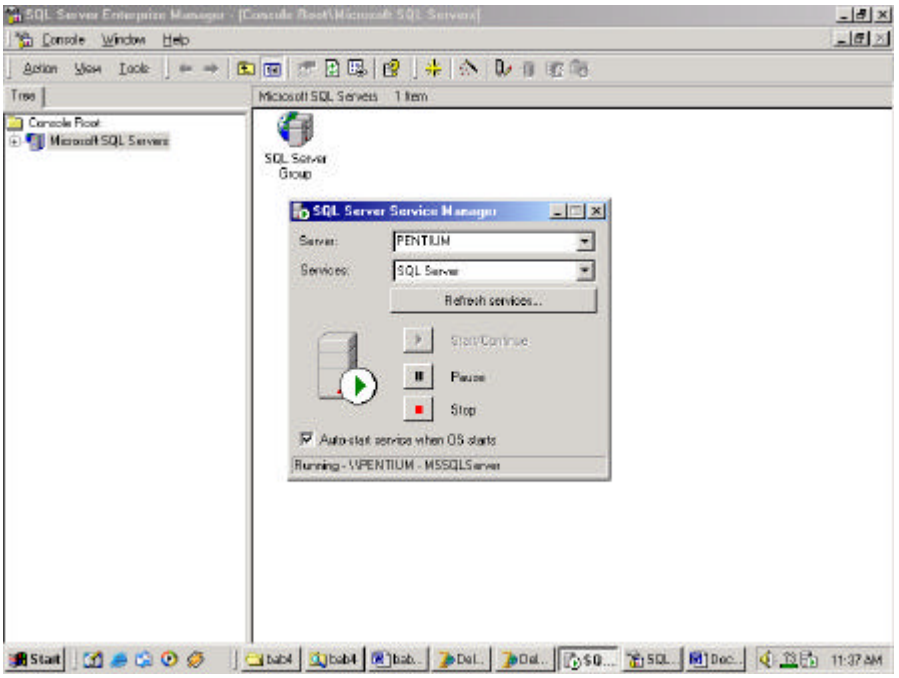
4. IMPLEMENTASI SISTEM

Setelah dalam dua sub bab sebelumnya yang membahas desain arsitektur dan *interface* program secara garis besar, maka pada bagian ini akan menguraikan barisan proses *coding* program.

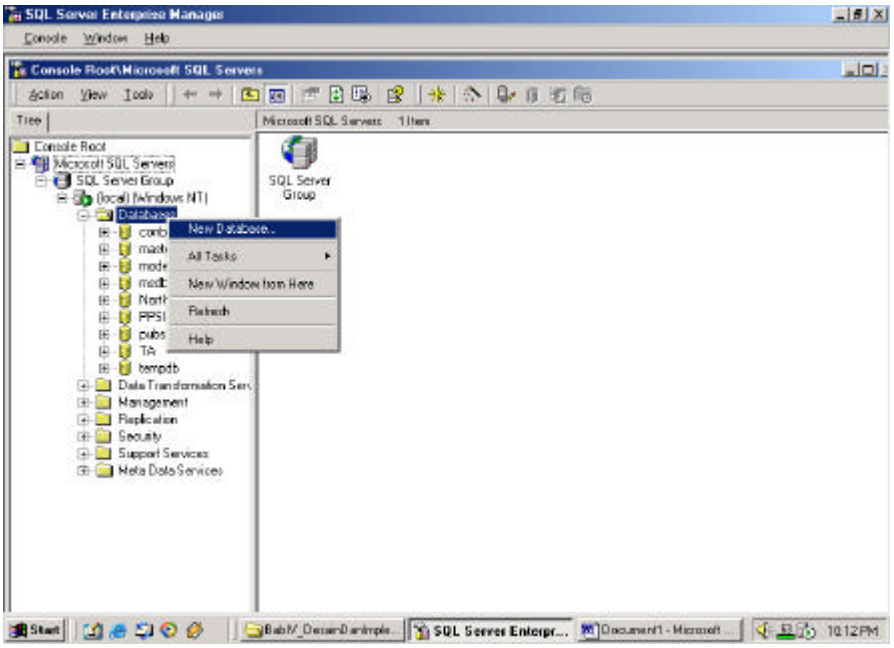
4.1 Koneksi pada SQL Server 2000



Gambar 4.1. Pembuatan Koneksi SQL Server

Untuk membangun sebuah koneksi, dilakukan dengan memilih *Microsoft SQL Server>Servise Manager*. Di dalamnya diinputkan nama komputer *server* dan *service* yang digunakan, misalnya *SQL Server*, setelah itu koneksi dijalankan dengan menekan *button start*.

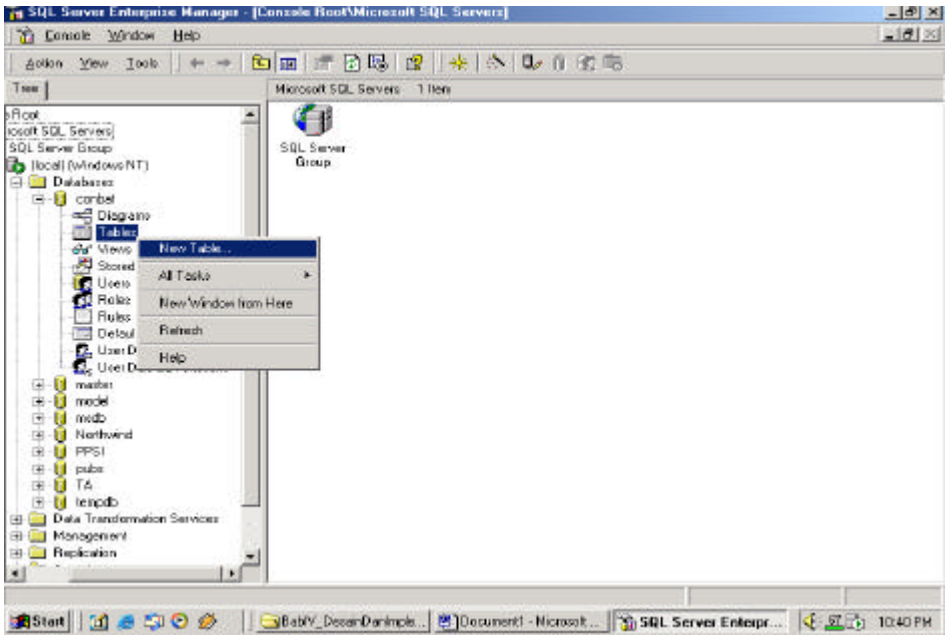
4.2. Pembuatan Database pada SQL Server 2000



