L U K U 7

Tietokantatuen lisääminen

Tässä luvussa

Useimmat sovellukset käsittelevät suurta, usein jaettua tietomäärää, joka on monesti varastoituna relaatiotietokannan hallintajärjestelmään (relational database management system, RDBMS). RDBMS on sovellus, jonka avulla voit hallita linkitetyissä tauluissa olevaa suurta tietomäärää. RDBMS-sovelluksiin kuuluu usein kyselykieli, jonka avulla voit nopeasti hakea tietoja tietokannasta. RDBMS -sovelluksia ovat esimerkiksi Oracle ja Microsoft SQL Server. Pienemmät yhden käyttäjän sovellukset voivat käyttää kevyempiä tietokantaohjelmia, kuten Microsoft Accessia, varastoidessaan ja hakiessaan tietoja. Pienempien tietokantojen toiminta periaate on pääsääntöisesti samanlainen kuin suurten RDBMS-sovellusten.

Tässä luvussa tutustutaan Microsoft Visual C++:n sovelluskehittäjille tarjoamiin tietokantaliittymiin. Näihin liittymiin kuuluvat MFC-luokkien ne ominaisuudet, jotka tukevat tietokantojen käsittelyä, ja ActiveX Data Objects (ADO), joka muodostaa Microsoftin standardiliittymän käytännössä kaikkeen ulkoiseen tietoon.

Ennen kuin aloitat

Ennen kuin aloitat tämän luvun läpikäymisen, sinun tulisi lukea luvut 2-6, ja tehdä niihin liittyvät harjoitukset.

Oppitunti 1: Tietokantayhteydet Visual C++:ssa

Tässä luvussa kuvataan Visual C++ -ohjelmoijille tarjolla olevat tietokantaliittymät ja kerrotaan myös, missä tilanteissa mitäkin niistä tulisi käyttää. Koska näiden tekniikoiden avulla voidaan olla yhteydessä relaatiotietokantoihin, sinulla tulisi olla perustiedot relaatiotietokantojen teoriasta ja SQL-kielestä, joka on RDBMS-yhteyksissä käytettävä komentokieli. Tässä luvussa relaatiotietokantoja ja SQL-kieltä käsitellään vain ylimalkaisesti.

Oppitunnin jälkeen:

- Tunnet Microsoftin tietokantojen käsittelystrategian kehittymisen.
- Tunnet käytettävissä olevat tietokantaliittymät ja osaat valita kulloiseenkin tilanteeseen parhaiten soveltuvan vaihtoehdon.
- Osaat kuvata relaatiotietokannan perusrakenteen ja osaat hakea tietoa RDBMS:stä käyttämällä yksinkertaisia SQL-lauseita.

Oppitunnin arvioitu kesto: 40 minuuttia

Tietokantaliittymät

Microsoft Windows -sovellusten käytössä olevien tietokantaliittymien määrä voi vaikuttaa ylivoimaiselta hallita. Mitä salaperäisillä nimillä varustetuista tekniikoista — DAO, ODBC, RDO, UDA, OLE DB tai ADO — tulisi käyttää tekeillä olevalle sovellukselle asetettujen vaatimusten täyttämiseksi?

Valintaa saattaa helpottaa näiden tekniikoiden historiallisen taustan tunteminen. Microsoftin tietokantaliittymät perustuivat työasematietokannoissa Data Access Objectsiin (DAO) ja Remote Data Objectsiin (RDO), joka käyttää Open Database Connectivity (ODBC) -arkkitehtuuria yhteyksissä asiakas palvelintieto-kantoihin. Tämä järjestelmä on nyt korvattu yhdellä mallilla, Universal Data Accessilla (UDA), jonka avulla voidaan käsitellä kaikkia tietokantoja.

Microsoftin UDA-järjestelmän tavoitteena on tarjota suorituskykyinen tietokantaliittymä niin relaatiotietokantoihin kuin muihinkin tietolähteisiin helppokäyttöisen työkalu- ja kieliriippumattoman ohjelmointirajapinnan kautta. UDA on toteutettu ADO:n välityksellä. ADO sisältää korkeatasoisen käyttöliittymän OLE DB:en, Microsoftin uusimpaan Component Object Model (COM)-pohjaiseen tietokanta-liittymätekniikkaan.

Vaikka voitkin edelleen käyttää mitä tahansa aikaisemmista C++:n tietokantaliittymistä, sinun tulisi käyttää UDA-tekniikkaa aina, kun olet tekemässä uutta sovellusta. Yleensä sinun tulisi käyttää ADOa kaikessa tietojenkäsittelyssä, koska ADO on helppokäyttöinen, sisältää monia tehokkaita ominaisuuksia ja toimii hyvin. Kokeneet COM-ohjelmoijat voivat käyttää OLE DB -rajapintoja suoraan saa-vuttaakseen optimaalisen suorituskyvyn ja tehoetuja. Visual C++ 6.0 sisältää *OLE DB Templates* -malleja, joukon malliluokkia, joissa on toteutettu monia usein käy-tettyjä OLE DB -rajapintoja, jotka helpottavat OLE DB -tekniikan käyttämistä.

Jos suunnittelet olemassaolevan DAO/ODBC-sovelluksen muuttamista ADO-sovellukseksi, sinun täytyy miettiä, ovatko muutoksesta saatavat hyödyt niin suuria, että muutoksesta aiheutuvan työn tekeminen on perusteltua. DAO tai RDO -koodi ei käänny suoraan ADO koodiksi. Joka tapauksessa muita tietokannan käsittelymenetelmiä käyttäen rakennetut ratkaisut ovat erittäin todennäköisesti toteutetta-vissa ADOlla. Lopulta sinun tulisi pyrkiä muuttamaan kaikki koodisi käyttämään ADOa, koska sen objektimalli on yksinkertaisempi ja joustavampi.

Data Access Objects

DAO, Microsoft Jet -tietokantamoottorin alkuperäinen ohjelmointirajapinta suunniteltiin alkujaan käytettäväksi Microsoft Visual Basic ja Microsoft Visual Basic for Applications -kielten kanssa. DAO käyttää Microsoft Jet -moottoria tarjotakseen joukon tietojenkäsittelyobjekteja, jotka koteloivat yleiset tietokantaobjektit kuten taulukot, kyselyt ja *tietuejoukot* (recordsets) (objektit, jotka sisältävät tietokantaan tehdyn kyselyn vastauksena saadut rivit).

DAO:a käytetään yleensä paikallisella työasemalla sijaitsevien tietolähteiden kuten Microsoft Accessin, Microsoft FoxPron ja Paradoxin käsittelemiseen, mutta sitä voidaan käyttää myös etätietolähteiden käsittelyyn.

Alimmalla tasolla DAO:n objektit ovat käytettävissä COM-rajapinnan kautta. C++- ohjelmoijat käsittelevät niitä kuitenkin todennäköisemmin MFC DAO -tietokantaluokkien (tarkemmin tämän luvun oppitunnilla 2) tai **dbDAO**-luokkien avulla. **dbDAO**-luokat lisäävät C++:aan samat DAO-toiminnot kuin Visual Basicissa ja niissä käytetään myös samanlaista syntaksia.

Open Database Connectivity

ODBC on yleinen ohjelmointirajapinta (API), jota asiakas/palvelintietokantojen, yleensä RDBMSien, kuten SQL Serverin tai Oraclen käsittelemiseen. ODBC:n yhdenmukainen rajapinta mahdollistaa suuren monikäyttöisyyden; yksi sovellus voi käsitellä erilaisia RDBMS-alustoja saman ohjelmakoodin avulla.

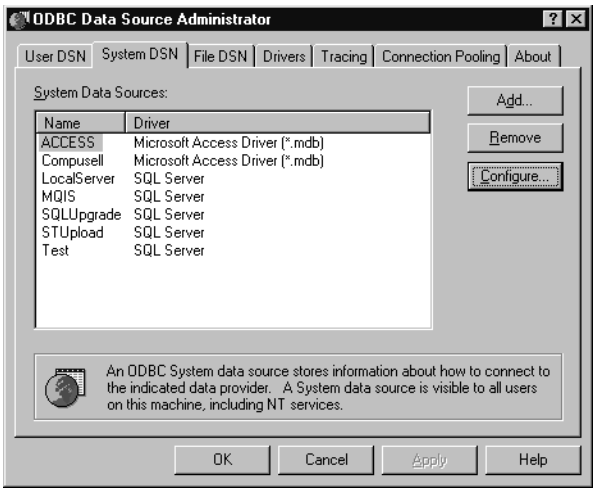
Tämä antaa ohjelmoijille mahdollisuuden rakentaa ja levittää asiakas/palvelin-sovelluksia ilman, että heidän tarvitsee suunnitella sovellusta tiettyyn RDBMS-ympäristöön tai tietää yksityiskohtia kaikista eri tietokantapalvelinalustoista, joita ohjelma saattaa käyttää. Kaikki mitä RDBMS:n käsittelyyn tarvitaan on ODBC-ajuri. Näitä ajureita toimittavat tietokantajärjestelmien kehittäjät ja ulkopuoliset toimittajat ja ne noudattavat avointa ODBC-standardia.

Koska eri RDBMS-alustojen kyvyt vaihtelevat ja koska ODBC-ajureiden kehittäjät voivat halutessaan rajoittaa toteutettavien toimintojen määrää, ODBC määrittelee ajureille kolme yhteensopivuustasoa (conformance), joiden avulla sovellus voi päätellä, mitkä ominaisuudet ajuri sille käytettäväksi tarjoaa:

- *Ydin* (Core) toiminnallisuus, joka kaikkien ODBC-ajureiden täytyy tarjota.
- *Taso 1* (Level 1) toiminnallisuus, joka pitää sisällään ydintason toiminnallisuuden lisäksi lisäominaisuuksia kuten RDBMS:ssä yleisesti saatavilla olevia tapahtumia.
- *Taso 2* (Level 2) toiminnallisuus, joka sisältää tason 1 toiminnallisuuden lisäk-si joukon lisätoimintoja kuten ODBC:n asynkronisen suorittamisen.

