4 DARBS AR DATU AVOTIEM

Šī laboratorijas darba mērķis ir iepazīties ar izstrādes vidē pieejamajiem vizuālajiem rīkiem, kas nodrošina manipulācijas ar datu avotiem. Tiek pieņemts, ka studenti ir pazīstami ar relāciju datu bāzu principiem un datu manipulēšanu (izgūšanu, modificēšanu un saglabāšanu), kā arī pārzina vaicājumu valodas SQL pamatus.

Padoms: *Laboratorijas darba izpildi var uzsākt uzreiz no sadaļas 4.3, atgriežoties pie apraksta sadaļām tikai tad, ja ir neskaidrības uzdevumu izpildīšanā.*

4.1 IESKATS TEHNOLOĢIJĀ ADO.NET

Tehnoloģiju ADO.NET veido klašu bibliotēka, kas realizē sadarbību ar dažāda veida datu avotiem, nodrošinot pieslēgšanos datu bāzes serveriem, vaicājumu sagatavošanu un nosūtīšanu, kā arī rezultātu saņemšanu un apstrādāšanu. Tehnoloģija ADO.NET ir galvenais mehānisms, kas tiek izmantots .NET lietojumprogrammās, lai piekļūtu un manipulētu ar datu avotiem.

Kaut arī tehnoloģijas nosaukumā ir abreviatūra ADO, tai ir ļoti maza saistība ar tehnoloģiju ADO (ActiveX Data Objects), jo ADO.NET ir pilnībā pārstrādāta klašu bibliotēka, kura nodrošina pilnvērtīgu XML atbalstu, kā arī pilnu kontroli par datu izgūšanu, modificēšanu un saglabāšanu datu avotos (starp kuriem, protams, var būt arī datu bāzes).

Tehnoloģijas ADO.NET funkcionalitāti var sadalīt divās loģiskās kategorijās: vienā, kas sadarbībai ar datu avotiem prasa pastāvīgu savienojumu (*connected layer*); un otrā, kurai pastāvīgs savienojums nav nepieciešams (*disconnected layer*). Pēdējā ir būtisks tehnoloģijas ADO.NET jaunievedums, kas ļauj realizēt mērogojamas darbvirsmas un tīmekļa lietojumprogrammas, jo savienojumu ar datu avotu veido tikai brīdī, kad nepieciešams datus izgūt vai saglabāt – kamēr notiek datu apskate vai labošana, savienojums tiek pārtraukts un darbs notiek ar lokālajā datu kešatmiņā saglabātām datu kopijām.

Kategorijas, kas prasa pastāvīgu savienojumu ar datu bāzi, galveno funkcionalitāti definē klases no vārdu telpas **System.Data.Common**: **DbConnection**, **DbCommand**, **DataAdapter**, **DbDataReader**, **DbParameter** un **DbTransaction**. Patiesībā visas šīs klases ir abstraktas – faktisko funkcionalitāti realizē no šīm klasēm atvasinātajās klasēs, kuras ir paredzētas noteiktam datu avotam. Apakšklašu kopu, kas realizē piekļuvi specifiskam datu avotam sauc par datu piegādātāju (*data provider*).

Kategorijas, ka neprasa pastāvīgu savienojumu ar datu avotu, galveno funkcionalitāti definē klases no vārdu telpas **System.Data**: **DataSet**, **DataTable**, **DataRow**, **DataColumn** un **DataView**. Šajā un turpmākajos laboratorijas darbos uzsvars tiek likts tieši uz šo kategoriju. Tomēr jāņem vērā, ka šīs kategorijas klases savai funkcionalitātei izmanto arī klases no pirmās kategorijas – brīžos, kad nepieciešama datu nolasīšana vai saglabāšana.

Savienojuma izveidošanu ar datu avotu nodrošina klases **DbConnection** apakšklases. Lai izveidotu savienojumu ar datu avota instanci (piemēram, noteiktu datubāzi noteiktā datu bāzu vadības sistēmā), tiek izmantota informācija no savienojuma teksta virknes. Šo virkni uzstāda klases **DbConnection** īpašībā **ConnectionString**. Katram datu piegādātājam šī virkne ir specifiskā formātā. Daži piemēri populārākajiem datu avotiem:

Datu avota tips	**Savienojuma virknes formāts**
Microsoft Access	Provider=Microsoft.Jet.Oledb.4.0; Data Source=C:\Work\database.mdb
Microsoft Access 2000 vai jaunāks	Provider=Microsoft.ACE.OLEDB.12.0; Data Source=C:\Work\database.accdb; Persist Security Info=False;
SQL Server	Server=ServerName\InstanceName; Initial Catalog=DataBaseName; Trusted_Connection=True;
SQL Server Compact	Data Source=C:\Work\DataBase.sdf; Persist Security Info=False;

MySQL	Server=ServerAddress; Database=DataBase; Uid=Username; Pwd=Password;

Padoms: *Lai uzzinātu specifiskāku informāciju par katras savienojuma virknes parametriem, var izmantot, piemēram, tīmekļa resursā http://www.connectionstrings.com/ pieejamo informāciju.*

Darbs ar datu avotiem bez pastāvīga savienojuma (turpmāk tekstā – atvienotais režīms) notiek pēc principa, ka savienojums tiek izveidots automātiski uz īsiem laika sprīžiem, kad nepieciešams nolasīt datus vai tos saglabāt datu avotā – pārējā laikā datu manipulēšana notiek lokālajā datu avota kešatmiņā, kurā informācija tiek ielādēta no datu avota, un no kura tiek saglabātas izmaiņas datu avotā.

Protokolu ar datu avota kešatmiņu definē abstrakta datu adaptera klase **DataAdapter**, bet pašu lokālo kešatmiņu realizē klases **DataSet** un **DataTable**. Datu adapteris ir kā starpnieks starp datu avotu un datu avota lokālo kešatmiņu, piedāvājot metodi **Fill()**, kas izgūst datus no datu avota un saglabā tos lokālajā kešatmiņā, un metodi **Update()**, kas veic pretējo darbību, saglabājot lokālajā kešatmiņā izmainītos datus datu avotā:

DataSet <---> DataAdapter <---> Datu avots

Klasei **DataAdapter** ir definēta abstrakta apakšklase **DbDataAdapter**, kas ir domāta speciāli datu bāzu datu avotiem, un definē īpašības **SelectCommand**, **InsertCommand**, **DeleteCommand** un **UpdateCommand**, kuras iekapsulē komandas, kas tiks izmantotas attiecīgi pie datu izgūšanas, pievienošanas, dzēšanas vai izmaiņas. No klases **DbDataAdapter**, savukārt, ir atvasinātas klases darbam ar noteiktu sistēmu datu bāzu datu avotiem, piemēram, **SqlDataAdapter** (Microsoft SQL Server datu bāzēm), **OracleDataAdapter** (Oracle datu bāzēm), **OleDbDataAdapter** (jebkurai datu bāzei, kurai ir realizēts OLE DB draiveris, piemēram, Microsoft Access tipa datu bāzēm).

Lokālās datu kešatmiņas realizācija tiek definēta ar datu kopas klases **DataSet** palīdzību. Klase **DataSet** iekapsulē tabulu (klase **DataTable**) un relāciju (klase **DataRelation**) kolekcijas.

Klase **DataSet** definē šādas būtiskākās īpašības un metodes:

Īpašība/Metode()	**Apraksts**
HasChanges()	Atgriež pazīmi, vai datu kopā dati ir izmainīti (t.sk. pievienoti jauni vai dzēsti esošie).
Reset()	Atiestata datu kopas stāvokli uz tādu, kāds tas bija sākotnēji.
DataSetName	Atgriež/uzstāda datu kopas nosaukumu.
Relations	Atgriež relāciju kolekciju, kas definē datu kopā esošo tabulu savstarpējās saites.
Tables	Atgriež tabulu kolekciju, kas apraksta tabulas, kuras atrodas datu kopā.
ReadXml(), WriteXml()	Definē metodes datu kopā esošo datu saglabāšanai un ielasīšanai XML formātā.

Klase **DataTable**, kura ir arī lokālās datu kešatmiņas realizētāja, definē vienas datu avota tabulas atspoguļojumu:

Īpašība/Metode()	**Apraksts**
AcceptChanges()	Apstiprina visas tabulas lokālajā kešatmiņā līdz šim veiktās izmaiņas.
Clear()	Izdzēš no lokālās datu kešatmiņas visus tabulas rakstus.
NewRow()	Atgriež atsauci uz jaunu tabulas raksta instanci, kuru izmanto tās lauku vērtību aizpildīšanai. Faktisku pievienošanu tabulas kešatmiņas rakstu kolekcijai veic ar tabulas īpašības **Rows** metodes **Add()** izsaukšanu (un tam sekojošu metodes **AcceptChanges()** izsaukšanu).