Gambar 4.2. Pembuatan Database Conbat

Untuk membangun sebuah *database* baru, maka dapat di-*click* kanan pada folder Database di *SQL Server Enterprise Manager* dengan memilih *new database*. Selain itu, perlu juga dibuat *login* untuk akses database tersebut dengan memilih folder *security*, pilih *Login click* kanan dipilih *new login*. Pada proses login ini kita inputkan nama yang akan mengakses, misalnya *Administrator* dan nama *database* yang akan diakses, misalnya conbat. Di dalam *database* ini memiliki beberapa fasilitas diantaranya pembuatan tabel, *store procedure*, diagram dan lain-lain.

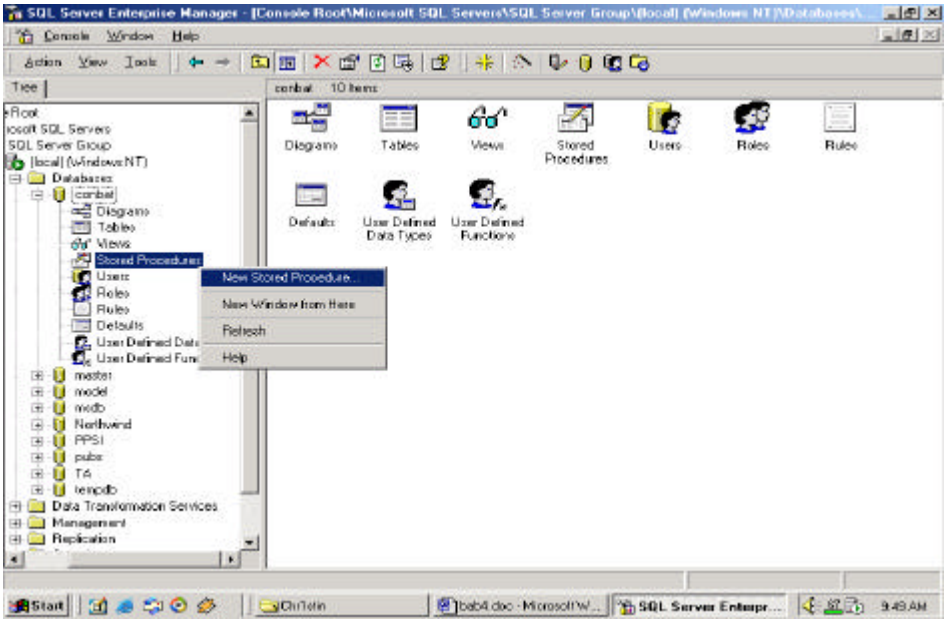
4.3. Pembuatan Tabel pada SQL Server 2000



Gambar 4.3. Pembuatan tabel pada SQL Sever

Untuk membuat sebuah tabel baru, pilih *tables* pada database kemudian di-click kanan *new table*. Selanjutnya untuk meng-edit attribut-attribut pada suatu tabel digunakan fasilitas *design table*. Dan untuk melihat isi dari tabel digunakan fasilitas *open table*.

4.4. Pembuatan Store Procedure pada SQL Server 2000



Gambar 4.4. Pembuatan Store Procedure

Di dalam Microsoft SQL Server 2000 terdapat fasilitas pembuatan *store procedure* dan *function* selain sebagai tempat penyimpanan tabel. Keuntungan yang diperoleh akses *database* lebih cepat dan program yang dibuat lebih singkat/pendek. Untuk membuat *store procedure* baru, pilih *stored procedured* kemudian di-*click* kanan *new store procedure*.

4.5. Proses Koneksi Database

Komponen *ADO* merupakan salah satu fasilitas yang dimiliki oleh *Borland Delphi* 7 untuk melakukan kontak atau hubungan dengan program-program aplikasi yang berfungsi sebagai penyimpanan *database*, seperti *Microsoft Access* maupun *Microsoft SQL Server* 2000. Dalam program ini, setiap *form* master, transakasi dan laporan berdiri sebagai satu *project* sendiri dan dalam setiap *project* dibangun satu koneksi dan *store procedure*. Berikut contoh koneksi database pada *form* master *customer*:

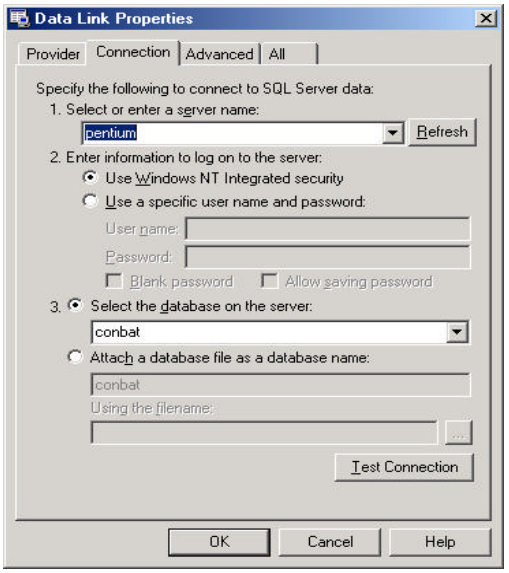
- ADOConnection (ADODB)
  - connection string

    provider=Microsoft OLE DB provider for SQL Server

    connection=server name : pentium

    database on server : conbat

    Provider=Microsoft OLE DB Provider for SQL Server
  - name=AdoCon
- ADOStoreProc (ADODB)
  - connection=AdoCon
  - name=spcustomer



