

Osnove klijentskih skupova podataka

PRETHODNA DVA POGLAVLJA SAM RAZMATRAO DBEXPRESS – JEDNOSMERNU TEHNOLOGIJU ZA BAZE PODATAKA. U PRAKTIČNIM PRIMENAMA VEĆINA APLIKACIJA PODRŽAVA DVOSMERNO KRETANJE KROZ SKUP PODATAKA. KAO ŠTO JE RANIJE ISTAKNUTO, BORLAND SA DVOSMERNIM SKUPOVIMA PODATAKA RADI POMOĆU TEHNOLOGIJE KOJA SE NAZIVA KLIJENTSKI SKUPOVI PODATAKA. U OVOM POGLAVLJU ĆETE SE UPOZNATI SA OSNOVNIM OPERACIJAMA KLIJENTSKIH SKUPOVA PODATAKA, UKLJUČUJUĆI I TO KAKO SU ONI KORISTAN ALAT. U NAREDnim POGLAVLJIMA ĆEMO SE POZABAVITI NAPREDNIJIM MOGUĆNOSTIMA KLIJENTSKIH SKUPOVA PODATAKA, UKLJUČUJUĆI I NAČIN NA KOJI MOŽETE POVEZATI KLIJENTSKI SKUP PODATAKA SA DBEXPRESS (ILI NEKOM DRUGOM) BAZOM PODATAKA PREKO USPOSTAVLJENE VEZE KAKO BISTE NAPRAVILI PRAVU VIŠESLOJNU APLIKACIJU.

Šta su klijentski skupovi podataka?

Klijentski skup podataka, kao što mu ime govori, je skup podataka koji se nalazi u klijentskoj aplikaciji (što je suprotno od servera aplikacije). Naziv je pomalo pogrešan, jer izgleda kao da opisuje da se klijentski skupovi podataka ne mogu koristiti van klijent/server ili višeslojne aplikacije. Međutim, kao što ćece saznati iz ovog poglavlja, klijentski skupovi podataka se mogu koristiti u drugim tipovima aplikacija, naročito u jednoslojnim aplikacijama za baze podataka.

NAPOMENA

Klijentski skupovi podataka su uvedeni u Delphiju 3 i predstavljali su način pravljenja višeslojnih aplikacija u Delphiju. Pošto su klijentski skupovi podataka počeli sve više da se koriste, unapređeni su tako da mogu da podrže dodatne mogućnosti. ■

Osnovna VCL/CLX klasa za klijentske skupove podataka je klasa `TCustomClientDataSet`. Obično nećete direktno raditi sa klasom `TCustomClientDataSet`, već sa njenim direktnim naslednikom, klasom `TClientDataSet` (u poglavlju 7, "Provajderi skupova podataka", ču Vas upoznati sa još jednim naslednikom klase `TCustomClientDataSet`). Radi lakšeg razumevanja i generalizacije, na klijentske skupove ču se generalno pozivati preko klase `TClientDataSet`.

Prednosti i nedostaci klijentskih skupova podataka

Klijentski skupovi podataka imaju brojne prednosti i nekoliko primetnih nedostataka. Prednosti su:

- Memorija. Klijentski skupovi podataka se u potpunosti nalaze u memoriji što ih čini pogodnim za privremene tabele.
- Brzina. Pošto se klijentski skupovi podaka nalaze u RAM memoriji, neverovatno su brzi.
- Efikasnost. Klijentski skupovi podataka na efikasan način zapisuju svoje podatke zbog čega ne zauzimaju mnogo resursa.
- Brzo indeksiranje. Klijentski skupovi podataka Vam omogućavaju da napravite i koristite indekse u toku rada što ih čini prilagodljivim.
- Automatska podrška poništavanja. Klijentski skupovi podataka obezbeđuju podršku za više nivoa poništavanja (engl. undo) što olakšava obavljanje what if operacija nad podacima. Podrška poništavanja operacija se razmatra u poglavlju 4, "Napredne operacije klijentskih skupova podataka".

Primetni nedostaci su:

- Memorija. Prednost klijentskih skupova podataka može biti nedostatak. Pošto se klijentski skupovi podataka nalaze u RAM memoriji, njihova veličina je ograničena slobodnom RAM memorijom.
- Jedan korisnik. Klijentski skupovi podataka su jednokorisnički, jer se nalaze u RAM memoriji.

Pošto budete razumeli klijentske skupove podataka, otkrićete da ovi takozvani nedostaci nemaju presudan uticaj na aplikaciju. Kako se klijentski skupovi podataka nalaze u RAM memoriji, to je ujedno prednost i nedostatak.

Pošto se u potpunosti nalaze u RAM memoriji, klijentski skupovi podataka su korisni za rad sa privremenim tabelama, tabelama za pretraživanje i za druge potrebe baze podataka koje nisu stalne. Klijentski skupovi podataka su neverovatno brzi jer se nalaze u RAM memoriji. Umetanje, uklanjanje, pretraživanje, sortiranje i prelazak sa sloga na slog se u klijentskim skupovima podataka odigravaju brzinom svetlosti.

S druge strane, potrebno je da preduzmete korake kojima treba da obezbedite da klijentski skupovi podataka ne postanu preveliki, jer ćete rasipati dragocenu RAM memoriju ako nameravate da radite sa ogromnim skupovima podataka. Na sreću, klijentski skupovi podataka svoje podatke zapisuju veoma kompaktno. (Ovu temu ću detaljnije razmatrati u odeljku "Podrška poništavanja" u poglavlju 7.)

Pošto se nalaze u memoriji, klijentske skupove podataka koristi samo jedan korisnik. Udaljeni računari nemaju pristup klijentskom skupu podataka na lokalnom računaru. U poglavlju 8, "DataSnap", ćete naučiti kako da klijentski skup povežete sa serverom aplikacije u troslojnoj konfiguraciji koja podržava višekorisničke operacije.

Pravljenje klijentskih skupova podataka

Korišćenje klijentskih skupova podataka u aplikaciji je slično korišćenju bilo kog drugog tipa skupa podataka, jer su oni izvedeni iz klase `TDataSet`.

Klijentske skupove podataka možete napraviti u vreme dizajniranja ili u vreme izvršavanja, što ćete videti u narednim odeljcima.

Pravljenje klijentskih skupova podataka u vreme dizajniranja

Klijentske skupove podataka ćete obično praviti u vreme dizajniranja. Da biste to učinili, na formular ili u modul podataka smestite komponentu `TClientDataSet` (koja se nalazi na kartici Data Access). Na ovaj način ćete napraviti komponentu, ali nećete podesiti definicije polja ili indeksa. Komponenti dodelite ime `cdsEmployee`.

Da biste za klijentski skup podataka napravili definicije polja, u editoru formulara dva puta kliknite komponentu `TClientDataSet`. Prikazaće se standardni Delphijev editor polja. Desnim tasterom miša kliknite editor polja i iz kontekst menija odaberite New Field ... kako biste napravili novo polje. Prikazaće se okvir za dijalog koji vidite na slici 3.1.



SLIKA 3.1 Upotrebite okvir za dijalog kako biste u skup podataka dodali polje.

Ukoliko ste upoznati sa editorom polja primetićete novi tip polja koji možete da upotrebite za klijentske skupove podataka, tip Aggregate. Agregatna polja ču detaljnije razmatrati u narednom poglavlju. Za sada treba da zapamtite da klijentskom skupu podataka možete dodati polja za podatke, izračunata ili interno izračunata polja - kao za bilo koji drugi skup podataka.

Razlika između klijentskih i ostalih skupova podataka jeste u tome što kada pravite polje podataka za tipičan skup podataka Vi pravite stalno polje koje odgovara polju u bazi podataka. Kada je u pitanju klijentski skup podataka, Vi fizički pravite polje u skupu podataka zajedno sa stalnim objektom polja. Ne postoji način na koji u vreme dizajniranja možete napraviti polje u klijentskom skupu podataka, a da ne napravite stalan objekat polja.

Polja podataka

Većina polja klijentskih skupova podataka će biti polja podataka. Polje podataka predstavlja polje koje je fizički deo baze podataka nasuprot izračunatim ili lookup poljima (koja ćemo razmatrati u narednim odeljcima). Izračunata i lookup polja možete zamisliti kao virtualna polja jer postoje u skupu podataka, a njihovi podaci potiču sa nekog drugog mesta.

Hajde da našem skupu podataka dodamo polje ID. U editoru polja, u polje Name unesite ID. Predite u polje Type i unesite Integer ili izberite iz liste. (Ime komponente je automatski napravljeno.) Polje Size je neaktivno jer su polja tipa Integer stalne veličine. U sekciji Field type je unapred odabrana opcija Data, a to je ono što želimo. Na slici 3.2 vidite kompletan okvir za dijalog.



SLIKA 3.2 Okvir za dijalog New Field nakon unošenja informacija za novo polje.

Kliknite OK kako biste klijentskom skupu podataka dodali polje. U editoru polja će se prikazati novo polje ID.

Dodajmo sada drugo polje, polje `LastName`. Desnim tasterom miša kliknite editor polja kako bi se prikazao okvir za dijalog New Field, pa u polje Name unsite `LastName`. U polju Type odaberite `String`. Nakon toga, u polje Size unesite 30 - to je maksimalan broj znakova koji može da se nađe u polju. Kliknite OK kako biste polje `LastName` dodali skupu podataka.

Na isti način skupu podataka dodajte polje `FirstName` dužine 20 znakova i polje Department tipa `Integer`. Konačno, dodajmo i polje `Salary`. Otvorite okvir za dijalog New Field. U polje Name unesite `Salary`. U polju Type odaberite `Currency`, pa kliknite OK. (Tip Currency automatski nalaže Delphiju da ga prikaže koristeći znak za dolar.)

Ukoliko ste pravilno obavili ove korake, editor polja će izgledati kao na slici 3.3.



SLIKA 3.3 Izgled editora polja nakon dodavanja pet polja.

Ovo je dovoljan broj polja za skup podataka. U narednom odeljku ću Vam pokazati kako da napravite izračunato polje.

Izračunata polja

Izračunata polja, kao što je ranije istaknuto, fizički nezauzimaju prostor u skupu podataka. Umesto toga, ona se izračunavaju na osnovu drugih podataka skupa podataka. Na primer, možete napraviti izračunato polje koje predstavlja zbir druga dva polja skupa podataka. U ovom odeljku ćete napraviti dva izračunata polja: standardno i interno izračunato polje.

NAPOMENA

Interne izračunata polja zapravo zauzimaju prostor u skupu podataka baš kao i standardna polja podataka. Zbog toga nad njima možete napraviti indekse kao što biste to uradili za polja podataka. Kasnije u ovom poglavlju ćemo razmatrati indekse. ■

Standardna izračunata polja

U ovom odeljku ćemo napraviti izračunato polje kojim se izračunava godišnji bonus za koji ćemo prepostaviti da je pet procenata plate zaposlenog.

Da biste napravili standardno izračunato polje, otvorite okvir za dijalog New Field (kao što ste to radili u prethodnom poglavlju). U polje Name unesite `Bonus`, a u polju Type odaberite `Currency`.

U sekciji `Field Type` odaberite opciju `Calculated`. Na ovaj način tražite da Delphi naprave izračunato polje, a ne polje podataka. Kliknite OK.

To su svi neophodni koraci za pravljenje izračunatog polja. Pozabavimo se sada internim izračunatim poljima.

Interna izračunata polja

Pravljenje internog izračunatog polja je gotovo identično sa pravljenjem standardnog izračunatog polja. Jedina razlika je to što u okviru za dijalog **NewField** u sekciji **Field Type** birate opciju **InternalCalc** umesto opcije **Calculated**.

Druga razlika između ova dva tipa izračunatih polja jeste što se standardna izračunata polja izračunavaju u toku rada svaki put kada je potrebna njihova vrednost, dok se interna izračunata polja izračunavaju jednom a njihova vrednost se čuva u RAM memoriji. (Naravno, interna izračunata polja se automatski ponovo izračunavaju ukoliko se promene polja na osnovu kojih se izračunavaju njihove vrednosti.)

Svojstvo **AutoCalcFields** skupa podataka određuje kada se polja ponovo izračunavaju. Ako svojstvo **AutoCalcFields** ima vrednost **True** (unapred zadata vrednost), izračunata polja se izračunavaju prilikom otvaranja skupa podataka, kada skup podataka pređe u režim izmena i svaki put kada se fokus promeni sa jedne na drugu komponentu koja prepoznaže podatke i kada se tekući slog izmeni. Ako svojstvo **AutoCalcFields** ima vrednost **False**, izračunata polja se izračunavaju kada se skup podataka otvorи, kada skup podataka pređe u režim izmena i kada se iz baze podataka slog upiše u skup podataka.