Lisätietoja ODBC:n toiminnallisuustasoista saat tekemällä haun Visual C++:n ohjeessa lauseella "Interface conformance levels".

Voit asentaa ja konfiguroida ODBC-ajureita käyttämällä Ohjauspaneelin ODBC-tietolähteet-toimintoa. ODBC-tietolähteet -toimintoa käyttämällä voi myös rekisteröidä *tietolähteiden nimiä* (Data Source Name DSN). DSN on yksilöivästi nimetty tietojoukko, jota ODBC-tietolähteet käyttävät yhdistäessään sovelluksesi tiettyyn ODBC-tietokantaan. DSN on rekisteröitävä siinä järjestelmässä, jossa sitä käytetään. DSN-nimet voidaan varastoida tiedostoon (*tiedostotietolähde*, file DSN) tai rekisteriin (*laitetietolähde*, machine DSN). Järjestelmän DNS-nimet voivat olla tietyn käyttäjän käytettävissä (*käyttäjätietolähde*, user DSN), tai kaikkien käyttäjien saatavissa (*järjestelmätietolähde*, system DSN). Kuvassa 7.1 ODBC-tietolähteet-toiminnon avulla tutkitaan järjestelmän-DSN:eja.



Kuva 7.1 ODBC-tietolähteet -toiminto

ODBC käyttää SQL-kieltä tietojen käsittelyyn. Kun sovellus tarvitsee tietoja tietolähteestä, se lähettää SQL-komennon ODBC Driver Managerille, joka sitten lataa tiedonsiirtoon tarvittavan ODBC-ajurin. Ajuri kääntää tämän jälkeen sovelluksen käyttämän SQL-lauseen DBMS:n käyttämään SQL-muotoon ja lähettää sen lopuksi tietokantapalvelimelle. DBMS noutaa tiedon ja lähettää sen sovellukselle ajurin ja Driver Managerin kautta.

ODBC sisältää *osoitinkirjaston* (cursor library), jossa on vieritettävä tietokanta-osoitin kaikille ajureille, jotka toteuttavat ODBC:n alimman yhteensopivuustason. Voit käyttää osoittimia selatessasi läpi tietokannasta haettuja rivejä.

C++-ohjelmoijat voivat käyttää ODBC API:a tietokantayhteyksien luomiseen, SQL-komentojen lähettämiseen, tuloksien vastaanottamiseen, virheiden seuraamiseen ja niin edelleen. ODBC API on hyvin dokumentoitu tapa asiakas/palvelin-sovellusten tekemiseen, mutta se on melko monimutkainen ja siihen kuuluu paljon ohjelmakoodia. Tämän seurauksena käytetään yleensä objektimalleja kuten ADO:a ja RDO:ta, tai MFC:n ODBC tietokantoja tukevia luokkia.

Remote Data Objects

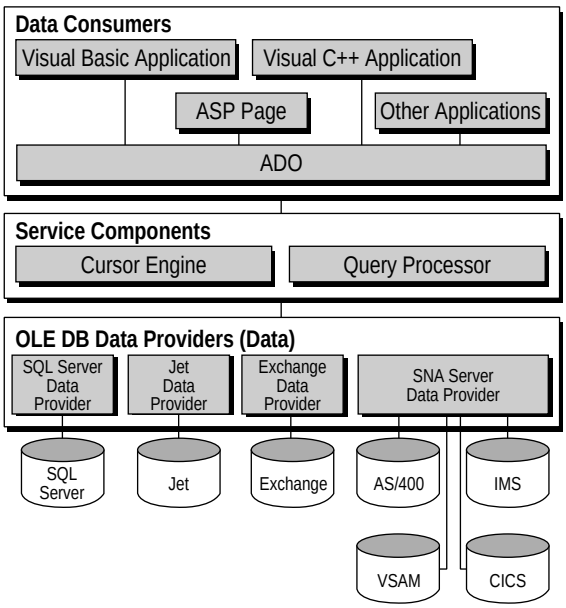
RDO on ODBC API:n päälle kasattu ohut objektikerros. RDO hyödyntää toimintojensa toteuttamisessa paljon ODBC-ajuria. RDO:n kautta voidaan luoda tietokantayhteys ODBC-tietolähteeseen käyttämällä DAO:n kaltaista objektimallia, ilman paikallisen tietokantatuen muistin määrälle asettamia lisävaatimuksia. RDO-objektimalli sisältää lisäominaisuuksia kuten palvelinosan osoittimia, ilman yhteyttä toimivat tietolähteet ja asynktroninen prosessointi.

Kuten DAO:n myös RDO:n objekteja voidaan käyttää COM-rajapinnan kautta. RDO:ssa on käytettävissä *Data Source Control*, joka on tietokantakyselyn ja saadun tietojoukon kapseloiva ActiveX-kontrolli. Data Source -kontrolli sisältää kontrollit, joiden avulla voit selata tietojoukkoa, esittää sen Microsoftin tietosidotuissa ActiveX-kontrolleissa kuten *DBGrid* tai *DBList* -kontrolleissa säilyttämät tiedot.

OLE DB

OLE DB on joukko COM-rajapintoja, jotka tarjoavat sovelluksille yhdenmukaisen liittymän erilaisiin tietolähteisiin varastoituihin tietoihin, riippumatta niiden tyypistä tai sijainnista. OLE DB on avoin määrittely, joka on suunniteltu lisäämään ODBC:n suosiota tarjoamalla avoimen standardin kaiken tyyppisen tiedon käsittelyyn. ODBC kehitettiin relaatiotietokantojen käsittelemistä varten. OLE DB sitä vastoin suunniteltiin sekä relaatio että ei-relaatio tietolähteitä varten. Näihin tietolähteisiin voivat kuulua mainframepalvelimet ja paikalliset tietokannat; sähköposti ja tiedostojärjestelmien varastot; taulukkolaskentaohjelmien taulukot ja projek-tihallintatyökalut; sekä räätälöidyt liiketoimintaobjektit.

Käsitteellisesti OLE DB:ssä on kolmentyyppisiä elementtejä: *asiakassovelluksia* (data consumers), *palvelinkomponentteja* (service components) ja *palvelinsovelluksia* (data providers) kuten kuvassa 7.2 on esitetty.



Kuva 7.2 OLE DB:n osat

Asiakassovellukset ovat ohjelmia tai komponentteja, jotka käyttävät palvelinsovellusten tarjoamia tietoja. Kaikki ADO:a käyttävät sovellukset ovat OLE DB -asiakassovelluksia.

Palvelinkomponentit käsittelevät tai kuljettavat tietoa laajentaen tietolähteen toimintoja. Palvelinkomponentit voivat olla esimerkiksi kyselyn muodostajia, jotka muodostavat optimoituja kyselyjä tai hakukoneita, jotka kokoavat tiedot sarjamuotoisesta vain yhteen suuntaan luettavasta tietokannasta molempiin suuntiin selattavaksi tietolähteeksi.

Palvelinsovellukset ovat sovelluksia kuten SQL Server tai Microsoft Exchange, tai järjestelmän osia kuten tiedostojärjestelmä tai dokumenttivarasto, jotka tarjoavat tietonsa toisten sovellusten käyttöön. Palvelinsovelluksilla on OLE DB -rajapinta, jota asiakassovellukset voivat käyttää suoraan. OLE DB -palveluntarjoaja on saatavissa ODBC:lle, joten monet olemassaolevista ODBC-tietolähteistä ovat OLE DB -asiakassovellusten käytettävissä.

ActiveX Data Objects

ADO on suunniteltu suorituskykyiseksi ja helppokäyttöiseksi sovellustason rajapinnaksi OLE DB:en. ADO on toteutettu niin, että se on kooltaan mahdollisimman pieni, verkossa tapahtuva liikenne on mahdollisimman vähäistä ja sovelluksen ja tietolähteen välillä on mahdollisimman vähän kerroksia. Kaikki tämä vaikuttaa osaltaan siihen, että käyttöliittymä on kevyt ja suorituskykyinen. ADO

sisältää COM Automation -rajapinnan, jonka kautta se on käytettävissä kaikissa johtavissa RAD-työkaluissa, sovelluskehitysympäristöissä ja skriptikielissä.

Koska ADO suunniteltiin yhdistämään RDO:n ja DAO:n parhaat ominaisuudet ja lopulta korvaamaan ne, siinä käytetään samanlaisia toimintatapoja yksinkertaistettujen toimintamallien avulla, jotta se olisi helpompi omaksua. ADO sisältää *ADO Data Controlin*, joka on parannettu versio RDO:n Data Source -kontrollista.

ADO:n toiminto, *Remote Data Service* (RDS) huolehtii yhteydettömien tietojoukkojen siirtämisestä asiakkaille HTTP-yhteyden samoin kuin *Distributed COM:*in (DCOM) yli, mahdollistaen näin täysin toiminnallisten tietojenkäsittelysovellusten toteuttamisen Web-sovelluksina. ADO käsitellään tarkemmin tämän luvun oppitunnilla 3.

ODBC tietolähteen hallinnan käyttäminen

Tässä käytännön harjoituksessa lisäät rekisteriin SQL Severin standardiasennuksen mukana tulevan *pubs* esimerkkitietokannan järjestelmätietolähteeksi ODBC-tietolähteen hallintaa käyttäen. Käytät tätä tietolähdettä tulevissa harjoituksissa käsitellessäsi ODBC-tietoja.

► **Tietolähteen lisääminen ODBC-tietolähteen hallinnalla**

1. Valitse **Käynnistä**-valikosta **Ohjauspaneeli**. Avaa **ODBC-tietolähteet**.
2. Napauta **Järjestelmätietolähde**-välilehdellä olevaa **lisää-**painiketta. Valitse SQL Server -ajuri esiin tulevasta luettelosta ja napauta **Valmis**.