ReadXml(), WriteXml()	Metodes tabulas kešatmiņā esošo datu saglabāšanai un ielasīšanai XML formātā.
RejectChanges()	Noraida visas tabulas kešatmiņā veiktās izmaiņas (līdz pēdējai metodes **AcceptChanges()** izsaukšanai).
Reset()	Atiestata tabulas kešatmiņas datus sākotnējā stāvoklī.
Select()	Atgriež tabulas kešatmiņas rakstu (apakš)kopu (kā klases **DataRow** instanču kolekciju), kas atbilst šīs metodes parametros norādītajai filtra izteiksmei. Filtra izteiksmē var izmantot tabulas lauku nosaukumus un to nosacījumus. Ir iespēja arī norādīt laukus pēc kuriem sakārtot atgriežamo rakstu kopu.
ChildRelations	Atgriež meitas saišu kolekciju (t.i. tabulas, kuru īpašībā **ParentRelations** ir norādīta šī tabula).
Columns	Atgriež tabulas lauku kolekcijas instanci (ar tipu **DataColumnCollection**).
Constraints	Atgriež tabulas lauku ierobežojumu kolekciju.
DataSet	Atgriež atsauci uz datu kopas instanci, kurā atrodas datu tabula.
ParentRelations	Atgriež tabulas mātes saišu kolekciju (t.i. tabulas, kuru īpašībā **ChildRelations** ir norādīta šī tabula).
PrimaryKey	Atgriež/uzstāda tabulas kolonnas, kuras definē tās primāro atslēgu.
Rows	Atgriež tabulas datu lokālajā kešatmiņā esošo rakstu kolekciju.
TableName	Atgriež/uzstāda tabulas nosaukumu datu kešatmiņā.

Katrs tabulas raksts tiek glabāts kolekcijas klasē **DataRowCollection**, kura nodrošina šādu funkcionalitāti:

Īpašība/Metode()	**Apraksts**
Add()	Rakstu kolekcijai pievieno jauna raksta instanci (ar tipu **DataRow**).
Clear()	Izdzēš visus tabulas rakstus no datu kešatmiņas.
Contains()	Ļauj noskaidrot, vai tabulas primārās atslēgas lauks (vai lauki) satur parametrā norādīto vērtību.
CopyTo()	Pārkopē tabulas rakstus vai nu uz masīva tipu, vai arī uz rakstu kolekcijas tipu, sākot ar parametrā norādīto raksta rindu.
Find()	Meklē un atgriež raksta instanci pēc parametrā norādītās primārās atslēgas vērtības.
IndexOf()	Atgriež parametrā norādītā raksta instances rindas numuru tabulas datu kešatmiņā.
InsertAt()	Ievieto jaunu raksta instanci norādītajā rindas numurā.
Remove(), RemoveAt()	Izdzēš norādīto raksta instanci no tabulas kešatmiņas.
Count	Atgriež rakstu skaitu tabulas datu kešatmiņā.
this[]	Ļauj atsaukties uz tabulas rakstu kolekciju kā masīvu, par indeksu izmantojot rindas numuru.

Katru tabulas rakstu datu kešatmiņā definē klases **DataRow** instances, kuras ļauj piekļūt raksta lauku vērtībām:

Īpašība/Metode()	**Apraksts**
AcceptChanges(), RejectChanges()	Apstiprina un noraida rakstā veiktās izmaiņas. Noraidīšana notiek tikai līdz pēdējai metodes **AcceptChanges()** izsaukšanai.
BeginEdit(), EndEdit()	Ieslēdz/izslēdz raksta lauku rediģēšanas režīmu. Rediģēšanas režīmā tiek atslēgti vairāki notikumi, kas ļauj vienlaicīgi mainīt vairāku tabulu rakstus un to laukus, neuztraucoties par pārbaudes trigeru nostrādāšanu.
CancelEdit()	Atceļ rakstā iesāktās izmaiņas un izslēdz rediģēšanas režīmu.
Delete()	Iezīmē rakstu kā dzēšamu. Faktiska izdzēšana tabulas datu kešatmiņā notiek tikai pēc **AcceptChanges()** izsaukšanas (vai arī tiek atcelta, ja tiek izsaukta metode **RejectChanges()**).
GetChildRows()	Atgriež meitas rakstu kolekciju, kas atbilst raksta primārajai atslēgai.
GetParentRows()	Atgriež mātes tabulas rakstu kolekciju.
IsNull() SetNull()	Atgriež pazīmi, vai parametrā norādītais lauks satur vērtību NULL. Uzstāda lauka vērtību uz NULL.
SetModified()	Uzstāda pazīmi, vai raksts ir modificēts.
SetParentRow()	Uzstāda saiti ar mātes raksta instanci.
HasErrors	Atgriež pazīmi, vai raksta lauki ir kļūdas stāvoklī.
this[]	Piekļūst raksta laukiem kā masīva elementiem pēc to indeksa vai nosaukuma. Raksta rediģēšanas režīmā indeksa otrajā parametrā var norādīt kuru lauka vērtību atgriezt – sākotnējo (**DataRowVersion.Original**), vai izmainīto (**DataRowVersion.Proposed**).

Vēlreiz jāuzsver, ka visas metodes, kas veic rakstu modificēšanu, to izpilda tikai datu kopas vai tabulas datu lokālajā kešatmiņā – faktiska datu aktualizēšana vai dzēšana no datu avota notiek tikai pēc datu adaptera metodes **Update()** izsaukšanas (tās parametrā norādot datu kopas vai tabulas instanci, kuras lokālo kešatmiņu nepieciešams aktualizēt).

Tabulas datu kešatmiņā katra lauka (kolonnas) īpašības tiek aprakstītas kā klases **DataColumn** instances:

Īpašība/Metode()	**Apraksts**
AllowDBNull	Atgriež/uzstāda pazīmi, vai tabulas lauks var saturēt vērtību NULL.
AutoIncrement, AutoIncrementSeed, AutoIncrementStep	Atgriež/uzstāda pazīmes, vai tabulas jauniem rakstiem lauka vērtība automātiski palielinās un ja jā tas par cik un no kādas vērtības sākot.
Caption	Atgriež/uzstāda lauku aprakstošu virsrakstu.
ColumnName	Atgriež/uzstāda lauka nosaukumu.
DataType	Atgriež/uzstāda lauka vērtības datu tipu.
Expression	Atgriež/uzstāda izteiksmi, kas tiek izmantota lauka vērtības aprēķināšanai. Izteiksmē var izmantot pārējo lauku nosaukumus, kā arī agregācijas funkcijas, kas darbojas uz noteiktu tabulas lauku.
MaxLength	Atgriež/uzstāda maksimāli pieļaujamo lauka vērtības garumu, ja tā tips ir teksta virkne.
Table	Atsauce uz tabulas instanci, kurā lauks ir definēts.

Unique	Atgriež/uzstāda pazīmi, vai lauka vērtības ir unikālas.

Vēl viena klase, kuru būtiski pieminēt, ir datu skata klase **DataView**. Tā papildina tabulas **DataTable** funkcionalitāti ar kārtošanas, filtrēšanas, meklēšanas (ne tikai pēc primārās atslēgas), rediģēšanas un navigācijas iespējām. Klases **DataView** instances par pamatu izmanto eksistējošu klases **DataTable** instanci, tāpēc datu kopijas klases **DataView** instancēs netiek glabātas – izgūstot datus, notiek griešanās pie **DataTable** instancē esošajiem faktiskajiem datiem.

Būtiskākās klases **DataView** īpašības un metodes:

Īpašība/Metode()	**Apraksts**
AddNew()	Pievieno un atgriež jaunu raksta **DataRowView** instanci. Pēc raksta lauku vērtību uzstādīšanas, jāizsauc atgrieztās instances metode **EndEdit()**.
AllowNew	Atgriež/uzstāda pazīmi, vai ir iespējams pievienot jaunus rakstus ar metodi **AddNew()**.
Delete()	Dzēš rakstu datu lokālajā kešatmiņā pēc parametrā norādītā rindas numura.
Find(), FindRows()	Meklē un atgriež rakstu pēc parametrā norādītās vērtības.
ToTable()	Atgriež **DataView** instanci atspoguļojošo rakstu kopijas datu tabulas veidā.
AllowDelete	Atgriež/uzstāda pazīmi, vai rakstus var dzēst.
AlowEdit	Atgriež/uzstāda pazīmi, vai rakstus var rediģēt.
Count	Atgriež rakstu skaitu, kas ir spēkā pēc **RowFilter** un **RowStateFilter** uzstādīšanas.
RowFilter	Atgriež/uzstāda tabulas rakstu filtrēšanas nosacījumu pēc rakstu lauku vērtībām.
RowStateFilter	Atgriež/uzstāda tabulas rakstu filtrēšanas nosacījumu pēc rakstu stāvokļiem.
Sort	Atgriež/uzstāda rakstu kārtošanas nosacījumus.
Table	Atgriež/uzstāda **DataView** pamatā izmantoto tabulas instanci.
this[]	Piekļūst datu skata rakstiem kā masīva elementiem. Katra **DataView** rinda tiek definēta kā klases **DataRowView** instance.

Kaut arī izmantojot apskatītās klases, var nodrošināt visbiežāk izmantoto funkcionalitāti, ADO.NET definē arī citas klases, informāciju par kurām, nepieciešamības gadījumā, var atrast izstrādes vides palīdzības dokumentācijā.