Gambar 4.5. Koneksi Database

4.6. Proses Pembuatan Store Procedure

Beberapa pengaksesan data dapat dilakukan dengan memanggil *procedure-procedure* yang dibuat di *store procedure* di SQL Server. Di dalam program ini terdapat 37 *store procedure*, antara lain:

Tabel 4.1. Store Procedure

Master	Customer	SpInsertCustomer
		SpUpdateCustomer
		SpDeleteCustomer
	Supplier	SpInsertSuplier
		SpUpdateSuplier
		SpDeleteSuplier
	Bahan Baku	SpInsertBahanBaku

		SpUpdateBahanBaku
		SpDeleteBahanBaku
	Barang	SpInsertBarang
		SpUpdateBarang
		SpDeleteBarang
Transaksi	Penjualan	SpInsertHeaderPenjualan
		SpInsertDetailPenjualan
		SpDeletePenjualan
		SpDeleteDetailPenjualan
	Pembelian	SpInsertHeaderSPBB
		SpInsertDetailSPBB
		SpDeleteSPBB
		SpDeleteDetailSPBB
	Pemesanan	SpInsertHeaderPemesanan
		SpInsertDetailPemesanan
		SpDeletePemesanan
		SpDeleteDetailPemesanan
	Produksi	SpInsertProduksi
		SpUpdateProduksi
		SpDeleteProduksi
	Pengambilan	SpInsertPengambilan
		SpUpdatePengambilan
		SpDeletePengambilan
	Retur	SpInsertHeaderRetur
		SpInsertDetailRetur
		SpDeleteRetur
		SpDeleteDetailRetur
Login		SpInsertUser
		SpUpdateUser
		SpDeleteUser

Berikut beberapa contoh program *store procedure*:

Segmen program 4.1. Store Procedure Insert Data Customer

```
CREATE Procedure spInsertCustomer
  @nama varchar (20),
  @alamat varchar (30),
  ….
As
  Declare @kdCust varchar (5)
  Declare @first2 varchar (2)
  Declare @jumlah int
  -- Auto Generate Kode Customer
  set @first2 = upper (left (@nama, 2))
```

```
  set @jumlah = (select count (kdCust) from tbCustomer where kdCust like @first2 +
"%")
 if (@jumlah = 0)
 begin
   set @kdCust = @first2 + "001"
 end
 else
 begin
   set @kdCust = (select max (kdCust) from tbCustomer where kdCust like @first2+"%")
   set @jumlah = cast (right (@kdCust, 3) as int) + 1
   if @jumlah < 10
     set @kdCust = @first2 + "00" + cast (@jumlah as varchar)
   else if @jumlah < 100
     set @kdCust = @first2 + "0" + cast (@jumlah as varchar)
   else
     set @kdCust = @first2 + cast (@jumlah as varchar)
 end
  insert into tbCustomer values (@kdCust, @nama, @alamat, @kota, @negara, @telp,
@kdPos, @email, @ket)
 return 0
GO
```

Prosedur ini digunakan untuk membuat kode *customer* secara otomatis diawali dengan 2 huruf pertama dari nama *customer*. Pemakaian prosedur ini berfungsi agar kode *customer* seragam dan sesuai urutan.

Segmen program 4.2. Store Procedure Update Data Customer

```
CREATE Procedure spUpdateCustomer
 @kdCust varchar (5),
 @nama varchar (20),
 ….
As
 set @kdCust = upper (@kdCust)
 update tbCustomer set
   nama = @nama,
   ….
 where kdCust = @kdCust
GO
```

Prosedur ini digunakan untuk melakukan *update* data *customer* tanpa mengganti kode *customer*.

Segmen program 4.3. Store Procedure Delete Data Customer

```
CREATE Procedure spDeleteCustomer
  @kdCust varchar (5)
As
  declare @jumlah int
  set @jumlah = (select count (kdCust) from tbHeaderPemesanan where kdCust = @kdCust)
  if @jumlah = 0
  begin
    set @jumlah = (select count (kdCust) from tbHeaderPenjualan where kdCust = @kdCust)
    if @jumlah = 0
    begin
      delete from tbCustomer where kdCust = @kdCust
      return 1
    end
    else
      return 0
  end
  else
    return 0
GO
```

Prosedur ini digunakan untuk melakukan *delete* data *customer* dengan syarat bahwa *customer* tersebut belum pernah melakukan transaksi pemesanan atau penjualan atau dengan kata lain belum terkait dengan tabel lain.

Segmen program 4.4. Store Procedure Insert Detail Penjualan

```
CREATE PROCEDURE spInsertDetailPenjualan
  @kdPenj varchar (6),
  @kdBarang varchar (5),
  @jumlah numeric (9)
AS
  declare @harga numeric (9)

  --update stock barang
     update tbBarang
     set jumlah = jumlah - @jumlah
     where kdBarang = @kdBarang

   -- Cari harga barang dari tabel barang
   set @harga = (select harga from tbBarang where kdBarang = @kdBarang)
   insert into tbDetailPenjualan values (@kdPenj, @kdBarang, @jumlah, @harga)
   return 0
GO
```

Prosedur ini berfungsi untuk mengupdate stok dengan mengurangi stok barang ketika terjadi penjualan. Selain itu prosedur ini juga akan mencari harga dari barang tersebut kemudian diinputkan ke dalam detail penjualan.