   Huomio Jos SQL Server -ajuria ei löydy luettelosta, sinun täytyy asentaa SQL Server ODBC -ajuri Visual C++ -rompulta. Valitse asennusikkunasta **Add/Remove** ja asenna ajuri **Data Access** -valintaryhmästä.

3. Kirjoita tietolähteen nimeksi **MyDSN**. Voit jättää **Description** ruudun tyhjäksi.
4. Valitse **(local)** SQL Serveriksi, johon yhteys muodostetaan ja napauta **Next**.
5. Jos olet asentanut Desktop-version SQL Serveristä, valitse **SQL Server authentication using a login ID and password entered by the user**. Kirjoita login ID ja salasana tähän näyttöön ja napauta **Next**.

   Huomio Käytä oletuslogin ID:tä, ja jätä salasanaruutu tyhjäksi, jos et ole määritellyt salasanaa tälle tilille. (Perusasennuksessa **sa** tilille ei ole määritelty salasanaa.)

   Jos olet asentanut SQL Server Standard -version, valitse **Windows NT authentication using the network login ID**, ja napauta **Next**.

6. Napauta valintaruutua saadaksesi **Change the default database to:** option käyttöön. Valitse **pubs** alasvetovalikosta ja napauta **Next**.

Huomio Jos pubs tietokantaa ei löydy sinun täytyy asentaa se SQL Serveriisi. Voit suorittaa asennuksen SQL Server Query Analyzerilla lataamalla ja suo-rittamalla *MSSQL7\ Install\InstPubs.sql* skriptin.

---

7. Jätä seuraavassa ruudussa oletus asetukset ennalleen ja napauta **Finish**.
8. Testaa tietolähde napauttamalla **Test Data Source**. Viimeisellä palautetulla rivillä tulisi lukea TESTS COMPLETED SUCCESSFULLY! Poistu **SQL Server ODBC Data Source Test** -dialogista napauttamalla **OK** ja lopeta sitten koko rekisteröintiprosessi napauttamalla uudestaan **OK**.

Huomaat, että DSN on lisätty järjestelmätietolähteet-luetteloon. Sulje ODBC tietolähteen hallinta ja Ohjauspaneeli.

Relaatiotietokantojen käsitteet

Vaikka relaatiotietokantojen lisäksi on olemassa muitakin tietokantajärjestelmiä, ja vaikka ADO on suunniteltu mahdollistamaan pääsyn relaatiotietokantojen lisäksi myös muihin tietokantoihin, relaatiomalli on tällä hetkellä yleisimmin käytetty järjestelmä. Tästä syystä ennen kuin aloitat työskentelyn tietolähteiden parissa, sinulla tulisi olla jonkinlainen käsitys relaatiotietokantojen teoriasta.

Relaatiotietokanta varastoi ja esittää tiedon taulukkojen kokoelmana. (Taulukoita käsitellään yksityiskohtaisemmin myöhemmin tässä jaksossa.) Tämän tyyppisen tietokannan looginen rakenne määritellään tekemällä taulukoiden välille yhteyksiä. *Relaatiotietokanta malli* antaa seuraavat edut:

- Organisoi tiedot taulukkojen kokoelmiksi ja tekee rakenteesta helpon ymmärtää.
- Tarjoaa suhteellisen täydellisen kielen tietojen määrittelemiseen, hakemiseen ja päivittämiseen.
- Sisältää tiedon eheyssäännöt, jotka määrittelevät tietokannoille yhdenmukaiset tilat parantaen tiedon luotettavuutta.

Relaatiotietokannan osat

Relaatiotietokanta esittää tiedon taulujen kokoelmana. Esimerkiksi SQL Serverin esimerkki tietokanta *pubs* sisältää liiketoimintatietoa, jollaista kustannustalo voisi kerätä. Pubs-tietokannassa on yksi taulu, jossa on lueteltu kaikki kirjoittajat ja toinen, jossa on lueteltu kaikki kirjanimikkeet. Kuvassa 7.3. on esitetty näkymä *authors* taulukosta.

au_id	au_lname	au_fname
172-32-1176	White	Johnson
213-46-8915	Green	Marjorie
238-95-7766	Carson	Cheryl
267-41-2394	O'Leary	Michael
274-80-9391	Straight	Dean
341-22-1782	Smith	Meander
409-56-7008	Bennet	Abraham
427-17-2319	Dull	Ann
472-27-2349	Gringlesby	Burt
486-29-1786	Locksley	Charlene
527-72-3246	Greene	Morningstar
648-92-1872	Blotchet-Halls	Reginald
672-71-3249	Yokomoto	Akiko

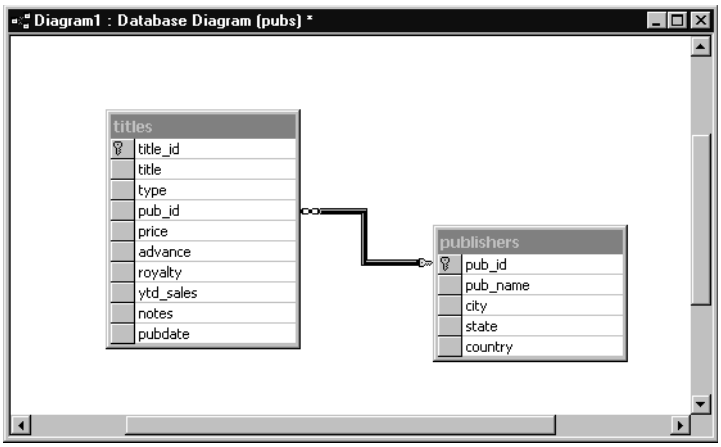
Kuva 7.3 pubs-tietokannan authors-taulu

Taulu on joukko loogisesti yhteenkuuluvaa tietoa ja se koostuu *riveistä* ja *sarakkeista*. Rivi (kutsutaan myös *tietueeksi*) sisältää yhden taulukkoon kirjatun kohteen tiedot. Esimerkiksi rivi authors-taulussa sisältää yhden kirjoittajan tiedot.

Rivi muodostuu useista sarakkeista (joita kutsutaan myös *kentiksi*). Jokainen sarake sisältää yhden riville kuuluvan tiedonosasen. Esimerkiksi author taulussa on sarake kirjoittajan tunnusta, etunimeä ja sukunimeä varten. Jos katsot kuvassa 7.3 esitettyä otosta author-taulukosta huomaat, että näille sarakkeille on annettu nimet *au_id*, *au_lname* ja *au_ fname*.

Toisin kuin sarakkeita, rivejä ei nimetä. Jotta tietokantataulun rivit voitaisiin yksilöidä, täytyy taulukolle määrätä *pääavain* (primary key). Pääavain on sarake tai joukko sarakkeita, joissa on jokaisella rivillä yksilöllinen arvo. Esimerkiksi kuvassa 7.3 näkyvä **au_id** sarake on pääavain. Kun sarake tai sarakejoukko on määritelty pääavaimeksi, RDBMS:ään rakennetut tiedon eheyssäännöt varmistavat sen, että samanlaisen avaimen sisältäviä rivejä ei voida taulukkoon syöttää.

Taulukkoon voidaan lisätä myös *viiteavaimia*, jotka määrittelevät taulujen välisiä yhteyksiä. Viiteavain viittaa toisen taulun pääavainkenttään. Esimerkiksi pubs-tietokannassa on *titles*-taulu, jossa on lueteltu julkaistut teokset. Titles-taulun yksi sarake on viiteavain, joka sisältää *publishers*-taulun pääavaimen arvon, joka vas-taa kyseisen teoksen kustantajan tietoja. Kuvassa 7.14 näet nämä yhteydet Visual Studio database diagram -työkalulla kuvattuina.



Taulukko 7.4 Titles ja publishers taulujen välinen suhde

Yhteyden kuvaamisessa käytetyt symbolit merkitsevät sitä, että publisher ja titles taulujen välillä on *yksi-moneen yhteys*. Toisin sanoen titles-taulussa voi **pub_id** sarakkeessa esiintyä sama arvo kuinka monella rivillä tahansa tai toisin ilmaistuna, titles-taulussa voi olla useita saman kustantajan julkaisemia kirjoja. Koska publisher-taulun **pub_id**-sarake on määritelty pääavaimeksi, publisher-taulussa jokaisella rivillä on **pub_id**-sarakkeessa erilainen arvo.

Structured Query Language

Structured Query Language (SQL) on tarkasti määritelty standardi kieli, jota käyttäen voidaan relaatiotietokantoja sekä päivittää että hallita ja tehdä niihin kyselyjä. International Standards Organization (ISO) -standardin mukaan SQL voi noutaa, järjestää ja suodattaa tietokannassa olevaa tietoa. Lisäksi voit SQL-komentoja käyttäen muuttaa ja poistaa tietokannassa olevia tietoja.

SQL:n perusteiden tunteminen on tärkeää, jotta saisit sovelluksesi toimimaan tehokkaasti tietokantojen kanssa. SQL:ää käyttämällä sovellus voi pyytää tietokantaa suorittamaan joitain tehtäviä sen sijaan, että suorittaisi ne itse. Mikä tärkeintä, SQL:ää tehokkaasti käyttämällä voidaan minimoida etätietokantapalvelimelta luettavan ja sinne kirjoitettavan datan määrä. Tehokkaasti käytettynä SQL minimoi myös verkon kautta siirrettävän tiedon määrän. Levy- ja verkkoliikenteen mini-moiminen on tärkeä asia parannettaessa sovelluksen suorituskykyä.

Sinun tulee olla selvillä siitä, että eri ympäristöissä tietokannoissa on erilaisia toteutuksia samoista SQL-toiminnoista, sekä kirjoitusasun että merkityksen suhteen. Jokaisessa SQL-toteutuksessa on omat versionsa eri tietotyypeille, eheyssäännöille ja kyselyjen optimoinnille.