4.2 DATU AVOTU ATBALSTS VIZUĀLAJOS KOMPONENTOS

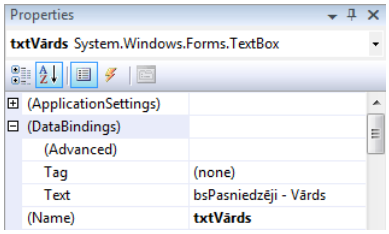
Šajā sadaļā tiek apskatītas datu avotu sasaistes iespējas ar kontrolēm un komponentiem.

Kaut gan datu izgūšanai no datu avotiem un to attēlošanai formu kontrolēs var izmantot iepriekšējā sadaļā aplūkotās ADO.NET klases un to metodes, tomēr kontrolēm ir realizēta piemērota funkcionalitāte, kas ļauj vienkāršot lielāko daļu no rutīnas darbībām. Turpmāk tiks apskatītas un, kur vien iespējams, izmantotas tieši automatizētās iespējas.

Katrai vizuālajai kontrolei ir iespēja piesaistīt datu avota lauku. Tas ir pateicoties tam, ka klase **Control**, kas ir visu vizuālo kontroļu bāzes klase, definē īpašību **DataBindings** ar tipu **ControlBindingsCollection**, kura, savukārt, satur klases **Binding** kolekciju. Klase **Binding** definē objekta atribūta un kontroles atribūta vērtības sasaistes vispārēju funkcionalitāti. Datu bāzes avota gadījumā ar objekta īpašību uztver datu bāzes tabulas vai datu skata lauku.

Sasaiste nodrošina, ka izmainoties kontroles atribūta vērtībai, tā tiks automātiski sinhronizēta ar tabulas lauka vērtību, un otrādi.

Īpašību logā kontroles atribūts **DataBindings** attēlojas ar kā **(DataBindings)**. Piemēram, ja formas redaktorā ir izvēlēta **TextBox** kontroles instance **txtVārds**, tad īpašību logā var būt tāds saturs:

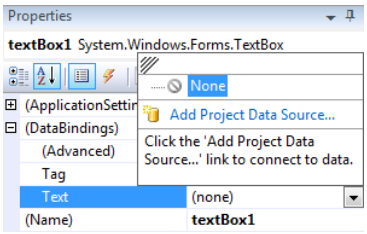


Attēlā redzams izvērsts atribūta **(DataBindings)** saturs ar šādiem apakšpunktiem:

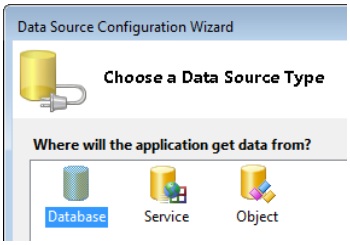
- **(Advanced)** – atver dialogu, kurā iespējams norādīt papildus nosacījumus saistīto īpašību sinhronizēšanai;
- Tālāk seko to kontroles īpašību uzskaitījums, kuru vērtības tiek sasaistītas ar datu avota lauku visbiežāk, kā arī papildus sasaistītie kontroles atribūti. Ja atribūta vērtība šajā uzskaitījumā ir **(none)**, tad atribūts nav sasaistīts datu avota lauku. Savukārt, ja ir vērtība formā *<sasaistes_avota_nosaukums>-<datu_lauka_nosaukums>*, tad tas nosaka, kuras datu tabulas kešatmiņas lauks ir piesaistīts kontroles vērtībai.

Lai kontroles atribūtam piesaistītu datu lauku, jārīkojas šādi:

- Zem īpašības **(DataBindings)** pie atbilstošās īpašības jānospiež kombinētā saraksta poga, atverot datu avota logu:

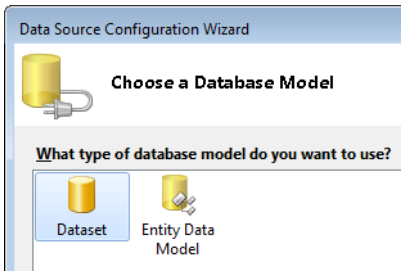


- Ja projektam vēl nav piesaistīts neviens datu avots, tad vienīgā iespēja to pievienot, ir nospiežot uz saites **Add Project Data Source...**, pēc kā atveras datu avotu konfigurācijas vednis, ar iespēju uzvēlēties datu avota veidu:



Datu bāzes datu avota gadījumā jāizvēlas **Database** un jāspiež poga **Next>**.

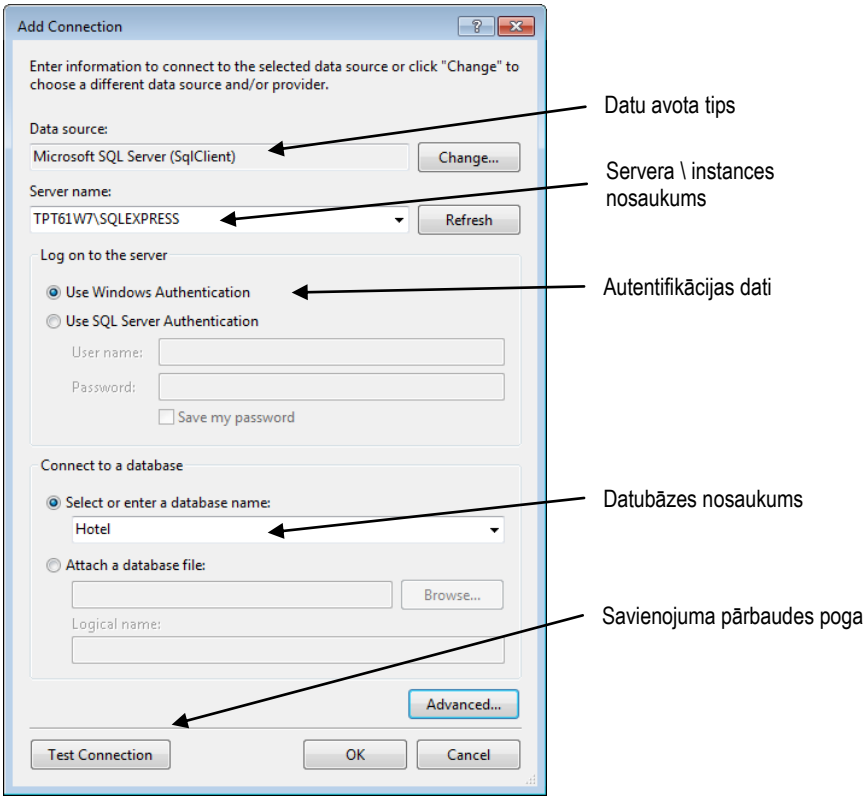
- Solis attiecas tikai uz Microsoft Visual Studio 2010 vai jaunāku versiju, iepriekšējās izstrādes vides versijās tas netiek piedāvāts:



Šajā solī jāizvēlas datu bāzes modeļa veids:

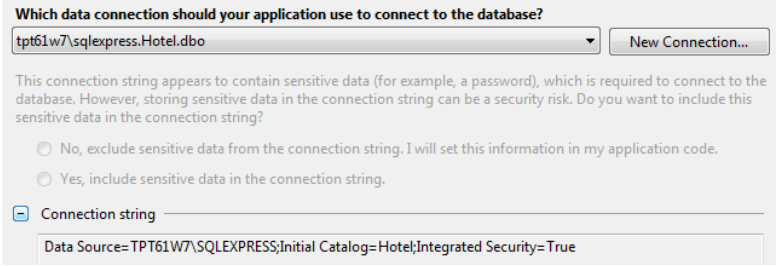
- **Dataset** – projektam tiks pievienots datu kopas artefakts. Šī tipa modelis tiek izvēlēts automātiski iepriekšējās izstrādes vides versijās;
- **Entity Data Model** – datu kopai par pamatu tiks izmantotas **Entity Framework** klases, kas nodrošina elastīgāku datu avota elementu kartēšanu klasēs un objektos. (Šajā laboratorijas darbā netiek apskatīta.)

- Nākošajā logā jāizvēlas noteikts datu avots un tā piekļūšanas nosacījumi. Ja nepieciešamais datu avots izstrādes vidē nav bijis iepriekš reģistrēts, tad jāspiež poga **New Connection...** un jānorāda nepieciešamā datu avota piekļuves nosacījumi, kas katram datu avotu veidam ir specifiski. Piemēram, piekļuves nosacījumu dialogs Microsoft SQL Server datu bāzēm ir šāds:

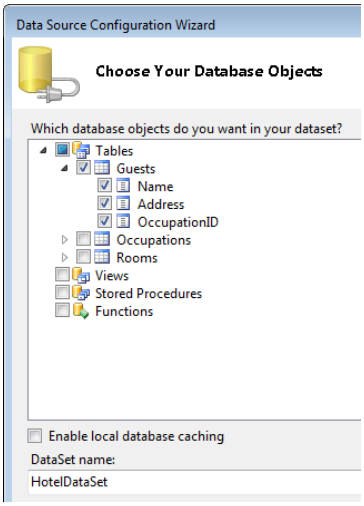


- Norādot visus prasītos parametrus un, ar pogas **Test Connection** nospiešanu, pārliecinoties par veiksmīgu savienojuma izveidošanu, dialogu aizver nospiežot pogu OK. (Turpmāk, kā piemērs, tiks izmantota hipotētiska viesnīcas uzskaites sistēmas datu bāze, kurā tiek uzglabāta informācija par viesiem, viesnīcas numuriem un rezervācijām.)