Segmen program 4.5. Store Procedure Delete Penjualan

```
CREATE PROCEDURE spDeletePenjualan
  @kdPenj varchar(6)
AS
  declare @kdBarang varchar(5)
  declare @jumlah numeric(9)
  declare cur cursor for
    select kdBarang, jumlah
    from tbDetailPenjualan
    where kdPenj = @kdPenj
    for read only

  open cur
  fetch next
    from cur
    into @kdBarang, @jumlah

  while (@@fetch_status = 0)
  begin
    --update stock barang
    update tbBarang
    set jumlah = jumlah + @jumlah
    where kdBarang = @kdBarang

    fetch next
      from cur
      into @kdBarang, @jumlah
  end
  close cur
  deallocate cur

  delete from tbDetailPenjualan where kdPenj = @kdPenj
  delete from tbHeaderPenjualan where kdPenj = @kdPenj
GO
```

Prosedur ini berfungsi untuk menghapus data penjualan yang termasuk di dalamnya detail penjualan. Variabel *cur* berfungsi untuk menampung semua barang yang ada di dalam detail penjualan. Ketika data barang dihapus maka stok dikembalikan lagi ke tabel

barang. Kemudian data dihapus dimulai dengan detail penjualan dahulu baru kemudian *header* penjualan.

Untuk menghapus data detail penjualan saja, prosedurnya sama dengan mengembalikan stok ke tabel barang, baru kemudian data detail penjualan dihapus.

4.7. Proses Pembuatan Procedure dan function

Pengaksesan data di delphi dilakukan dengan memanggil *procedure-procedure/function-function* yang dibuat di setiap unit master, transaksi dan laporan. *Procedure* dan *function* yang ada antara lain:

Tabel 4.2. Procedure dan Function

Master • Customer • Supplier • Bahan Baku • Barang	procedure btnSimpanClick(Sender: TObject);
	procedure sbtnTambahClick(Sender: TObject);
	procedure sbtnUbahClick(Sender: TObject);
	procedure sbtnHapusClick(Sender: TObject);
	procedure btnBatalClick(Sender: TObject);
	procedure FormShow(Sender: Tobject);
	procedure btnKdCustClick(Sender: TObject);
	procedure clearFields;
Transaksi • Pembelian • Pemesanan • Penjualan • Retur	procedure stgDetailKeyDown(Sender: TObject; var Key: Word; Shift: TShiftState);
	procedure sbtnTambahClick(Sender: TObject);
	procedure sbtnUbahClick(Sender: TObject);
	procedure sbtnHapusClick(Sender: TObject);
	procedure FormShow(Sender: Tobject);
	procedure btnKdCustClick(Sender: TObject);
	procedure edtKdCustExit(Sender: TObject);
	procedure stgDetailSelectCell(Sender: TObject; ACol, ARow: Integer; var CanSelect: Boolean);
	procedure btnSimpanClick(Sender: TObject);
	procedure btnKdPenjClick(Sender: TObject);
	procedure btnBatalClick(Sender: TObject);
	procedure clearFields;
	procedure columnChanged (fromIndex, toIndex : integer);
	procedure countTotal();
Transaksi • Produksi • Pengambilan	procedure sbtnTambahClick(Sender: TObject);
	procedure sbtnUbahClick(Sender: TObject);
	procedure sbtnHapusClick(Sender: TObject);

	Procedure upJumlahChanging(Sender: TObject; var AllowChange: Boolean);
	procedure btnKdBBClick(Sender: TObject);
	procedure FormShow(Sender: Tobject);
	procedure btnSimpanClick(Sender: TObject);
	procedure btnKdAmbilClick(Sender: TObject);
	procedure BtnClearClick(Sender: TObject);
	procedure btnBatalClick(Sender: TObject);
	procedure clearFields;
	procedure changeButton;
Laporan (10 laporan)	procedure FormShow(Sender: Tobject);
	procedure FormCreate(Sender: Tobject);
Login	procedure btnSimpanClick(Sender: TObject);
	procedure sbtnTambahClick(Sender: TObject);
	procedure sbtnUbahClick(Sender: TObject);
	procedure sbtnHapusClick(Sender: TObject);
	procedure btnBatalClick(Sender: TObject);
	procedure FormShow(Sender: Tobject);
	procedure btnKdUserClick(Sender: TObject);
	procedure clearFields;
	function check (index : byte) : String;
ULibConbat	procedure showError (const value : string);
	procedure convert (bil : integer; var hasil : string);
	function terbilang (bil : integer) : string;

4.8. Proses Pengaksesan Database

Setelah melakukan koneksi dengan *database*, maka tahap selanjutnya adalah mengakases *database* yang telah terhubung tersebut. Model dari pengaksesan *database* dapat berupa memasukkan/tambah data, mencari data, mengubah data, dan menghapus data. Dalam program ini, setiap *form* master dan transaksi terdapat fasilitas pengaksesan tersebut. Berikut contoh program Tambah, Ubah dan Hapus data *customer*:

Segmen Program 4.6. Prosedur Tambah Data Customer

```
procedure TfrmMstCust.btnSimpanClick(Sender: TObject);
begin
  if sbtnTambah.down then
  begin
    if (edtNama.Text <> '') and (edtAlamat.text <> '') and (edtKota.Text <> '') and
      (cboNegara.itemIndex > -1) then
    begin
```

```
      if MessageDlg('Apakah data akan disimpan ?', mtConfirmation, mbOKCancel, 0) =
mrOK then
     begin
      adoCon.BeginTrans;
      try
       spCustomer.close;
       spCustomer.ProcedureName := 'spInsertCustomer';
       spCustomer.Parameters.Clear;
       with spCustomer.parameters do
       begin
         CreateParameter('nama', ftString, pdInput, 20, edtNama.text);
         ….
         ….
       end;
       spCustomer.ExecProc;
       adoCon.CommitTrans;
      except
       adoCon.RollbackTrans;
      end;
      clearFields;
     end;
```

Prosedur ini dijalankan saat penekanan *speedbutton* tambah dan *button* simpan, disini *user* diwajibkan untuk mengisis semua *field* yang wajib diisi, jika tidak akan ditampilkan pesan error. Selanjutnya ditampilkan pesan konfirmasi “apakah data akan disimpan?”, jika ditekan Ok maka akan dijalankan koneksi AdoCon dan *store procedure* spcustomer untuk menjalankan *store procedure* spInsertCustomer di SQL Server. Setelah data disimpan, dijalankan prosedur *clearfields* untuk membersihkan layar.