Postoje dva razloga zbog kojih biste upotrebili interno umesto standardnog izračunatog polja. Ako skup podataka želite da indeksirate po izračunatom polju, morate koristiti interno izračunato polje. (Kasnije u ovom poglavlju ćemo detaljno razmatrati indekse.) Takođe, interno izračunato polje ćeće koristiti ako je potrebno relativno mnogo vremena za izračunavanje njegove vrednosti. Pošto se ova polja izračunavaju samo jednom i pošto se čuvaju u RAM memoriji, interna izračunata polja se ne moraju izračunavati onoliko često koliko standardna izračunata polja.

Dodajmo našem skupu podataka interno izračunato polje. Polje ćemo nazvati **Name** i u njemu ćeće se nalaziti vrednosti polja **FirstName** i **LastName**. Kasnije će nam verovatno biti potreban indeks nad ovim poljem pa ćemo ga učiniti internim izračunatim poljem.

Otvorite okvir za dijalog **New Field** i u polje **Name** unesite **Name**, a u polju **Type** odaberite **String**. U polje **Size** unesite 52 (što je maksimalna veličina za vrednosti polja imena, prezimena, zareza i razmaka koji će razdvajati ime i prezime).

U sekciji **Field Type** odaberite **InternalCalc** pa kliknite **OK**.

Obezbedivanje vrednosti za izračunata polja

Za sada imamo četiri izračunata polja. Potrebno je da napravimo kod za izračunavanje njihovih vrednosti. Komponenta **TClientDataSet**, kao i svi Delphijevi skupovi podataka, ima metod **OnCalcFields** za koji moramo da napravimo telo.

Ponovo kliknite klijentski skup podataka i u Object Inspector kliknite karticu **Events**. Dva puta kliknite dogadjaj **OnCalcFields** kako biste napravili obradu događaja.

Prvo ćemo izračunati vrednost polja **Bonus**. Za obradu događaja unesite odgovarajući kod. Obrada će izgledati ovako:

```
procedure TForm1.cdsEmployeeCalcFields(DataSet: TDataSet);
begin
  cdsEmployeeBonus.AsFloat := cdsEmployeeSalary.AsFloat * 0.05;
end;
```

Ovo je bilo lako - vrednost polja **Salary** smo pomnožili sa pet procenata (0.05), a dobijenu vrednost smo zapisali u polje **Bonus**.

Hajde da sada izračunamo vrednost polja **Name**. Prvi pokušaj bi (razumljivo) izgledao ovako:

```
procedure TForm1.cdsEmployeeCalcFields(DataSet: TDataSet);
begin
  cdsEmployeeBonus.AsFloat := cdsEmployeeSalary.AsFloat * 0.05;
  cdsEmployeeName.AsString := cdsEmployeeLastName.AsString + ', ' +
    cdsEmployeeFirstName.AsString;
end;
```

Rešenje radi, ali nije naročito efikasno. Polje **Name** se izračunava svaki put kada se izračunava polje **Bonus**. Međutim, prisetite se da nije neophodno izračunavati interno izračunato polje onoliko često koliko i standardno izračunato polje. Na sreću, mi možemo proveriti svojstvo **State** skupa podataka kako bismo utvrdili da li moramo ili ne moramo izračunati interno izračunato polje:

```
procedure TForm1.cdsEmployeeCalcFields(DataSet: TDataSet);
begin
  cdsEmployeeBonus.AsFloat := cdsEmployeeSalary.AsFloat * 0.05;

  if cdsEmployee.State = dsInternalCalc then
    cdsEmployeeName.AsString := cdsEmployeeLastName.AsString + ', ' +
      cdsEmployeeFirstName.AsString;
end;
```

Zapamtite da se polje **Bonus** izračunava svaki put, a da se polje **Name** izračunava samo kada nas Delphi obaveste da to treba uraditi.

Lookup polja

Lookup polja su konceptualno slična izračunatim poljima jer nisu fizički deo skupa podataka. Ipak, umesto zahteva za izračunavanjem vrednosti lookup polja, Delphi vrednost dobijaju iz drugog skupa podataka. Pogledajmo primer.

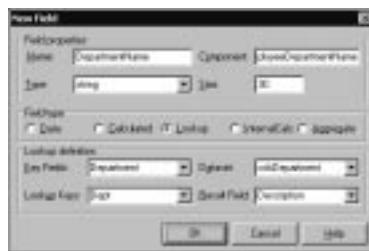
Ranije smo u skupu podataka napravili polje **Department**. Napravimo novi skup podataka **Department** u kome će biti zapisane informacije o odeljenju.

Na formular smestite komponentu **TClientDataSet** i dodelite joj ime **cdsDepartment**. Dodajte dva polja: **Dept** (celobrojno polje) i **Description** (string dužine 30 znakova).

Otvorite editor polja za skup podataka **cdsEmployee** tako što ćete dva puta kliknuti skup podataka. Otvorite okvir za dijalog New Field. Polju dodelite ime **DepartmentName**, u polju Type odaberite **String**, a u polje **Size** unesite 30.

U sekciji **Field Type** odaberite **Lookup**. Primetićete da su sada aktivna dva polja u sekciji **Lookup definition**. U polju **Key Fields** odaberite **Department**. U polju **Dataset** odaberite **cdsDepartment**.

Druga dva polja u sekciji **Lookup definition** su dostupna. U polju **Lookup Keys** odaberite **Dept**. U polju **Result Field** odaberite **Description**. Završni izgled okvira za dijalog vidite na slici 3.4.



SLIKA 3.4 Dodavanje lookup polja skupu podataka.

Za lookup polja je važno zapamtiti da **Key Field** predstavlja polje osnovnog skupa podataka koje se obraća lookup skupu podataka. Polje **Dataset** se obraća lookup skupu podataka. Polje **Lookup Keys** predstavlja polje **Key field** u lookup skupu podataka. Polje **Result** je polje lookup skupa podataka iz koga se dobijaju vrednosti.

Da biste skup podataka napravili u vreme dizajniranja, komponentu **TClientDataSet** možete kliknuti desnim tasterom miša i iz kontekst menija odabratи Create DataSet.

Pošto ste naučili kako da u vreme dizajniranja napravite klijentski skup podataka, pogledajmo šta je sve potrebno za pravljenje klijentskog skupa podataka u vreme izvršavanja.

Pravljenje klijentskog skupa podataka u vreme izvršavanja

Pravljenje klijentskog skupa podataka počinjemo sledećim kodom:

```
var
  CDS: TClientDataSet;
begin
  CDS := TClientDataSet.Create(nil);
  try
    // Do something with the client dataset here
  finally
    CDS.Free;
  end;
end;
```

Pošto napravite klijentski skup podataka obično dodajete polja, ali klijentski skup podataka možete i učitati sa diska (kao što ćete naučiti kasnije u odeljku "Trajni klijentski skupovi").

Dodavanje polja klijentskom skupu podataka

Da biste u vreme izvršavanja dodali polja, koristite svojstvo `FieldDefs` skupa podataka. Svojstvo `FieldDefs` ima dva metoda za dodavanje polja: `AddFieldDef` i `Add`.

AddFieldDef

Metod `TFieldDefs.AddFieldDef` je definisano na sledeći način:

```
function AddFieldDef: TFieldDef;
```

Kao što vidite, metod `AddFieldDef` nema parametre i vraća objekat `TFieldDef`. Kada imate objekat `TFieldDef` možete podesiti njegova svojstva kao u sledećem isečku koda.

```
var
  FieldDef: TFieldDef;
begin
  FieldDef := ClientDataSet1.FieldDefs.AddFieldDef;
  FieldDef.Name := 'Name';
  FieldDef.DataType := ftString;
  FieldDef.Size := 20;
  FieldDef.Required := True;
end;
```

Add

Brži način za dodavanje polja klijentskom skupu podataka je korišćenje metoda `TFieldDef.Add` koji je definisan na sledeći način:

```
procedure Add(const Name: string; DataType: TFieldType; Size: Integer = 0;
  Required: Boolean = False);
```

Metodu `Add` zadajete ime polja, tip podataka, veličinu (za stringovna polja) i zastavicu kojom označavate da li je polje neophodno kao parametar. Korišćenjem metoda `Add`, prethodni isečak koda postaje samo jedan red:

```
ClientDataSet1.FieldDefs.Add('Name', ftString, 20, True);
```

Zbog čega bi ikada koristili metod `AddFieldDef` kada možete koristiti metod `Add`? Jedan od razloga je to što objekat `TFieldDef` ima nekoliko naprednijih svojstava (kao što su preciznost polja, da li je samo za čitanje i nekoliko drugih atributa) pored četiri svojstva koje ima metod `Add`. Ukoliko želite da za polje zadate ova svojstva morate ih zadati pomoću objekta `TFieldDef`. Za više detalja pogledajte dokumentaciju o objektu `TFieldDef`.

Pravljenje skupa podataka

Pošto napravite definicije polja, potrebno je da u memoriji napravite prazan skup podataka. Da biste to uradili pozovite `TClientDataSet.CreateDataSet`:

```
ClientDataSet1.CreateDataSet;
```

Kao što vidite, klijentske skupove podataka je lakše napraviti u vreme dizajniranja nego u vreme izvršavanja. Međutim, ako često u memoriji pravite privremene skupove podataka ili ako je potrebno da klijentski skup podataka napravite u jedinici koja nema formular, klijentski skup podataka možete napraviti u vreme izvršavanja uz minimalan napor.

Pristupanje poljima

Bez obzira na koji način napravite klijentski skup podataka, moraćete da pristupite informacijama polja - zato što treba da ih prikažete, zato što treba da izračunate neka polja ili da dodate ili modifikuјete novi slog.

U Delphiju postoji nekoliko načina za pristupanje informacijama polja. Najlakši način je korišćenje stalnih polja.

Stalna polja

Ranije u ovom poglavljju, kada smo koristili editor polje za pravljenje polja, takođe smo pravili objekte stalnih polja za ta polja. Na primer, kada smo dodali polje LastName, Delphi je napravio stalan objekat polja cdsEmployeeLastName.

Kada znate ime objekta polja lako možete dobiti sadržaj polja koristeći familiju metoda AsXXX. Na primer, da biste polju pristupili kao da je string, koristili biste svojstvoAsString:

```
ShowMessage('The employee''s last name is ' +  
    cdsEmployeeLastNameAsString);
```

Da biste platu zaposlenog dobili u obliku pokretnog zareza, koristili biste svojstvoAsFloat:

```
Bonus := cdsEmployeeSalary.AsFloat * 0.05;
```

Da biste saznali koja sva svojstva pristupa možete koristiti, pogledajte VCL/CLX izvorni kod i Delphijevu dokumentaciju.

NAPOMENA

Prilikom pristupanja polju niste ograničeni na njegov osnovni format. Na primer, ako se u polju Salary nalazi tip podataka Currency to ne znači da ne možete da pokušate da mu pristupite kao da je string. Sledeći red koda prikazuje platu zaposlenog u obliku Currency:

```
ShowMessage('Your salary is ' + cdsEmployeeSalaryAsString);
```

Polju tipa string možete pristupiti kao da je celobrojno ako znate da se u polju nalazi celobrojna vrednost. Međutim, ako polju pokušate da pristupite kao da je celobrojno (ili nekog drugog tipa podataka), a u polju se ne nalazi kompatibilan tip podataka, Delphi poziva izuzetak. ■

Polja koja nisu stalna

Ako u vreme dizajniranja napravite skup podataka, verovatno nećete imati objekte stalnih polja. U tom slučaju postoje nekoliko metoda koje možete koristiti za rad sa vrednošću polja.

Prvi metod je FieldByName. Metod FieldByName kao parametar ima ime polja, a vraća privremeni objekat polja. Sledeći isečak koda prikazuje prezime zaposlenog koristeći metod FieldByName:

```
ShowMessage('The employee''s last name is ' +  
ClientDataSet1.FieldByName('LastName').AsString);
```

UPOZORENJE

Ako metod `FieldByName` pozovete koristeći ime nepostojećeg polja, Delphi poziva izuzetak. ■

Drugi način za pristupanje polju skupa podataka je pomoću metoda `FindField`:

```
if ClientDataSet1.FindField('LastName') <> nil then  
ShowMessage('Dataset contains a LastName field');
```

Koristeći ovu tehniku možete napraviti stalna polja u skupu podataka koji je napravljen u vreme izvršavanja.

```
var  
fldLastName: TField;  
fldFirstName: TField;  
begin  
...  
fldLastName := cds.FindField('LastName');  
fldFirstName := cds.FindField('FirstName');  
...  
ShowMessage('The last name is ' + fldLastName.AsString);  
end;
```

Na kraju, možete koristiti svojstvo `Fields` skupa podataka. Svojstvo `Fields` sadrži spisak `TField` objekata skupa podataka, što je ilustrovano sledećim kodom:

```
var  
Index: Integer;  
begin  
for Index := 0 to ClientDataSet1.Fields.Count - 1 do  
ShowMessage(ClientDataSet1.Fields[Index].AsString);  
end;
```

Obično svojstvu `Fields` nećete direktno pristupati. U opštem slučaju nije dobro prepostaviti da je dato polje prvo polje u listi `Fields`. Međutim, postoje slučajevi kada je pogodno koristiti svojstvo `Fields`. Na primer, ako imate dva klijentska skupa podataka koji imaju istu strukturu, onda iz jednog skupa podataka slog možete dodati drugom skupu podataka koristeći sledeći kod:

```
var  
Index: Integer;  
begin  
ClientDataSet2.Append;  
for Index := 0 to ClientDataSet1.Fields.Count - 1 do  
ClientDataSet2.Fields[Index].AsVariant :=  
ClientDataSet1.Fields[Index].AsVariant;  
ClientDataSet2.Post;
```

```
end;
```

U narednom odeljku detaljno ćemo razmatrati dodavanje slogova skupu podataka.