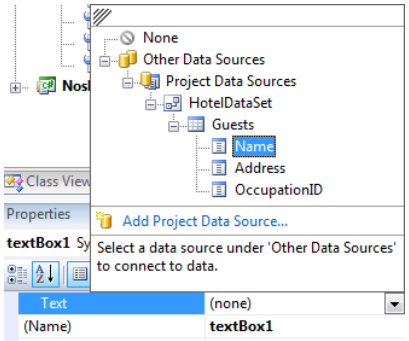
- Pēc savienojuma dialoga aizvēršanas, kombinētajā sarakstā tiek norādīta jaunizveidotā saite, bez zemāk – tai atbilstoša savienojuma virkne:



- Nospiežot pogu **Next>** tiek piedāvāts saglabāt savienojuma virkni kā lietojumprogrammas parametru, ko parasti arī dara.
- Nākamajā solī jāizvēlas datubāzē pieejamos objektus, kuriem nepieciešams piekļūt no lietojumprogrammas, kā arī datu kopas klases nosaukumu, caur kuras instanci tiks organizēta piekļuve izvēlētajiem datubāzes objektiem:



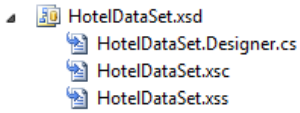
- Nospiežot pogu **Finish**, vedņa dialogs aizveras.
- Pēc datu avota saites un datu kopas instances izveidošanas, ir iespēja pabeigt datu avota tabulas lauka piesaisti formas kontrolei – izvēršot pievienoto datu kopu **HotelDataSet** līdz pat nepieciešamam laukam **Name** un to izvēloties:



Pēc tabulas lauka izvēles, kontroles **textBox1** datu sasaistes vērtība atribūtam **Text** kļūst par 'guestsBindingSource – Name'.

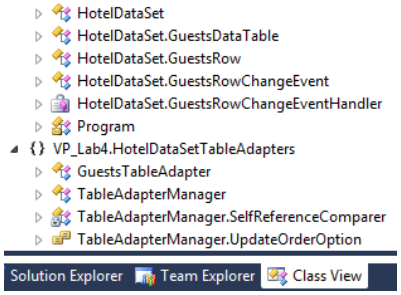
- Šāda soļu secība jāatkārto katrai formas kontrolei, tikai, ja kontroles atribūta sasaiste notiek caur to pašu datu kopu, sākot no iepriekšējā punkta.

Apskatīsim detalizētāk, kādas vēl izmaiņas, pēc datu avota piesaistīšanas, automātiski ir veiktas projektā. Pirmkārt, projekta pārlūka logā ir ievietotas jaunas virsotnes, kas apraksta datu avota piesaistes vednī izvēlētos datubāzes objektus:



Virsotnē ar paplašinājumu **.xsd** tiek saglabāta piesaistīto tabulu entīšu-saišu diagramma, kurā redzamas arī saites starp tabulām, ja tādas pastāv. Zem tās, virsotnē ar paplašinājumu **.Designer.cs** ir automātiski noģenerēts C# pirmkoda fails, kurā definētas vairākas klases, caur kuru instanču metodēm tiek realizēta piekļuve izvēlētajam datu avotam. Failos ar paplašinājumu **.xsc** un **.xss** tiek glabāta informācija par piesaistīto tabulu diagramma elementu izvietojumu un parametriem.

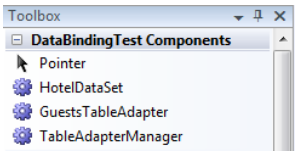
Atverot klašu skata logu, redzams, ka projektā ir izveidotas vairākas jaunas klases:



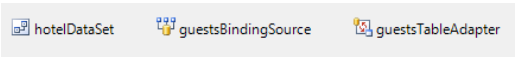
Jaunizveidoto klašu funkcijas ir šādas:

Klase	**Apraksts**
Klases, kas realizē datu kopu:	
HotelDataSet	Klases **DataSet** apakšklase, kas iekapsulē izvēlētos datu avota objektus.
GuestsDataTable	Klases **DataTable** apakšklase, kas iekapsulē izvēlēto datubāzes objektu – tabulu Guests.
GuestsRow	Klases **DataRow** apakšklase, kas iekapsulē tabulas Guests rakstu. Šajā klasē katrs tabulas Guests laukam ir iekapsulēts īpašībā, caur kuru var nolasīt un izmainīt lauka vērtību.
GuestsRowChangeEvent, GuestsRowChangeEventHandler	Klases **EventArgs** apakšklase, kas iekapsulē rindas izmainīšanas notikuma parametru, ļaujot caur to piekļūt izmainītās rindas instancei, kā arī delegāts (atsauce uz funkciju) šī notikuma apstrādāšanai.
Vārdu telpā HotelDataSetTableAdapters ir definēti adapteri darbam ar datu avota objektiem:	
GuestsTableAdapter	Komponents (t.i. klases **Component** pēctecis), kas iekapsulē tabulas Guests adapteri, nodrošinot komandas datu izgūšanai, modificēšanai, dzēšanai un saglabāšanai.
TableAdapterManager	Komponents, ar mērķi vienkāršot vairāku tabulu adapteru pārvaldību, koordinējot datu kopas sadarbību ar tabulu adapteriem, lai nodrošinātu hierarhisku datu aktualizēšanas scenāriju (piem., vispirms tiek saglabāti primārās atslēgas tabulas raksti, un tikai pēc tam pakārtoto tabulu raksti).

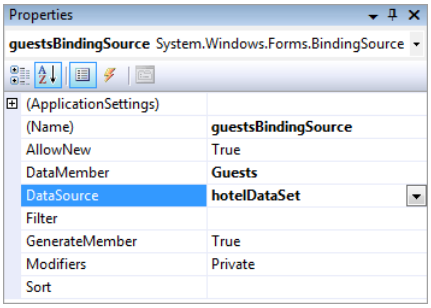
Uz to ka projektā ir izveidoti jauni komponenti liecina tas, ka pēc projekta nokompilēšanas rīku logā tiek parādītas jaunas komponentu ikonas:



Bet formas redaktora nevizuālo komponentu laukumā tiek ievietotas šādas komponentu instances:



Komponenti **hotelDataSet** un **guestsTableAdapter** ir attiecīgi datu kopas **HotelDataSet** un datu bāzes tabulas adaptera instances, bet komponents **guestsBindingSource** ir kategorijas **Data** komponenta **BindingSource** instance, kura tiek izmantota kā starpnieks starp datu kopu **hotelDataSet** un formā izvietotajām kontrolēm, uz ko norāda šī komponenta īpašību loga saturs:



Komponents **BindingSource** nodrošina interfeisu navigācijai pa piesaistītā datu avota (jeb vispārīgā gadījumā - saraksta) elementiem. Būtiskākās šī komponenta īpašības un metodes:

Īpašība/Metode()	**Apraksts**
MoveNext() MovePrevious() MoveFirst() MoveLast()	Izmaina tekošo datu avota elementu, pārejot attiecīgi uz nākošo, iepriekšējo, pirmo vai pēdējo.
Position	Atgriež/uzstāda tekošo datu avota elementu pēc tā indeksa. Pirmajam elementa atbilst indekss 0.
Count	Atgriež datu avotā esošo elementu skaitu.

Current	Atgriež tekošā elementa instanci. Ja piesaistītajā datu avotā nav neviena elementa, tad atgrieztā vērtība ir null. Lai noskaidrotu faktisko instances tipu var izmantot metodes GetType() vai ToString(). Ja avots ir datu kopa **DataSet**, tad instance ir ar tipu **DataRowView**, piem.:
	`DataRowView drv = guestsBindingSource.Current as DataRowView;`
	Pašai raksta instancei piekļūst caur **DataRowView** īpašību **Row**, piem.:
	`HotelNamespace.HotelDataSet.GuestsRow gr = drv.Row as HotelNamespace.HotelDataSet.GuestsRow;`

DataSource	Atgriež/uzstāda datu avota instanci.
DataMember	Atgriež/uzstāda datu avota instances tabulu/sarakstu, pa kuru arī notiks navigācija.
Filter	Uzstāda datu avota elementu filtrēšanas nosacījumu, piem.: `guestsBindingSource.Filter = "Name = 'Jānis Bērziņš'";` Datu bāzu avota gadījumā par filtra izteiksmi var izmantot faktiski tādas pašas izteiksmes, kādas pieejamas SQL vaicājuma nosacījuma WHERE daļā.
Sort	Atgriež/uzstāda piesaistīto datu avotu elementu sakārtošanas nosacījumus.
Add()	Pievieno datu avotam parametrā norādīto elementu.
AddNew()	Atgriež jauna elementa instanci.
EndEdit()	Apstiprina tekošajā elementā veiktās izmaiņas datu avotā.
CancelEdit()	Atceļ visas līdz šim tekošajā elementā veiktās izmaiņas.