Segmen Program 4.7. Prosedur Ubah Data Customer

```
else if sbtnUbah.down then
  begin
    if (edtNama.Text <> '') and (edtAlamat.text <> '') and (edtKota.Text <> '') and
      (cboNegara.itemIndex > -1) then
    begin
       if MessageDlg('Apakah data akan diubah ?', mtConfirmation, mbOKCancel, 0) =
mrOK then
     begin
      adoCon.BeginTrans;
      try
       spCustomer.close;
       spCustomer.ProcedureName := 'spUpdateCustomer';
       spCustomer.Parameters.clear;
```

```
        with spCustomer.Parameters do
        begin
          CreateParameter('kdCust', ftString, pdInput, 5, edtKdCust.text);
          ….
          ….
        end;
        spCustomer.ExecProc;
        adoCon.CommitTrans;
      except
        adoCon.RollbackTrans;
      end;
      clearFields;
    end;
```

Prosedur ini dijalankan saat penekanan *speedbutton* tambah dan *button* ubah, disini *user* diwajibkan untuk mengisis semua *field* yang wajib diisi, jika tidak akan ditampilkan pesan error. Selanjutnya ditampilkan pesan konfirmasi “apakah data akan diubah?”, jika ditekan Ok maka akan dijalankan koneksi AdoCon dan *store procedure* spcustomer untuk menjalankan *store procedure* spUpdateCustomer di SQL Server. Setelah data disimpan, dijalankan prosedur *clearfields* untuk membersihkan layar.

Segmen Program 4.8. Procedure Hapus Data Customer

```
else if sbtnHapus.down then
  begin
     if MessageDlg('Apakah data akan dihapus ?', mtConfirmation, mbOKCancel, 0) =
mrOK then
    begin
      adoCon.BeginTrans;
      try
        spCustomer.close;
        spCustomer.ProcedureName := 'spDeleteCustomer';
        spCustomer.Parameters.Clear;
        spCustomer.Parameters.CreateParameter('kdCust', ftString, pdInput, 5,
edtKdCust.text);
        spCustomer.ExecProc;
        adoCon.CommitTrans;
      except
        adoCon.RollbackTrans;
      end;
      clearFields;
    end;
```

Prosedur ini dijalankan saat penekanan *speedbutton* tambah dan *button* simpan, disini *user* diwajibkan untuk mengisis semua *field* yang wajib diisi, jika tidak akan ditampilkan

pesan error. Selanjutnya ditampilkan pesan konfirmasi “apakah data akan hapus?”, jika ditekan Ok maka akan dijalankan koneksi AdoCon dan *store procedure* spcustomer untuk menjalankan *store procedure* spDeleteCustomer di SQL Server. Setelah data disimpan, dijalankan prosedur *clearfields* untuk membersihkan layar.

Untuk proses pencarian data digunakan komponen khusus Qsoft: DBBrowser. Salah satu fasilitas yang terdapat pada komponen ini adalah *browsing* data. Hal ini sangat berguna karena *user* tidak mingkin mengingat kode yang ada. Selain itu user juga tidak perlu menginputkan data satu persatu secara manual. Berikut contoh pemakaian komponen DBBrowser untuk mencari data *customer*:

- Connection=AdoCon
- Name=dbbKdCust
- Table name=tbcustomer

Segmen Program 4.9. Prosedur Cari Data Customer

```
procedure TfrmMstCust.btnKdCustClick(Sender: TObject);
var
  list : TStringList;
begin
  if (DbbKdCust.Execute) then
  begin
    list := DbbKdCust.Hasil.AsList;
    edtKdCust.text := list[0];
    edtNama.text := list[1];
    edtAlamat.text := list[2];
    edtKota.text := list[3];
    cboNegara.ItemIndex := 0;
     while UpperCase (cboNegara.Items [cboNegara.ItemIndex]) <> UpperCase (list [4])
do cboNegara.itemIndex := cboNegara.itemIndex + 1;
    edtTelepon.text := list[5];
    edtKdPos.text := list[6];
    edtEmail.text := list[7];
    mmoKet.text := list[8];
  end;
end;
```

Prosedur ini dijalankan saat penekanan *button* kode *customer*, disini akan ditampilkan *form* Browse Database yang berisi data-data *customer*. Setelah *user* memilih salah satu *customer*, maka data-datanya akan ditampilkan ke *form* master *customer* dan *form* tersebut ditutup.

4.9. Proses Penampilan Form

Program yang dibuat dengan bahasa pemrograman *Borland Delphi* 7, memiliki kelebihan di mana dapat melibatkan penggunaan beberapa *form*, sehingga antara *form* yang satu dan yang lainnya meskipun terpisah tetap terintegrasi ke dalam satu modul program.