SQL SELECT -komento

SQL:n **SELECT**-komento hakee tietokannasta tietoja ja antaa tuloksena löydettyjen rivien luettelon. **SELECT**- komento jaetaan kolmeen pääosaan:

- **SELECT** -lista antaa sinulle mahdollisuuden määrittää kyselyyn mukaan otettavat sarakkeet.
- **FROM**-ehto antaa sinulle mahdollisuuden määritellä taulut, joista **SELECT**-listassa määrätyt sarakkeet haetaan.
- **WHERE** Valinnaisen **WHERE**-ehdon avulla voit määritellä suodatusehdot, joiden mukaan rajoitetaan haettavien rivien määrää. Voit suodattaa useampiin sarakkeisiin pohjautuvia kyselyjä.

SELECT-komennon tulee sisältää vähintään seuraavat osat:

```
SELECT sarakkeet FROM taulut
```

Suorittaessaan tämän operaation tietokanta etsii ensin määrätyn taulun tai taulut ja hakee valitut sarakkeet. Voit valita taulun kaikki sarakkeet käyttämällä tähteä (*). Esimerkiksi seuraava SQL-komento hakee authors-taulun kaikki sarakkeet ja rivit:

```
SELECT *
FROM authors
```

Taulun kaikkien tietojen hakeminen ei useinkaan ole tehokasta. Lisäämällä **WHERE**-ehto komennon loppuun, voidaan määrätä, että vain tietyn ehdon täyttävät rivit noudetaan. Seuraava esimerkkikatkelma palauttaa authors-taulukon kaikki rivit, joiden au_lname kentän arvo on Ringer:

```
SELECT *
FROM authors
WHERE au_lname = 'Ringer'
```

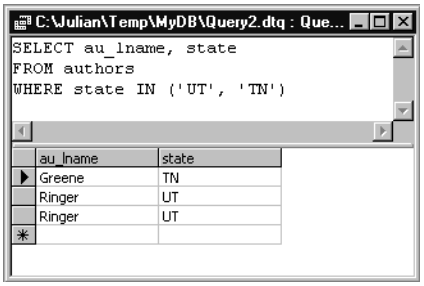
Huomaa, että nimi Ringer on tässä esimerkissä kahden heittomerkin (') välissä. Heittomerkkejä käytetään, jos **WHERE**-ehto on merkkijono. Tässä tapauksessa kentän au_lname arvot ovat merkkijonoja. Kun **WHERE**-ehdossa määritellään numeroarvo, heittomerkkejä ei käytetä kuten seuraavassa nähdään:

```
SELECT *
FROM titles
WHERE royalty = 10
```

IN-operaattori

Käyttämällä IN-operaattoria **WHERE**-ehdossa, voit poimia ne rivit, joiden **WHERE**-ehdossa määrätyssä sarakkeessa on listassa lueteltu arvo. Voit käyttää IN-operaattoria esimerkiksi poimiaksesi niiden kirjoittajien sukunimet ja kotiosavaltiot, jotka asuvat Utahissa tai Tennesseessä, kuten seuraavassa esimerkissä ja kuvassa 7.5 on tehty.

```
SELECT au_lname, state
FROM authors
WHERE state IN (‘UT’, ‘TN’)
```



Kuva 7.5 IN-operaattorin käyttäminen **WHERE**-ehdon arvojen suodattamiseen

BETWEEN-operaattori

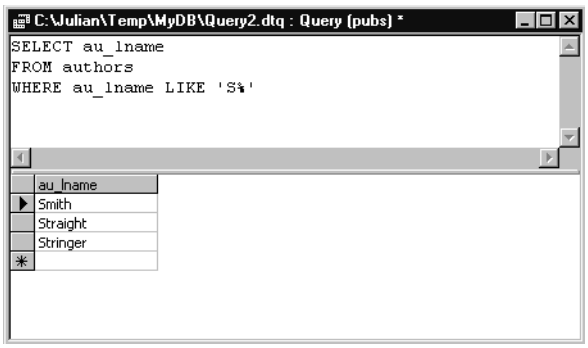
Tämä operaattori palauttaa rivijoukon, jonka **WHERE**-parametri on kahden annetun ehdon välisellä alueella. Huomaa, että päivämäärät on annettu *'yyyymmdd'* muotoisena merkkijonona.

```
SELECT title_id, title, pubdate
FROM titles
WHERE pubdate BETWEEN ‘19910601’ AND ‘19910630’
```

LIKE-operaattori

Voit etsiä määrittämääsi mallia vastaavia arvoja sarakkeista käyttämällä **LIKE**-operaattoria. Voit määritellä arvon kokonaan, esimerkiksi **LIKE ‘Smith’**, tai voit käyttää jokerimerkkejä etsiessäsi arvoja tietyltä alueelta (esimerkiksi, **LIKE ‘Sm%’**). Seuraavassa esimerkissä, joka on esitetty myös kuvassa 7.6, haetaan kirjoittajien sukunimiä, jotka alkavat S-kirjaimella.

```
SELECT au_lname
FROM authors
WHERE au_lname LIKE ‘S%’
```



Kuva 7.6 **LIKE**-operaattoria käyttävän SQL-komennon syntaksi

ORDER BY -ehto

Tavallisesti rivit haetaan siinä järjestyksessä kuin ne tietokantaan on syötetty. Valinnaisen **ORDER BY** -ehdon avulla kyselyn tulos voidaan järjestää määrätyn sarakkeen tai sarakkeiden sisällön mukaan nousevaan tai laskevaan järjestykseen. ASC-optio määrittää järjestyksen nousevaksi ja DESC-optio merkitsee laskevaa järjestystä. Oletuksena käytetään nousevaa järjestystä (A:sta Z:aan, 0:sta 9:ään). Seuraava esimerkki hakee sarakkeet authors-taulukosta ja järjestää ne sukunimen perusteella laskevaan järjestykseen:

```
SELECT *
FROM authors
ORDER BY au_lname DESC
```

Oppitunnin yhteenveto

Visual C++ -sovellukset voivat käyttää useita erilaisia rajapintoja käsitellessään ulkoisissa tietokannoissa olevia tietoja. Muutamina viime vuosina Microsoftin tietokantojen käsittelystrategia on vaihtunut mallipohjaisesta DAO:sta ja ODBC-pohjaisesta RDO:sta uudempaan yhteen malliin, joka tunnetaan nimellä UDA. UDA perustuu OLE DB:hen, joka on kokoelma COM-rajapintoja, jotka tarjoavat suorituskykyisen yhteyden kaikentyyppisiin tietolähteisiin - sekä relaatiotietokantoihin että muunlaisiin lähteisiin kuten sähköposti- tai tiedostojärjestelmiin. Vaikka voitkin käyttää OLE DB -rajapintoja suoraan, Microsoft suosittelee ADO:n käyttämistä, koska se on tehokkaampi, helppokäyttöinen ja korkeatasoinen käyttöliittymä OLE DB -tietolähteisiin.

Visual C++ tukee edelleen DAO:a ja RDO:ta, joten voit edelleen tukea ja ylläpitää aikaisempia sovelluksia, jotka käyttävät näitä tekniikoita. Sinun tulisi kuitenkin käyttää UDA:a aina uutta sovellusta tehdessäsi.

DAO hyödyntää Microsoft Jet -tietokantaohjainta paikallisten tietolähteiden käsit-telyyn tarkoitetuissa objekteissaan. DAO-objektimalli sisältää tavallisimmat tietokantaobjektit kuten taulut, kyselyt ja tietojoukot. DAO tarjoaa objektinsa käytettäviksi COM-rajapinnan kautta, mutta Visual C++ -ohjelmoijat käyttävät yleensä MFC:n DAO-tietokantaluokkia.

RDO mahdollistaa ODBC-tietolähteiden käsittelemisen DAO:a muistuttavan objektimallin kautta. ODBC:tä on käytetty paljon matalantason API:na asiakas/palvelin-tietolähteisiin. RDO sisältää monia tehokkaita ominaisuuksia kuten palvelinosan osoittimet ja asynkroninen tiedonsiirto.

SQL-standardiin perustuvana tietojen käsittelymenetelmänä ODBC on tärkeä tekniikka — ODBC-pohjaisia sovelluksia on tehty lukemattomia. ODBC-pohjaiset sovellukset ovat yhteydessä tietokantoihin ODBC-ajureiden kautta. Ajureita jakelevat RDBMS-toimittajat ja kolmanen osapuolen kehittäjät ja ne tehdään avoimen ODBC-standardin mukaisesti. Voit käyttää ODBC Tietolähteen hallintaa ODBC-ajureiden asentamiseen ja konfigurointiin, sekä DSN:ien rekisteröimiseen käyttöjärjestelmään.

Visual C++ -ohjelmoijat voivat käyttää ODBC API:a tietokantayhteyden muodostamiseen, SQL-komentojen lähettämiseen, tulosten vastaanottamiseen, virheiden selvittämiseen, yhteyden purkamiseen ja niin edelleen. On kuitenkin helpompaa käyttää jotain objektimallia kuten ADOa tai RDO:ta, tai käyttää MFC:n ODBC-ietokantoja tukevia luokkia.

OLE DB on avoin määrittely, joka suunniteltiin tukemaan ODBC:n menestystä kaikenlaisen tiedon käsittelyyn sopivan avoimen standardin avulla. Käsitteellisesti OLE DB:ssä on kolmen tyyppisiä elementtejä: *asiakassovelluksia*, *palvelin-komponentteja* ja *palvelinsovelluksia*.