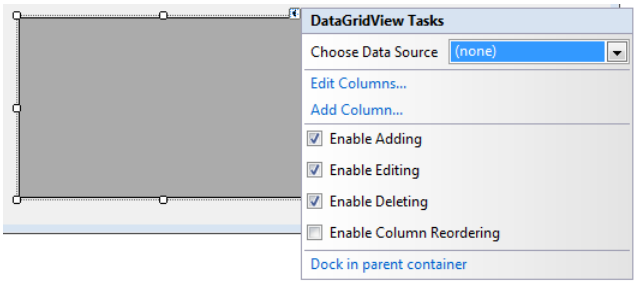
Kā minēts no tabulā uzskaitītajām īpašībām, izmantojot komponentu **guestsBindingSource**, datu tabulas vai skata kešatmiņas rakstiem var norādīt papildus filtrēšanas (atribūts **Filter**) un kārtošanas (atribūts **Sort**) nosacījumus.

Visbeidzot, formas notikuma apstrādātājā **Load** automātiski tiek ievietots pirmkods, kas izgūst un ielādē datus no datu bāzes tabulas Guests un saglabā tos datu tabulas lokālajā kešatmiņā:

```
guestsTableAdapter.Fill(hotelDataSet.Guests);
```

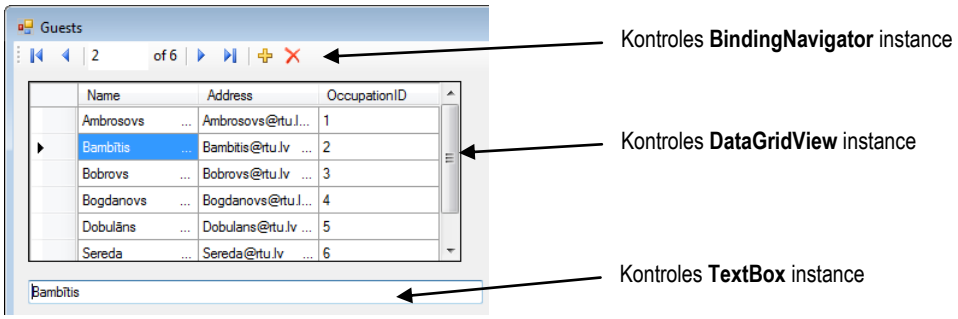
Izstrādes vides komponentu loga kategorijā **Data** ir pieejami vēl divi līdz šim neapskatīti komponenti – **DataGridView** un **BindingNavigator**.

Kontrole **DataGridView** nodrošina plašu funkcionalitāti datu atspoguļošanai tabulas veidā, ar iespēju veikt rakstu modificēšanu, pievienošanu un dzēšanu. Uzvietojot kontroli uz formas, tiek uzreiz piedāvāts piesaistīt datu avotu, un norādīt rakstu labošanas nosacījumus:



Kombinētā saraksta laukā **Choose Data Source** var izvēlēties starp jau formā esošajiem datu avotiem (komponenta **BindingSource** instancēm) vai jauna avota pievienošanu. Jebkurā gadījumā, pēc datu avota izvēles, kontrolē automātiski tiks izveidoti datu avota tabulas lauki, bet, palaižot programmu uz izpildi un parādot formu – kontrolē tiks parādīti faktiskie tabulas dati. Tas viss tiek nodrošināts bez nevienas manuālas pirmkoda rindiņu rakstīšanas nepieciešamības!

Otra neapskatītā kontrole – **BindingNavigator** nodrošina rīku joslu, ar kontrolēm, kas ļauj pārvietoties pa datu tabulas rakstiem kešatmiņā, kā arī pievienot jaunu un izdzēst esošos. Lai iegūtu minēto funkcionalitāti, atliek tikai uzvietot kontroles **BindingNavigator** instanci uz formas virsmas un kontroles atribūta **BindingSource** vērtībā norādīt esoša komponenta **BindingSource** instanci. Piemērs, kā var izskatīties forma palaišanas režīmā ar tās kontrolēm piesaistītiem datu avotiem:



Vēlreiz jāuzsver, ka šādu funkcionalitāti var realizēt bez nevienas manuālas pirmkoda rindiņas rakstīšanas nepieciešamības.

Pēc tam, kad ir veiktas nepieciešamās manipulācijas ar datu avota lokālo kešatmiņu, datu aktualizēšanu pašā datu avotā veic, piemēram, ar šādu izteiksmi:

```
guestsTableAdapter.Update(hotelDataSet.Guests);
```

Tabulas adaptera metode **Update()** ir pārlādēta, ļaujot saglabāt gan atsevišķa raksta instanci, gan rakstu instanču kopu, gan datu kopu, gan datu tabulu, gan arī norādīt raksta visu lauku vērtības.

Laboratorijas darbā datu avotu pielietošanas scenāriji tiks demonstrēti ar SQL Server Compact datu bāzu vadības sistēmu. Šī sistēma ir vienkāršota Microsoft SQL Server versija, kas paredzēta izmantošanai vienlietotāja lietotājprogrammās vai mobilajās ierīcēs ar maksimālo datu bāzes izmēra ierobežojumu līdz 4GB. SQL Server Compact ir failu orientēta – visa informācija par datubāzi un tajā esošajiem datiem tiek glabāta vienā datubāzes failā ar paplašinājumu **.sdf**. Līdz ar to, lai piekļūtu datu bāzei, nepieciešamas tikai tiesības uz atbilstošo datu bāzes failu.

Vēl viena priekšrocība, kāpēc demonstrēšanas nolūkiem ir ērti izmantot tieši SQL Server Compact DBVS ir tā, ka tās administrēšanai nav nepieciešamas administratora grupas tiesības, bet izstrādes vide piedāvā ērtus grafiskos līdzekļus tabulu un relāciju izveidošanai. Izstrādes vide piedāvā arī līdzīgus vizuālos rīkus, tikai ar papildus iespējām, Microsoft SQL Server 2000 un jaunākām versijām.

Nobeigumā apskatīsim SQL Server Compact datubāzes datu tipu attēlojums biežāk izmantotajos C# datu tipos:

Datu bāzes lauka datu tips	**C# datu tips**	**Vērtību diapazons**
bit	bool	false (0), true (1)
tinyint	byte	0..255
datetime	datetime	No 1753. gada 1. janvāra līdz 9999. gada 31. decembrim ar precizitāti līdz ~3.33 milisekundēm.
numeric	decimal	$-10^{38}+1$ .. $10^{38}-1$
float	double	–1.79E+308 .. 1.79E+308
integer	int	-2^{31} .. $2^{31}-1$
nvarchar	string	0 .. 4000 rakstzīmes

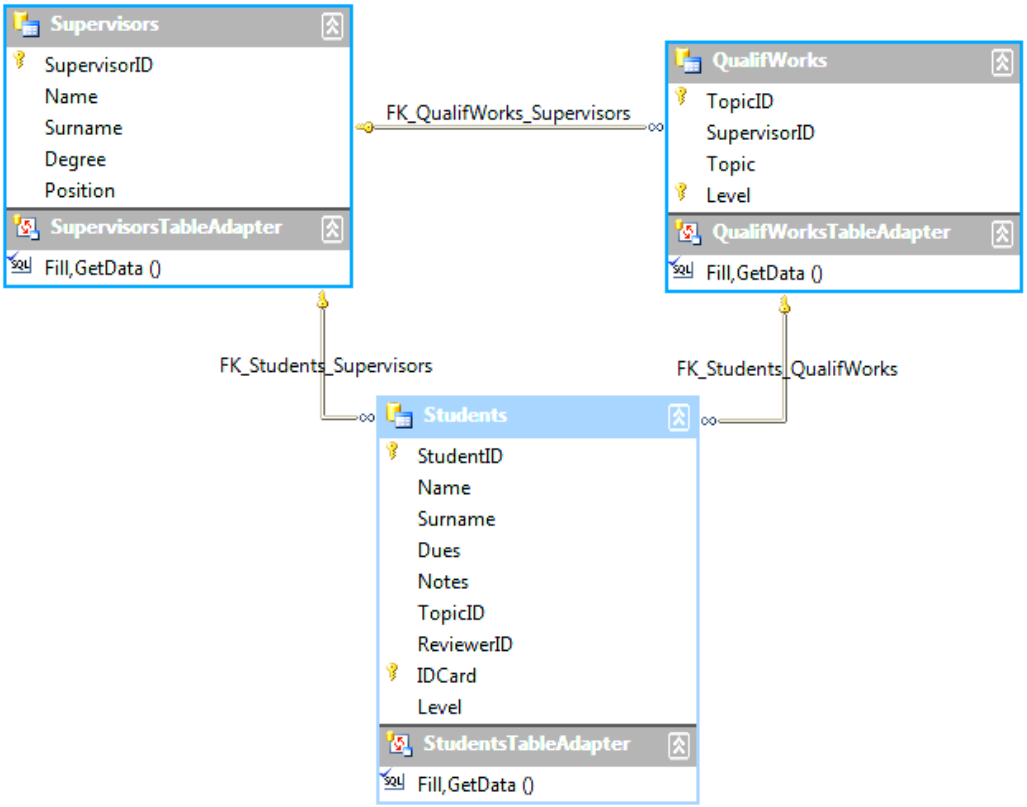
Padoms: *Detalizētāks iespējamo datubāzes tabulu lauku tipu apraksts pieejams, piemēram, http://technet.microsoft.com/en-us/library/ms172424.aspx.*

4.3 UZDEVUMS PASNIEDZĒJA VADĪBĀ

Izstrādāt noslēguma (bakalaura, maģistra) darbu uzskaites lietojumprogrammu, kas ļautu apstrādāt informāciju par:

- studentiem;
- studentu izstrādātajiem noslēguma darbiem;
- noslēguma darbu vadītājiem;
- vadītāju piedāvātajām noslēguma darbu tēmām;
- sasaisti starp studentiem, to izstrādātajām tēmām, noslēguma darbu vadītājiem un recenzentiem.