Popunjavanje i rad sa klijentskim skupovima podataka

Pošto napravite klijentski skup podataka (u vreme dizajniranja ili u vreme izvršavanja), u njega treba da smestite podatke. Postoji nekoliko načina da to uradite: podatke možete smestiti pomoću koda, slogove skupa podataka možete učitati iz drugog skupa podataka ili skup podataka možete učitati iz datoteke ili toka. U narednim odeljcima ćemo razmatrati ove načine kao i načine izmene i uklanjanja slogova.

Popunjavanje pomoću koda

Osnovni način unošenja podataka u klijentski skup podataka je korišćenje metoda `Append` i `Insert` koje podržavaju svi skupovi podataka. Razlika između ovih metoda je što metod `Append` nov slog dodaje na kraj skupa podataka, a metod `Insert` nov slog smešta ispred tekućeg sloga.

Za umetanje slogova uvek koristim metod `Append` jer se izvršava nešto brže od metoda `Insert`. Ako je skup podataka indeksiran, nov slog se automatski sortira u pravilnom redosledu.

Sledeći isečak koda pokazuje kako da klijentskom skupu podataka dodate nov slog:

```
cdsEmployee.Append; // You could use cdsEmployee.Insert; here as well
cdsEmployee.FieldByName('ID').AsInteger := 5;
cdsEmployee.FieldByName('FirstName').AsString := 'Eric';
cdsEmployee.Post;
```

Izmena slogova

Menjanje sadržaja postojećeg sloga je gotovo identično sa dodavanjem novog sloga. Umesto da pozovete metod `Append` ili `Insert` kako biste umetnuli nov slog, Vi pozivate metod `Edit` kako biste skup podataka prebacili u režim izmena. Sledecim kodom se menja ime u tekućem slogu.

```
cdsEmployee.Edit; // Edit the current record
cdsEmployee.FieldByName('FirstName').AsString := 'Fred';
cdsEmployee.Post;
```

Uklanjanje slogova

Da biste uklonili tekući slog, pozovite metod `Delete` na sledeći način:

```
cdsEmployee.Delete;
```

Ako želite da iz skupa podataka uklonite sve slogove, pozivate metod `EmptyDataSet`:

```
cdsEmployee.EmptyDataSet;
```

Popunjavanje pomoću drugog skupa podataka

Skupovi podataka dbExpressa su jednosmerni pa se kroz njih ne možete kretati unazad. Zbog toga nisu kompatibilni sa dvosmernim kontrolama koje prepoznaju podatke kakva je **TDBGrid**. Međutim, komponenta **TClientDataSet** svoje podatke može dobiti iz drugog skupa podataka (uključujući skupove podatka dbExpressa, BDE skupove podataka ili druge klijentske skupove podataka) pomoću *provajdera*. Koristeći ovu osobinu, klijentski skup podataka možete učitati iz jednosmernog dbExpress skupa podataka, a zatim povezati komponentu **TDBGrid** sa klijentskim skupom podataka obezbeđujući dvosmernu podršku.

Zaista, ova mogućnost je veoma moćna i važna tako da čini osnovu Delphijeve višeslojne podrške za baze podataka.

Popunjavanje iz datoteke ili toka: stalni klijentski skupovi podataka

Iako se klijentski skupovi podataka nalaze u RAM memoriji Vi ih možete sačuvati u datoteci ili toku i ponovo ih kasnije učitati čime ih činite stalnim. Ovo je treći način popunjavanja klijentskog skupa podataka.

Da biste skup podataka sačuvali u datoteci koristite metod **SaveToFile** koji je definisan na sledeći način:

```
procedure SaveToFile(const FileName: string = '';
Format: TDataPacketFormat = dfBinary);
```

Slično, da biste skup podataka sačuvali u toku, pozivate metod **SaveToStream** koji je definisan na sledeći način:

```
procedure SaveToStream(Stream: TStream; Format: TDataPacketFormat = dfBinary);
```

Metod **SaveToFile** prihvata ime datoteke u koju zapisujete podatke. Ukoliko ne zadate ime datoteke, podaci se zapisuju u datoteku čije ime se dobija iz svojstva **FileName** klijentskog skupa podataka. Metodi **SaveToFile** i **SaveToStream** imaju parametar kojim se zadaje format koji želite da koristite prilikom zapisivanja. Klijentski skupovi podataka se mogu zapisati u jednom od tri oblika: binarnom ili jednom od XML oblika. U tabeli 3.1 su navedeni formati koje možete zadati.

Tabela 3.1: Formati paketa podataka koje možete koristiti za učitavanje i zapisivanje.

Vrednost	Opis
dfBinary	Podaci se zapisuju u binarnom formatu.
dfXML	Podaci se zapisuju u XML formatu. Posebni znaci se zapisuju pomoću escape sekvenце.
dfXMLUTF8	Podaci se zapisuju u XML formatu. Posebni znaci se zapisuju koristeći UTF8.

Kada se klijentski skupovi podataka zapisuju na disk, tumače se kao *MyBase* datoteke. MyBase zapisuje po jedan skup podataka u datoteku ili tok, izuzev ako koristite ugnježdene *skupove podataka*.

N A P O M E N A

Ako poznajete Microsoft ADO tehnologiju, onda znate da ADO omogućava da skup podataka učinite stalnim tako što koristite XML format. XML formati koje koriste ADO i MyBase nisu kompatibilni. Drugim rečima, ne možete ADO skup podataka sačuvati na disk u XML formatu, a zatim ga učitati u klijentski skup podataka (ili obrnuto). ■

Ponekada će biti potrebno da izračunate koliko bajtova je potrebno za zapisivanje podataka koji se nalaze u klijentskom skupu podataka. Na primer, možda je potrebno da proverite koliko slobodnog prostora ima na flopi disku pre nego što sačuvate podatke ili je možda potrebno da unapred rezervišete memoriju za tok. U ovakvim situacijama treba da proverite vrednost svojstva **DataSetSize**:

```
if ClientDataSet1.DataSize > AvailableSpace then
  ShowMessage('Not enough room to store the data');
```

Vrednost svojstva **DataSetSize** uvek odgovara veličini prostora koji je neophodan za zapisivanje podataka u binarnom formatu (**dfBinary**). Obično je za zapisivanje u XML formatu potrebno više prostora, verovatno dva puta više (ili čak i više od toga).

N A P O M E N A

Jedan način za izračunavanje koliko prostora je potrebno za zapisivanje skupa podataka u XML formatu je da skup podataka zapišete u tok u memoriji, pa da zatim dobijete veličinu rezultujućeg toka. ■

Primer: pravljenje, popunjavanje i rad sa klijentskim skupom podataka

Sledećim primerom je pokazano kako da napravite, popunite i radite sa klijentskim skupom podataka u vreme izvršavanja. Kodom je omogućeno da skup podataka zapišete na disk i da ga učitate. U listingu 3.1 je kompletan izvorni kod CDS (ClientDataset) aplikacije.

Listing 3.1 Aplikacija CDS - MainForm.pas.

```
unit MainForm;

interface

uses
  SysUtils, Types, IdGlobal, Classes, QGraphics, QControls, QForms, QDialogs,
  QStdCtrls, DB, DBClient, QExtCtrls, QGrids, QDBGrids, QActnList;

const
  MAX_RECS = 10000;

type
  TfrmMain = class(TForm)
    DataSource1: TDataSource;
    pnlClient: TPanel;
```

Listing 3.1 Aplikacija CDS - MainForm.pas.

```
  pnlBottom: TPanel;
  btnPopulate: TButton;
  btnSave: TButton;
  btnLoad: TButton;
  ActionList1: TActionList;
  btnStatistics: TButton;
  Populate1: TAction;
  Statistics1: TAction;
  Load1: TAction;
  Save1: TAction;
  DBGrid1: TDBGrid;
  lblFeedback: TLabel;
  procedure FormCreate(Sender: TObject);
  procedure Populate1Execute(Sender: TObject);
  procedure Statistics1Execute(Sender: TObject);
  procedure Save1Execute(Sender: TObject);
  procedure Load1Execute(Sender: TObject);
  private
    { Private declarations }
    FCDS: TClientDataSet;
  public
    { Public declarations }
  end;

var
  frmMain: TfrmMain;

implementation

{$R *.xfm}

procedure TfrmMain.FormCreate(Sender: TObject);
begin
  FCDS := TClientDataSet.Create(Self);
  FCDS.FieldDefs.Add('ID', ftInteger, 0, True);
  FCDS.FieldDefs.Add('Name', ftString, 20, True);
  FCDS.FieldDefs.Add('Birthday', ftDateTime, 0, True);
  FCDS.FieldDefs.Add('Salary', ftCurrency, 0, True);
  FCDS.CreateDataSet;
  DataSource1.DataSet := FCDS;
end;

procedure TfrmMain.Populate1Execute(Sender: TObject);
const
```

Listing 3.1 Aplikacija CDS - MainForm.pas.

```

FirstNames: array[0 .. 19] of string = ('John', 'Sarah', 'Fred', 'Beth',
  'Eric', 'Tina', 'Thomas', 'Judy', 'Robert', 'Angela', 'Tim', 'Traci',
  'David', 'Paula', 'Bruce', 'Jessica', 'Richard', 'Carla', 'James',
  'Mary');
LastNames: array[0 .. 11] of string = ('Parker', 'Johnson', 'Jones',
  'Thompson', 'Smith', 'Baker', 'Wallace', 'Harper', 'Parson', 'Edwards',
  'Mandel', 'Stone');

var
  Index: Integer;
  t1, t2: DWord;
begin
  RandSeed := 0;

  t1 := GetTickCount;
  FCDS.DisableControls;
  try
    FCDS.EmptyDataSet;
    for Index := 1 to MAX_RECS do begin
      FCDS.Append;
      FCDS.FieldByName('ID').AsInteger := Index;
      FCDS.FieldByName('Name').AsString := FirstNames[Random(20)] + ' ' +
        LastNames[Random(12)];
      FCDS.FieldByName('Birthday').AsDateTime := StrToDate('1/1/1950') +
        Random(10000);
      FCDS.FieldByName('Salary').AsFloat := 20000.0 + Random(600) * 100;
      FCDS.Post;
    end;
    FCDS.First;
  finally
    FCDS.EnableControls;
  end;
  t2 := GetTickCount;
  lblFeedback.Caption := Format('%d ms to load %.0n records',
    [t2 - t1, MAX_RECS * 1.0]);
end;

procedure TfrmMain.Statistics1Execute(Sender: TObject);
var
  t1, t2: DWord;
  msLocateID: DWord;
  msLocateName: DWord;
begin
  FCDS.First;
  t1 := GetTickCount;

```

Listing 3.1 Aplikacija CDS - MainForm.pas.

```
FCDS.Locate('ID', 9763, []);
t2 := GetTickCount;
msLocateID := t2 - t1;

FCDS.First;
t1 := GetTickCount;
FCDS.Locate('Name', 'Eric Wallace', []);
t2 := GetTickCount;
msLocateName := t2 - t1;

ShowMessage(Format('%d ms to locate ID 9763' +
#13'%d ms to locate Eric Wallace' +
#13'%0n bytes required to store %0n records',
[msLocateID, msLocateName, FCDS.DataSize * 1.0, MAX_RECS * 1.0]));
end;

procedure TfrmMain.Save1Execute(Sender: TObject);
var
  t1, t2: DWord;
begin
  t1 := GetTickCount;
  FCDS.SaveToFile('C:\Employee.cds');
  t2 := GetTickCount;
  lblFeedback.Caption := Format('%d ms to save data', [t2 - t1]);
end;

procedure TfrmMain.Load1Execute(Sender: TObject);
var
  t1, t2: DWord;
begin
  try
    t1 := GetTickCount;
    FCDS.LoadFromFile('C:\Employee.cds');
    t2 := GetTickCount;
    lblFeedback.Caption := Format('%d ms to load data', [t2 - t1]);
  except
    FCDS.Open;
    raise;
  end;
end;

end.
```

U aplikaciji postoji pet metoda i svaki vredi proučiti:

- Metod `FormCreate` u vreme izvršavanja pravi klijentski skup podataka i njegovu strukturu. Skup podataka bi bilo lakše napraviti u vreme dizajniranja, ali sam ja želeo da Vam pokažem kod koji je neophodan da bi se to uradilo u vreme izvršavanja. Kod pravi četiri polja: `Employee ID`, `Name`, `BirthDay` i `Salary`.
- Metod `Populate1Execute` na slučajan način učitava u klijentski skup podataka 10,000 zaposlenih. Na početku metoda sam za `RandomSeed` zadao vrednost `0` kako bih obezbeđio da se dobiju isti podaci kada se aplikacija više puta pokrene

N A P O M E N A

Delphijev generator slučajnih brojeva obično koristi tekući datum i vreme. Zadavanjem konstantne vrednosti može se obezbediti dosledno generisanje slučajnih brojeva prilikom svakog pokretanja programa. ■

- Metod meri koliko vremena je potrebno za generisanje 10,000 zaposlenih. Na mom računaru je bilo potrebno oko pola sekunde.
- Metodom `Statistics1Execute` se meri vreme potrebno za izvršavanje nekoliko `Locate` operacija i izračunava se veličina prostora koji je neophodan za zapisivanje podataka na disk (u binarnom formatu). Metod `Locate` će razmatrati kasnije u ovom poglavlju.
- Metod `Save1Execute` zapisuje podatke na disk u datoteku `C:\Employee.cds`. Ekstenzija `.cds` je standardna ekstenzija, mada je ne morate obavezno koristiti, za klijentske skupove podataka koji su sačuvani u binarnom formatu. Klijentski skupovi podataka koji se zapisuju u XML formatu imaju ekstenziju `.xml`.

