:
081|         s = str(field.value().toDouble())
082|     elif field.type() == QVariant.Int:
083|         s = str(field.value().toInt())
084|     else:
085|         s = field.value().toString()
086|     e.setText( s )
087|     return e
088|
089| def setCellContentFromEditor(self, row, col):
090|     self.isiDariEditor()
091|
092| def clearCell(self, row, col):
093|     if self._cursor.seek(row):
094|         self.ubah().field(col).clear()
095|         self.updateCell(row, col)
096|
097| def insertRows(self, row, count=1):
098|     if not self.simpan() or self._cursor.size() < 1:
099|         return
100|     Grid.insertRows(self, row)
101|     self.tambah()
102|     r = self.cellGeometry(row-1,0)
103|     r.setRight( self.width() )
```

```
104|         r.setBottom( self.height() )
105|         self.repaintContents(r)
106|
107|     def removeRow(self, row):
108|         if not self.adaPrimaryKey(): return
109|         if self._cursor.seek( row ):
110|             self._cursor.primeDelete()
111|             if self._cursor.delRecords():
112|                 self._refresh()
113|                 self.setStatus( self.Browse )
114|                 Grid.removeRow( self, row )
115|                 r = self.cellGeometry( self.currentRow(), 0 )
116|                 r.setRight( self.width() )
117|                 self.repaintContents(r)
118|             else: self.salah()
119|         else: Grid.removeRow( self, row )
120|
121|     def keyPressEvent(self, e):
122|         if e.key() == Qt.Key_Up: self.naik()
123|         elif e.key() == Qt.Key_Escape: self.batal()
124|         elif e.key() == Qt.Key_F4: self.simpan()
125|         elif e.key() == Qt.Key_F5: self.perbaharui()
126|         else: Grid.keyPressEvent(self, e)
127|
128|     def paintCell(self, painter, row, col, cr, selected)
129|         if not self._cursor: return
130|         if row == self.editRow:
131|             field = self._cursor.editBuffer().field(col)
132|             elif self.status == self.Insert:
```

```
133|         if row > self.editRow: r = row - 1
134|         else:                   r = row
135|         if self._cursor.seek(r):
136|             field = self._cursor.field(col)
137|         else:
138|             field = None
139|     elif self._cursor.seek(row):
140|         field = self._cursor.field(col)
141|     else: field = None
142|
143|     s = self.strField( field )
144|
145|     if field and field.type() in [QVariant.Int,
146|         QVariant.Double]:
147|         align = QPainter.LTR
148|     else: align = QPainter.RTL
149|
150|     if row == self.editRow:
151|         if self.status == self.Insert:
152|             painter.fillRect( 0,0, cr.width(), cr.height(),
153|                 self.brushInsert )
154|         else:
155|             painter.fillRect( 0,0, cr.width(), cr.height(),
156|                 self.brushUpdate )
157|     elif row == self.currentRow() and \
158|         self._cursor.size() > 0:
159|         painter.fillRect( 0,0, cr.width(), cr.height(),
160|             self.brushBrowse )
161|     else:
```

```
162|         painter.eraseRect( 0,0, cr.width(), cr.height()
163|
164|     painter.setPen( self.penBox )
165|     painter.drawLine( 0, cr.height()-1, cr.width(),
166|         cr.height()-1 )
167|     painter.drawLine( cr.width()-1, cr.height()-1,
168|         cr.width()-1, 0 )
169|     painter.setPen( self.penText )
170|     painter.drawText(2,2, cr.width()-4, cr.height()-4,
171|         align, s)
172|
173| def strField(self, field):
174|     if not field or field.isNull(): s = ""
175|     elif field.type() == QVariant.Date:
176|         s = field.value().toDate().toString(self.Tgl)
177|     elif field.type() == QVariant.DateTime:
178|         s = field.value().toDateTime().toString(self.Wkt)
179|     elif field.type() == QVariant.Time:
180|         s = field.value().toTime().toString(self.Jam)
181|     elif field.type() == QVariant.Double:
182|         s = str(ribu( field.value().toDouble()))
183|     elif field.type() == QVariant.Int:
184|         s = str(ribu( field.value().toInt()))
185|     else: s = str(field.value().toString())
186|     return s
187|
188| def setSqlCursor(self, cursor):
189|     if cursor:
190|         if cursor.sort().isEmpty() and \
```

```
191|         not cursor.primaryIndex().isEmpty():
192|             index = QSqlIndex()
193|             for i in range( cursor.primaryIndex().count() ):
194|                 index.append( cursor.field(i) )
195|             cursor.setSort( index )
196|         cursor.select()
197|         self.setNumCols( cursor.count() )
198|         self.setNumRows( cursor.size() )
199|         self._row = self.currentRow()
200|         self._col = self.currentColumn()
201|         for col in range( cursor.count() ):
202|             self.horizontalHeader().setLabel(col,
203|                 cursor.field(col).name() )
204|         else:
205|             self.setNumCols(0)
206|             self.setNumRows(0)
207|         self._cursor = cursor
208|
209|     def adaPrimaryKey(self):
210|         if self._cursor.primaryIndex().isEmpty():
211|             self.pesan("Tabel tanpa Primary Key tidak dapat \
212| diubah atau dihapus.")
213|         else: return 1
214|
215|     def setStatus(self, status):
216|         if self.status == status: return
217|         self.status = status
218|         if status == self.Browse: self.editRow = -1
219|         else: self.editRow = self.currentRow()
```

```
220|         r = self.cellGeometry(self.currentRow(),0)
221|         r.setRight( self.width() )
222|         self.repaintContents(r)
223|
224|     def tambah(self):
225|         b = self._cursor.primeInsert()
226|         for col in range(b.count()): b.field(col).clear()
227|         self.setStatus( self.Insert )
228|         if self.afterInsert: self.afterInsert(self._cursor)
229|         return b
230|
231|     def ubah(self):
232|         if self.status != self.Browse:
233|             return self._cursor.editBuffer()
234|         if self._cursor.size() == 0: return self.tambah()
235|         if not self.adaPrimaryKey(): return
236|         if self._cursor.seek( self.currentRow() ):
237|             b = self._cursor.primeUpdate()
238|             self.setStatus( self.Update )
239|             return b
240|
241|     def simpan(self):
242|         if self.status == self.Browse: return 1
243|         if self.editorAktif():
244|             self.isiDariEditor()
245|             self.clearCellWidget( self.currentRow(),
246|                 self.currentColumn() )
247|         if self.beforePost: self.beforePost(self._cursor)
248|         if self.status == self.Insert:
```



```
278| def batal(self):
279|     if self.status != self.Browse:
280|         status = self.status
281|         self.setStatus( self.Browse )
282|         if status == self.Insert:
283|             Grid.removeRow( self, self.currentRow() )
284|             r = self.cellGeometry(self.currentRow(),0)
285|             r.setRight( self.width() )
286|             r.setBottom( self.height() )
287|             self.repaintContents(r)
288|
289| def salah(self):
290|     pesan = "%s\nTekan Escape untuk membatalkan." % \
291|         self._cursor.lastError().databaseText()
292|     self._cursor.select()
293|     if self._row != self.currentRow():
294|         r = self.cellGeometry( self.currentRow(), 0 )
295|         r.setRight( self.width() )
296|         self.setCurrentCell( self._row, self._col )
297|         self.repaintContents(r)
298|     self.pesan( pesan )
299|
300| def _refresh(self):
301|     self._cursor.select()
302|     self.repaintContents()
303|
304| # Berguna pada cursor berkondisi pada select()-nya
305| def perbaharui(self):
306|     if self.cellWidget( self.currentRow(),
```

```
307|         self.currentColumn() ): return
308|     if self.simpan():
309|         self._cursor.select()
310|         self.setNumRows( self._cursor.size() )
311|
312|     def pesan(self, s):
313|         QMessageBox.warning(self, "Perhatian", s)
314|
315|     # Tentukan jumlah karakter maksimum untuk field string
316|     def setMaxLength(self, fieldname, length):
317|         self._maxLength.update( {fieldname:length} )
318|
319|     # Tentukan uppercase atau tidak untuk field string
320|     def setCase(self, fieldname, case):
321|         self._case.update( {fieldname:case} )
322|
323|     def editorAktif(self):
324|         return self.cellWidget( self.currentRow(),
325|             self.currentColumn() )
326|
327|     def isiDariEditor(self):
328|         s = str(self.editorAktif().text())
329|         field = self._cursor.editBuffer().field(
330|             self.currentColumn() )
331|         if field.type()==QVariant.Date: s = tanggal(s)
332|         elif field.type()==QVariant.Time: s = jam(s)
333|         elif field.type()==QVariant.DateTime: s=waktu(s)
334|         elif field.type() in [QVariant.Int,
335|             QVariant.Double]:
```