Contoh program untuk menampilkan *form*:

Segmen Program 4.10. Penampilan Form Master Customer

```
procedure TFormCustomer.Click(Sender: TObject);
begin
  formPenjualan.Visible:=true;
  …..
  …..
end;
```

4.10. Proses Penampilan Pesan Error

Dalam program ini, untuk menampilkan pesan-pesan error disimpan suatu unit khusus UlibConbat dengan tujuan menyimpan prosedur-prosedur yang sering

digunakan. Dalam pemakaiannya pesan error dinyatakan dalam bentuk konstanta. Berikut unit UlibConbat yang menjalankan prosedur untuk menampilkan pesan error:

Segmen Program 4.11. Penampilan Pesan Error

```
unit uLibConbat;

interface
uses
  Dialogs;
procedure showError (const value : string);
implementation
  procedure showError (const value : string);
  begin
    MessageDlg(value, mtError, [mbOk], 0);
  end;
end.
```

4.11. Proses Penghitungan Sub Total

Dalam melakukan perhitungan sub total, tiap kali diinputkan satu item barang yang dibeli maka secara otomatis sub total akan ditampilkan, begitu pula jika ada item barang yang dihapus, maka sub total otomatis langsung ter-*update.* Dalam program ini, proses perhitungan sub total dilakukan pada transaksi pemesanan, pembelian, penjualan dan retur. Berikut contoh program penghitungan sub total penjualan:

Segmen Program 4.12. Penghitungan Sub Total Penjualan

```
procedure TfrmPenjualan.countTotal();
var
  i : integer;
  total : integer;
begin
  total := 0;
  for i := 1 to stgDetail.RowCount-1 do
  begin
    if (stgDetail.Cells[_subtotal, i] <> '') then
      total := total + strtoint(stgDetail.cells[_subtotal, i]);
  end;
  edtTotal.Text := intToStr (total);
  lblTerbilangKata.Caption := terbilang(total);
end;
```

4.12. Proses Menampilkan Total dalam bentuk Kata/Terbilang

Agar lebih akurat, dibuat sebuah prosedur konversi untuk merubah *numeric* sub total menjadi kata/terbilang. Kemudian untuk menampilkan terbilang kata ini dibuat suatu fungsi terbilang. Prosedur dan fungsi ini disimpan di dalam unit UlibConbat. Berikut unit UlibConbat yang menjalankan prosedur konversi dan fungsi terbilang:

Segmen Program 4.13. Prosedur konversi angka menjadi kata

```
unit uLibConbat;
interface
uses
  Dialogs;
const
  angka : array [-1..10] of string =
    ('se', '', 'satu', 'dua', 'tiga',
     'empat', 'lima', 'enam', 'tujuh', 'delapan', 'sembilan', 'sepuluh');
procedure convert (bil : integer; var hasil : string);
function terbilang (bil : integer) : string;

implementation

  procedure convert (bil : integer; var hasil : string);
  var
    temp : integer;
  begin
    if bil <= 10 then hasil := hasil + angka [bil]
    else if bil < 20 then
    begin
      temp := bil mod 10;
      if temp = 1 then hasil := hasil + angka [-1] + 'belas'
      else hasil := hasil + angka [temp] + ' belas';
    end
    else if bil < 100 then
    begin
      temp := bil div 10;
      hasil := hasil + angka [temp] + ' puluh ';
      convert (bil mod 10, hasil);
    end
    else if bil < 1000 then
    begin
      temp := bil div 100;
      if temp = 1 then hasil := hasil + angka [-1] + 'ratus '
```

```
    else hasil := hasil + angka [temp] + ' ratus ';
    convert (bil mod 100, hasil);
  end
  else if bil < 10000 then
  begin
    temp := bil div 1000;
    if temp = 1 then hasil := hasil + angka [-1] + 'ribu '
    else hasil := hasil + angka [temp] + ' ribu ';
    convert (bil mod 1000, hasil);
  end
  else if bil < 100000 then
  begin
    convert (bil div 1000, hasil);
    hasil := hasil + ' ribu ';
    convert (bil mod 1000, hasil);
  end
  else if bil < 1000000 then
  begin
    convert (bil div 1000, hasil);
    hasil := hasil + ' ribu ';
    convert (bil mod 1000, hasil);
  end
  else if bil < 10000000 then
  begin
    temp := bil div 1000000;
    hasil := hasil + angka [temp] + ' juta ';
    convert (bil mod 1000000, hasil);
  end
  else if bil < 100000000 then
  begin
    convert (bil div 1000000, hasil);
    hasil := hasil + ' juta ';
    convert (bil mod 1000000, hasil);
  end
  else if bil < 1000000000 then
  begin
    convert (bil div 1000000, hasil);
    hasil := hasil + ' juta ';
    convert (bil mod 1000000, hasil);
  end;
end;
```

Segmen program 4.14. Funsi untuk menampilkan terbilang kata

```
function terbilang (bil : integer) : string;

 var
   hasil : string;
 begin
```

```
    hasil := '';
    if bil > 0 then convert (bil, hasil)
    else if bil = 0 then hasil := 'nol';
    terbilang := hasil;
  end;
end.
```

4.13. Proses Memasukkan Detail dalam Form Transaksi

Detail barang/bahan baku yang dibeli ditampilkan dalam sebuah StringGrid yang terdiri dari kolom nomer, kode barang, nama barang, jumlah, harga dan total. Untuk pengisian kode barang/bahan baku dalam detail dapat dilakukan dengan 2 cara:

- Pencarian secara otomatis dengan fasilitas dbbrowser

  Cara mencari kode barang/bahan baku dilakukan dengan menekan F1 pada kolom kode.

- Pengisian secara manual

  Dilakukan dengan mengisikan langsung ke kolom kode.

Setelah memilih kode barang, secara otomatis akan ditampilkan kode, nama dan harga barang. Untuk menginputkan jumlah barang dilakukan dengan menekan tombol tab/enter. Setelah mengiputkan jumlah otomatis akan dihitung totalnya. Sedangkan untuk menghapus barang dalam detail, ditekan Ctrl+Del. Berikut contoh program memasukkan detail barang dalam transaksi penjualan:

Segmen Program 4.15. Penginputan Detail Barang dalam Form Penjualan

```
procedure TfrmPenjualan.columnChanged (fromIndex, toIndex : integer);
var
  i : byte;
  err : integer;
  retVal : integer;
  totalStock : integer;
begin
  if fromIndex = _kdBrg then
  begin
    tbTemp.close;
    tbTemp.TableName := 'tbBarang';
    tbTemp.open;
```

```
   if tbTemp.Locate('kdBarang', stgDetail.Cells [stgDetail.col, stgDetail.row],
[locaseInSensitive]) then
   begin
    with stgDetail do
    begin
      cells [_brg, Row] := tbTemp.fields [1].asString;
      cells [_harga, Row] := tbTemp.fields [3].asString;
      cells [_stock, Row] := tbTemp.fields[2].AsString;
      if (cells[_updstock, row] = '') then cells [_updstock, Row] := '0';
    end;
    stgDetail.col := _jumlah   //kolom aktif dipindah ke jml
   end
   else
   begin
    showError(_errKdBarang);
    for i := 2 to stgDetail.colCount-1 do
    begin
      stgDetail.cells [i, stgDetail.Row] := '';
    end;
   end;
  end;
  if fromIndex = _jumlah then
  begin
   if stgDetail.cells [_kdBrg, stgDetail.row] <> '' then
   begin
    val (stgDetail.cells [_jumlah, stgDetail.row], retVal, err);
    if err = 0 then
    begin
      totalStock := strtoint(stgDetail.cells[_stock, stgDetail.row]) +
strtoint(stgDetail.cells[_updstock, stgDetail.row]);
      if (strtoint(stgDetail.cells[_jumlah, stgDetail.row]) > totalstock) then
      begin
       showMessage('Stock tidak Mencukupi. Stock hanya tersisa ' +
inttostr(totalstock) + ' buah');
        stgDetail.cells[_jumlah, stgDetail.row] := inttostr(totalstock);
      end;
      stgDetail.Cells [_subTotal, stgDetail.row] := intToStr (strToInt
(stgDetail.cells [_jumlah, stgDetail.row]) * strToInt (stgDetail.Cells [_harga,
stgDetail.row]));
      stgDetail.col := _kdBrg;
      countTotal();
    end
    else showError (_errJumlah);
   end
   else stgDetail.col := _kdBrg;
  end;
end;
```

Segmen Program 4.16. Aturan Penekanan Tombol dalam Mengisi Detail Barang

```
procedure TfrmPenjualan.stgDetailKeyDown(Sender: TObject; var Key: Word;
  Shift: TShiftState);
var
  list : tStringList;
  i, j : integer;
begin
  if stgDetail.Col in [_kdBrg, _jumlah] then
  begin
    if ssCtrl in shift then
    begin
      case key of
        78 : // ctrl N
        begin
          with stgDetail do
          begin
              if (cells [1, row] <> '') and (cells [2, row] <> '') and (cells [3, row] <> '')
and (cells [4, row] <> '') and (cells [5, row] <> '') then
            begin
              rowCount := rowCount + 1;
              col := 1;
              row := rowCount-1;
              cells [0, row] := '  ' + intToStr (rowCount-1) + '.';
            end
            else showError (_errColumn);
          end;
        end;
        46 : // ctrl Del
          with stgDetail do
          begin
            total := total - strToFloat (stgDetail.Cells [_subTotal, row]);
            edtTotal.Text := floatToStr (total);
            for i := row+1 to rowCount-1 do
            begin
              for j := 0 to colCount-1 do
              begin
                cells [j, i-1] := cells [j, i];
              end;
            end;
            for i := 0 to colCount-1 do cells [i, rowCount-1] := '';
            rowCount := rowCount-1;
          end;
    end;
  end
  else
  begin
    case (key) of
```

```
      112 : //F1
      begin
        if dbbBarang.Execute then
        begin
          list := dbbBarang.hasil.AsList;
          with stgDetail do
          begin
            cells [_kdBrg, Row] := list [0];
            cells [_brg, Row] := list [1];
            cells [_Harga, Row] := list [3];
          end;
        end;
      end;
      9, 13 :  //tab
      begin
        if stgDetail.col = _kdBrg then columnChanged(_kdBrg, _jumlah)
        else columnChanged(_jumlah, _kdBrg);
      end;
    end;
  end;
 end;
end;
```

4.14. Proses Menampilkan Data Master dalam Form Transaksi

Untuk menampilkan data master dalam *form* transaksi dapat dilakukan secara manual atau secara otomatis dengan fasilitas *dbbrowse*. Berikut contoh menampilkan data master *customer* dalam *form* transaksi penjualan:

Segmen Program 4.17. Menampilkan Data Customer secara Manual dalam Form Penjualan

```
procedure TfrmPenjualan.edtKdCustExit(Sender: TObject);
begin
 tbTemp.Close;
 tbTemp.TableName := 'tbCustomer';
 tbTemp.Open;
 if tbTemp.Locate('kdCust', edtKdCust.text, [loCaseInsensitive]) then
 begin
   with tbTemp do
   begin
     lblNama.caption := Fields [1].asString;
     ….
   end;
 end
 else
 begin
   showError(_errKdCust);
   edtKdCust.SetFocus;
```

```
    lblNama.caption := '';
    ….
  end;
end;
```

Untuk menampilkan data *customer* secara manual, digunakan tabel *temporary* yang berfungsi untuk menampung sementara *field-field* data *customer* berdasarkan kode yang diinputkan yang kemudian ditampilkan pada label-label data *customer* yang ada dalam *form* penjualan. Bila kode yang dimasukkan tidak ada, maka akan ditampilkan pesan error dan semua *field* dikosongkan.

Segmen Program 4.18. Menampilkan Data Customer secara Otomatis dalam Form Penjualan

```
procedure TfrmPenjualan.btnKdCustClick(Sender: TObject);
var
  list : tStringList;
begin
  if dbbKdCust.execute then
  begin
    list := dbbKdCust.hasil.asList;
    edtKdCust.text := list [0];
    lblNama.caption := list [1];
    lblAlamat.caption := list[2];
    lblKota.caption := list [3];
    lblNegara.Caption := list [4];
    lblTelepon.caption := list [5];
    lblEMail.Caption := list [7];
  end;
end;
```

Untuk menampilkan data *customer* secara otomatis, dapat dilakukan dengan menekan *button* kecil disamping kode *customer*. Setelah itu akan ditampilkan *form browsing database* yang berisi data-data semua *customer* tersebut. Untuk menampilkan di *form* transaksi penjualan dapat dilakukan dengan men-*double click* data *customer* yang ingin ditampilkan atau menekan *button* kembali.