N A P O M E N A

Postarajte se da kliknete kontrolu `Save`, jer se datoteka koja se pravi (`C:\EMPLOYEE.CDS`) koristi u sledećim primerima aplikacija ovog poglavlja kao i u nekim primerima sledećeg poglavlja.

- Metod `Load1Execute` učitava podatke iz datoteke u klijentski skup podataka. Ako metod `LoadFromFile` ne može da se izvrši (verovatno zbog toga što datoteka ne postoji ili nije odgovarajućeg formata), klijentski skup podataka ostaje zatvoren. Zbog toga otvaram klijentski skup podataka kada se pozove izuzetak.

Na slici 3.5 vidite kako se CDS aplikacija izvršava na mom računaru. Obratite pažnju na vreme na koja su potrebna za pronalaženje sloga. Čak i prilikom pretraživanja celog skupa podataka radi pronalaženja ID-a 9763 na mom računaru je potrebno oko 10 ms.



SLIKA 3.5 Aplikacija CDS u vreme izvršavanja.

Kretanje kroz klijentske skupove podataka

Skup podataka je neupotrebljiv bez načina za kretanje unapred i/ili unazad kroz podatke. Delphijevi skupovi podataka imaju veliki broj metoda za kretanje kroz skup podataka. U narednom odeljku se razmatra Delphijeva podrška za kretanje kroz skup podataka.

Sekvecijalno (redno) kretanje

Osnovni način kretanja kroz skup podataka je redno kretanje unapred ili unazad. Na primer, kroz skup podataka se redno krećete prilikom štampanja izveštaja ili iz nekog drugog razloga. U Delphiju postoje četiri jednostavna metoda za kretanje kroz skup podataka:

- Metodom **First** se prelazi na prvi slog u skupu podataka. Metod **First** se uvek uspešno izvršava, čak i kada je skup podataka prazan. Ako je skup podataka prazan, metod **First** u svojstvo **EOF** (kraj datoteke) skupa podataka zapisuje vrednost **True**.
- Metodom **Next** se prelazi na sledeći slog skupa podataka (ukoliko svojstvo **EOF** nema vrednost **True**). Ukoliko svojstvo **EOF** ima vrednost **True**, metod **Next** se neće izvršiti. Ukoliko se pozivom metoda **Next** prelazi na kraj datoteke onda se u svojstvo **EOF** zapisuje vrednost **True**.
- Metodom **Last** se prelazi na poslednji slog skupa podataka. Metod **Last** se uvek uspešno izvršava, čak i kada je skup podataka prazan. Ako je skup podataka prazan, metod **First** u svojstvu **BOF** (početak datoteke) skupa podataka zapisuje vrednost **True**.
- Metodom **Prior** se prelazi na prethodni slog skupa podataka (ukoliko svojstvo **BOF** nema vrednost **True**). Ukoliko svojstvo **BOF** ima vrednost **True**, metod **Prior** se neće izvršiti. Ukoliko se pozivom metoda **Prior** prelazi na početak datoteke onda se u svojstvo **BOF** zapisuje vrednost **True**.

Sledećim isečkom koda je pokazano kako za kretanje kroz skup podataka možete koristiti ove metode:

```
if not ClientDataSet1.IsEmpty then begin
    ClientDataSet1.First;
    while not ClientDataSet1.EOF do begin
        // Process the current record

        ClientDataSet1.Next;
    end;

    ClientDataSet1.Last;
    while not ClientDataSet1.BOF do begin
        // Process the current record

        ClientDataSet1.Prior;
    end;
end;
```

Nasumično kretanje

Pored metoda `First`, `Last` i `Prior` (kojima se obezbeđuje sekvenčalno kretanje kroz skup podataka), komponenta `TClientDataSet` obezbeđuje još dva načina za direktni prelazak na slog: pomoću obeleživača i brojeva slogova.

Obeleživači

Obeleživač se koristi kada je klijentski skup podataka veoma sličan obeleživaču za knjigu: označava se pozicija u skupu podataka tako da se na nju kasnije možete brzo vratiti.

Postoje tri operacije koje možete obaviti sa obeleživačima: zadati obeleživač, vratiti se na njega i ukloniti ga. Sledećim kodom je pokazano kako da obavite sve tri operacije:

```
var
    Bookmark: TBookmark;
begin
    Bookmark := ClientDataSet1.GetBookmark;
    try
        // Do something with ClientDataSet1 here that changes the current record
        ...
        ClientDataSet1.GotoBookmark(Bookmark);
    finally
        ClientDataSet1.FreeBookmark(Bookmark);
    end;
end;
```

U skupu podataka možete napraviti neograničen broj obeleživača. Međutim, imajte na umu da obeleživač rezerviše malu količinu memorije pa morate ukloniti sve obeleživače koristeći metod `FreeBookmark`, inače će Vaša aplikacija dovesti do gubitka memorije.

Postoji drugi skup operacija sa obeleživačima koje možete koristiti umesto operacija `GetBookmark/GotoBookmark/FreeBookmark`. Sledećim kodom je pokazan drugačiji način obelžavanja:

```
var
  BookmarkStr: string;
begin
  BookmarkStr := ClientDataSet1.Bookmark;
  try
    // Do something with ClientDataSet1 here that changes the current record
    ...
  finally
    ClientDataSet1.Bookmark := BookmarkStr;
  end;
end;
```

Obeleživač koji se dobija svojstvom `Bookmark` je `string`, pa ne morate brinuti o njegovom uklanjanju kada Vam više nije potreban. Delphi automatski uklanja obeleživač kada izade iz opsega, što se inače čini za sve stringove.

Brojevi slogova

Za klijentske skupove podataka možete koristiti drugi način direktnog prelaženja na slog skupa podataka: zadavanjem vrednosti za svojstvo `RecNo` skupa podataka. Svojstvo `RecNo` sadrži broj koji odgovara broju tekućeg sloga koji je relativan u odnosu na početak skupa podataka.

Svojstvo `RecNo` možete pročitati kako biste utvrdili tekući apsolutni broj sloga i možete zapisati vrednost u svojstvo `RecNo` kako biste zadali tekući slog. Treba da zapamtite dve važne stvari koje se tiču svojstva `RecNo`:

- Ako pokušate da u svojstvo `RecNo` upišete broj koji je manji od jedan ili broj koji je veći od ukupnog broja slogova u skupu podataka će doći do pozivanja izuzetka `At beginning of table` odnosno `At end of table`.
- Broj sloga bilo kojeg sloga nije konstantan. Na primer, ako promenite indeks skupa podataka promeniće se brojevi svih slogova u skupu podataka.

NAPOMENA

Broj slogova skupa podataka možete utvrditi čitanjem svojstva `RecordCount` skupa podataka. Kada zadajete vrednost svojstvu `RecNo` nikada nemojte zadavati vrednost koja je veća od vrednosti koja se nalazi u svojstvu `RecordCount`. ■

Ipak, ako svojstvo `RecNo` pažljivo koristite, možete mu naći primenu. Na primer, recimo da korisnik aplikacije želi da ukloni sve slogove koji se nalaze između slogova John Smith i Fred Jones. Sledećim kodom je pokazano kako da uklonite slogove:

```

var
  RecNoJohn: Integer;
  RecNoFred: Integer;
  Index: Integer;
begin
  if not ClientDataSet1.Locate('Name', 'John Smith', []) then
    raise Exception.Create('Cannot locate John Smith');
  RecNoJohn := ClientDataSet1.RecNo;

  if not ClientDataSet1.Locate('Name', 'Fred Jones', []) then
    raise Exception.Create('Cannot locate Fred Jones');
  RecNoFred := ClientDataSet1.RecNo;

  if RecNoJohn < RecNoFred then
    // Locate John again
    ClientDataSet1.RecNo := RecNoJohn;

  for Index := 1 to Abs(RecNoJohn - RecNoFred) + 1 do
    ClientDataSet1.Delete;
end;

```

Prethodnim kodom se prvo pronalaze granični slogovi i zapisuju se njihovi apsolutni brojevi slogova u skupu podataka. Zatim se prelazi na slog sa manjim brojem sloga. Ako se Fred nalazi pre Johna, skup podataka je već pozicioniran na slog sa manjim brojem.

Pošto se slogovi sekvencijalno numerišu, možete oduzeti dva broja sloga (i dodati jedan) kako biste odredili koliko slogova treba ukloniti. Prilikom uklanjanja slogova sledeći slog postaje tekući, pa će jednostavna for petlja biti dovoljna za uklanjanje slogova.

Imajte na umu da svojstvo `RecNo` neće biti prvi izbor za kretanje kroz skup podataka, ali je zgodno znati da ga možete upotrebiti ako je neophodno.

Listing 3.2 je kompletan izvorni kod aplikacije kojom su pokazani različiti metodi za kretanje kroz klijentske skupove podataka.

Listing 3.2 Aplikacija Navigate - MainForm.pas.

```

unit MainForm;

interface

uses
  SysUtils, Classes, QGraphics, QControls, QForms, QDialogs, QStdCtrls,
  DB, DBClient, QExtCtrls, QActnList, QGrids, QDBGrids, QDBCtrls;

type
  TfrmMain = class(TForm)
    DataSource1: TDataSource;

```

Listing 3.2 Aplikacija Navigate - MainForm.pas.

```
  pnlClient: TPanel;
  pnlBottom: TPanel;
  btnFirst: TButton;
  btnLast: TButton;
  btnNext: TButton;
  btnPrior: TButton;
  DBGrid1: TDBGrid;
  ClientDataSet1: TClientDataSet;
  btnSetRecNo: TButton;
  DBNavigator1: TDBNavigator;
  btnGetBookmark: TButton;
  btnGotoBookmark: TButton;
  procedure FormCreate(Sender: TObject);
  procedure btnNextClick(Sender: TObject);
  procedure btnLastClick(Sender: TObject);
  procedure btnSetRecNoClick(Sender: TObject);
  procedure btnFirstClick(Sender: TObject);
  procedure btnPriorClick(Sender: TObject);
  procedure btnGetBookmarkClick(Sender: TObject);
  procedure btnGotoBookmarkClick(Sender: TObject);
  private
    { Private declarations }
    FBookmark: TBookmark;
  public
    { Public declarations }
  end;

var
  frmMain: TfrmMain;

implementation

{$R *.xfm}

procedure TfrmMain.FormCreate(Sender: TObject);
begin
  ClientDataSet1.LoadFromFile('C:\Employee.cds');
end;

procedure TfrmMain.btnFirstClick(Sender: TObject);
begin
  ClientDataSet1.First;
end;
```

Listing 3.2 Aplikacija Navigate - MainForm.pas.

```
procedure TfrmMain.btnPriorClick(Sender: TObject);
begin
  ClientDataSet1.Prior;
end;

procedure TfrmMain.btnNextClick(Sender: TObject);
begin
  ClientDataSet1.Next;
end;

procedure TfrmMain.btnLastClick(Sender: TObject);
begin
  ClientDataSet1.Last;
end;

procedure TfrmMain.btnSetRecNoClick(Sender: TObject);
var
  Value: string;
begin
  Value := '1';
  if InputQuery('RecNo', 'Enter Record Number', Value) then
    ClientDataSet1.RecNo := StrToInt(Value);
end;

procedure TfrmMain.btnGetBookmarkClick(Sender: TObject);
begin
  if Assigned(FBookmark) then
    ClientDataSet1.FreeBookmark(FBookmark);

  FBookmark := ClientDataSet1.GetBookmark;
end;

procedure TfrmMain.btnGotoBookmarkClick(Sender: TObject);
begin
  if Assigned(FBookmark) then
    ClientDataSet1.GotoBookmark(FBookmark)
  else
    ShowMessage('No bookmark set!');
end;

end.
```

Na slici 3.6 vidite program u vreme izvršavanja.