- Palvelinsovellukset ovat ohjelmia, jotka antavat tietonsa toisten sovellusten käyttöön. Ne julkistavat myös OLE DB -rajapinnat, joita palvelinkomponentit ja asiakassovellukset voivat käsitellä suoraan. OLE DB -sovitin ODBC:lle mahdollistaa monien olemassa olevien ODBC-tietolähteiden käyttämisen OLE DB -asiakassovelluksille.
- Asiakassovellukset ovat sovelluksia tai komponentteja, jotka käyttävät tietolähteitä. Kaikki sovellukset, jotka käyttävät ADO:a, ovat OLE DB -asiakassovelluksia.
- Palvelinkomponentit ovat elementtejä, jotka prosessoivat tai siirtävät tietoja laajentaen tietolähteiden toiminnallisuutta.

ADO on suunniteltu suorituskykyiseksi, helppokäyttöiseksi sovellustason rajapinnaksi OLE DB:hen. ADO julkistaa COM Automaatio -rajapinnan, jonka kautta se on useiden ohjelmointityökalujen ja skriptikielien käytettävissä.

Jotta voisit käyttää ODBC:tä ja ADO:a tehokkaasti, sinulla tulisi olla perustiedot tietokantojen teoriasta ja SQL:stä, joka on kieli, jolla kontrolloidaan RDBMS:ää ja ollaan vuorovaikutuksessa sen kanssa.

Relaatiotietokannassa tiedot esitetään *riveistä* ja *sarakkeista* koostuvina *taulukoina*. Rivi sisältää yhden taulukon tietueen tiedot. Jokainen sarake sisältää yhden tietueeseen kuuluvan tiedon.

Taulukon rivit tulisi yksilöidä määrittämällä taulukolle *pääavain*. Pääavain on sarake tai sarakkeiden yhdistelmä, jonka arvo on takuulla erilainen jokaisella taulukon rivillä. Voit määritellä myös *viiteavaimia*, joiden avulla voidaan määrittää taulukoiden välisiä yhteyksiä. Viiteavain “osoittaa” yhteystaulukon pääavaimeen.

Muista, että SQL on kyselyiden tekemiseen ja tietokantojen päivittämiseen sekä hallinnointiin tarkoitettu standardi kieli. SQL:n avulla tietoja voidaan sekä lukea tietokannasta, että kirjoittaa tietokantaan. Sinun tulisi tuntea SQL:n perusteet, jotta voisit tehdä sovelluksia, jotka käsittelevät tietokantoja tehokkaasti. Käyttämällä SQL:ää oikein saat tietokantapalvelimen suorittamaan tietojen käsittelyn sovelluk-sesi puolesta ja varmistat sen, että tiedot siirretään tehokkaasti ilman ylimääräistä verkkoliikennettä ja levynkäsittelyä.

Sinun tulisi muistaa, että eri ympäristöissä tietokannat toteuttava samat SQL-toiminnot eri tavoin.

SQL:n **SELECT**-komento hakee tietokannasta tietoja ja antaa tuloksena löydettyjen rivien luettelon. **SELECT**-komento jaetaan kolmeen pääosaan: **SELECT**, **FROM** ja **WHERE**.

- **SELECT**:n avulla voit määrätä, mitkä sarakkeet poimitaan kyselyyn.
- **FROM** antaa mahdollisuuden määrittää taulut, joista **SELECT**-osassa määritellyt sarakkeet haetaan.
- **WHERE**-ehdon avulla voit asettaa rivien määrää rajoittavia suodattimia.Voit suodattaa useampiin sarakkeisiin pohjautuvia kyselyitä. **WHERE**-ehtoa voidaan tarkentaa **IN**, **BETWEEN** ja **LIKE** -operaattoreiden avulla.

Voit myös järjestää kyselyn rivit **ORDER BY** -ehdon avulla määrätyn sarakkeen tai useiden sarakkeiden arvojen mukaiseen nousevaan tai laskevaan järjestykseen.

Oppitunti 2: MFC:n tietokantatuki

MFC sisältää luokat, jotka tukevat DAO:n tai ODBC:n kautta tapahtuvaa tietojen käsittelyä. Tällä oppitunnilla selviää, kuinka voit käyttää näitä luokkia ulkoisissa tietokannoissa olevan tiedon noutamiseen ja käsittelemiseen.

Tämän oppitunnin jälkeen:

- Tunnet MFC DAO ja ODBC -tietokantaluokat ja kuinka ne yhteistoiminnallaan mahdollistavat pääsyn paikallisiin ja asiakas/palvelin-tietokantoihin.
- Tiedät, kuinka käytät AppWizardia **CRecordView** tai **CDaoRecordView** -luokkaan perustuvien tietokantasovellusten luomiseen.
- Tiedät, kuinka käytät MFC-tietokantaobjekteja tietokantakyselyjen tekemiseen suoraan sovelluksesi koodista.
- Tiedät, kuinka suodatat tietojoukkoa suoritusaikaisesti annettujen parametrien perusteella.

Oppitunnin arvioitu kesto: 50 minuuttia

MFC:n tietokantaluokat

MFC:hen kuuluu kaksi erillistä tietokantojen käsittelyyn pystyvää tietokantaluokkien ryhmää: toinen käyttää yhteyksissä DAO:a ja toinen ODBC:tä. DAO:a käytetään yleensä käsiteltäessä paikallisia tietokantoja ja ODBC:tä käsiteltäessä palvelimilla si-jaitsevia nimettyjä relaatiotietokantoja.

Molemmat ryhmät ovat samanlaisia ja perustuvat yhteiseen ohjelmointimalliin. DAO ja ODBC -tietokantaluokat eroavat toisistaan usein vain nimen ja joidenkin suhteellisen pienten toteutuksen yksityiskohtien osalta. Taulukossa 7.1 on luettelo tärkeimmistä ODBC-luokista, niiden DAO-vastineista, lyhyt kuvaus niiden toiminnoista:

Taulukko 7.1 ODBC-luokat ja niiden DAO-vastineet

ODBC-luokka	DAO-luokka	Funktio
CDatabase	CDaoDatabase	Kapseloi yhteyden etätietolähteeseen tai paikalliseen tietokantaan.
CRecordset	CDaoRecordset	Kapseloi tietokannan taulukosta valitun tietojoukon.
CRecordView	CDaoRecordView	Tarjoaa dialogi-pohjaisen lomakenäkymän suoraan tietojoukko-objektiin.

Nämä luokat toimivat yhteistyössä keskenään mahdollistaakseen tietojen hakemisen tietolähteestä niin, että tieto voidaan esittää selaamista tai päivittämistä varten dialogi-pohjaisessa näkymässä. Seuraavat jaksot kuvaavat tarkemmin

kunkin luokan tehtävät. Pidä mielessä, että koska ODBC ja DAO -versiot luokista ovat hyvin samanlaisia, puhumme usein luokista yhteisesti sen sijaan, että erottelisimme ne. Kiinnitämme huomiosi merkittäviin eroihin silloin, kun niitä ilmenee.

CDatabase ja CDaoDatabase

CDatabase-luokkaa käytetään tyypillisesti luotaessa yhteyksiä ODBC-tietolähteisiin kuten SQL Server -tietokantoihin. **CDaoDatabase**-luokkaa puolestaan käytetään luotaessa yhteyksiä paikallisiin tietolähteisiin kuten Access-tietokantoihin.

Tietokantayhteyden muodostamisessa on kaksi vaihetta. Ensin luodaan tietokantaobjekti ja sen jälkeen kutsutaan objektin **Open()**-jäsenfunktiota. Molempien luokkien **Open()**-funktiot ovat samanlaiset. **CDatabase::Open()**-funktiolle on määriteltävä DSN tai yhteysmerkkijono; **CDaoDatabase::Open()**-funktiolle määritellään tietokannan tiedostonimi. **Open()**-funktiolle välitetään argumentteja, joiden avulla voidaan määrittää, avataanko yhteys yksinoikeudella tai vain-luku-tilassa.

ODBC-yhteyksissä käytetään **CDatabase::OpenEx()**-funktiota. **OpenEx()**-funktion syntaksi on seuraava:

```
CDatabase::OpenEx (LPCTSTR lpszConnectString,
    DWORD dwOptions = 0);
```

lpszConnectString-parametri on yhteysmerkkijono, joka määrittelee DSN:n. Jos Windows NT:n käyttäjätunnistusta ei voida hyödyntää tietolähteessä, käyttäjänimi ja salasanatieto välitetään yhteysmerkkijonossa. Jos annat *lpszConnectString*-parametrinä **NULL**-arvon, suorituksen aikana avautuu dialogi, joka pyytää syöttä-mään DSN:n, käyttäjätunnuksen ja salasanan.

Toinen parametri, dwOptions, on bittijonoarvo, joka määrittää tilan, jossa tietolähde avataan. Oletusarvo on **0**, joka määrittelee tietolähteen avattavaksi jaetussa tilassa ja kirjoitusoikeudella. Muut käytettävissä olevat arvot ovat yhdistelmiä taulukussa 7.2 esitettävistä arvoista.

Taulukko 7.2 Valinnaiset parametrit CDatabase::OpenEx-jäsenfunktiolle

dwOptions parametri	Merkitys
CDatabase::openReadOnly	Avaa tietolähteen vain-luku-tilassa
CDatabase::useCursorLib	Lataa ODBC-osoitinkirjaston
CDatabase::noOdbcDialog	Ei näytä ODBC yhteysdialogia, vaikka yhteysmerkkijonossa olisi annettu riittämättömästi tietoja
CDatabase::forceOdbcDialog	Näyttää aina ODBC-yhteysdialogin

Tämä koodi esittää, kuinka voit avata yhteyden *pubs* DSN:ään, käyttämällä **sa**-tiliä ja salasanaa *password*, kutsumalla **OpenEx()**-funktiota.

```
CDatabase db;
db.OpenEx("DSN=pubs;UID=sa;PWD=password",0);
```

CDaoDatabase-luokka sisältää useita ominaisuuksia, joiden avulla voit käyttää Microsoft Jet -tietokantamoottorille ominaista toiminnallisuutta. Voit esimerkiksi yhdistää joukon **CDaoTableDef**-objekteja **CDaoDatabase**-objektin kanssa. Nämä ovat taulukon määrittelyobjekteja, joita käyttämällä voit luoda ja muokata Jet-yhteensopivissa tietokantatiedostoissa olevia taulukoita.