```
336|         s = angka(s)
337|         if s: s = str(s)
338|         if s: field.setValue( QVariant(s) )
339|         elif s == "": field.clear()
340|
341| if __name__ == "__main__":
342|     from login import FormLogin
343|     from qtsql import QSqlCursor
344|     import sys
345|
346|     tablename = sys.argv[1]
347|     app = QApplication([])
348|     if FormLogin(None).db.isOpen():
349|         cursor = QSqlCursor( tablename )
350|         cursor.select()
351|         fm = DBGrid(None)
352|         fm.setSqlCursor( cursor )
353|         fm.show()
354|         app.setMainWidget(fm)
355|         app.exec_loop()
```

Untuk mencobanya jalankan:

```
$ python dbgrid.py pegawai
```

Fungsi `beginEdit()` merupakan event yang dipanggil menjelang edit-mode. Ini saat yang tepat untuk membolehkan atau tidak suatu cell di-edit. Bila diizinkan fungsi ini harus mengembalikan widget sebagai editor untuk memasukkan data yang diperoleh lewat `createEditor()`.

Untuk menyimpan data yang telah dimasukkan tekan F4, atau langsung tekan panah atas / bawah. Secara umum dikatakan bahwa setiap pergantian baris membuat record tersimpan. Bisa juga menekan F5 yang sekaligus merupakan tombol *refresh*, yaitu *query ulang* terhadap tabel barang.

23.5.6 Urutkan - Sort

Sebelumnya kita sudah menyimpan daftar barang dalam sebuah file. Sekarang daftar barang tersebut akan disimpan dalam sebuah tabel barang dengan tambahan field *id* sebagai kode barang.

```
CREATE TABLE barang(  
  id INTEGER NOT NULL PRIMARY KEY,  
  nama VARCHAR(20) NOT NULL UNIQUE,  
  harga FLOAT NOT NULL);
```

UNIQUE pada field *nama* berarti field tersebut juga merupakan *identitas record* selain field pada **PRIMARY KEY** (field *id*). Makna lainnya adalah tidak boleh ada nama barang yang sama.

Catatan Perumusan kode barang diserahkan sepenuhnya pada para pemakai.

DBGrid akan mengurutkan (sorting) data pada tabel berdasarkan **primary key** yang ada. Untuk pencarian nama barang biasanya dibutuhkan nama yang terurut, bukan kode barangnya. *Source* fungsi `setSqlCursor()` sudah memberitahu bagaimana mengurutkan data berdasarkan field tertentu. Kini hal yang sama akan dibuat untuk field *nama*.

```

sqlindex.py
-----
01| from qt import *
02| from qtsql import QSqlCursor, QSqlIndex
03| from login import FormLogin
04| from dbgrid import DBGrid
05|
06| app = QApplication([])
07| if FormLogin(None).db.isOpen():
08|     c = QSqlCursor( "barang" )
09|     index = QSqlIndex()
10|     index.append( c.field("nama" ) )
11|     c.setSort( index )
12|     fm = DBGrid(None)
13|     fm.setSqlCursor(c)
14|     fm.show()
15|     app.setMainWidget(fm)
16|     app.exec_loop()

```

Meski DBGrid secara default mengurutkan data berdasarkan primary key, namun ia - melalui `setSqlCursor()` - juga memeriksa terlebih dahulu apakah *cursor*-nya sudah memiliki index. Bila sudah ia akan menggunakan index yang ada.

Pada SQL pengurutan pada contoh di atas berarti

```
SELECT * FROM barang ORDER BY nama;
```

Berkaitan dengan sort ini, Form Pencarian pada halaman 168 akan diubah sedikit karena sumber datanya berasal dari database.

```
caritabel.py
-----
01| from qt import *
02| from dbgrid import DBGrid
03| from qtsql import QSqlCursor, QSqlIndex
04| from linecase import LineCase
05|
06| class FormCari(QDialog):
07|     def __init__(self, parent, cursor, searchfieldname):
08|         QDialog.__init__(self, parent)
09|         self.setCaption( cursor.name() )
10|         layout = QVBoxLayout(self)
11|         self.teks = LineCase(self, LineCase.Upper)
12|         self.tabel = DBGrid(self)
13|         layout.addWidget(self.teks)
14|         layout.addWidget(self.tabel)
15|         self._field = cursor.field(searchfieldname)
16|         index = QSqlIndex()
17|         index.append( self._field )
18|         cursor.setSort(index)
19|         self._cursor = cursor
20|         self.tabel.setSqlCursor( cursor )
21|         self.kolomCari = cursor.position(searchfieldname)
22|         self.teks.teksBerubah = self.teksBerubah
23|
24|     def keyPressEvent(self, e):
25|         if e.key() == Qt.Key_Escape: self.escape()
26|         elif e.state() == Qt.ControlButton and \
27|             e.key() == Qt.Key_PageDown: self.lanjut()
```

```
28|     elif e.key() in [Qt.Key_Return, Qt.Key_Enter]:
29|         self.enter()
30|     else: self.tabel.keyPressEvent( e )
31|
32| def enter(self):
33|     self._cursor.seek( self.tabel.currentRow() )
34|     self.accept()
35|
36| def escape(self):
37|     if self.teks.text().isEmpty(): self.reject()
38|     else: self.teks.clear()
39|
40| def teksBerubah(self, w):
41|     if self.teks.text().isEmpty(): return
42|     if self.cari() < 0 and self.cari(0) < 0:
43|         QMessageBox.warning(self, "Perhatian",
44|             "%s tidak ditemukan" % self.teks.text())
45|
46| def cari(self, awalan=1, mulaiBaris=-1):
47|     s = "%s" % self.teks.text().upper()
48|     brs = mulaiBaris
49|     ketemu = 0
50|     while brs < self.tabel.numRows()-1:
51|         brs = brs + 1
52|         self._cursor.seek(brs)
53|         data = self.tabel.strField( self._field )
54|         t = QString(data).upper()
55|         if awalan:
56|             ketemu = t.find(QRegExp("^"+s)) > -1
```



```
57|         else: ketemu = t.find(s) > -1
58|         if ketemu:
59|             if self.tabel.currentRow() != brs:
60|                 self.tabel.setCurrentCell(brs, self.kolomCari)
61|             return brs
62|         return -1
63|
64|     def lanjut(self):
65|         if self.teks.text().isEmpty(): return
66|         if self.cari(0, self.tabel.currentRow() ) < 0:
67|             self.cari(0)
68|
69|
70| if __name__ == "__main__":
71|     from login import FormLogin
72|
73|     app = QApplication([])
74|     if FormLogin(None).db.isOpen():
75|         c = QSqlCursor( "barang" )
76|         fm = FormCari(None, c, "nama" )
77|         fm.tabel.setColumnWidth(1,200)
78|         fm.tabel.setCase("nama", LineCase.Upper)
79|         fm.tabel.setMaxLength("nama",20)
80|         fm.showMaximized()
81|         fm.exec_loop()
82|         if fm.result() == QDialog.Accepted:
83|             print fm._cursor.field("id").value().toInt()
```

23.5.7 Nomor Urut - Autoincrement

Bila Anda menginginkan kode barang “sekedar kode”, maka solusi nomor urut (autoincrement) bisa menjadi pilihan. Artinya pemakai tidak perlu mengisikan kode barang karena akan diisi secara otomatis sesuai dengan nomor urut terakhir. Untuk mendapatkannya gunakan fungsi `MAX()` pada SQL.

```
SELECT MAX(id) FROM barang;
```

Namun cara ini tidak dianjurkan karena semakin banyak record semakin lama proses pencarian nilai maksimum. Sebagai gantinya dibuatlah sebuah tabel untuk mencatat nomor terakhir.

```
CREATE TABLE urutan(  
  tabel VARCHAR(32) NOT NULL PRIMARY KEY,  
  nomor INTEGER NOT NULL);
```

Bila tabel barang sudah terisi, dapatkan nomor terakhirnya dengan `MAX()`. Misalkan diperoleh angka 142, maka isilah tabel urutan dengan query berikut:

```
INSERT INTO urutan(tabel,nomor) VALUES ('barang',142);
```

Namun bila masih kosong gantilah dengan angka 0.

Selanjutnya nomor urut ini akan dimasukkan pada field `id` *se saat* sebelum disimpan. Event ini bisa kita dapatkan pada `beforePost()` yang memang disediakan untuk validasi tambahan.