4.15. Proses Menampilkan Pracetak dan Cetak

Untuk *form* transaksi pembelian, penjualan, pemesanan dan retur terdapat fasilitas pracetak dan cetak. Fasilitas ini berfungsi antara lain:

- Form pembelian sebagai nota pembelian
- Form penjualan sebagai nota penjualan dan nota tagih

- Form pemesanan sebagai nota pemesanan dan surat jalan
- Form retur sebagai nota retur

Tombol “Cetak” dan “praCetak” baru dapat dijalankan setelah user melakukan pengaksesan *database* dapat berupa tambah data, mengubah data atau menghapus data. Komponen yang digunakan untuk fasilitas ini adalah *QuickReport*. Berikut contoh program untuk men-set *QuickReport* dalam transaksi penjualan:

Segmen Program 4.19. Men-set *QuickReport* dalam Transaksi Penjualan

```
procedure TfrmPenjualan.setQR();
begin
  with qtemp do
  begin
    close;
    sql.Clear;
    Parameters.CreateParameter('kdPenj', ftString, pdInput, 6, edtKdPenj.Text);
    sql.Text := 'select hd.tglpenj, hd.tglbayar, hd.total, ' +
                'c.nama as namacust, c.alamat, ' +
                'b.nama as namabrg, dt.jumlah, dt.harga ' +
                'from tbCustomer c, tbBarang b, tbHeaderPenjualan hd,
tbDetailPenjualan dt ' +
                'where hd.kdpenj = :kdPenj and ' +
                'hd.kdpenj = dt.kdpenj and ' +
                'hd.kdcust = c.kdcust and ' +
                'b.kdbarang = dt.kdbarang';
    open;
    qrTglPenj.DataField := 'tglpenj';
    qrTglBayar.DataField := 'tglbayar';
    qrNamaCust.DataField := 'namacust';
    qrAlamat.DataField := 'alamat';
    qrNamaBrg.DataField := 'namabrg';
    qrJumlah.DataField := 'jumlah';
    qrHarga.DataField := 'harga';
    qrSubtotal.Expression := 'qtemp.harga * qtemp.jumlah';
    qrTotal.DataField := 'total';
  end;
end;
```

Untuk menampilkan data transaksi penjualan pertama dibuat parameter kode penjualan kemudian dilakukan pengecekan pada setiap tabel yang berhubungan dengan transaksi penjualan, antara lain tabel *customer*, tabel barang, tabel header penjualan dan tabel detail penjualan. Setelah itu setiap *field* penjualan ditampilkan ke dalam *QuickReport*.

Segmen Program 4.20. Pracetak Data Transaksi Penjualan

```
procedure TfrmPenjualan.btnPraCetakClick(Sender: TObject);
begin
  setQR();
  QuickRep1.Preview;
end;
```

Prosedur ini untuk menampilkan pracetak transaksi penjualan.

Segmen Program 4.21. Mencetak data transaksi penjualan

```
procedure TfrmPenjualan.btnCetakClick(Sender: TObject);
begin
  setQR();
  QuickRep1.Print;
end;
```

Prosedur ini untuk melakukan cetak transaksi penjualan.

4.16. Proses Penampilan Report

Laporan adalah salah satu unsur terpenting dalam dunia kerja sebuah perusahaan, karena laporan merupakan hasil pengolahan data menjadi sebuah informasi yang sangat berguna bagi penggunanya. Program ini memiliki fasilitas laporan dalam bentuk pengarsipan data dan *print out*. Laporan yang dibuat antara lain:

Tabel 4.3. Report

Master	Laporan Customer
	Laporan Supplier
	Laporan Bahan Baku
	Laporan Barang
Transaksi	Laporan Pembelian
	Laporan Pengambilan
	Laporan Produksi
	Laporan Pemesanan

	Laporan Penjualan
	Laporan Retur

Berikut contoh program pembuatan report :

Segmen Program 4.22. Menampilan Laporan Customer

```
procedure TfrmLapCustomer.FormCreate(Sender: TObject);
begin
  with qycustomer do
  begin
    close;
    sql.clear;
    sql.text := 'select * from tbCustomer';
    open;
    qrNama.DataField := 'nama';
    qrAlamat.DataField := 'alamat';
    qrKota.Datafield := 'kota';
    qrTelp.DataField := 'telepon';
    qrEmail.DataField := 'email';
    qrKeterangan.DataField := 'keterangan';
    qrnegara.DataField := 'negara';
    First;
  end;
  QuickRep1.Preview;
end;
```

Segmen Program 4.23. Menampilan Laporan Pengambilan Bahan Baku

```
procedure TfrmLapPengambilan.Button1Click(Sender: TObject);
begin
  QRDBText3.DataField := 'jumlah';
  QRDBText4.DataField := 'keterangan';
  case cbUrut.ItemIndex of
    0 : begin
          QRGroup1.Expression := 'nama';
          qrlblGroup.Caption := 'Nama Bahan Baku : ';
          qrtxtGroup.DataField := 'nama';
          QRLabel1.Caption := 'Tanggal Pengambilan ';
          QRDBText1.DataField := 'tglambil';
        end;
    1 : begin
          QRGroup1.Expression := 'tglambil';
          qrlblGroup.Caption := 'Tanggal Pengambilan : ';
          qrtxtGroup.DataField := 'tglambil';
```

```
        QRLabel1.Caption := 'Nama Bahan Baku ';
        QRDBText1.DataField := 'nama';
      end;
  end;

  QuickRep1.Preview;
end;
```

Prosedur ini untuk menampilkan laporan pengambilan bahan baku yang dapat dipilih dengan format laporan diurutkan berdasarkan nama atau tanggal ambil.