Indeksi klijentskih skupova podataka

Do sada u klijentskom skupu podataka nismo napravili ni jedan indeks pa se možda pitate da li su (i zašto) uopšte neophodni kada se sekvencialna pretraživanja skupa podataka (pomoću metoda `Locate`) tako brzo obavljaju.

Indeksi se u klijentskim skupovima podataka koriste iz najmanje tri razloga:

- Da bi se obezbedio brži pristup podacima. Jedna operacija `Locate` se obavlja veoma brzo, ali ako je potrebno da obavite hijlade operacija `Locate` postoji primetno poboljšanje performansi kada se koriste indeksi.
- Da bi se omogućilo da se klijentski skupovi podataka mogu sortirati u toku rada. Ovo je naročito korisno kada podatke uređujete u komponenti koja prepoznaje podatke.
- Radi implementiranja agregatnih polja.

	Name	BirthDate	Salary
41	Jenner Bahney	6/25/1951	\$71,500.00
49	Winston Journe	24/5/1972	\$99,200.00
68	Ford Rags	8/28/1987	\$63,400.00
81	Thomas Waller	3/15/1988	\$21,000.00
92	Paula Ulmer	3/28/1987	\$81,400.00
93	Tobias Johnson	6/16/1979	\$28,300.00
94	Ezra Mandel	7/13/1989	\$46,300.00
95	Magg Johnson	1/29/1988	\$71,000.00
96	John Tsoi	1/23/1/1983	\$41,000.00
97	Stacie Balone	1/23/1/1982	\$47,000.00
98	Paula Edmonds	7/5/1985	\$29,200.00

SLIKA 3.6 Aplikacija *Navigate* demonstrira različite tehnike kretanja.

Pravljenje indeksa

Indeksi se, kao i definicije polja, mogu praviti u vreme dizajniranja ili u vreme izvršavanja. Za razliku od definicija polja koje se obično prave u vreme dizajniranja, indekse pravite i uklanjate u vreme izvršavanja. Na primer, neki indeksi se koriste veoma kratko - recimo, za pravljenje izveštaja u kome su podaci o određenom redosledu. U tom slučaju ćete verovatno napraviti indeks, upotrebiti ga, a zatim ukloniti. Ukoliko Vam je indeks stalno potreban, bolje je napraviti ga u vreme dizajniranja (ili napraviti ga prvi put kada Vam je potreban i ne ukloniti ga).

Pravljenje indeksa u vreme dizajniranja

Da biste napravili indeks u vreme dizajniranja, kliknite komponentu `TClientDataSet` koja se nalazi na formularu ili u modulu podataka. U Object Inspectoru dva puta kliknite svojstvo `IndexDefs`. Prikazaće se editor indeksa.

Da biste u klijentskom skupu podataka napravili indeks, desnim tasterom miša kliknite editor indeksa i iz kontekst menija odaberite Add. Drugi način za pravljenje indeksa je da kliknete ikonu Add koja se nalazi na paleti alata ili da pritisnete taster Ins.

Zatim se vratite u Object Inspector i podesite odgovarajuća svojstva indeksa. U tabeli 3.2 su navedena svojstva indeksa.

Tabela 3.2: Svojstva indeksa.

Svojstvo	Opis
Name	Ime indeksa. Preporučujem Vam da za imena indeksa korisite prefiks by (recimo, byName, byState i tako dalje).
Fields	Lista polja odvojenih tačkom-zarez koja čine indeks. Primer: 'ID' ili 'Name;Salary'.
DescFields	Lista polja koja sa se nalaze u svojstvu Fields koja treba da se sortiraju u opadajućem redosledu. Na primer, da biste obavili rastuće sortiranje po imenu, a opadajuće po plati, svojstvo Fields treba da ima vrednost 'Name;Salary', a svojstvo DescFields vrednost 'Salary'.
CaseInsensitive	Lista polja koja sadržana u svojstvu Fields koja treba da budu indeksirana tako da se ne obraća pažnja na velika i mala slova. Na primer, ako je indeks napravljen nad imenom i prezimenom, a za ime i prezime se u obzir ne uzimaju velika i mala slova, onda svojstvo Fields treba da ima vrednost 'Last;First', a svojstvo CaseInsensitive vrednost 'Last;First'.
GroupingLevel	Svojstvo se koristi za sumiranje.
Options	Svojstvo se koristi za zadavanje dodatnih opcija indeksa. Ove opcije su objašnjene u tabeli 3.3.
Expression	Svojstvo se ne može iskoristiti za klijentske skupove podataka.
Source	Svojstvo se ne može iskoristiti za klijentske skupove podataka.

U tabeli 3.3 su navedene različite opcije indeksa koje se mogu zadati poocu svojstva Options.

Tabela 3.3: Opcije indeksa.

Svojstvo	Opis
IxPrimary	Indeks je primarni indeks skupa podataka.
IxUnique	Indeks je jedinstven.
IxDescending	Indeks ima opadajući redosled.
IxCASEInsensitive	Indeks ne obraća pažnju na velika i mala slova.
IxExpression	Svojstvo se ne može iskoristiti za klijentske skupove podataka.
IxNonMaintained	Svojstvo se ne može iskoristiti za klijentske skupove podataka.

Za jedan skup podataka možete napraviti više indeksa. Dakle, nad poljem EmployeeName možete imati i rastući i opadajući indeks.

Pravljenje indeksa u vreme izvršavanja

Nasuprot definicijama polja (koja se obično prave u vreme dizajniranja), definicije indeksa se često prave u vreme izvršavanja. Postoji nekoliko valjanih razloga zbog kojih se to radi:

- Indeksi se mogu brzo i lako napraviti i ukloniti. Dakle, ako Vam je indeks potreban veoma kratko (na primer, za štampanje izveštaja u određenom redosledu), pravljenje i uklanjanje indeksa prema potrebama smanjuje korišćenje memorije.
- Informacije o indeksu se ne zapisuju u datoteku ili tok kada je klijentski skup podataka stalan. Kada iz datoteke ili toka učitate klijentsku bazu podataka onda morate pomoći koda ponovo napraviti indekse.

Da biste napravili indeks koristite metod `AddIndex` klijentskog skupa podataka. Metod `AddIndex` ima tri obavezna parametra kao i tri opciona parametra i definisan je na sledeći način:

```
procedure AddIndex(const Name, Fields: string; Options: TIndexOptions;
  const DescFields: string = ''; const CaseInsFields: string = '';
  const GroupingLevel: Integer = 0);
```

Parametri metoda odgovaraju svojstvima objekta `TIndexDef` koja su navedena u tabeli 3.2. Sledeći isečak koda pokazuje kako da napravite jedinstven indeks nad prezimenima i imenima:

```
ClientDataSet1.AddIndex('byName', 'Last;First', [ixUnique]);
```

Kada odlučite da Vam indeks više nije potreban (zapamtite, uvek ga kasnije možete ponovo napraviti), možete ga ukloniti koristeći metod `DeleteIndex`. Metod `DeleteIndex` ima samo jedan parametar: ime indeksa koji treba da se ukloni. Sledeći red koda pokazuje kako da uklonite indeks koji je napravljen prethodnim isečkom koda:

```
ClientDataSet1.DeleteIndex('byName');
```

Korišćenje indeksa

Pravljenjem indeksa se ne obavlja nikakvo sortiranje u skupu podataka već se pravi indeks nad podacima. Pošto napravite indeks, aktivirajte ga zadavanjem vrednosti svojstvu `IndexName` skupa podataka na sledeći način

```
ClientDataSet1.IndexName := 'byName';
```

Ukoliko je nad skupom podataka definisano dva ili više indeksa lako ih možete menjati zadavanjem vrednosti svojstvu `IndexName`. Ako ne želite da korisite indeks i da se vratite na unapred zadati redosled slogova onda svojstvu `IndexName` možete dodeliti prazan string kao što je pokazano sledećim isečkom koda:

```
// Do something in name order
ClientDataSet1.IndexName := 'byName';

// Do something in salary order
ClientDataSet1.IndexName := 'bySalary';

// Switch back to the default ordering
```

```
ClientDataSet1.IndexName := '';
```

Postoji i drugi način za određivanje indeksa u vreme izvršavanja. Umesto pravljenja indeksa i zadavanja vrednosti za svojstvo `IndexName`, možete zadati vrednost svojstva `IndexFieldNames`. Svojstvo `IndexFieldNames` prihvata listu polja odvojenih tačkom-zarez koja čine indeks. Sledećim kodom je pokazano kako možete upotrebiti ovo svojstvo:

```
ClientDataSet1.IndexFieldNames := 'Last;First';
```

Iako je svojstvo `IndexFieldNames` brže i lakše koristiti nego svojstva `AddIndex/IndexName`, ta jednostavnost ima svojsih nedostataka. Tačnije,

- Ne možete zadati opcije indeksa, recimo da indeks bude jedinstven ili opadajući.
- Ne možete zadati nivo grupisanja ili napraviti sumiranje.
- Prilikom prelaženja sa jednog na drugi indeks (promenom vrednosti svojstva `IndexFieldNames`), stari indeks se automatski uklanja. To se događa toliko brzo da ne može primetiti, ali ipak treba znati da se događa. Kada indeks pravite pomoću svojstva `AddIndex`, indeks se održava sve dok ga eksplicitno ne uklonite koristeći metod `DeleteIndex`.

N A P O M E N A

Iako u istoj aplikaciji možete koristiti svojstva `IndexName` i `IndexFieldNames`, za oba svojstva ne možete istovremeno zadati vrednost. Ako zadate vrednost za svojstvo `IndexName` automatski se uklanja vrednost svojstva `IndexFieldNames`, a zadavanje vrednosti svojstva `IndexFieldNames` uklanja vrednost svojstva `IndexName`. ■

Dobijanje informacija o indeksu

U Delphiju postoji nekoliko različitih metoda za dobijanje informacija o indeksu skupa podataka. Ove metode razmatram u narednim odeljcima.

Metod `GetIndexNames`

Najjednostavniji metod za dobijanje informacija o indeksu je metod `GetIndexNames`. Metod `GetIndexNames` ima samo jedan parametar, objekat `TStrings`, u koji se zapisuje rezultujuće ime indeksa. Sledeći isečak koda pokazuje kako da učitate listu sa imenima svih indeksa koji su definisani u skupu podataka:

```
ClientDataSet1.GetIndexNames(ListBox1.Items);
```

U P O Z O R E N J E

Ako prethodni kod izvršite nad skupom podataka za koji niste definisali indekse, primetićete da su unapred već definisana dva indeksa: `DEFAULT_ORDER` i `CHANGEINDEX`. Indeks `DEFAULT_ORDER` se koristi interno da bi se sloganovi dobili u redosledu koji nije indeksiran. Indeks `CHANGEINDEX` se koristi interno kako bi se obezbedila podrška poništavanja koju ćemo razmatrati kasnije u ovom poglavlju. Nemojte uklanjati ove indekse. ■

Objekat TIndexDefs

Ako želite da dobijete detaljnije informacije o indeksu možete koristiti objekat `TIndexDefs`. Objekat `TIndexDefs` sadrži listu svih indeksa kao i informacije o svakom od njih (kao što su sva polja koja čine indeks, koja polja su u opadajućem redosledu i tako dalje).

Sledeći isečak koda pokazuje kako da informacijama o indeksu pristupite pomoću objekta `TIndexDefs`.

```
var
  Index: Integer;
  IndexDef: TIndexDef;
begin
  ClientDataSet1.IndexDefs.Update;

  for Index := 0 to ClientDataSet1.IndexDefs.Count - 1 do begin
    IndexDef := ClientDataSet1.IndexDefs[Index];
    ListBox1.Items.Add(IndexDef.Name);
  end;
end;
```

Obratite pažnju na poziv `IndexDefs.Update` pre nego što se pomoću petlje prođe kroz definicije indeksa. Ovaj poziv je neophodan kako bi se obezbedilo da interna `IndexDefs` lista bude ažurirana. Kada ovog poziva ne bi bilo, lista `IndexDefs` možda ne bi sadržala informacije o indeksima koji su u skorije vreme napravljeni.

Naredna aplikacija pokazuje kako da napravite indeksiranje u toku rada za komponentu `TDBGrid`. Takođe, ona sadrži kod za dobijanje detaljnih informacija o svim indeksima koji su definisani u skupu podataka.