```
barang.py
-----
01| from qt import *
02| from qtsql import QSqlCursor, QSqlIndex
03| from caritabel import FormCari
04| from linecase import LineCase
05|
06| class FormBarang(FormCari):
07|     def __init__(self, parent):
08|         self.barang = QSqlCursor("barang")
09|         FormCari.__init__(self, parent, self.barang, "nama")
10|         self.tabel.setColumnWidth(1,200)
11|         self.tabel.setCase("nama", LineCase.Upper)
12|         self.tabel.setMaxLength("nama",20)
13|         self.urutan = QSqlCursor("urutan")
14|         self.urutan.select("tabel='barang'")
15|         self.urutan.next()
16|         self.tabel.afterInsert = self.afterInsert
17|         self.tabel.beforePost = self.beforePost
18|         self.tabel.afterPost = self.afterPost
19|         self.nama = ""
20|
21|     def afterInsert(self, cursor):
22|         self.tabel.setFocus()
23|         self.tabel.setCurrentCell(self.tabel.currentRow(),1)
24|
25|     def beforePost(self, cursor):
26|         if self.tabel.status == self.tabel.Insert:
27|             n = self.urutan.value("nomor").toInt() + 1
```

```
28|         cursor.editBuffer().setValue("id", QVariant(n))
29|         b = self.urutan.primeUpdate()
30|         b.setValue("nomor", QVariant(n))
31|         self.urutan.update()
32|         self.urutan.select()
33|         self.urutan.next()
34|         self.nama = cursor.editBuffer().value("nama").\
35|             toString()
36|
37|     def afterPost(self, cursor):
38|         self.teks.setText( self.nama )
39|         self.teks.setFocus()
40|         self.teks.selectAll()
41|
42|
43| if __name__ == "__main__":
44|     from login import FormLogin
45|     app = QApplication([])
46|     if FormLogin(None).db.isOpen():
47|         fm = FormBarang(None)
48|         fm.showMaximized()
49|         fm.exec_loop()
```

Bab 24

Kasir III

Karena sudah masuk tema database, Kasir III akan dilengkapi fitur untuk menyimpan transaksi penjualan.

24.1 Struktur Tabel

Transaksi disimpan dalam dua tabel. Tabel pertama bernama penjualan yang berisi *header* transaksi:

```
CREATE TABLE penjualan(  
  id INTEGER NOT NULL PRIMARY KEY,  
  wkt DATETIME NOT NULL);
```

`id` merupakan identitas transaksi. Field ini bersifat *autoincrement*, sehingga kita perlu mendaftarkan nilai terakhirnya dalam

tabel urutan:

```
INSERT INTO urutan (tabel, nomor) VALUES ('penjualan', 0);
```

dan `wkt` adalah waktu transaksi, yaitu tanggal dan jam transaksi.

Sedangkan tabel kedua berisi rinciannya, yaitu daftar barang yang dijual:

```
CREATE TABLE penjualan_barang(
  id_penjualan INTEGER NOT NULL REFERENCES penjualan(id),
  id_barang INTEGER NOT NULL REFERENCES barang(id),
  jumlah FLOAT NOT NULL,
  harga FLOAT NOT NULL,
  PRIMARY KEY (id_penjualan, id_barang) );
```

Nilai `id_penjualan` harus terdaftar pada field `id` di tabel `penjualan`, begitu pula dengan `id_barang` harus terdaftar pada field `id` di tabel `barang`.¹ Hal ini ditunjukkan dengan adanya “perintah” `REFERENCES`. Baik `id_penjualan` maupun `id_barang` dikategorikan sebagai *foreign key*.

Berbeda dengan tabel-tabel sebelumnya, tabel `penjualan_barang` memiliki dua field sebagai primary key, yaitu `id_penjualan` dan `id_barang`. Ini artinya dalam sebuah transaksi tidak boleh tercatat dua barang yang sama di dua record. Namun struktur ini bisa jadi memiliki kelemahan. Misalkan ada penjualan dua

¹Field yang menjadi referensi harus merupakan primary key di tabel referensi.

buah semangka dengan berat masing-masing 3 dan 4 kg. Tentu saja kita tidak ingin mencatatnya sebagai sebuah semangka dengan berat 7 kg karena ini bisa membuat kekeliruan dalam menghitung banyaknya barang.

Membuang primary key juga tidak dianjurkan karena hanya membuat tabel tersebut “hanya bisa” di-INSERT. Jadi kita tambahkan saja sebuah field bertipe integer namun dengan ukuran yang lebih kecil, yaitu SMALLINT.² Kalau sudah terlanjut membuat tabelnya, maka bisa dihapus dengan perintah berikut:

```
DROP TABLE penjualan_barang;
```

lalu buat kembali:

```
CREATE TABLE penjualan_barang(  
  id_penjualan INTEGER NOT NULL REFERENCES penjualan(id),  
  nomor SMALLINT NOT NULL,  
  id_barang INTEGER NOT NULL REFERENCES barang(id),  
  jumlah FLOAT NOT NULL,  
  harga FLOAT NOT NULL,  
  PRIMARY KEY (id_penjualan, nomor) );
```

Field nomor merupakan nomor urut per transaksi yang akan diisi otomatis (autoincrement).

24.2 Daftar Barang

Kasir III memanfaatkan Form Barang untuk mencari ID barang, sekaligus untuk menambah data barang baru atau mengubah

²Jangkauan SMALLINT adalah -32767 hingga 32767.

Gambar 24.1: Kasir

harga. Untuk menampilkannya cukup menekan tombol huruf A-Z. Huruf tersebut memicu tampilnya Form Barang sekaligus menyertakan huruf tadi.

24.3 Penyimpanan Data

Pada saat pemakai mendata barang yang dibeli pelanggan, data tidak langsung dimasukkan dalam tabel, melainkan ditampung terlebih dahulu. Setelah pembayaran (mencetak struk) barulah data disimpan. Teknik ini diambil karena pertimbangan kecepatan layanan pada para pelanggan.

24.4 Pencetakan

Struk disimpan ke sebuah file yang telah ditentukan lokasinya, untuk kemudian dicetak ke printer. Pada saat printer mencetak, pemakai sudah dapat memasukkan data kembali. Untuk struk yang cukup panjang dengan antrian pembeli yang juga panjang hal ini tentu sangat menghemat waktu.

Meski program ini mewajibkan penggunaan printer, namun ketiadaannya tidak menghalangi proses pemasukan data, karena pencetakan diserahkan sepenuhnya pada sistem operasi, tidak peduli pencetakan berhasil atau tidak.

24.5 Program

Agar form lebih informatif, program ini juga menyertakan `WaktuDigital`.³ Silahkan ubah font-nya apabila dianggap terlalu kecil.

```
kasir3.py
-----
001| from qt import *
002| from qtsql import QSqlCursor, QSqlQuery
003| from waktudigital import WaktuDigital
004| from valuegrid import ValueGrid
005| from barang import FormBarang
006| from fungsi import ribu
007| from string import rjust, ljust
008| import os
009|
010| class LineControl(QLineEdit):
011|     def __init__(self, parent):
012|         QLineEdit.__init__(self, parent)
013|         self.setValidator( QDoubleValidator(self) )
014|         self.tombolDitekan = None
015|
016|     def keyPressEvent(self, e):
017|         if self.tombolDitekan: self.tombolDitekan( e )
018|
019|     # Kursor / Focus control tetap di sini
020|     def focusOutEvent(self, e):
021|         self.setFocus()
```

³Lihat source-nya pada halaman 143.