Minēto informāciju uzglabāt relāciju datu bāzē, ar šādu struktūru:



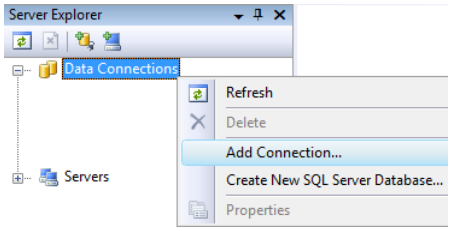
Piezīme: Papildus šīs lietojumprogrammas prasības tiks izvirzītas turpmākajos laboratorijas darbos.

Šim nolūkam, izstrādes vidē izveidot jaunu projektu 'DataModel' ar sagatavi **Class Library**, pašu risinājumu nosaucot par 'QualificationWorks' (t.i. noslēguma darbi). Šī projekta mērķis būs uzturēt artefaktus, izmantojot kurus, notiks piekļūšana datu avotam – t.i. datu pieejas slānis (*data access layer*).

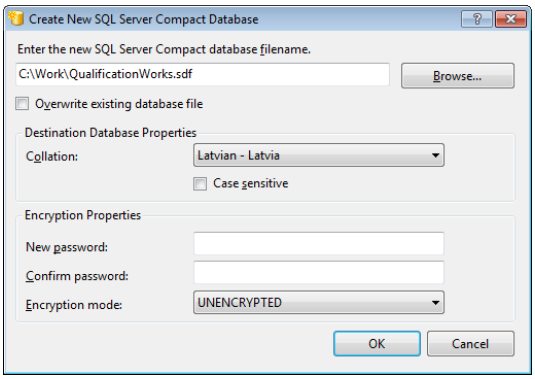
Projekta pārlūkā no projekta **DataModel** izdzēst pēc noklusēšanas izveidoto klasi **Class1.cs**.

Izveidot jaunu lokālo (t.i. SQL Server Compact) datubāzes failu:

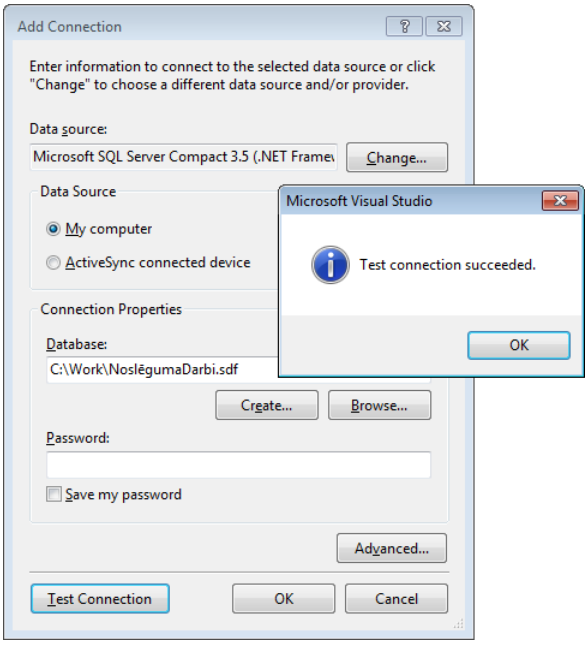
- Izstrādes vides serveru pārlūka logā (atvērt ar galvenās izvēlnes komandu **View | Server Explorer**) uz virsotnes **Data Connections** konteksta izvēlnes izvēlēties punktu **Add Connection...** :



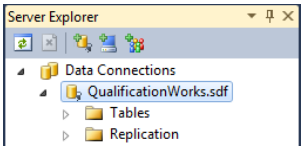
- Dialoga laukā **Data source** izvēlēties 'Microsoft SQL Server Compact 3.5'
- Nospiest zemāk esošo pogu **Create...** un dialogā aizpildīt laukus ar datubāzes vārdu (C:\Work\QualificationWorks.sdf) un salīdzināšanas nosacījumu (Latvian – Latvia):



- Ar OK aizvērt jaunas datubāzes faila izveides dialogu, apstiprinot, ka noklusēto paroli (kas ir tukša) nav nepieciešams mainīt.
- Datubāzes savienojuma izveides dialogā nospiest pogu **Test Connection** un pārliecināties, ka savienojums ir veiksmīgs:

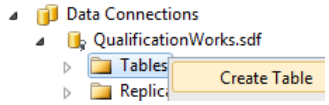


- Ar OK aizvērt savienojuma izveidošanas dialogu. Serveru pārlūka logā tiks parādīta jaunizveidotā saite uz datubāzes failu:



Jaunizveidotajā datubāzē **QualificationWorks.sdf**, izveidot jaunu tabulu **Supervisors** (noslēguma darbu vadītāji):

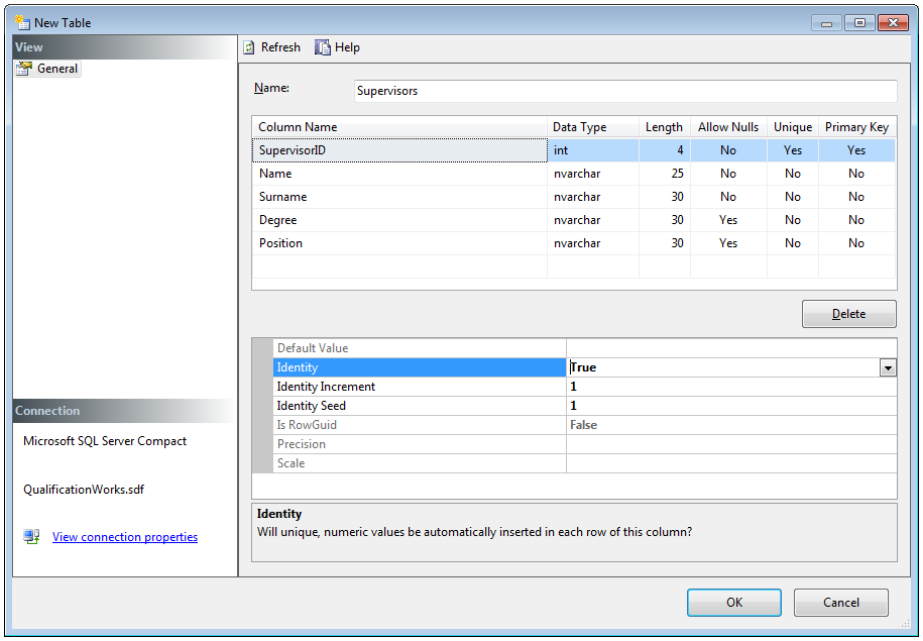
- Serveru pārlūkā izvērst virsotni **QualificationWorks.sdf** un uz apakšvirsotnes **Tables** konteksta izvēlnes izvēlēties punktu **Create Table**:



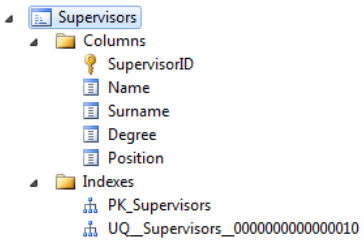
- Jaunas tabulas **Supervisors** izveides dialogu aizpildīt ar šādu lauku definīcijām:

Lauks	**Tips**	**Garums**	**Nulls?**	**Unikāls?**	**Primārā atslēga?**	**Citi**
SupervisorID	int	4	No	Yes	Yes	Identity = True
Name	nvarchar	25	No	No	No	
Surname	nvarchar	30	No	No	No	
Degree	nvarchar	30	Yes	No	No	
Position	nvarchar	30	No	No	No	

Ievērot, ka laukam **SupervisorID** detalizētas informācijas sarakstā īpašība **Identity** ir jāuzstāda uz **True**, bet īpašības **IdentityIncremenent** un **IdentitySeed** uz 1. Pārliecināties, ka visiem pārējiem tabulas laukiem īpašība **Identity** vērtība ir **False** :



- Ar OK apstiprināt tabulas **Supervisors** izveidošanu – jaunizveidotā tabula tiks attēlota servera pārlūka logā zem virsotnes **Tables**:



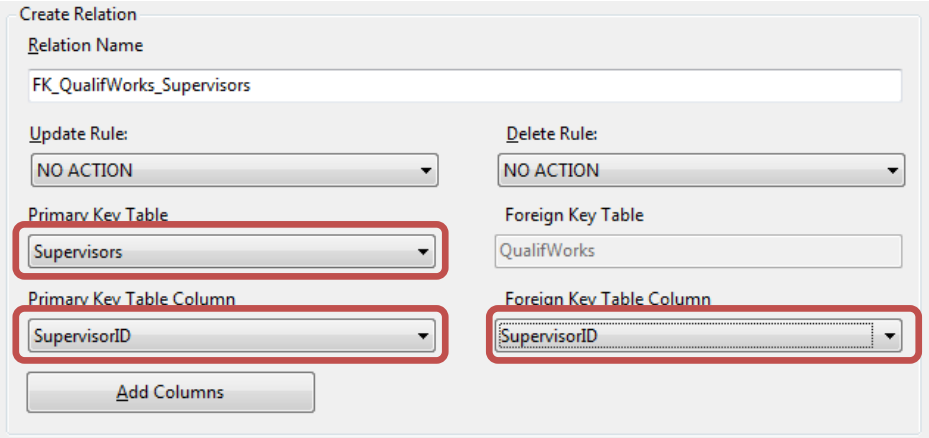
Redzams, ka papildus izveidotajiem laukiem, automātiski ir izveidoti indeksi **PK_Supervisors** un **UQ_Supervisors_**, kas atbilst primārās atslēgtas laukam **SupervisorID**, nodrošinot tā vērtību unikalitāti.

Līdzīgā veidā izveidot tabulu **QualifWorks** (noslēguma darbu tēmas), ar šādu specifikāciju:

Lauks	Tips	Garums	Nulls?	Unikāls?	Primārā atslēga?	Citi
TopicID	int	4	No	Yes	Yes	Identity = True
SupervisorID	int	4	No	No	No	
Topic	nvarchar	200	No	No	No	
Level	int	4	No	No	No	

Starp tabulām **Supervisors** un **QualifWorks** izveidot relāciju 'viens-pret-daudziem':

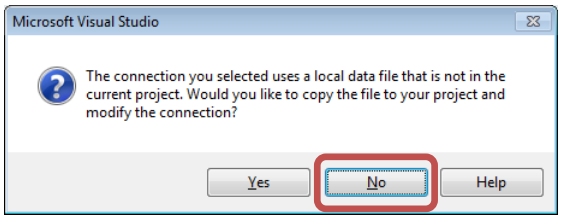
- Servera pārlūka logā uz tabulas **QualifWorks** virsotnes (t.i. no veidojamās relācijas 'daudziem' puses) konteksta izvēlnes izvēlēties punktu **Table Properties**;
- Dialogā, kas tiks atvērts, izvēlēties kategoriju **Add Relations** un aizpildīt to ar šādu informāciju (saites nosaukumu formējot no *FK_ārējās-atslēgas-tabula_primārās-atslēgas-tabula*, t.i. 'FK_QualifWorks_Superviors'):



- Nospiest pogu **Add Columns** un pēc tam pogu **Add Relation**;
- Ar OK nospiešanu, apstiprināt relācijas izveidošanu, un pēc tam ar OK nospiešanu aizvērt relācijas izveidošanas dialogu.

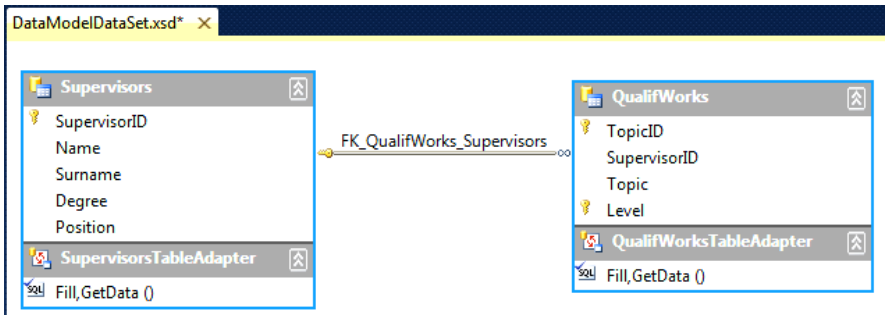
Projektam **DataModel** pievienot jaunu **DataSet** artefaktu:

- Projekta pārlūka logā uz projekta **DataModel** nosaukuma virsotnes konteksta izvēlnes izvēlēties punktu **Add | New Item...** un dialogā izvēlēties datu kopas sagatavi **DataSet**;
- Datu kopu nosaukt par 'DataModelDataSet.xsd' un ar pogu **Add** apstiprināt tās pievienošanu projektam. Tiek automātiski atvērts tukšs datu kopas projektēšanas logs (**Dataset Designer**);
- No servera pārlūka loga ar peli uz datu kopas projektēšanas logu uzvilkt tabulas **Supervisors** un **QualifWorks**;
- Ja, atlaižot peli datu kopas projektēšanas logā, tiek izvadīts ziņojums:

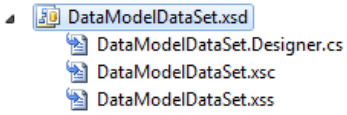


Tad tajā obligāti jāizvēlas poga **No**. Tas nepieciešams tāpēc, lai izstrādes vide datu bāzi nevis iekļautu projekta artefaktos, bet gan tikai atsauktos uz C:\Work\QualificationWorks.sdf;

- Izvietojot abas tabulas datu kopas projektētāja logā, automātiski tiks norādīta arī iepriekš izveidotā saite 'viens-pret-daudziem':



- Pēc tabulu uzvietošanas, projekta pārlūka logā tiek izveidotas jaunas virsotnes:



- Bet, pats būtiskākais ir tas, ka pēc tabulu norādīšanas datu kopa projektēšanas logā, un pēc tam sekojošas projekta nokompilēšanas, automātiski tiek izveidotas C# klases darbam ar uzvietotajām tabulām **Supervisors** un **QualifWorks**, no kurām būtiskākās ir:
  - SupervisorsDataTable
  - QualifWorksDataTable
  - SupervisorsRow
  - QualifWorksRow

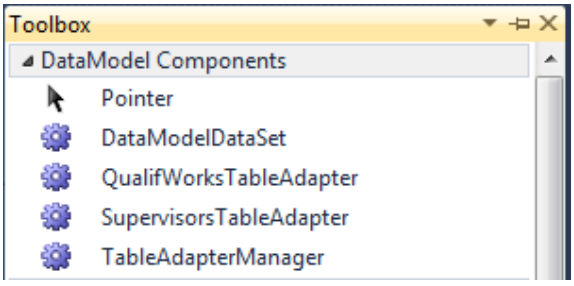
Šīs klases iekapsulē, attiecīgi tabulas un tabulas rakstu abstrakcijas. Pie tam šo klašu realizācijas nav triviālas – to metodēs ir realizētas dažādas papildus pārbaudes, lai nerastos kļūdas vai datubāzes nekonsistence.

Risinājumam **QualificationWorks** pievienot jaunu projektu ar sagatavi **Windows Forms Application**, kuru nosaukt par 'QualifWorksClient', un kurš realizēs prezentācijas un biznesa loģikas slāņus.

Jaunizveidoto projektu **QualifWorksClient** uzstādīt kā sāknēšanas projektu (**StartUp Project**), bet tā atsaucēs norādīt iepriekš jau izveidoto projektu **DataModel**.

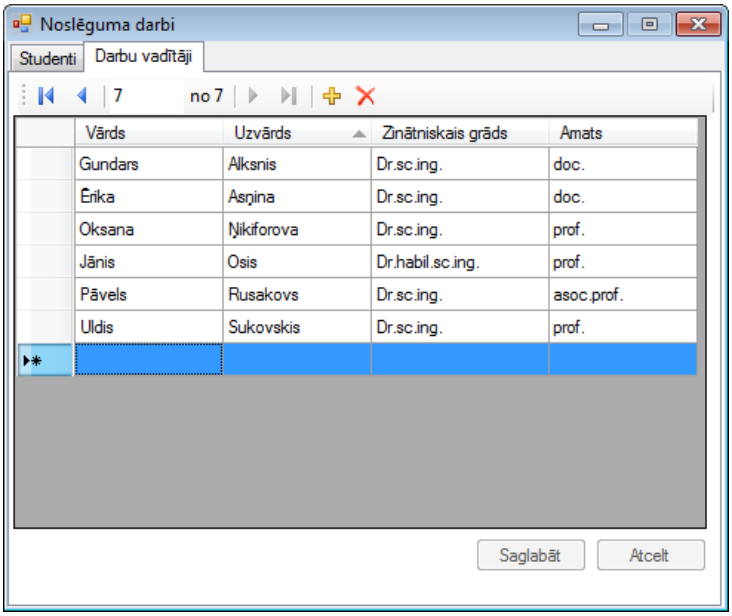
Nokompilēt risinājumu, lai pārliecinātos, ka tajā nav kļūdas!

Ievērot, ka pēc risinājuma veiksmīgas nokompilēšanas, projekta **QualifWorksClient** komponentu logā kļūst pieejami jauni komponenti:



Lai no projekta **QualifWorksClient** varētu piekļūt tabulām, kas ir izveidotas datu bāzē QualificationWorks.sdf, tiks izmantoti tieši šie komponenti.