Na slici 3.7 vidite kako aplikacija `CDSIndex` u vreme izvršavanja prikazuje informacije o indeksima za klijentski skup podataka o zaposlenima.

Listing 3.3 je kompletan izvorni kod aplikacije `CDSIndex`.

Slika 3.7 Aplikacija `CDSIndex` Vam pokazuje kako da u toku rada napravite indeks.

Listing 3.3 Aplikacija CDSIndex - MainForm.pas.

```
unit MainForm;

interface

uses
  SysUtils, Classes, QGraphics, QControls, QForms, QDialogs, QStdCtrls,
  DB, DBClient, QExtCtrls, QActnList, QGrids, QDBGrids;

type
  TfrmMain = class(TForm)
    DataSource1: TDataSource;
```

Listing 3.3 Aplikacija CSDIndex - MainForm.pas.

```
  pnlClient: TPanel;
  DBGrid1: TDBGrid;
  ClientDataSet1: TClientDataSet;
  pnlBottom: TPanel;
  btnDefaultOrder: TButton;
  btnIndexList: TButton;
  ListBox1: TListBox;
  procedure FormCreate(Sender: TObject);
  procedure DBGrid1TitleClick(Column: TColumn);
  procedure btnDefaultOrderClick(Sender: TObject);
  procedure btnIndexListClick(Sender: TObject);
private
  { Private declarations }
public
  { Public declarations }
end;

var
  frmMain: TfrmMain;

implementation

{$R *.xfm}

procedure TfrmMain.FormCreate(Sender: TObject);
begin
  ClientDataSet1.LoadFromFile('C:\Employee.cds');
end;

procedure TfrmMain.DBGrid1TitleClick(Column: TColumn);
begin
try
  ClientDataSet1.DeleteIndex('byUser');
except
end;

  ClientDataSet1.AddIndex('byUser', Column.FieldName, []);
  ClientDataSet1.IndexName := 'byUser';
end;

procedure TfrmMain.btnDefaultOrderClick(Sender: TObject);
begin
  // Deleting the current index will revert to the default order
  try
```

Listing 3.3 Aplikacija CSDIndex - MainForm.pas.

```
ClientDataSet1.DeleteIndex('byUser');
except
end;

ClientDataSet1.IndexFieldNames := '';
end;

procedure TfrmMain.btnAddIndexListClick(Sender: TObject);
var
  Index: Integer;
  IndexDef: TIndexDef;
begin
  ClientDataSet1.IndexDefs.Update;

  ListBox1.Items.BeginUpdate;
  try
    ListBox1.Items.Clear;
    for Index := 0 to ClientDataSet1.IndexDefs.Count - 1 do begin
      IndexDef := ClientDataSet1.IndexDefs[Index];
      ListBox1.Items.Add(IndexDef.Name);
    end;
  finally
    ListBox1.Items.EndUpdate;
  end;
end;

end.
```

Kod za dinamičko sortiranje tabele u vreme izvršavanja se nalazi u metodu `DBGrid1TitleClick`. Prvo, metod pokušava da ukloni privremeni indeks `byUser`, ukoliko postoji. Ako ne postoji, poziva se izuzetak. Prava aplikacija ne bi maskirala izuzetke. Umesto toga morala bi da presretne određene izuzetke koji se dobijaju pozivom `DeleteIndex`, a ostale bi morala da prijavi korisniku.

Metodom se zatim pravi nov indeks `byUser` koji postaje tekući indeks.

NAPOMENA

Iako ovaj kod radi, on je ipak osnovni kod. Ne postoji podrška za sortiranje po više kolona i ne postoji vizuelna indikacija po kojim kolonama je izvršeno sortiranje. Preporučujem Vam da pogledate elegantno rešenje `TCDSDBGrid` koje je napravio Džon Kaster (John Kaster) (ID 15099 u Code Centralu na adresi <http://codecentral.borland.com>). ■

Filteri i opsezi

Filteri i opsezi su načini ograničavanja količine podataka koji su u skupu podataka vidljivi, slično odredbi WHERE u SQL iskazu. Osnovna razlika između filtera, opsega i odredbe WHERE leži u tome što, kada primenite filter ili opseg Vi zapravo fizički ne menjate koji podaci se nalaze u skupu podataka. Filterima i opsezima se samo ograničava količina podataka koju možete videti.

Opsezi

Opsezi su korisni kada se podaci koje želite da ograničite nalaze u uzastopnom nizu slogova. Na primer, recimo da skup podataka sadrži podatke koji su prikazani u tabeli 3.4.

Tabela 3.4: Primer podataka za opsege i filtere.

ID	Ime	Datum rođenja	Plata
4	Bill Peterson	3/28/1957	\$60,000.00
2	Frank Smith	8/25/1963	\$48,000.00
3	Sarah Johnson	7/5/1968	\$52,000.00
1	John Doe	5/15/1970	\$39,000.00
5	Paula Wallace	1/15/1971	\$36,500.00

Podaci u ovoj skraćenoj tabeli su indeksirani po datumu rođenja. Opsezi se mogu koristiti samo kada nad skupom podataka postoji aktivan indeks.

Prepostavimo da želite da prikažete sve zaposlene koji su rođeni između 1960 i 1970 godine. Pošto su podaci indeksirani po datumu rođenja, na skup podataka na sledeći način možete primeniti opseg:

```
ClientDataSet1.SetRange(['1/1/1960'], ['12/31/1970']);
```

Opsezi su ponašaju kao segmenti, što znači da su granice opsega uključene u opseg. U prethodnom primeru, zaposleni koji su rođeni prvog januara 1960 ili 31. decembra 1970 su deo opsega.

Da biste uklonili opseg na sledeći način pozovite CancelRange:

```
ClientDataSet1.CancelRange;
```

Filteri

Za razliku od opsega, pre primene filtera nije neophodno postojanje indeksa. Filteri klijentskog skupa podataka su veoma moći i nude mnoge karakteristike SQL-a kao i nekoliko opcija koje čak ne postoje u SQL-u. U tabelama 3.5 do 3.10 su navedene različite funkcije i operacije koje možete koristiti u filteru.

Tabela 3.5: Operatori poređenja koje možete koristiti u filteru.

Funkcija	Opis	Primer
=	Jednako	Name = 'John Smith'
<>	Različito	ID <> 100
<	Manje	Birthday < '1/1/1980'
>	Veće	Birthday > '12/31/1960'
<=	Manje ili jednako	Salary <= 80000
>=	Veće ili jednako	Salary >= 40000
BLANK	Polje sa praznim stringom (ne koristi se za proveru NULL vrednosti)	Name = BLANK
IS NULL	Provera NULL vrednosti	Birthday IS NULL
IS NOT NULL	Provera ne-NUL vrednosti	Birthday IS NOT NULL

Tabela 3.6: Logički operatori koje možete koristiti u filteru.

Funkcija	Primer
And	(Name = 'John Smith') and (Birthday = '5/6/1964')
Or	(Name = 'John Smith') or (Name = 'Julie Mason')
Not	Not (Name = 'John Smith')

Tabela 3.7: Aritmetički operatori koje možete koristiti u filteru.

Funkcija	Opis	Primer
+	Sabiranje. Može se koristiti za brojeve, stringove ili datum/vreme.	Birthday + 30 < '1/1/1960' Name + 'X' = 'SmithX' Salary + 10000 = 100000
-	Oduzimanje. Može se koristiti za brojeve ili datum/vreme.	Birthday - 30 > '1/1/1960' Salary - 10000 > 40000
*	Množenje. Može se koristiti samo za brojeve.	Salary * 0.10 > 5000
/	Deljenje. Može se koristiti samo za brojeve.	Salary / 10 > 5000

Tabela 3.8: Stringovne funkcije koje možete koristiti u filteru.

Funkcija	Opis	Primer
Upper	Sva slova prelaze u velika slova.	Upper(Name) = 'JOHN SMITH'
Lower	Sva slova prelaze u mala slova.	Lower (Name) = 'john smith'
SubString	Vraća deo stringa.	SubString(Name, 6) = 'Smith' SubString(Name, 1, 4) = 'john'
Trim	Iz stringa odstranjuje vodeće i završne znake.	Trim(Name) Trim(Name, '.')
TrimLeft	Iz stringa odstranjuje vodeće znake.	TrimLeft (Name, '.')
TrimRight	Iz stringa odstranjuje završne znake.	TrimRight (Name) TrimRight (Name, '.')

Tabela 3.9: Funkcije za datum/vreme koje možete koristiti u filteru.

Funkcija	Opis	Primer
Year	Funkcija iz datuma izdvaja godinu.	Year(Birthday) = 1970
Month	Funkcija iz datuma izdvaja mesec.	Month(Birthday) = 1
Day	Funkcija iz datuma izdvaja dan.	Day(Birthday) = 15
Hour	Funkcija iz vremena u 24-satnom prikazu izdvaja sate.	Hour(Appointment) = 18
Minute	Funkcija iz vremena u 24-satnom prikazu izdvaja minute.	Minute(Appointment) = 30
Second	Funkcija iz vremena u 24-satnom prikazu izdvaja sekunde.	Second(Appointment) = 0
GetDate	Funkcija daje tekući datum i vreme.	Appointment < GetDate
Date	Funkcija iz vrednosti datum/vreme izdvaja datum.	Date(Appointment)
Time	Funkcija iz vrednosti datum/vreme izdvaja vreme.	Time(Appointment)

Tabela 3.10: Ostale funkcije i operatori koje možete koristiti u filteru.

Funkcija	Opis	Primer
LIKE	Delimično poređenje stringa.	Name LIKE '%Smith%'
IN	Provera višestrukih vrednosti.	Year(Birthday) IN (1960, 1970, 1980)
*	Delimično poređenje stringa.	Name = 'John*'

Da biste filtrirali podatke skupa podataka svojstvu `Filter` zadajte string, a zatim svojstvu `Filtered` zadajte vrednost `True`. Na primer, sledećim isečkom koda se filtriraju svi zaposleni čija imena počinju slovom M:

```
ClientDataSet1.Filter := 'Name LIKE ' + QuotedStr('M%');
ClientDataSet1.Filtered := True;
```

Da biste kasnije prikazali samo zaposlene čija imena počinju slovom P treba samo da na sledeći način promenite filter:

```
ClientDataSet1.Filter := 'Name LIKE ' + QuotedStr('P%');
```

Da biste uklonili filter, svojstvu `Filtered` dodelite vrednost `False`. Da biste uklonili filter svojstvu `Filter` ne morate dodeliti prazan string (što znači da poslednji filter možete uključiti ili isključiti promenom vrednosti `True` u vrednost `False` svojstva `Filtered`).

Naprednije kriterijume filtriranja možete primeniti koristeći događaj `OnFilterRecord` skupa podataka (umesto da koristite svojstvo `Filter`). Na primer, pretpostavimo da želite da filtrirate sve zaposlene u čijim prezimenima postoji Smith. Dakle, da želite da prikažete zaposlene Smith, Smithe i ostale. Pretpostavimo da možete da koristite funkciju Soundex, tada možete na sledeći način napisati metod za filtriranje:

```
procedure TForm1.ClientDataSet1FilterRecord(DataSet: TDataSet;
  var Accept: Boolean);
begin
  Accept := Soundex(DataSet.FieldByName('LastName').AsString) =
    Soundex('Smith');
end;
```

Ako parametru `Accept` dodelite vrednost `True`, slog se uključuje u filter. Ako parametru `Accept` dodelite vrednost `False`, slog ostaje sakriven.

Pošto napravite obradu događaja `OnFilterRecord` možete svojstvu `TClientDataSet.Filtered` dodeliti vrednost `True`. Nije neophodno zadavati vrednost svojstva `Filter`.

Sledećim primerom su pokazane razne tehnike filtriranja i korišćenja opsega.

Listing 3.4 je izvorni kod za glavni formular.