```
022|
023|
024| class FormKasir(QDialog):
025|     Huruf = range(ord("A"),ord("Z")+1)
026|     Huruf = Huruf + range(ord("a"),ord("z")+1)
027|     GridKeys = [Qt.Key_Up, Qt.Key_Down,
028|                 Qt.Key_PageUp, Qt.Key_PageDown,
029|                 Qt.Key_Delete]
030|
031|     def __init__(self, parent):
032|         QDialog.__init__(self, parent)
033|         self.setCaption("Kasir III")
034|         layout = QVBoxLayout(self)
035|         layout.setAutoAdd(1)
036|         jam = WaktuDigital(self)
037|         self.lcd = QLCDNumber(self)
038|         self.lcd.setSegmentStyle( QLCDNumber.Flat )
039|         self.lcd.setMinimumHeight(150)
040|         self.lcd.setNumDigits(10)
041|         self.teks = LineControl(self)
042|         self.tabel = ValueGrid(self)
043|         self.tabel.setNumCols(4)
044|         self.tabel.hideColumn(0) # id barang
045|         self.tabel.horizontalHeader().setLabel(1, "Nama")
046|         self.tabel.horizontalHeader().setLabel(2, "Jumlah")
047|         self.tabel.horizontalHeader().setLabel(3, "Harga")
048|         self.fmBrg = FormBarang(self)
049|         self.brg = QSqlCursor( "barang" )
050|         self.urutan = QSqlCursor( "urutan" )
```

```
051|         self.urutan.select( "tabel='penjualan'" )
052|         self.query = QSqlQuery()
053|         self.output = "/tmp/kasir.txt"
054|         self.reset()
055|         self.teks.tombolDitekan = self.tombolDitekan
056|         self.tabel.beforeDelete = self.beforeDelete
057|
058|     def tombolDitekan(self, e):
059|         if e.key() in [Qt.Key_Return, Qt.Key_Enter]:
060|             self.cariKode()
061|         elif e.key() == Qt.Key_Asterisk: self.ubahJml()
062|         elif e.key() == Qt.Key_Plus: self.bayar()
063|         elif e.ascii() in self.Huruf:
064|             self.formBarang( e.text() )
065|         elif e.key() == Qt.Key_Escape: self.teks.clear()
066|         elif e.key() in self.GridKeys:
067|             if e.key() == Qt.Key_Down and \
068|                 self.tabel.currentRow() == \
069|                 self.tabel.numRows()-1:
070|                 return
071|             self.tabel.keyPressEvent( e )
072|         else: QLineEdit.keyPressEvent( self.teks, e )
073|
074|     def resizeEvent(self, e):
075|         self.tabel.setColumnWidth( 1, self.tabel.width() -
076|             self.tabel.columnWidth(2) -
077|             self.tabel.columnWidth(3) - 100)
078|
079|     def reset(self):
```

```
080|     self.total = 0
081|     self.jml = 1.0
082|     self.tabel.setData([])
083|     self.lcd.display(0)
084|     self.teks.clear()
085|     self.perluReset = 0
086|
087|     def ubahJml(self):
088|         try: self.jml = float(str(self.teks.text()))
089|         except: return
090|         # dua desimal saja
091|         self.jml = round(self.jml*100)/100
092|         self.lcd.display(self.jml)
093|         self.teks.clear()
094|
095|     def beforeDelete(self, row):
096|         self.total = self.total - self.tabel.data[row][3]
097|         self.lcd.display(ribu(self.total))
098|
099|     def cariKode(self):
100|         where = "id='%s'" % self.teks.text()
101|         self.brg.select(where)
102|         if self.brg.next(): self.tambah( self.brg )
103|         else:
104|             self.pesan( "Kode barang %s tidak ada" % \
105|                 self.teks.text() )
106|             self.teks.selectAll()
107|
108|     def tambah(self, cursor):
```

```
109|         if self.perluReset: self.reset()
110|         harga = int( self.jml *
111|             cursor.value("harga").toDouble() )
112|         self.total = self.total + harga
113|         self.lcd.display(ribu(self.total))
114|         rec = []
115|         rec.append( cursor.value("id").toInt() )
116|         rec.append( str(cursor.value("nama").toString() ) )
117|         rec.append( self.jml )
118|         rec.append( harga )
119|         if self.tabel.data:
120|             self.tabel.turun()
121|             row = self.tabel.currentRow()
122|             self.tabel.data[row] = list(rec)
123|         else: self.tabel.data.append( list(rec) )
124|         self.tabel.repaintContents()
125|         self.teks.clear()
126|         self.jml = 1.0
127|
128|     def timerEvent(self, e):
129|         self.killTimers()
130|         self.fmBrg.teks.setText( self._teks )
131|
132|     def formBarang(self, teks):
133|         self._teks = teks # untuk timer
134|         self.fmBrg.showMaximized()
135|         # Bila data yang dicari terlalu ke bawah, current
136|         # record tidak tampak. Timer digunakan agar pada
137|         # saat form barang tampil, huruf dimasukkan untuk
```

```
138|         # dicari.
139|         self.startTimer( 500 )
140|         self.fmBrg.exec_loop()
141|         if self.fmBrg.result() == QDialog.Accepted:
142|             self.tambah( self.fmBrg._cursor )
143|
144|     def bayar(self):
145|         try: b = int(float(str(self.teks.text())))
146|         except ValueError:
147|             self.pesan("%s bukan angka" % self.teks.text())
148|             return
149|         k = int(b - self.total)
150|         self.lcd.display(ribu(k))
151|         self.teks.clear()
152|
153|         # Header
154|         self.urutan.select()
155|         self.urutan.next()
156|         id_penjualan = \
157|             self.urutan.value("nomor").toInt()+1
158|         sql = """"UPDATE urutan SET nomor=%s
159|             WHERE tabel='penjualan'"""" % id_penjualan
160|         if not self.executeQuery( sql ): return
161|
162|         # Antisipasi multiuser, tanggal mengambil
163|         # dari database, menghindari perbedaan setting
164|         # tanggal antar komputer client
165|         if not self.executeQuery( "SELECT NOW()" ): return
166|         self.query.next()
```

```
167|         tgl = self.query.value(0).toDateTime()
168|         self.struk = "%s\n%s\n" % (
169|             tgl.toString("dd-MM-yyyy hh:mm"), "-" * 35 )
170|
171|         # Cetak
172|         nomor = 0
173|         daftarSql = []
174|         for rec in self.tabel.data:
175|             nomor = nomor + 1
176|             kode, nama, jml, harga = rec
177|             daftarSql.append( ""INSERT INTO penjualan_barang
178|                 (id_penjualan, nomor, id_barang, jumlah, harga)
179|                 VALUES (%s, %s, %s, %s, %s)"" % \
180|                 ( id_penjualan, nomor, kode, jml, harga ) )
181|             self.cetakBarang( nama, jml, harga )
182|         self.struk = "%s%s\n" % (self.struk, "-" * 35)
183|         self.cetakAkhir( "TOTAL", self.total )
184|         self.cetakAkhir( "BAYAR", b )
185|         self.cetakAkhir( "KEMBALI", k )
186|         self.cetakFile()
187|         self.perluReset = 1
188|
189|         # Simpan ke database
190|         sql = ""INSERT INTO penjualan (id, tgl)
191|             VALUES (%s,%s)"" % ( id_penjualan,
192|                 tgl.toString(Qt.ISODate) )
193|         if not self.execQuery( sql ): return
194|         for sql in daftarSql:
195|             if not self.execQuery( sql ): return
```

```
196|
197| def pesan(self, s):
198|     QMessageBox.warning(self, "Perhatian", s)
199|
200| def execQuery(self, sql):
201|     ok = self.query.execQuery( sql )
202|     if ok: return ok
203|     else: self.pesan ( "%s\n%s" % (
204|         self.query.lastError().databaseText(),
205|         self.query.lastQuery() ) )
206|
207| def cetakBarang(self, nama, jml, subtotal):
208|     self.struk = "%s%s%s\n" % ( self.struk,
209|         ljust(nama[:20], 20),
210|         rjust(ribu(jml), 7),
211|         rjust(ribu(subtotal),8) )
212|
213| def cetakAkhir(self, nama, total):
214|     self.struk = "%s%s%s\n" % ( self.struk,
215|         ljust(nama,10),
216|         rjust(ribu(total),25) )
217|
218| def cetakFile(self):
219|     s = "%s\n%s%s" % (chr(15), self.struk, "\n"*7)
220|     f = open(self.output, "w")
221|     f.write(s)
222|     f.close()
223|     os.system("killall cat")
224|     os.system("cat %s > /dev/lp0 &" % self.output)
```