Tietokantaobjektit tulisi aina sulkea **Close()**-funktiota käyttämällä sen jälkeen, kun niiden käsittely on päättynyt. Tietokantayhteydet ovat arvokkaita resursseja, joita tulisi käyttää säästeliäästi.

CRecordset ja CDaoRecordset

Tietojoukko kapseloi joukon tietokannasta valittuja tietueita. Tietuejoukko on verrattavissa dokumenttiin dokumentti/näkymä -arkkitehtuurissa, koska se pitää sisällään tiedon, jonka tietuenäkymäobjekti näyttää.

Tietojoukko mahdollistaa siirtymisen tietueesta toiseen, tietueiden päivittämisen (tietueiden lisäämisen, muokkaamisen ja poistamisen), valintojen karsimisen suodattimilla, valittujen tietueiden järjestämisen ja valintojen parametrisoimisen suorituksen aikana hankitun tai lasketun tiedon perusteella. **Recordset**-objektit sisältävät jäsenmuuttujat, jotka vastaavat valitun tietokantataulukon sarakkeita. Näiden muuttujien arvot päivitetään vastaamaan valittuna olevan tietokannan rivin tietoja käyttäjän liikkuessa tietojoukossa.

Tiedonsiirto tietojoukon muuttujien ja vastaavien valitun tietolähteen taulukon sarakkeiden välillä on toteutettu *Record Field Exchange* (RFX) mekanismilla. RFX vastaa dialogin kontrollien ja dialogiluokan välisessä tiedonsiirrossa käytettyä DDX-mekanismia.

Huomio DAO:n yhteydessä, RFX:ää kutsutaan DFX:ksi — joka saadaan lyhennyksenä sanoista DAO Record Field Exchange.

Tietojoukko-objektien luominen on yhteydessä tietokantaobjektiin. Sinun tulisi välittää olemassa olevan tietokantaobjektin osoite tietojoukon muodostimelle, kuten seuraavassa ODBC-luokkia käyttävässä esimerkissä on tehty:

```
CDatabase db;
db.OpenEx("DSN=pubs;UID=sa;PWD=password",0);
CRecordset rs(&db);
```

Jos periytät tietojoukkoluokan **CRecordset** tai **CDaoRecordset** -luokasta, voit ylikuormittaa jäsenfunktiot, jotka käsittelevät yhteysinformaatiota ja lähettää NULL-arvon muodostimelle luodessasi objektin ilmentymää. Jos teet näin, MFC luo väliaikaisen tietokantaobjektin, joka liitetään tietojoukkoon käyttämällä määrittelemääsi yhteysinformaatiota.

Kun tietojoukko-objekti on luotu, lähetetään kysely ja täytetään tietojoukko kyselyn palauttamilla tietueilla kutsumalla **Open()** jäsenfunktiota. Seuraavassa koodissa kutsutaan edellisessä esimerkissä luodun **CRecordset**-objektin **Open()**-funktiota:

```
rs.Open(CRecordset::dynaset, “SELECT * FROM authors”,
    CRecordset::none);
```

Ensimmäinen **CRecordset::Open()**-funktion parametreistä määrittelee vastaanotettavaa tietoa varten määriteltävän tietokantaosoittimen tyypin taulukossa 7.3 kuvatulla tavalla.

Taulukko 7.3 CRecordset::Open-funktion kohdistintyypit

Parametri	Tarkoitus
CRecordset::dynaset	Luo luonteeltaan dynaamisen tietojoukon, jota voidaan vierittää molempiin suuntiin. Tietojen järjestys määritellään tietojoukkoa avattaessa. Dynasetit soveltuvat suurimpaan osaan tietokantaoperaatioista, koska ne minimoivat muistin käytön varastoimalla vain tulosjoukon avaimet ja koska ne pystyvät esittämään suurimman osan muiden käyttäjien tekemistä muutoksista. Ainoat muutokset, jotka eivät näy, ovat tietueiden lisäykset.
CRecordset::snapshot	Luo tietojoukon, jossa muiden käyttäjien tietolähteeseen tekemät muutokset eivät näy, mutta jota voidaan selata sekä eteen että taakse päin. Snapshot on käyttökelpoinen, kun haluat varmistua, ettei tietojoukko muutu käytön aikana, esimerkiksi luodessasi raporttia.
CRecordset::dynamic	Luo tietojoukon, joka on luonteeltaan todella dynaaminen. Voit selata tietojoukkoa eteen ja taaksepäin ja kaikki käyttäjien tekemät muutokset näkyvät tietojoukossa. Dynaaminen tietokantaosoitin kuluttaa eniten resursseja. Dynamic on käyttökelpoinen silloin, kun tulosjoukon täytyy olla jatkuvasti ajantasalla.
CRecordset::forwardonly	Luo vain-luku-tyyppisen tietojoukon, jota voidaan selata vain eteenpäin. Forward-only-osoitin on nopein ja vähiten muistia tarvitseva osoitin. Monet tietokannan käsittelytehtävät sisältävät yksittäisen tulosjoukon läpikäynnin, jolloin tämä osoitin on käyttökelpoinen.

Open-funktion toinen parametri on SQL-komentojono. Viimeinen parametri on bittimaski, jonka avulla voidaan määritellä, onko tietojoukko vain-luku vai vain-lisäys-tyyppinen, sallitaanko useiden rivien muokkaaminen ja niin edelleen. Arvo **CRecordset::none** ilmaisee, että mitään optiota ei ole asetettu. Täydellisen listan tämän parametrin arvoista saat tekemällä haun Visual C++:n ohjeeseen hakusa-nalla "**CRecordset::Open**".

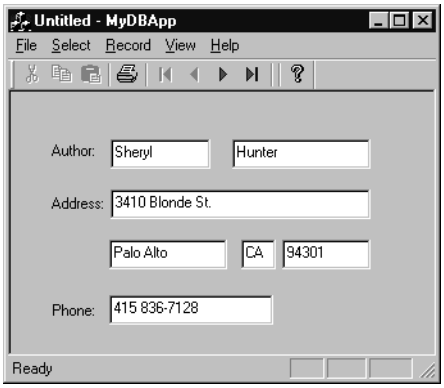
CDaoRecordset::Open() on samanlainen kuin **CRecordset::Open()**. **CDaoRecordset**-luokan avulla voidaan luoda vain dynaset, snapshot, tai ***Table*** -tietojoukko. **Table**-tietojoukko on päivitettävissä oleva tietojoukko, joka edustaa yhden tietokantataulun tietueita. Snapshot-tietojoukot ovat **CDaoRecordset**-luokassa vain-luku-tyyppisiä.

Kun lopetat työskentelyn tietojoukolla, sinun täytyy kutsua lopuksi tietojoukon **Close()**-funktiota. Varmista, että suljet tietojoukon *ennen* kuin yrität sulkea siihen liittyvän tietokantaobjektin.

CRecordView ja CDaoRecordView

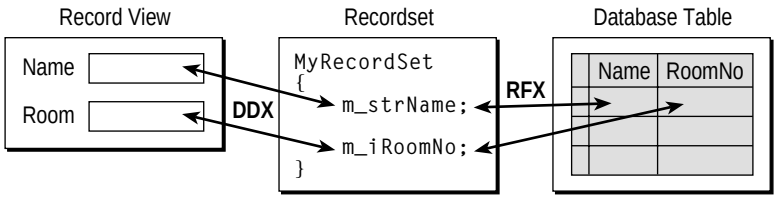
Tietuenäkymäluokat antavat keinot tietojoukon sisältämien tietojen tarkastelemiseen. **CRecordView** ja **CDaoRecordView** on periytetty ***CFormView***-luokasta, joka sisältää sovelluksille tarkoitetun, dialogiin perustuvan työalueresurssin. Tällaista sovellusta kutsutaan usein *lomakepohjaiseksi* (forms-based) sovellukseksi.

Dialogin kontrollit voidaan tietuenäkymäluokkien avulla yhdistää dialogiin liittyvän tietojoukon jäsenmuututtujiin. Käyttäjä voi katsella tietueen tietoja lomakkeen kontrolleista kuten kuvasta 7.7 nähdään.



Kuva 7.7 **CRecordView**-pohjainen sovellus

Tiedonsiirto tietojoukosta valitun tietueen ja tietuenäkymän kontrollin välillä hoidetaan DDX/DDV-mekanismin (dialog data exchange/dialog data validation) avulla. Mekanismi on toteutettu tietuenäkymäluokissa. **CWnd::UpdateData()**-metodia kutsutaan tietuenäkymäluokan **OnMove()** funktiosta, jota kutsutaan, kun käyttäjä siirtyy valittuna olleesta tietueesta. Jos tietojoukko ei ole vain-luku-tilassa, tietojoukon tiedot päivitetään tietuenäkymäkontrollien sisältämiä arvoja vastaaviksi. RFX/DFX-mekanismi välittää nämä muutokset tietolähteen tauluihin. Kuva 7.8 havainnollistaa ODBC-tietokantaobjektien ja tiedonvaihtomekanismin välistä suhdetta.



Kuva 7.8 DDX ja RFX

Tietokannan virheet

Ulkoisten resurssien kuten tietokantojen käsitteleminen sisältää suuria riskejä. Tietokanta voi olla offline-tilassa tai käyttäjältä puuttuvat tarvittavat tietokannan käsittelyn käyttöoikeudet. Tietokantojen ja tietojoukkojen avaamisessa käytetyt komennot tulisi aina sijoittaa **try… catch…**-lohkoon, jonka avulla voidaan saada käsittelyyn kaikki mahdolliset virheet. MFC:llä on omat, erityisesti tietokantoja varten tarkoitetut poikkeusluokat, jotka kertovat tietokantavirheen luonteen. Luokat ovat samanlaisia kuin **CFileException**-luokka.