Listing 3.4 Aplikacija RangeFilter - MainForm.pas.

```
unit MainForm;

interface

uses
  SysUtils, Classes, QGraphics, QControls, QForms, QDialogs, QStdCtrls,
  DB, DBCClient, QExtCtrls, QGrids, QDBGrids;

type
  TfrmMain = class(TForm)
```

Listing 3.4 Aplikacija RangeFilter - MainForm.pas.

```
DataSource1: TDataSource;
pnlClient: TPanel;
pnlBottom: TPanel;
btnFilter: TButton;
btnRange: TButton;
DBGrid1: TDBGrid;
ClientDataSet1: TClientDataSet;
btnClearRange: TButton;
btnClearFilter: TButton;
procedure FormCreate(Sender: TObject);
procedure btnFilterClick(Sender: TObject);
procedure btnRangeClick(Sender: TObject);
procedure btnClearRangeClick(Sender: TObject);
procedure btnClearFilterClick(Sender: TObject);
private
  { Private declarations }
public
  { Public declarations }
end;

var
  frmMain: TfrmMain;

implementation

uses FilterForm, RangeForm;

{$R *.xfm}

procedure TfrmMain.FormCreate(Sender: TObject);
begin
  ClientDataSet1.LoadFromFile('C:\Employee.CDS');

  ClientDataSet1.AddIndex('bySalary', 'Salary', []);
  ClientDataSet1.IndexName := 'bySalary';
end;

procedure TfrmMain.btnFilterClick(Sender: TObject);
var
  frmFilter: TfrmFilter;
begin
  frmFilter := TfrmFilter.Create(nil);
  try
    if frmFilter.ShowModal = mrOk then begin
```

Listing 3.4 Aplikacija RangeFilter - MainForm.pas.

```
ClientDataSet1.Filter := frmFilter.Filter;
ClientDataSet1.Filtered := True;
end;
finally
  frmFilter.Free;
end;
end;

procedure TfrmMain.btnClearFilterClick(Sender: TObject);
begin
  ClientDataSet1.Filtered := False;
end;

procedure TfrmMain.btnRangeClick(Sender: TObject);
var
  frmRange: TfrmRange;
begin
  frmRange := TfrmRange.Create(nil);
  try
    if frmRange.ShowModal = mrOk then
      ClientDataSet1.SetRange([frmRange.LowValue], [frmRange.HighValue]);
  finally
    frmRange.Free;
  end;
end;

procedure TfrmMain.btnClearRangeClick(Sender: TObject);
begin
  ClientDataSet1.CancelRange;
end;

end.
```

Kao što vidite, glavni formular sa diska učitava skup podataka o zaposlenima, pravi indeks nad poljem `Salary` i aktivira ga. Zatim korisniku omogućava da na skup podataka primeni opseg filter ili i opseg i filter.

Listing 3.5 sadrži izvorni kod za filter formulara. Filter formulara omogućava korisniku da odabere polje nad kojim će biti primenjen filter i da unese vrednost na osnovu koje će biti obavljeno filtriranje.

Listing 3.5 Aplikacija RangeFilter - FilterForm.pas.

```
unit FilterForm;

interface

uses
  SysUtils, Classes, QGraphics, QControls, QForms, QDialogs, QStdCtrls,
  QExtCtrls;

type
  TfrmFilter = class(TForm)
    pnlClient: TPanel;
    pnlBottom: TPanel;
    Label1: TLabel;
    cbField: TComboBox;
    Label2: TLabel;
    cbRelationship: TComboBox;
    Label3: TLabel;
    ecValue: TEdit;
    btnOk: TButton;
    btnCancel: TButton;
  private
    function GetFilter: string;
    { Private declarations }
  public
    { Public declarations }
    property Filter: string read GetFilter;
  end;

implementation

{$R *.xfm}

{ TfrmFilter }

function TfrmFilter.GetFilter: string;
begin
  Result := Format('%s %s ''%s'''',
    [cbField.Text, cbRelationship.Text, ecValue.Text]);
end;

end.
```

Jedini interesantan deo koda je funkcija `GetFilter` koja spaja vrednosti tri ulazne kontrole u string koji šalje glavnoj aplikaciji.

Listing 3.6 je izvorni kod za opseg formulara. Za opseg formulara korisnik treba da unese donju i gornju granicu plata.

Listing 3.6 Aplikacija RangeFilter - RangeForm.pas.

```
unit RangeForm;

interface

uses
  SysUtils, Classes, QGraphics, QControls, QForms, QDialogs, QExtCtrls,
  QStdCtrls;

type
  TfrmRange = class(TForm)
    pnlClient: TPanel;
    pnlBottom: TPanel;
    Label1: TLabel;
    Label2: TLabel;
    ecLower: TEdit;
    ecUpper: TEdit;
    btnOk: TButton;
    btnCancel: TButton;
    procedure btnOkClick(Sender: TObject);
  private
    function GetHighValue: Double;
    function GetLowValue: Double;
    { Private declarations }
  public
    { Public declarations }
    property LowValue: Double read GetLowValue;
    property HighValue: Double read GetHighValue;
  end;

implementation

{$R *.xfm}

{ TfrmRange }

function TfrmRange.GetHighValue: Double;
begin
  Result := StrToFloat(ecUpper.Text);
end;

function TfrmRange.GetLowValue: Double;
```

Listing 3.6 Aplikacija RangeFilter - RangeForm.pas.

```

begin
  Result := StrToFloat(ecLower.Text);
end;

procedure TfrmRange.btnOkClick(Sender: TObject);
var
  LowValue: Double;
  HighValue: Double;
begin
  try
    LowValue := StrToFloat(ecLower.Text);
    HighValue := StrToFloat(ecUpper.Text);

    if LowValue > HighValue then begin
      ModalResult := mrNone;
      ShowMessage('The upper salary must be >= the lower salary');
    end;
  except
    ModalResult := mrNone;
    ShowMessage('Both values must be a valid number');
  end;
end;

end.

```

Na slici 3.8 vidite kako se izvršava aplikacija RangeFilter.

The screenshot shows a Delphi application window titled 'Delphi Range Filter'. It contains a table with columns: 'Name', 'Address', and 'Salary'. The data is as follows:

Name	Address	Salary
5323 Steve Edwards	3/30/1967	\$41,000.00
3419 Bruce Edwards	6/10/1962	\$41,000.00
7852 Paul Edwards	7/17/1976	\$41,000.00
6801 David Edwards	2/18/1973	\$41,000.00
2793 Anna Edwards	11/22/1962	\$41,000.00
5209 Tracy Edwards	5/26/1971	\$41,000.00
6338 Bruce Edwards	3/30/1966	\$41,400.00
7944 Bruce Edwards	6/9/1967	\$41,500.00
4987 Tracy Edwards	7/28/1974	\$41,600.00
2776 Paul Edwards	8/17/1986	\$41,600.00
8128 Bruce Edwards	3/28/1988	\$41,600.00
6481 Victoria Edwards	7/6/1978	\$41,600.00
414 David Edwards	4/14/1972	\$41,700.00

At the bottom of the window, there are buttons for 'Save File', 'Open File', 'Set Range', and 'Close Range'.

SLIKA 3.8 Aplikacija RangeFilter nad skupom podataka koristi opsege i filtere.

Pretraživanje

Pored filtriranja neinteresantnih slogova iz klijentskog skupa podataka, komponenta `TClientDataSet` ima brojne metode za brzo pronalaženje zadatog sloga. Za neke od tih metoda neophodno je da indeks nad skupom podataka bude aktivan dok za ostale nije. Metodi za pretraživanje su detaljno objašnjeni u narednim odeljcima.

Tehnike neindeksiranog pretraživanja

U ovom odeljku će razmatrati tehnike pretraživanja za koje nije neophodno postojanje aktivnog indeksa nad klijentskim skupom podataka. Umesto da koriste indeks, ovi metodi obavljaju sekvencijalno pretraživanje nad skupom podataka u potrazi za prvim sloganom koji zadovoljava zadate uslove.

Metod Locate

Metod `Locate` je verovatno najopštiji metod pretraživanja komponente `TClientDataSet`. Metod `Locate` možete koristiti za pronalaženje sloga na osnovu bilo kog polja ili kombinacije polja. Metod `Locate` možete koristiti i za pronalaženje sloganova na osnovu delimičnog poklapanja sa uslovima i pronalaženje sloganova bez obzira na velika i mala slova.

```
function Locate(const KeyFields: string; const KeyValues: Variant;
  Options: TLocateOptions): Boolean; override;
```

Prvi parametar, parametar `KeyFields`, označava polje (ili polja) koja se pretražuju. Kada pretražujete više polja odvajajte ih tačkom-zarez (na primer, 'Name;Birthday').

Drugi parametar, parametar `KeyValues`, predstavlja vrednosti koje tražite. Broj vrednosti mora odgovarati broju polja. Ako se pretraživanje obavlja po samo jednom polju, ovaj parametar ima vrednost koja se traži. Da biste pretraživanje proveli po više vrednosti te vrednosti morate proslediti kao niz podataka tipa variant. Jedan od načina na koji to možete uraditi jeste pozivanje funkcije `VarArrayOf`:

```
VarArrayOf(['John Smith', '4/15/1965'])
```

Poslednji parametar, parametar `Options`, određuje kako se pretraživanje sprovodi. U tabeli 3.11 su navedene sve opcije.

Tabela 3.11: Opcije metoda Locate.

Vrednost	Opis
<code>loPartialKey</code>	Parametar <code>KeyValues</code> ne mora da sadrži potpuno poklapanje. Metod <code>Locate</code> pronalazi prvi slogan čije polje počinje vrednošću koja se nalazi u parametru <code>KeyValues</code> .
<code>loCaseInsensitive</code>	Metod <code>Locate</code> ne obraća pažnju na velika i mala slova kada pretražuje stringovna polja.

Obe opcije se odnose samo na stringovna polja. One se ignorisu ukoliko se pretraživanje sproveodi na poljima koja nisu stringovna.

Metod `Locate` daje vrednost `True` ako se pronađe slog koji odgovara uslovima, a vrednost `False` ako se ne pronađe. Ako se pronađe slog onda on postaje tekući slog.

Sledećim primerima su ilustrovane ove opcije:

```
ClientDataSet1.Locate('Name', 'John Smith', []);
```

Prethodnim kodom se pronalazi slog u kome je ime jednako 'John Smith'.

```
ClientDataSet1.Locate('Name', 'JOHN', [loPartialKey, loCaseInsensitive]);
```

Prethodnim kodom se pronalazi slog u kojem ime počinje sa 'JOHN'. Pronalaze se slogovi 'John Smith', 'Johnny Jones' i 'JOHN ADAMS', ali ne i slog 'Bill Johnson'.

```
ClientDataSet1.Locate('Name;Birthday', VarArrayOf(['John', '4/15/1965']), [loPartialKey]);
```

Prethodnim kodom se pronalazi slog u kojem ime počinje sa 'John', a datum rođenja je 15. april 1965 godine. U ovom slučaju se opcija `loPartialKey` odnosi samo na ime. Iako se datum rođenja prosleđuje kao string, polje na koje se odnosi je tipa podataka Date, pa se za ovo polje opcija `loPartialKey` ignoriše.

Metod Lookup

Metod `Lookup` je sličan metodu `Locate` izuzev što ne pomera tekući pokazivač sloga. Umesto toga metod `Lookup` vraća vrednost jednog ili više polja u slogu. Takođe, metod `Lookup` nema parametar `Options` tako da ne možete obaviti pretraživanje na osnovu delimične vrednosti ili pretraživanje kojim se u obzir ne uzimaju velika i mala slova.

Metod `Lookup` je definisan na sledeći način:

```
function Lookup(const KeyFields: string; const KeyValues: Variant;
  const ResultFields: string): Variant; override;
```

Parametri `KeyFields` i `KeyValues` označavaju polja koja se pretražuju i vrednosti koje se traže kao kod metoda `Locate`. Parametar `ResultFields` određuje polja za koja želite da dobijete podatke. Na primer, da biste dobili datum rođenja zapislenog čiji je ime John Doe, mogli biste da napišete sledeći kod:

```
var
  V: Variant;
begin
  V := ClientDataSet1.Lookup('Name', 'John Doe', 'Birthday');
end;
```

Sledećim kodom se dobija ime i datum rođenja zapislenog čiji je ID jednak 100.

```
var
  V: Variant;
begin
  V := ClientDataSet1.Lookup('ID', 100, 'Name;Birthday');
end;
```

Ako se traženi slog ne pronađe, promenljiva V ima vrednost NULL. Ako parametar `ResultFields` sadrži samo jedno polje, onda se pri povratku iz metoda `Lookup` u promenljivoj V nalazi vrednost tipa variant koja predstavlja vrednost polja koje je navedeno parametrom `ReturnFields`. Ako parametar `ResultFields` sadrži više imena polja, onda se pri povratku iz metoda `Lookup` u promenljivoj V nalazi niz vrednosti tipa variant koja predstavljaju vrednosti polja navedenih u parametru `ResultFields`.

NAPOMENA

Detaljnije razmatranje o nizovima tipa variant možete naći u mojoj knjizi Delphi COM Programming, Macmillan Technical Publishing. ■

Sledećim isečkom koda je pokazano kako možete pristupiti rezultatima koji su dobijeni metodom `Lookup`.

```
var
  V: Variant;
begin
  V := ClientDataSet1.Lookup('ID', 100, 'Name');
  if not VarIsNull(V) then
    ShowMessage('ID 100 refers to ' + V);

  V := ClientDataSet1.Lookup('ID', 200, 'Name;Birthday');
  if not VarIsNull(V) then
    ShowMessage('ID 200 refers to ' + V[0] + ', born on ' +
DateToStr(V[1]));
end;
```

Tehnike indeksiranog pretraživanja

Tehnike pretraživanja koje sam prethodno pomenuo ne zahtevaju postojanje aktivnog indeksa (zapravo, za korišćenje tih tehniki nije uopšte potrebno da nad skupom podataka postoji indeks), ali komponenta `TDataSet` podržava nekoliko operacija pretraživanja sa indeksima.