```
225|
226|
227| if __name__ == "__main__":
228|     from login import FormLogin
229|     app = QApplication([])
230|     if FormLogin(None).db.isOpen():
231|         fm = FormKasir(None)
232|         fm.showMaximized()
233|         fm.exec_loop()
```

Ada yang menarik dalam program ini dimana `LineControl` merupakan pusat kendali. Event `focusOutEvent()` membuatnya selalu menjadi “pusat perhatian”. Jadi meski Anda mengklik `ValueGrid` untuk mengalihkan fokus kursor, tetap saja kursor berada di `LineControl`.

Penggunaan timer pada program ini merupakan trik karena `DBGrid` tidak menampilkan current record sebelum form-nya tampil. Padahal untuk menampilkan form dibutuhkan `exec_loop()` yang akan membuat proses “terhenti” hingga form ditutup. Oleh karena itu dibutuhkan “petugas lain” yang bekerja beberapa saat setelah form tampil. Tugasnya adalah memasukkan huruf ke dalam teks pencarian agar current record berada pada nama awal barang yang dicari. Ketemu atau tidak selanjutnya timer tidak dibutuhkan lagi dengan memanggil fungsi `killTimers()`.

`Qt.ISODate` membuat `QDateTime.toString()` menampilkan format waktu sesuai standar ISO, yaitu berturut-turut: tahun, bulan, tanggal, jam, menit, dan detik. Pola ini dibutuhkan dalam perintah SQL untuk menerima masukan field waktu.

Latihan Tambahkan nama toko dalam header struk. Nama

toko ini tersimpan dalam database.

24.6 Laporan

Ada beberapa laporan yang dapat dibuat berdasarkan struktur tabel yang ada. Laporan ini kerap dibutuhkan untuk analisa bisnis guna meningkatkan kualitas layanan dan juga kuantitas penjualan - tentunya.

24.6.1 Barang Terlaris

Laporan ini menampilkan nama barang dengan urutan mulai dari yang terlaris. Manfaatnya adalah untuk memprioritaskan pengadaan barang tertentu yang paling sering dibeli pelanggan.

```
SELECT b.nama, COUNT(*)
FROM penjualan_barang pb, barang b
WHERE pb.id_barang = b.id
GROUP BY 1 ORDER BY 2 DESC;
```

Bila Anda berniat untuk membuat program tampilannya, bisakan menyertakan periode transaksi. Periode ini berupa dua masukan tanggal: awal dan akhir, sehingga para pemakai mendapat keleluasaan menentukan periode: harian, mingguan, bulanan, tahunan, dsb. Contoh:

```
SELECT b.nama, COUNT(*)
FROM penjualan_barang pb, barang b, penjualan p
WHERE pb.id_barang = b.id
```

```
    AND pb.id_penjualan = p.id
    AND (p.tgl BETWEEN '2003/2/1' AND '2003/2/28')
GROUP BY 1 ORDER BY 2 DESC;
```

Option DESC (*descending*) digunakan untuk mengurutkan data dari yang nilainya paling tinggi ke yang paling kecil.

24.6.2 Total Penjualan Harian

Laporan ini menampilkan tanggal transaksi beserta total nilainya. Berikut ini perintah dalam PostgreSQL:

```
SELECT date(tgl), SUM(pb.harga)
FROM penjualan p, penjualan_barang pb
WHERE p.id = pb.id_penjualan
GROUP BY 1 ORDER BY 1;
```

sedangkan dalam MySQL:

```
SELECT FROM_DAYS(TO_DAYS(p.tgl)), SUM(pb.harga)
FROM penjualan p, penjualan_barang pb
WHERE p.id = pb.id_penjualan
GROUP BY 1 ORDER BY 1;
```

24.6.3 Rata-rata Penjualan Harian

Laporan ini bisa digunakan untuk mengukur pangsa pasar dari sudut potensi pelanggan atau kalau boleh dikatakan sebagai daya beli. Gunakan fungsi AVG() sebagai pengganti SUM() pada contoh di atas.

24.6.4 Jam Sibuk

Untuk mengetahui jam-jam kepadatan pembeli jalankan perintah berikut:

```
SELECT EXTRACT(HOUR FROM tgl), COUNT(*)  
FROM penjualan GROUP BY 1 ORDER BY 1 DESC;
```

Bisa digunakan sebagai acuan untuk mempersiapkan tenaga tambahan guna menghindari antrian panjang.

Bibliografi

- [1] Sugiana, Owo, *Pemrograman Dasar dengan Python*, Jakarta, 2001
- [2] Lutz, Mark, *Programming Python*, O'Reilly, 1996
- [3] http://groups.yahoo.com/group/id_python/files/pytut.id.html
- [4] Sugiana, Owo, *Aplikasi Rumah Sakit Pertamina Jaya - Referensi Teknis*, Jakarta, 2000
- [5] PyQt, www.riverbankcomputing.co.uk/pyqt
- [6] Python, www.python.org
- [7] Qt, www.qt.org
- [8] KDE, www.kde.org
- [9] PostgreSQL, www.postgresql.org
- [10] MySQL, www.mysql.org

- [11] RedHat, www.redhat.com
- [12] Linux, www.linux.org
- [13] Infolinux, www.infolinux.co.id

Indeks

- * perkalian, 39
- ** pangkat, 39
- + penjumlahan, 39
- pengurangan, 39
- / pembagian, 38, 39
- % - string formatting, 91
- &, QLabel, 114
- __doc__, 61
- __init__(), 84
- __init__(), Grid, 165
- __init__(), QTable, 160
- __name__, 95

- acak, bilangan acak, 61
- accept(), QCloseEvent, 152
- accept(), QDialog, 145
- activateNextCell(), QTable, 160
- addColumn(), QDataTable, 193

- addDatabase(), QSqlDatabase, 184
- addDays(), QDate, 140
- addDays(), QDateTime, 143
- addMonths(), QDate, 140
- addMonths(), QDateTime, 143
- addSecs(), QDateTime, 143
- addSecs(), QTime, 139
- addStretch(), QBoxLayout, 134
- addWidget(), QGridLayout, 137
- addWidget(), QLayout, 132
- addYears(), QDate, 140
- addYears(), QDateTime, 143
- akar, 74
- Alt, keyboard, 117
- analyst, dokumentasi, 24
- and, 55
- angka(), 162, 198

- antarmuka, Python antarmuka C, 93
- append(), list, 44
- argv, 71
- Arial, font, 96
- array, associative array, 48
- array, tipe data, 22
- ascending, QComboBox, 105
- ASCII, 71
- at(), QSqlQuery, 190
- autocompletion, QComboBox, 105
- autowarp, QTextEdit, 89
- AVG(), SQL, 226

- background, 167
- backslash, 41
- bahasa pemrograman, 18
- baris, list dua dimensi, 47
- beginEdit(), QTable, 210
- Belanda, 21
- black, 98
- blok program, 51
- blue, 98
- BMP, 24
- bold, font, 96
- boolean, 98
- break, 62
- buffer pada QSqlCursor, 197

- bug, 24

- C, 18, 22
- C++, 18, 22
- C++, bahasa library Qt, 93
- capslock, 55
- caption, 85
- cast, 43
- cat, 70, 71, 152
- cell, current cell, 157
- cell, elemen QTable, 157
- cellWidget(), QTable, 166
- chdir(), 72
- checkbox, QCheckBox, 99
- checked, QCheckBox, 99
- child, 89
- chmod, 70
- chr(), 71
- clear(), 93
- clear(), QSqlField, 197
- clearCell(), QTable, 161, 166
- clearEdit(), QComboBox, 189
- clicked(), sinyal QButtonGroup, 102
- clicked(), sinyal QPushButton, 93
- close(), file, 70, 121
- closeEvent(), QWidget, 152
- combobox, QComboBox, 105

- command line, 28
- connect(), QObject, 93
- connection ID, 179
- console, 72
- console environment, text editor, 29
- constructor, 84
- container, 123
- Control, keyboard, 117
- count(), QButtonGroup, 102
- COUNT(), SQL, 225
- Courier, 149
- Courier, font, 96
- CREATE DATABASE, SQL, 180, 181
- CREATE TABLE, SQL, 185
- createEditor(), QTable, 166, 210
- cross, karakter, 35
- CSV, 76
- current column, QTable, 157
- current directory, 72
- current record, 189
- current row, QTable, 157
- currentChanged(), QTable, 157
- currentColumn(), QTable, 161
- currentDate(), QDate, 140
- currentDateTime(), QDateTime, 121, 142
- currentItem(), QComboBox, 106, 189
- currentRow(), QTable, 161
- currentText(), QComboBox, 106
- currentTime(), QTime, 139
- CursorTable, class, 194
- cyan, 98

- darkBlue, 98
- darkCyan, 98
- darkGray, 98
- darkGreen, 98
- darkMagenta, 98
- darkRed, 98
- darkYellow, 98
- database, 185
- database file, 179
- database server, 179
- databaseText(), QSqlError, 184
- date(), QDateTime, 142
- DATETIME ke DATE, SQL, 226
- day(), QDate, 140
- dayOfWeek(), QDate, 140
- dayOfYear(), QDate, 140

- daysInMonth(), QDate, 140
- daysInYear(), QDate, 140
- daysTo(), QDate, 141
- daysTo(), QDateTime, 143
- DBGGrid, class, 199, 224
- debugging, 42
- debugging, arti, 24
- debugging, proses, 24
- def, 63
- deklarasi variabel, 35
- del, dictionary, 49
- del, list, 44
- DELETE, SQL, 185
- Delphi, Borland, 199
- delRecords(), QSqlCursor, 197
- descending, QComboBox, 105
- desimal, 40
- development cycle, 18
- device, 70
- dir(), info properti modul, 61
- dir(), info properti objek, 50
- direktori aktif, 72
- disable, 110
- display(), QLCDNumber, 112
- dokumentasi modul, 61
- dokumentasi, manfaat, 24
- DOS Prompt, 28
- dotmatrix, 76
- double-click, 107
- drawLine(), QPainter, 168
- drawText(), QPainter, 168
- driver, 179
- driver database, 179
- DROP TABLE, SQL, 218
- edit-mode, QTable, 166
- editBuffer(), QSqlCursor, 197
- editor, pada QComboBox, 105
- efisien, 23
- elif, 53
- else, 53
- embeddable, 22
- enable, 110
- enter, karakter, 69
- Enter, keyboard, 115
- eraseRect(), QPainter, 168
- error, 73
- event, 94, 117
- event driven programming, 91
- except, 74
- exception, 73
- exec(), 66, 161
- exec__loop(), 84
- exec_loop(), QDialog, 224
- execQuery(), QSqlQuery, 189
- extensible, 22

- EXTRACT(), SQL, 226
- faktorial, bilangan, 60
- faktorial, fungsi, 65
- false, 52
- field(), QSqlRecord, 197
- field, file teks, 76
- field, list dua dimensi, 47
- file, 69, 76
- file, close(), 121
- file, flush(), 121
- file, write(), 121
- find(), QButtonGroup, 102
- find(), QString, 173
- first(), QSqlQuery, 190
- float(), integer ke float, 38
- float(), string ke float, 43
- float, tipe data, 23, 37
- flush(), file, 121
- focus widget, 158
- focused, 110
- focusOutEvent(), QWidget, 224
- font printer, 70
- font(), QWidget, 97
- font, fixed, 149
- fonts, daftar font pada kwrite, 97
- for di dalam for, 60
- for, in range(), 60
- for, perbandingan, 61
- for, perulangan, 59
- form, 84
- form manager, QApplication, 84
- format(), locale, 40
- formatting, string, 91
- frame, 123
- fungsi, nilai keluaran, 64
- fungsi, nilai masukan, 64
- fungsi, penyisipan, 93
- fungsi, rekursif, 65
- ganjil, bilangan ganjil, 55
- gedit, 29
- getcwd(), 72
- getDouble(), QDialog, 154
- getInteger(), QDialog, 154
- getOpenFileName(), QFileDialog, 148
- getSaveFileName(), QFileDialog, 148
- getText(), QDialog, 154
- Gnome, 29
- grafis, modus grafis, 76
- GRANT, SQL, 191

- gray, 98
- green, 98
- Grid, `__init__()`, 165
- Grid, class, 157, 161, 198
- GROUP BY, SQL, 225
- Guido, 21

- halaman, ganti halaman, 70
- `has_key()`, dictionary, 78
- `height()`, `QRect`, 168
- Helvetica, font, 96
- highlight, 167
- home directory, 117
- `horizontalHeader()`, `QTable`, 156
- `hour()`, `QTime`, 139
- huruf, ukuran, 70

- icon, 149
- if, 51
- `ignore()`, `QCloseEvent`, 152
- in, for, 59
- in, operator, 54
- indent, aturan penulisan, 34
- Indonesia, 21, 41
- input, `QLineEdit`, 88
- `insert()`, list, 44
- INSERT, SQL, 185
- `insertItem()`, `QComboBox`, 189
- `insertItem()`, `QListBox`, 108
- `insertRows()`, `QTable`, 160, 166
- `insertStrList()`, `QComboBox`, 106
- `insertTab()`, `QTabWidget`, 129
- install, 29
- `int()`, pembulatan, 40
- `int()`, string ke integer, 43, 55
- integer, tipe data, 23, 37
- interactive interpreter, 51
- interactive mode, 35
- interface, Python: C interface, 93
- internet, 21, 22
- interpreter, `exec()`, 66
- interpreter, lingkup sys, 72
- `intValue()`, `QLCDNumber`, 117
- IP, Internet Protocol, 179
- `isChecked()`, `QCheckBox`, 100
- `isEmpty()`, `QString`, 111
- ISODate, Qt, 225
- italic, font, 96

- `jam()`, string ke `QDateTime`, 199
- jam, `QTime`, 139
- Java, 18, 22

- JPEG, 24