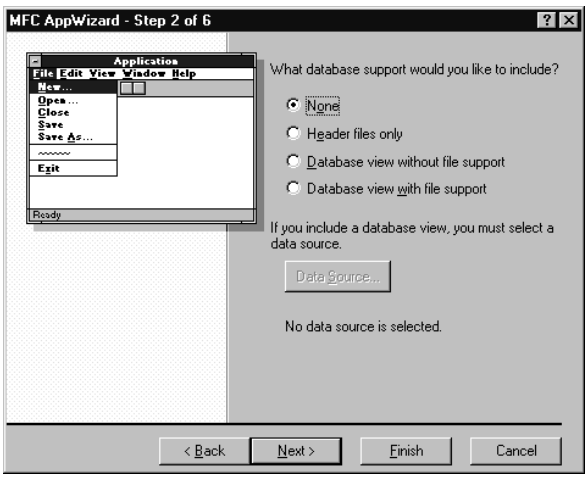
CDatabase::Open() ja **CRecordset::Open()**-funktiot nostavat **CDBException**-poikkeuksen virheen sattuessa. **CDaoDatabase::Open()** ja **CDaoRecordset::Open()**-funktiot nostavat **CDaoException**-poikkeuksen. Tietokantoihin liittyvästä poikkeuskäsittelystä on esimerkki myöhemmin tällä oppitunnilla.

Tietokantasovelluksen tekeminen AppWizardilla

Jotta saisit paremman käsityksen tietokanta-, tietojoukko- ja tietuenäkymäluokkien yhteistoiminnasta, teemme seuraavaksi tietokantasovelluksen AppWizardia käyttämällä. Teemme lomakepohjaisen sovelluksen, joka hakee ja vie tietoja SQL Serverin *pubs*-mallitietokantaan käyttämällä luvussa 1 luotua ODBC-tietolähdettä. Vaikka seuraavissa esimerkeissä käytetäänkin ODBC-luokkia, esitetyt tekniikat voidaan helposti sijoittaa projektiin, joka perustuu DAO-luokkiin.

► **MyDBApp-sovelluksen luominen**

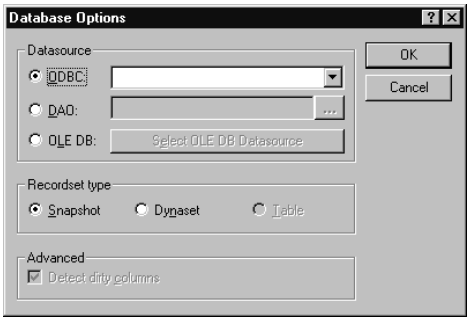
1. Luo **MFC AppWizard (exe)** -projekti valitsemalla **File** Visual C++:n File-valikosta. Anna projektille nimi **MyDBApp**.
2. Valitse **Single document** AppWizardin vaiheessa 1. Siirry kuvassa 7.9 esitettyyn vaiheeseen 2 napauttamalla **Next**.



Kuva 7.9 AppWizardin tietokantavalinnat

Vaiheessa 2 voit valita neljästä vaihtoehdosta sovellukseesi luotavan tietokantatuen tason. Jos tietokantatukea ei tarvita, valitse ensimmäinen vaihtoehto. Valitsemalla toisen vaihtoehdon saat lisättyä projektiisi tietokantoihin liittyvät header-tiedostot, joiden avulla voit käyttää tietokantaluokkia ja luoda tietojoukkoja manuaalisesti. Valitse kolmas tai neljäs vaihtoehto, jos haluat tehdä dokumentti/näkymä-pohjaisen sovelluksen, jossa dokument-tiluokka pitää tietojoukon sisällään; näkymäluokka periytetään joko **CRecordView** tai **CDAORecordView** -luokasta. Kolmas vaihtoehto ei sisällä serialisointiin tarkoitettuja rutiineja ja sitä käytetään tavallisesti, kun halutaan luoda yksinkertainen lomakepohjainen sovellus, jonka avulla voidaan tutkia ja/tai muuttaa tietokannassa olevia tietoja. Käytä neljättä vaihtoehtoa, jos tarvitset tukea sarjamuotoisille tietolähteille.

3. Valitse vaihtoehto **Database view without file support**. Koska olet valinnut tietokantanäkymän sovellukseesi, sinun täytyy valita projektiin luotavalle tietojoukkoluokalle tietolähde. Napauta **Data Source**. Kuvassa 7.10 näkyvä dialogi **Database Options** avautuu.



Kuva 7.10 **Database Options** -dialogi

4. Varmista, että **ODBC**-optio on valittuna. Näet kaikki tietokoneelle rekisteröidyt tietolähteet avaamalla alasvetovalikon. Valitse **MyDSN**.
5. Varmista, että tietojoukkotyypeistä on valittuna **Snapshot**, ja napauta **OK**. Huomaa, että SQL Server -palvelun täytyy olla toiminnassa, jotta tämä vaihe voitaisiin suorittaa onnistuneesti loppuun. **Select Database Tables** -dialogi avautuu.
6. Napauta ensin **dbo.authors** ja sitten **OK** (**dbo** on SQL Serverin tauluun liittämä *omistajan nimi*).
7. Jäljellä oleviin AppWizardin ruutuihin ei tehdä muutoksia, joten voit jatkaa napauttamalla **Finish**-painiketta. Luo projekti napauttamalla **New Project Information** -dialogissa **OK.**

Luo projekti napauttamalla OK-painiketta New Project Information -dialogissa. Uuden projektin **IDD_MYDBAPP_FORM**-dialogimalli avautuu dialogieditoriin. **IDD_MYDBAPP_FORM** on dialogimalli, johon tietuenäkymäluokka **CMyDBAppView** perustuu. Mallin avautuminen tässä vaiheessa vihjaa, että sinun täytyy lisätä siihen tietojen esittämisessä tarvittavat kontrollit ennen kuin siinä voidaan näyttää tietojoukon tietoja.

Ennen kuin luot dialogimallin tietueiden katselua varten, sinun kannattaa tutkia hetki AppWizardin luomaa koodia. Saat sen avulla käsityksen siitä, kuinka tietokantasovellus on toteutettu.

► **Luodun koodin tarkasteleminen**

1. Tutki tietojoukon luokkamäärittelyä avaamalla MyDBAppSet.h. Huomaa AppWizardin luokkaan lisäämät RFX-jäsenmuuttujat, joita on yksi jokaista author-taulun saraketta kohden.
2. Avaa **MyDBAppSet.cpp** ja etsi **CMyDBAppSet::DoFieldExchange()**-funktio. Huomaa, että se on samanlainen kuin DDX:n käyttämä DoDataExchange()-funktio. AppWizard lisää tähän funktioon kutsun sopivan tyyppiseen RFX-funktioon jokaista authors-taulun saraketta kohden.
3. **GetDefaultSQL()**-jäsenfunktio, joka on juuri **DoFieldExchange()**-funktion yläpuolella, määrittelee **SELECT**-kommennon **FROM**-määreen, johon tietojoukko perustuu:

```
CString CMyDBAppSet::GetDefaultSQL()
{
    return _T("[dbo].[authors]");
}
```

Huomaa, että ODBC-objektin nimet on kirjoitettu hakasulkuihin. Vaikka tämä on tarpeen vain, jos nimet sisältävät välilyönteja, käytetään automaattisesti muodostetussa koodissa aina hakasulkuja.

Voit määritellä suodattimen tietojoukkoosi sijoittamalla tietojoukkoluokan m_strFilter-jäsenmuuttujaan merkkijonon, joka määrittelee **WHERE**-ehdon.

Voit myös järjestää tietojoukon yhdistämällä m_strSort-jäsenmuuttujaan **ORDER BY** -ehdon määrittelevän merkkijonon.

4. **GetDefaultConnect()**-jäsenfunktio, joka on juuri ennen **GetDefaultSQL()**-funktiota, palauttaa yhteysmerkkijonon, joka määrittelee tietojoukon käyttämän tietolähteen.

```
CString CMyDBAppSet::GetDefaultConnect()
{
    return _T("ODBC;DSN=MyDSN");
}
```

Framework ei tee **CDatabase**-luokkaan pohjautuvaa objektia projektiisi. Tietojoukon muodostin saa **NULL**-parametrin, joka aiheuttaa väliaikaisen **CDatabase**-objektin luomisen ja käyttöönoton.

Jos käytät tietolähteen SQL Server -oikeuksien tarkistustoimintoa, voit estää **SQL Server Login** -dialogin ilmestymisen näytölle ohjelmaa suoritettaessa lisäämällä tunnistusinformaation seuraavaan yhteysmerkkijonoon:

```
CString CMyDBAppSet::GetDefaultConnect()
{
    return _T("ODBC;DSN=MyDSN;UID=sa;PWD=");
}
```

Esitetyssä merkkijonossa oletetaan, että **sa**-tilille ei ole määritelty salasanaa.

5. Avaa lopuksi MyDBAppDoc.h tiedosto, niin pääset tutkimaan dokumenttiluokan määrittelyä. Huomaa, että **CMyDBAppDoc**-luokka sisältää julkisen jäsenmuuttujan m_myDBAppSet. **CMyDBApp**-sovellus pohjautuu dokumentti/näkymämalliin - dokumenttiluokka sisältää sovelluksen tiedot, jotka näkymäluokka esittää näytöllä (**CMyDBAppView**-luokka).

Saat tietojoukon toimimaan tehokkaammin, jos poistat ne RFX-jäsenmuuttujat ja funktiot, jotka liittyvät sellaisiin authors-taulun sarakkeisiin, joita sovelluksesi ei tarvitse. Seuraava harjoitus näyttää, kuinka tämä tehdään ClassWizardin avulla.