Metod `FindKey`

Metod `FindKey` pronalazi odgovarajući slog koristeći polje ključa tekućeg indeksa. Na primer, ako u skupu podataka postoji indeks nad poljem `ID`, metod `FindKey` pronalazi slog za koji polje `ID` u potpunosti odgovara uslovu pretraživanja. Ako u skupu podataka postoji indeks na poljima imena i prezimena, metod `FindKey` pronalazi slog za koji polja imena i prezimena u potpunosti odgovaraju uslovu pretraživanja.

Metod `FindKey` ima samo jedan parametar kojim se zadaje vrednost(i) koja se traži. Metod vraća vrednost tipa Boolean kojom se označava da li je pronađen odgovarajući slog. Ako se ne pronađe ni jedan slog onda se ne pomera pokazivač sloga. Ako se slog pronađe, on postaje tekući slog.

Parametar metoda `FindKey` je zapravo niz vrednosti, pa nije neophodno te vrednosti navoditi između uglastih zagrada, što je pokazano sledećim primerom:

```

if ClientDataSet.FindKey([25]) then
  ShowMessage('Found ID 25');

...
if ClientDataSet.FindKey(['Doe', 'John']) then
  ShowMessage('Found John Doe');

```

Potrebito je samo da obezbedite vrednosti tekućeg indeksa. Zbog toga bi trebalo da indeks odredite pre pozivanja metoda `FindKey`. Sledećim isečkom koda pokazano je kako da to uradite:

```

ClientDataSet1.IndexName := 'byID';
if ClientDataSet.FindKey([25]) then
  ShowMessage('Found ID 25');

...
ClientDataSet1.IndexName := 'byName';
if ClientDataSet.FindKey(['Doe', 'John']) then
  ShowMessage('Found John Doe');

```

Metod FindNearest

Metod `FindNearest` radi na sličan način kao metod `FindKey`, izuzev što pronalazi prvi slog koji je veći ili jednak prosleđenim vrednostima. Pronalaženje zavisi od vrednosti svojstva `KeyExclusive`.

Ako svojstvo `KeyExclusive` ima vrednost `False` (unapred zadata vrednost), metod `FindNearest` pronalazi prvi slog koji je jednak ili veći od prosleđenih vrednosti. Ako svojstvo `KeyExclusive` ima vrednost `True`, metod `FindNearest` pronalazi prvi slog koji je veći od prosleđenih vrednosti.

Ako metod `FindNearest` ne pronađe ni jedan slog, on pomera pokazivač sloga na kraj skupa podataka.

Metod GotoKey

Metod `GotoKey` obavlja istu funkciju kao i metod `FindKey`, izuzev što Vi zadajete vrednost pretraživanja polja pre pozivanja metoda `GotoKey`. Sledećim isečkom koda je pokazano kako da upotrebite metod `GotoKey`:

```

ClientDataSet1.IndexName := 'byID';
ClientDataSet1.SetKey;
ClientDataSet1.FieldByName('ID').AsInteger := 25;
ClientDataSet1.GotoKey;

```

Ako se indeks sastoji iz više polja, na sledeći način zadajete svako polje nakon poziva `SetKey`:

```

ClientDataSet1.IndexName := 'byName';
ClientDataSet1.SetKey;
ClientDataSet1.FieldByName('First').AsString := 'John';
ClientDataSet1.FieldByName('Last').AsString := 'Doe';
ClientDataSet1.GotoKey;

```

Posle pozivanja metoda `GotoKey`, možete koristiti metod `EditKey` kako biste promenili vrednosti ključa koje ste koristili za pretraživanje. Na primer, sledeći isečak koda pokazuje kako da pronadete slog John Doe, pa da zatim pronadete slog John Smith. U oba sloga ime je identično tako da se menja samo deo koji se odnosi na prezime koje treba zadati prilikom drugog pretraživanja.

```
ClientDataSet1.IndexName := 'byName';
ClientDataSet1.SetKey;
ClientDataSet1.FieldByName('First').AsString := 'John';
ClientDataSet1.FieldByName('Last').AsString := 'Doe';
ClientDataSet1.GotoKey;
// Do something with the record

// EditKey preserves the values set during the last SetKey
ClientDataSet1.EditKey;
ClientDataSet1.FieldByName('Last').AsString := 'Smith';
ClientDataSet1.GotoKey;
```

Metod `GotoNearest`

Metod `GotoNearest` radi slično kao metod `GotoKey`, izuzev što pronalazi prvi slog koji je veći ili jednak vrednosti(ma) koje se prosleđuju. Da li će vrednosti koje se pronalaze biti veće ili jednake prosleđenim vrednostima zavisi od vrednosti svojstva `KeyExclusive`.

Ako svojstvo `KeyExclusive` ima vrednost `False` (unapred zadata vrednost), metod `GotoNearest` pronalazi prvi slog koji je jednak ili veći od vrednosti koje su zadate nakon poziva metoda `SetKey` ili `EditKey`. Ako svojstvo `KeyExclusive` ima vrednost `True`, metod `GotoNearest` pronalazi prvi slog koji je veći od vrednosti koje su zadate nakon poziva metoda `SetKey` ili `EditKey`.

Ako metod `GotoNearest` ne pronađe ni jedan slog, on pomera pokazivač sloga na kraj skupa podataka.

Sledeći primer pokazuje kako da u skupu podataka obavite pretraživanje sa i bez indeksa.

Listing 3.7 je izvorni kod aplikacije `Search`, programa koji ilustruje različite tehnike pretraživanja sa i bez indeksa koje podržava komponenta `TClientDataSet`.

Listing 3.7 Aplikacija Search - `MainForm.pas`.

```
unit MainForm;

interface

uses
  SysUtils, Classes, Variants, QGraphics, QControls, QForms, QDialogs,
  QStdCtrls, DB, DBClient, QExtCtrls, QActnList, QGrids, QDBGrids;

type
  TfrmMain = class(TForm)
```

Listing 3.7 Aplikacija Search - MainForm.pas.

```
DataSource1: TDataSource;
pnlClient: TPanel;
pnlBottom: TPanel;
btnSearch: TButton;
btnGotoBookmark: TButton;
btnGetBookmark: TButton;
btnLookup: TButton;
DBGrid1: TDBGrid;
ClientDataSet1: TClientDataSet;
btnSetRecNo: TButton;
procedure FormCreate(Sender: TObject);
procedure btnGetBookmarkClick(Sender: TObject);
procedure btnGotoBookmarkClick(Sender: TObject);
procedure btnSetRecNoClick(Sender: TObject);
procedure btnSearchClick(Sender: TObject);
procedure btnLookupClick(Sender: TObject);
private
  { Private declarations }
  FBookmark: TBookmark;
public
  { Public declarations }
end;

var
  frmMain: TfrmMain;

implementation

uses SearchForm;

{$R *.xmf}

procedure TfrmMain.FormCreate(Sender: TObject);
begin
  ClientDataSet1.LoadFromFile('C:\Employee.cds');

  ClientDataSet1.AddIndex('byName', 'Name', []);
  ClientDataSet1.IndexName := 'byName';
end;

procedure TfrmMain.btnGetBookmarkClick(Sender: TObject);
begin
  if Assigned(FBookmark) then
    ClientDataSet1.FreeBookmark(FBookmark);
```

Listing 3.7 Aplikacija Search - MainForm.pas.

```
FBookmark := ClientDataSet1.GetBookmark;
end;

procedure TfrmMain.btnGotoBookmarkClick(Sender: TObject);
begin
  if Assigned(FBookmark) then
    ClientDataSet1.GotoBookmark(FBookmark)
  else
    ShowMessage('No bookmark assigned');
end;

procedure TfrmMain.btnSetRecNoClick(Sender: TObject);
var
  Value: string;
begin
  Value := '1';
  if InputQuery('RecNo', 'Enter Record Number', Value) then
    ClientDataSet1.RecNo := StrToInt(Value);
end;

procedure TfrmMain.btnSearchClick(Sender: TObject);
var
  frmSearch: TfrmSearch;
begin
  frmSearch := TfrmSearch.Create(nil);
  try
    if frmSearch.ShowModal = mrOk then begin
      case TSearchMethod(frmSearch.grpMethod.ItemIndex) of
        smLocate:
          ClientDataSet1.Locate('Name', frmSearch.ecName.Text,
            [loPartialKey, loCaseInsensitive]);
        smFindKey:
          ClientDataSet1.FindKey([frmSearch.ecName.Text]);
        smFindNearest:
          ClientDataSet1.FindNearest([frmSearch.ecName.Text]);
        smGotoKey: begin
          ClientDataSet1.SetKey;
          ClientDataSet1.FieldByName('Name').AsString :=
            frmSearch.ecName.Text;
          ClientDataSet1.GotoKey;
        end;
      end;
    end;
  finally
    frmSearch.Free;
  end;
end;
```

Listing 3.7 Aplikacija Search - MainForm.pas.

```

smGotoNearest: begin
    ClientDataSet1.SetKey;
    ClientDataSet1.FieldByName('Name').AsString :=
        frmSearch.ecName.Text;
    ClientDataSet1.GotoNearest;
end;
end;
end;
finally
    frmSearch.Free;
end;
end;
end;

procedure TfrmMain.btnAddClick(Sender: TObject);
var
    Value: string;
    V: Variant;
begin
    Value := '1';
    if InputQuery('ID', 'Enter ID to Lookup', Value) then begin
        V := ClientDataSet1.Lookup('ID', StrToInt(Value), 'Name;Salary');
        if not VarIsNull(V) then
            ShowMessage(Format('ID %s refers to %s, who makes %s',
                [Value, V[0], FloatToStrF(V[1], ffCurrency, 10, 2)]));
    end;
end;
end;

```

Listing 3.8 je izvorni kod formulara za pretraživanje. Jedini interesantan deo koda je TSearchMetod koji je definisan na početku jedinice. Koristi se za određivanje metoda koji treba pozvati radi pretraživanja.

Listing 3.8 Aplikacija Serach - SearchForm.pas.

```

unit SearchForm;

interface

uses
    SysUtils, Classes, QGraphics, QControls, QForms, QDialogs, QExtCtrls,
    QStdCtrls;

type
    TSearchMethod = (smLocate, smFindKey, smFindNearest, smGotoKey,

```

Listing 3.8 Aplikacija Serach - SearchForm.pas.

```
    smGotoNearest);

TfrmSearch = class(TForm)
  pnlClient: TPanel;
  pnlBottom: TPanel;
  Label1: TLabel;
  ecName: TEdit;
  grpMethod: TRadioGroup;
  btnOk: TButton;
  btnCancel: TButton;
private
  { Private declarations }
public
  { Public declarations }
end;

implementation

{$R *.xfm}

end.
```

Na slici 3.9 vidite aplikaciju Search u vreme izvršavanja.



SLIKA 3.9 Aplikacija Search demonstrira pretražvanje sa i bez indeksa.

Zaključak

Komponenta `TClientDataSet` je veoma moćan skup podataka koji se nalazi u memoriji. Podržava brojne operacije sortiranja i pretraživanja visokih preformansi. Navešću nekoliko ključnih stvari koje treba da zapamtitate:

- Klijentski skup podataka možete napraviti u vreme dizajniranja i u vreme izvršavanja. U ovom poglavlju ste naučili kako da klijentske skupove podataka zapišete na disk kako biste ih koristili u aplikacijama za baze podataka.
- Tri osnovna načina popunjavanja klijentskog skupa podataka su:
 - Pomoću metoda `Append` i `Insert`.
 - Pomoću drugog skupa podataka.
 - Pomoću datoteke ili toka (to jest, pomoću stalnih klijentskih skupova podataka).
- Kroz Delphijeve skupove podataka se možete kretati na razne načine: sekvencijalno (redno), pomoću obeleživača i pomoću brojeva slogova.
- U skupu podataka možete napraviti indekse koji Vam olakšavaju i ubrzavaju sortiranje slogova u zadatom redosledu i pronalaženje slogova koji zadovoljavaju zadate uslove.
- Filteri i opsezi se mogu koristiti za ograničavanje količine podataka koja je vidljiva u skupu podataka. Opsezi se koriste kada su relevantni podaci zapisani u uzastopnim slogovima skupa podataka. Za razliku od opsega, filtere možete koristiti i kada u skupu podataka ne postoji indeks.
- Metodi `Locate` i `Lookup` su tehnike za pronalaženje slogova u klijentskom skupu podataka za koje nije potreban indeks. Metodi `FindKey`, `FindNearest`, `GotoKey` i `GotoNearest` su tehnike pretraživanja koje koriste indeks.

U narednom poglavlju ću razmatrati naprednije karakteristike klijentskih skupova podataka.