- kalkulator, 114
- kalkulator, contoh, 66
- kalkulator, interactive mode, 38
- karakter ASCII, 71
- karakter, font, 96
- kate, 29
- KDE, 29, 81, 113
- key(), QKeyEvent, 117
- keypad, 115
- keyPressEvent(), QWidget, 117
- keys(), dictionary, 49
- killTimers(), QObject, 225
- kolom, list dua dimensi, 47
- konversi, kesalahan, 55
- konversi, string ke float, 43
- kutip ganda, karakter, 35, 41
- kutip tunggal pada query, 189
- kutip tunggal, karakter, 41
- kwrite, 29, 33, 131
- kwrite, melihat daftar font, 97
- larik, nama lain list, 43
- last(), QSqlQuery, 190
- lastError(), QSqlDatabase, 184
- layouter, 131
- len(), jumlah elemen list, 45
- len(), panjang string, 43
- library, 81
- lightGray, 98
- LineCase, class, 95, 96
- LineCase, event teksBerubah, 96
- linecase, modul, 94
- LineCase, teksBerubah, 95
- LineControl, 224
- Linux, 21, 27, 93, 180
- list dua dimensi, 47, 155, 165, 173
- list(), menyalin isi list, 48
- ljust(), string, 78
- locale, modul, 40
- logika, bentuk logika, 52, 55
- logika, kesalahan logika, 73
- login, database server, 179
- loop, 59
- looping, 84
- lp0, 152
- lp0, printer, 70
- magenta, 98
- maintenance, 22
- margin, aturan penulisan, 34

- math, modul, 38
- MAX(), SQL, 213
- memori, alokasi memori, 24
- memori, rekursif, 65
- minute(), QTime, 139
- month(), QDate, 140
- mouse, 113
- MS-SQL, 179
- msec(), QTime, 139
- multidimensi, list, 45, 60
- multiline, 88
- multiuser, 181
- MySQL, 179–181
- mysql, MySQL client, 181, 193

- next(), QSqlQuery, 189, 190
- non-visual class, 90
- None, logika false, 52
- None, nilai keluaran fungsi, 65
- None, objek hampa, 65
- not, 55
- NOT, SQL, 185
- NULL, SQL, 185, 197

- object oriented, 21
- object oriented programming, 84

- objek, berorientasi objek, 21, 23
- objek, tipe data, 23
- objek-tampak, QWidget, 87
- ODBC, 179
- open(), file, 69
- OpenGL, 18
- operator bilangan, 39
- option, 25
- optional, istilah, 189
- or, 56
- Oracle, 179
- ord(), 71
- os, modul, 71, 72
- out of paper, 71
- override, 94
- owner, 89

- paintCell(), QTable, 161, 166
- panel, 123
- parent, 89
- parent, font dan warna, 99
- Pascal, bahasa pemrograman, 18
- Pascal, perbandingan, 34
- pass, 166
- pecahan, bilangan pecahan, 37

- pedoman menulis program, 23
- Perl, 22
- PHP, 34
- pico, 29
- plaintext, 91
- pointer, 76
- popup-list, pada QComboBox, 105
- postgres, superuser PostgreSQL, 180
- PostgreSQL, 179, 180
- prev(), QSqlQuery, 190
- PRIMARY KEY, SQL, 185, 210
- primeDelete(), QSqlCursor, 197
- primeInsert(), QSqlCursor, 197
- primeUpdate(), QSqlCursor, 197
- print, 33, 36
- print, tanpa enter, 69
- print, menampilkan QString, 91
- printer, 70
- programmer, dokumentasi, 24
- psql, PostgreSQL client, 180, 193
- PyQt, 81
- Python, 81
- Python - C++, perbandingan, 191
- Python - Java, 22
- Python - Perl, 22
- Python - Tcl, 22
- Python, alasan memilih, 21
- Python, bahasa pemrograman, 18
- Python, berorientasi objek, 23
- Python, dengan Qt, 93
- python, interactive interpreter, 36
- Python, perbandingan bahasa, 22
- Python, sejarah nama, 23
- Python, text editor, 30
- PyQt, 93
- QApplication, 83, 84
- QApplication, setMainWidget(), 83
- QBoxLayout, 134, 136, 137
- QBoxLayout, addStretch(), 134
- QBoxLayout, setDirection(), 136

- QBrush, setColor(), 167
- QBrush, setStyle(), 167
- QButtonGroup, 101, 123
- QButtonGroup, clicked(), 102
- QButtonGroup, count(), 102
- QButtonGroup, find(), 102
- QButtonGroup, selected(), 102
- QButtonGroup, setButton(), 104
- QCheckBox, 99
- QCheckBox, isChecked(), 100
- QCheckBox, pedoman, 100
- QCloseEvent, accept(), 152
- QCloseEvent, ignore(), 152
- QColor, warna, 98
- QComboBox, 105
- QComboBox, clearEdit(), 189
- QComboBox, currentItem(), 106, 189
- QComboBox, currentText(), 106
- QComboBox, insertItem(), 189
- QComboBox, insertStrList(), 106
- QComboBox, pedoman, 106
- QComboBox, removeItem(), 189
- QComboBox, setAutoCompletion(), 106
- QComboBox, setEditable(), 106
- QComboBox, text(), 106
- QDataTable, 192–194
- QDataTable, addColumn(), 193
- QDataTable, setSqlCursor(), 193
- QDate, 140
- QDate, addDays(), 140
- QDate, addMonths(), 140
- QDate, addYears(), 140
- QDate, currentDate(), 140
- QDate, day(), 140
- QDate, dayOfWeek(), 140
- QDate, dayOfYear(), 140
- QDate, daysInMonth(), 140
- QDate, daysInYear(), 140
- QDate, daysTo(), 141
- QDate, month(), 140
- QDate, toString(), 141
- QDate, year(), 140
- QDateTime, 142
- QDateTime, addDays(), 143
- QDateTime, addMonths(), 143
- QDateTime, addSecs(), 143
- QDateTime, addYears(), 143
- QDateTime, currentDateTime(), 121, 142

- QDateTime, date(), 142
- QDateTime, daysTo(), 143
- QDateTime, secsTo(), 143
- QDateTime, time(), 142
- QDateTime, toString(), 121, 225
- QDial, 126
- QDial, setValue(), 126
- QDialog, 145
- QDialog, accept(), 145
- QDialog, exec_loop(), 224
- QDialog, reject(), 145
- QDialog, result(), 145
- QFileDialog, 146
- QFileDialog, getOpenFileName(), 148
- QFileDialog, getSaveFileName(), 148
- QFont, 96
- QFont, setBold(), 98
- QFont, setFamily(), 97
- QFont, setItalic(), 98
- QFont, setPointSize(), 97
- QFont, setUnderline(), 98
- QFrame, 123
- QGridLayout, 136, 137
- QGridLayout, addWidget(), 137
- QHBoxLayout, 131, 134
- QHeader, setLabel(), 156
- QInputDialog, 146
- QInputDialog, getDouble(), 154
- QInputDialog, getInteger(), 154
- QInputDialog, getText(), 154
- QKeyEvent, 117
- QKeyEvent, key(), 117
- QKeyEvent, state(), 117
- QKeyEvent, text(), 117
- QLabel, 88, 89
- QLabel, setAutoResize(), 98
- QLabel, setBuddy(), 114
- QLayout, 131, 137
- QLayout, addWidget(), 132
- QLayout, setAutoAdd(), 132
- QLayout, setMargin(), 137
- QLayout, setSpacing(), 137
- QLCDNumber, 111
- QLCDNumber, display(), 112
- QLCDNumber, intValue(), 117
- QLCDNumber, value(), 117
- QLineEdit, 88, 89, 166
- QLineEdit, pada QComboBox, 105
- QLineEdit, pada QTable, 160
- QLineEdit, textChanged(), 98, 111

- QListBox, insertItem(), 108
- QListBox, removeItem(), 108
- QListBox, selected(), 108
- QListBox, setSelectionMode(), 109
- QListBox, sort(), 108
- QMessageBox, 146, 149
- QMessageBox, warning(), 152
- QObject, 93
- QObject, killTimers(), 225
- QObject, startTimer(), 143
- QObject, timer, 143
- QObject, timerEvent(), 143
- QPainter, 168
- QPainter, drawLine(), 168
- QPainter, drawText(), 168
- QPainter, eraseRect(), 168
- QPainter, setPen(), 168
- QPen, 165, 168
- QPen, setColor(), 167
- QPushButton, 88, 89, 134
- QRadioButton, 101
- QRadioButton, setChecked(), 102
- QRect, height(), 168
- QRect, setBottom(), 168
- QRect, setHeight(), 168
- QRect, setLeft(), 168
- QRect, setRight(), 168
- QRect, setTop(), 168
- QRect, setWidth(), 168
- QRect, setX(), 168
- QRect, setY(), 168
- QRect, width(), 168
- QRect, x(), 168
- QRect, y(), 168
- QRegExp, 173
- QScrollView, 167
- QScrollView, repaintContents(), 167
- QSpinBox, 124
- QSpinBox, setValue(), 126
- QSpinBox, valueChanged(), 126
- QSqlCursor, 191
- QSqlCursor, delRecords(), 197
- QSqlCursor, editBuffer(), 197
- QSqlCursor, primeDelete(), 197
- QSqlCursor, primeInsert(), 197
- QSqlCursor, primeUpdate(), 197
- QSqlCursor, select(), 191
- QSqlCursor, update(), 197
- QSqlCursor, value(), 192
- QSqlDatabase, 184

- QSqlDatabase, addDatabase(), 184
- QSqlDatabase, lastError(), 184
- QSqlDatabase, setDatabaseName(), 184
- QSqlDatabase, setHostName(), 184
- QSqlDatabase, setPassword(), 184
- QSqlDatabase, setUsername(), 184
- QSqlError, 184
- QSqlError, databaseText(), 184
- QSqlField, clear(), 197
- QSqlQuery, 188, 191
- QSqlQuery, at(), 190