Projektā **QualifWorksClient** realizēt saskarni darbam ar darbu vadītāju tabulu, ar šādu gala rezultātu:



Šim nolūkam, atvērt formas **Form1.cs** redaktoru un uzstādīt īpašības:

Īpašība:	**Vērtība:**
(Name)	frmQualifWorks
FormBorderStyle	FixedSingle
MaximizeBox	False
Text	Noslēguma darbi
Size	490; 405

Risinājuma pārlūkā pārsaukt failu **Form1.cs** par **frmQualifWorks.cs**.

Uz formas **frmQualifWorks** uzvietot kategorijas **Containers** kontroli **TabControl** un uzstādīt īpašības:

Īpašība:	**Vērtība:**
(Name)	tbcQualifWorks
Dock	Fill

Kontrolē **tbcQualifWorks** divām izveidotajām cilnēm uzstādīt īpašības:

Īpašība:	**Vērtība:**
(Name)	tbpSupervisors
Text	Darbu vadītāji

Īpašība:	**Vērtība:**
(Name)	tbpStudents
Text	Studenti

Uz formas komponentu laukuma uzvietot komponentes **DataModelDataSet**, **SupervisorsTableAdapter** un **QualifWorksTableAdapter**, un pārdēvēt tās attiecīgi par **dsDataModel**, **taSupervisors** un **taQualifWorks**.

Uz formas komponentu laukuma uzvietot kategorijas **Data** komponentu **BindingSource** un uzstādīt īpašības (ievērot, ka īpašību **DataSource** nepieciešams uzstādīt pirms īpašības **DataMember** uzstādīšanas):

Īpašība:	**Vērtība:**
(Name)	bsSupervisors
DataSource	dsDataModel
DataMember	Supervisors
Sort	Surname

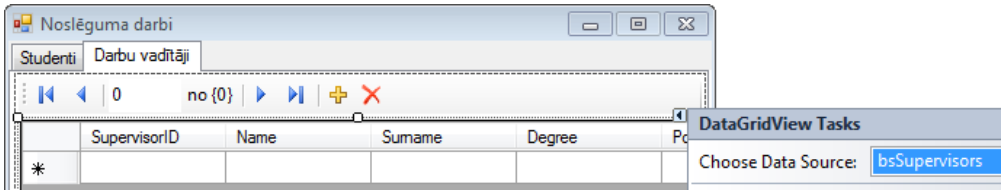
Uz formas cilnes **tbpSupervisors** uzvietot kontroli **BindingNavigator** un uzstādīt īpašības:

Īpašība:	**Vērtība:**
(Name)	bnSupervisors
BindingSource	bsSupervisors
CountItemFormat	no {0}

Uz cilnes **tbpSupervisors** uzvietot kontroli **DataGridView** un uzstādīt īpašības:

Īpašība:	**Vērtība:**
(Name)	dgvSupervisors
Anchor	Top, Bottom, Left
DataSource	bsSupervisors
Location	0; 28
MultiSelect	False
SelectionMode	FullRowSelect
Size	468; 279

Ievērot, ka pēc šīm darbībām formas redaktorā komponenta **dgvSupervisors** ietvaros tiek automātiski izveidotas un parādītas tabulas **Supervisors** kolonnas:



Izmantojot komponenta **dgvSupervisors** īpašību **Columns**, atvērt kolonnu redaktora dialogu un veikt šādus labojumus:

- Izvēloties kolonnu **SupervisorID**, nospiest pogu **Remove**, lai izņemtu to no rādāmo kolonnu kopas;
- Izvēloties kolonnu **Name**, uzstādīt īpašības:

Īpašība:	**Vērtība:**
HeaderText	Vārds

- Izvēloties kolonnu **Surname**, uzstādīt īpašības:

Īpašība:	**Vērtība:**
HeaderText	Uzvārds

- Izvēloties kolonnu **Degree**, uzstādīt īpašības:

Īpašība:	**Vērtība:**
ColumnType	DataGridViewComboBoxColumn
DisplayStyle	ComboBox
DisplayStyleForCurrentCellOnly	True

HeaderText	Zinātniskais grāds
Items	Atvērt elementu saraksta dialogu un norādīt šādas rindas: Dr.habil.sc.ing. Dr.sc.ing. Mg.sc.ing.
Width	120

- Izvēloties kolonnu **Position**, uzstādīt īpašības:

Īpašība:	**Vērtība:**
ColumnType	DataGridViewComboBoxColumn
DisplayStyle	ComboBox
DisplayStyleForCurrentCellOnly	True
HeaderText	Amats
Items	Atvērt elementu saraksta dialogu un norādīt: akad. asist. asoc.prof. doc. lekt. prof.

Uz cilnes **tbpSupervisors** uzvietot kontroli **Button** un uzstādīt īpašības:

Īpašība:	**Vērtība:**
(Name)	btnUpdateSupervisors
Text	Saglabāt
Location	312; 313

Pogai **btnUpdateSupervisors** realizēt notikuma **Click** apstrādātāju (un aiz tā definēto metodi **UpdateSupervisors()**), kas ļauj saglabāt datu kopā veiktās izmaiņas datu bāzē:

```
private void btnUpdateSupervisors_Click(object sender, EventArgs e)
{
    UpdateSupervisors("Vai tiešām saglabāt veiktās izmaiņas?");
}

private DialogResult UpdateSupervisors(string message)
{
    DialogResult result = MessageBox.Show(message
                                          , "Darbu vadītāju datu saglabāšana"
                                          , MessageBoxButtons.YesNoCancel
                                          , MessageBoxIcon.Question);
    if (DialogResult.Yes == result)
    {
        //atceras tekošā raksta pozīciju
        int index = bsSupervisors.Position;
        taSupervisors.Update(dsDataModel. Supervisors);
        taSupervisors.Fill(dsDataModel. Supervisors);
        //atjauno tekošā raksta pozīciju
        bsSupervisors.Position = index;
    }

    return result;
}
```

Uz cilnes **tbpSupervisors** uzvietot kontroli **Button** un uzstādīt īpašības:

Īpašība:	**Vērtība:**
(Name)	btnCancelSupervisors
Text	Atcelt

Location	393; 313

Pogai **btnCancelSupervisors** realizēt notikuma apstrādātāju **Click**, kas atceļ datu kopas lokālajā kešatmiņā līdz šim veiktās un nesaglabātās izmaiņas:

```
private void btnCancelSupervisors_Click(object sender, EventArgs e)
{
    if (DialogResult.Yes == MessageBox.Show(
        "Vai tiešām atcelt veiktās izmaiņas?"
      , "Darbu vadītāju datu saglabāšana", MessageBoxButtons.YesNo
      , MessageBoxIcon.Question))
    {
        dsDataModel. Supervisors.RejectChanges();
    }
}
```

Izvēlēties cilni **tbpSupervisors** un realizēt tās notikuma apstrādātāju **Leave**, kas aktivizējas, ja notiek pāreja uz citu cilni un pārbauda, vai datu kopā ir veiktas izmaiņas, un atgādina lietotājam par to saglabāšanas nepieciešamību:

```
private void tbpSupervisors_Leave(object sender, EventArgs e)
{
    if (dsDataModel.HasChanges())
        UpdateSupervisors("Ir nesaglabāti labojumi darbu vadītāju datos."
                          + "Vai saglabāt izmaiņas datu bāzē?");
}
```

Izvēlēties formu **frmQualifWorks** un realizēt tās notikuma **Load** apstrādātāju (un aiz tā sekojošo metodi **Idle()**):

```
private void frmQualifWorks_Load(object sender, EventArgs e)
{
    taSupervisors.Fill(dsDataModel. Supervisors);

    Application.Idle += new EventHandler(Idle);
}

void Idle(object sender, EventArgs e)
{
    // pogas Saglabāt un Atcelt iespējo tikai tad, ja datu kopā
    // ir veiktas izmaiņas
    btnUpdateSupervisors.Enabled = dsDataModel.HasChanges();
    btnCancelSupervisors.Enabled = btnUpdateSupervisors.Enabled;
}
```

Realizēt formas **frmQualifWorks** notikuma **FormClosing** apstrādātāju, kas pie formas aizvēršanas pārbauda, vai datu kopā **dsDataModel** nav veiktas izmaiņas un piedāvā lietotājam tās saglabāt – ja lietotājs izvēlas atcelt, tad formas aizvēršana tiek atcelta):

```
private void frmQualifWorks FormClosing(object sender
                                        , FormClosingEventArgs e)
{
    if (dsDataModel.HasChanges())
        if (DialogResult.Cancel == UpdateSupervisors("Ir nesaglabāti"
           + " labojumi darbu vadītāju datos."
           + " Vai saglabāt izmaiņas datu bāzē?"))
            e.Cancel = true;
}
```

Nokompilēt risinājumu un palaist uz izpildi projektu **QualifWorksClient**, lai pārliecinātos, ka cilnē 'Darbu vadītāji' ir iespējams ievadīt jaunas rindas un, nospiežot pogu 'Saglabāt', tās tiek saglabātas datu bāzē.