- QSqlQuery, execQuery(), 189
- QSqlQuery, first(), 190
- QSqlQuery, last(), 190
- QSqlQuery, next(), 189, 190
- QSqlQuery, prev(), 190
- QSqlQuery, seek(), 190
- QSqlQuery, value(), 189
- QSqlRecord, 191
- QSqlRecord, buffer QSqlCursor, 197
- QSqlRecord, field(), 197
- QString, 90
- QString, find(), 173
- QString, isEmpty(), 111
- Qt, 81
- Qt, dengan Python, 93
- Qt, ISODate, 225
- qt, modul, 90
- QTable, 155, 192
- QTable, __init__(), 160
- QTable, activateNextCell(), 160
- QTable, beginEdit(), 210
- QTable, cellWidget(), 166
- QTable, clearCell(), 161, 166
- QTable, createEditor(), 166, 210
- QTable, currentChanged(), 157
- QTable, currentColumn(), 161
- QTable, currentRow(), 161
- QTable, horizontalHeader(), 156
- QTable, insertRows(), 160, 166
- QTable, paintCell(), 161, 166
- QTable, removeRow(), 160, 166
- QTable, resizeData(), 165
- QTable, setCellContentFromEditor(), 166

- QTable, setColumnWidth(),
173
- QTable, setCurrentCell(), 161
- QTable, setHide(), 157
- QTable, setNumCols(), 156,
166
- QTable, setNumRows(), 156,
160
- QTable, setText(), 157
- QTable, text(), 157
- QTable, updateCell(), 166
- QTabWidget, 123, 127
- QTabWidget, insertTab(), 129
- QTextEdit, 88, 89, 131
- QTime, 139
- QTime, addSecs(), 139
- QTime, currentTime(), 139
- QTime, hour(), 139
- QTime, minute(), 139
- QTime, msec(), 139
- QTime, second(), 139
- QTime, secsTo(), 139
- QTime, toString(), 139
- qtsql, modul, 179
- qtable, modul, 155
- query, 179, 185
- QVariant, 189, 190
- QVariant, toDateTime(), 190
- QVariant, toDouble(), 190
- QVariant, toInt(), 190
- QVariant, toString(), 190
- QVBoxLayout, 131, 134
- QWidget, 83, 84, 87, 111,
123
- QWidget, closeEvent(), 152
- QWidget, focusOutEvent(),
224
- QWidget, font(), 97
- QWidget, keyPressEvent(),
117
- QWidget, setEnabled(), 111
- QWidget, setFocus(), 93
- QWidget, setMinimumHeight(),
132, 134
- QWidget, show(), 84
- QWidget, showMaximized(),
134
- QWidget, warna, 99
- radiobutton, pedoman, 104
- radiobutton, QRadioButton,
101
- random(), random, 61
- random, modul, 61
- range(), 60
- raw_input(), 51
- read, file, 69
- read-only, QLabel, 88

- read-only, QListBox, 110
- readlines(), 69, 75, 76
- ready, status printer, 70
- record, list dua dimensi, 47
- red, 98
- reimplementation, 94, 160
- reject(), QDialog, 145
- rekursif, 65, 127
- removeItem(), QComboBox, 189
- removeItem(), QListBox, 108
- removeRow(), QTable, 160, 166
- rename(), modul os, 93
- repaintContents(), QScrollView, 167
- reserved word pada text editor, 30
- reset, istilah, 115
- reset, QButtonGroup, 102
- reset, QRadioButton, 101
- resize(), QWidget, 88
- resizeData(), QTable, 165
- result(), QDialog, 145
- return, 64
- Return, keyboard, 115
- reverse(), list, 47
- RGB, Red Green Blue, 99
- ribu(), 162
- ribuan, 40
- rjust(), string, 78
- root, 70
- root, superuser Linux, 118
- root, superuser MySQL, 180
- round(), 40
- runtime, 23, 104
- script, arti, 24
- scrolling, 106
- second(), QTime, 139
- secsTo(), QDateTime, 143
- secsTo(), QTime, 139
- seek(), file, 76
- seek(), QSqlQuery, 190
- segmentation fault, 90
- select(), QSqlCursor, 191
- SELECT, SQL, 185
- selected(), QButtonGroup, 102
- selected(), QListBox, 108
- self, 85
- setAutoAdd(), QLayout, 132
- setAutoCompletion(), QComboBox, 106
- setAutoResize(), QLabel, 98
- setBold(), QFont, 98
- setBottom(), QRect, 168
- setBuddy(), QLabel, 114

- setButton(), QButtonGroup, 104
- setCaption(), 85
- setCellContentFromEditor(), QTable, 166
- setChecked(), QRadioButton, 102
- setColor(), QBrush, 167
- setColor(), QPen, 167
- setColumnWidth(), QTable, 173
- setCurrentCell(), QTable, 161
- setDatabaseName(), QSqlDatabase, 184
- setDefault(), QPushButton, 145
- setDirection(), QBoxLayout, 136
- setEditable(), QComboBox, 106
- setEnabled(), QWidget, 111
- setFamily(), QFont, 97
- setFocus(), QWidget, 93
- setFont(), QWidget, 97
- setGeometry(), QWidget, 88
- setHeight(), QRect, 168
- setHide(), QTable, 157
- setHostName(), QSqlDatabase, 184
- setItalic(), QFont, 98
- setLabel(), QHeader, 156
- setLeft(), QRect, 168
- setMainWidget(), QApplication, 83
- setMargin(), QLayout, 137
- setMinimumHeight(), QWidget, 132, 134
- setNumCols(), QTable, 156, 166
- setNumRows(), QTable, 156, 160
- setPaletteBackgroundColor(), QWidget, 99
- setPaletteForegroundColor(), QWidget, 99
- setPassword(), QSqlDatabase, 184
- setPen(), QPainter, 168
- setPointSize(), QFont, 97
- setRight(), QRect, 168
- setSelectionMode(), QListBox, 109
- setSpacing(), QLayout, 137
- setSqlCursor(), QDataTable, 193
- setStyle(), QBrush, 167
- setText(), 89
- setText(), QTable, 157

- setTop(), QRect, 168
- setUnderline(), QFont, 98
- setUsername(), QSqlDatabase, 184
- setValue(), QDial, 126
- setValue(), QSpinBox, 126
- setWidth(), QRect, 168
- setX(), QRect, 168
- setY(), QRect, 168
- shape, bidang gambar, 168
- sheet, 155
- shell tools, 22
- Shift, keyboard, 117
- show(), QWidget, 84
- showMaximized(), QWidget, 134
- showmodal, 146
- SIGNAL(), qt, 93
- siklus pengembangan, 18
- sinyal, 91, 94
- sistem kepegawaian, 100
- sistem KTP, 100
- sistem operasi, 27
- situs, 21
- slice, pemenggalan list, 45
- slot, 93
- sort(), list, 47
- sort(), QListBox, 108
- spinbox, 124
- splitfields(), string, 78
- spreadsheet, 76, 155, 161
- SQL, 18, 185
- startTimer(), QObject, 143
- state(), QKeyEvent, 117
- storage, 24
- str(), 42
- str(), QString menjadi string, 91
- string, modul, 61
- strip(), string, 78
- su, 70
- SUM(), SQL, 226
- superuser database, 180
- superuser sistem operasi, 118
- Sybase, 179
- syntax error, 73
- sys, modul, 71, 72
- system(), os, 72, 93

- tabel, 155
- tabel, dari file, 76
- tabel, list dua dimensi, 47, 60
- TableFile, file menjadi tabel, 170
- tanggal(), string ke QTime, 199
- Tanggal, class, 141

- tanggal, `QDate`, 140
- `Tcl`, 22
- teks, file teks, 76
- teks, modus teks, 76
- teks, pengolahan, 22
- `teksBerubah`, event `LineCase`, 96
- `teksBerubah`, `LineCase`, 95
- `templatel`, 180
- text editor, 29
- `text()`, 89
- `text()`, `QComboBox`, 106
- `text()`, `QKeyEvent`, 117
- `text()`, `QTable`, 157
- text, CSV, 76
- `TEXT`, SQL, 185
- `textChanged()`, `QLineEdit`, 95, 98, 111
- `time()`, `QDateTime`, 142
- time, modul, 61
- timer, 143
- `timerEvent()`, `QObject`, 143
- `toDateTime()`, `QVariant`, 190
- `toDouble()`, `QVariant`, 190
- toggle, `QCheckBox`, 99
- `toInt()`, `QVariant`, 190
- `toString()`, `QDate`, 141
- `toString()`, `QDateTime`, 121, 225
- `toString()`, `QTime`, 139
- `toString()`, `QVariant`, 190
- touch, 93
- true, 52, 55, 56
- try, 55, 74
- `type()`, info tipe data, 50
- ukuran huruf, 71
- underline, font, 96
- `UNIQUE`, SQL, 210
- Unix, 21
- `update()`, dictionary, 49
- `update()`, `QSqlCursor`, 197
- `UPDATE`, SQL, 185
- `updateCell()`, `QTable`, 166
- user, 88
- `value()`, `QLCDNumber`, 117
- `value()`, `QSqlCursor`, 192
- `value()`, `QSqlQuery`, 189
- `valueChanged()`, `QSpinBox`, 126
- `ValueGrid`, class, 161, 224
- `values()`, dictionary, 49
- values, dictionary, 48
- `VARCHAR`, SQL, 185
- variabel, 37
- variabel, milik fungsi, 65
- variant, `QVariant`, 190

vi, 29
virtual, 186
visual object, 87

wadah, 123
waktu(), string ke QDateTime, 199
WaktuDigital, class, 143, 219
warna, QColor, 98
warning(), QMessageBox, 152
while, 60, 73, 85
white, 98
widget, istilah, 87
width(), QRect, 168
window manager, 81
Windows, 21
write(), file, 121
write, file, 69

x(), QRect, 168

y(), QRect, 168
year(), QDateTime, 140
yellow, 98

ZeroDivisionError, 73
zfill(), string, 76