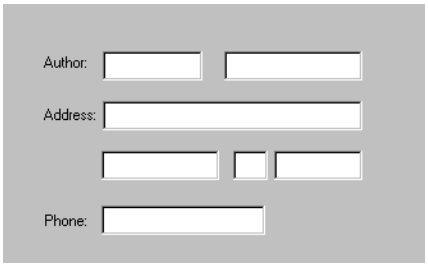
► **Ylimääräisten kenttien poistaminen tietojoukosta**

1. Avaa ClassWizard painamalla CTRL+W, ja valitse **Member Variables** -välilehti.
2. Valitse **CMyDBAppSet**-luokan nimi. Valitse jäsenmuutujien listasta **m_au_id**, joka vastaa [au_id]-saraketta. Napauta **Delete Variable**.
3. Tee sama toimenpide **m_contract**-muuttujalle, joka vastaa [contract]-saraketta. Tallenna muutokset napauttamalla **OK**-painiketta.

Seuraavaksi palaat IDD_MYDBAPP_FORM dialogimalliin, jossa lisätään tietojoukon muuttujien sisältämien tietojen esittämisessä tarvittavat kontrollit.

► **Tietuenäkymädialogimallin mukauttaminen**

1. Lisää dialogieditorilla kontrolleja IDD_MYDBAPP_FORM dialogimalliin. Lisää kolme staattista tekstikontrollia ja seitsemän muokkausruutua kuvan 7.11 mallin mukaisesti.



Kuva 7.11 IDD_MYDBAPP_FORM-dialogimalli

2. Yhdistä seuraavat tunnisteet muokkausruutuihin:
   - **IDC_AU_FNAME**
   - **IDC_AU_LNAME**
   - **IDC_AU_ADDRESS**
   - **IDC_AU_CITY**
   - **IDC_AU_STATE**
   - **IDC_AU_ZIP**
   - **IDC_AU_PHONE**

Seuraavaksi yhdistetään ClassWizardin avulla tietojoukkoluokan jäsenmuuttujat näihin tunnisteisiin.

► **Tietojoukon muuttujien yhdistäminen tietuenäkymädialogin kontrollien tunnisteisiin**

1. Valitse **CMyDBAppView**-luokka ClassWizardin **Member Variables** -välilehdeltä.
2. Napauta **Control IDs** -ruudussa kohtaa **IDC_AU_ADDRESS**. Napauta **Add Variable**.
3. Avaa **Member variable name** -alasvetovalikko, josta näet luettelon tietojoukkoluokan jäsenmuuttujista, jotka voidaan yhdistää tämän tietuenäkymän kontrolleihin. Valitse **m_pSet->m_address** -muuttuja.
4. Varmista, että **Value** on valittu **Category**-ruudusta, ja että **CString** on valittu **Variable** type -ruudusta, ja napauta lopuksi OK.
5. Liitä muut loput tietojoukon muuttujista kontrollien tunnisteisiin toistamalla edelliset vaiheet Aseta IDC_AU_STATE kontrollille syöttörajoite, joka sallii korkeintaan kahden merkin syöttämisen.

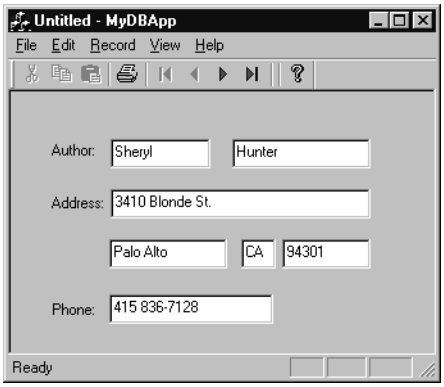
6. Tallenna muutokset ja sulje ClassWizard napauttamalla **OK**.

Tutustu ClassWizardin tekemiin muutoksiin käymällä MyDBAppView.h ja MyDBAppView.cpp-tiedostot läpi. Huomaa, että et ole lisännyt yhtään jäsenmuut-tujaa **CMyDBAppView**-luokkaan. Olet pelkästään liittänyt kontrolleja **CMyDBAppSet**-luokan olemassaoleviin jäsenmuuttujiin. ClassWizard toteuttaa yhdistämisen käyttämällä **DDX_FieldText()**-funktiota tietuenäkymän **DoDataExchange()**-funktion sisällä, seuraavaan tapaan:

```
void CMyDBAppView::DoDataExchange(CDataExchange* pDX)
{
    CRecordView::DoDataExchange(pDX);
    //{{AFX_DATA_MAP(CMyDBAppView)
    DDX_FieldText(pDX, IDC_AU_ADDRESS, m_pSet->m_address, m_pSet);
    DDX_FieldText(pDX, IDC_AU_CITY, m_pSet->m_city, m_pSet);
    DDX_FieldText(pDX, IDC_AU_FNAME, m_pSet->m_au_fname, m_pSet);
    DDX_FieldText(pDX, IDC_AU_LNAME, m_pSet->m_au_lname, m_pSet);
    DDX_FieldText(pDX, IDC_AU_PHONE, m_pSet->m_phone, m_pSet);
    DDX_FieldText(pDX, IDC_AU_STATE, m_pSet->m_state, m_pSet);
    DDV_MaxChars(pDX, m_pSet->m_state, 2);
    DDX_FieldText(pDX, IDC_AU_ZIP, m_pSet->m_zip, m_pSet);
    //}}AFX_DATA_MAP
}
```

Huomaa, että standardia DDV-tarkistusfuktiota voidaan edelleen käyttää.

Käännä ja käynnistä MyDBApp-sovellus. Sovelluksen tulisi näyttää samalta kuin kuvassa 7.12, vaikka saatatkin nähdä eri tietueen tiedot.



Kuva 7.12 MyDBApp-sovellus

Huomaa, että **CRecordView**-luokka lisää työkaluriviin joukon painikeita, joiden avulla voit selata tietojoukon teitueita eteen ja taakse sekä hypätä tietojoukon alkuun tai loppuun. Framework yhdistää näiden painikkeiden kontrollitunnisteet

tietuenäkymän OnMove()-funktioon. **OnMove()**-funktion perustoteutus kutsuu **CWnd::UpdateData(TRUE)**-funktiota tallentaessaan tietueisiin tehdyt muutokset, kun siirryt pois tietueesta ja kun siirryt tietueeseen, se päivittää näkymän tiedot kutsumalla **CWnd::UpdateData(FALSE)**-funktiota. **UpdateData** päivittää näkymän kutsumalla sen **DoDataExchange()**-funktiota.

Tietojoukon suodattaminen

Mainitsimme aikaisemmin, että voit suodattaa tietojoukkoa asettamalla **WHERE**-ehdon määrittävän merkkijonon tietojoukon m_strFilter-jäsenmuuttujaan. Seuraavassa harjoituksessa näet, kuinka voit soveltaa tätä tekniikkaa MyDBApp- sovellukseen niin, että authors-taulun tiedot saadaan suodatettua asuinpaikan mukaan. Harjoituksessa tehdään myös valikkokomento, jonka kautta käyttäjä pääsee valitsemaan asuinpaikkoja luettelosta, joka pohjautuu authors-taulun tietoihin. Valittua osavaltiokoodia käytetään tietojoukon suodattamiseen.

► **MyDBApp-sovelluksen käyttöliittymän päivittäminen**

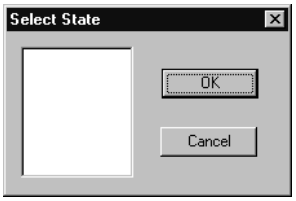
1. Avaa **IDR_MAINFRAME** valikko menueditoriin MyDBApp-projektissa. Poista **Edit**-valikko, jota ei käytetä MyDBApp-sovelluksessa.
2. Tee **Edit**-valikon paikalle uusi valikko, jonka otsikoksi tulee **&Filter**. Sijoita tähän valikkoon komento, jonka otsikoksi asetat tekstin **&State**. Tämä komento pitää yhdistää tunnisteeseen **ID_FILTER_STATE**. Sulje menueditori.
3. Poista **IDR_MAINFRAME**-työkaluriviltä **Edit**-valikon komentoja vastaavat painikkeet (**ID_EDIT_CUT**, **ID_EDIT_COPY** ja **ID_EDIT_PASTE**) työkaluruvieditoria käyttämällä. Muista, että poistaminen tehdään vetämällä painikkeet pois työkaluriviltä. Sulje työkaluruvieditori.
4. Avaa ClassWizard ja valitse **Message Maps** -välilehti. Valitse **CMyDBAppDoc**-luokka ja luo tapahtumakäsittelijä **ID_FILTER_STATE**-tunnisteelle. Hyväksy funktion oletusnimi **OnFilterState()**.
5. Tallenna muutokset ja sulje ClassWizard napauttamalla **OK**.

Seuraavaksi luot **Select State** -dialogin, joka näyttää luettelon authors-taulun osavaltiotunnuksista.

► **Select State -dialogin luominen**

1. Napauta hiiren kakkospainikkeella Dialog-kansiota ResourceViewissä. Valitse pikavalikosta **Insert Dialog** -kometo, jolla luodaan uusi dialogimalliresurssi. Tämä resurssi yhdistetään tunnisteeseen **IDD_STATEDIALOG**, ja otsikoksi asetetaan **Select State**.

2. Lisää dialogiin luetteloruutukontrolli ja yhdistä se tunnisteeseen **IDC_STATELIST**. Järjestä kontrollit niin, että dialogi muistuttaa kuvaa 7.13.



Kuva 7.13 **Select State** -dialogi

3. Avaa ClassWizard painamalla CTRL+W, ja luo **IDD_STATEDIALOG**-mallia vastaava dialogiluokka. Anna luokkalle nimi **CStateDialog**.
4. Valitse **IDC_STATELIST**-tunniste ClassWizardin **Member Variables**-väliledeltä. Lisää m_strState-niminen CString-muuttuja ja varastoi luetteloruudusta valittu arvo napautamalla **Add Variable**. Lisää m_statelist-niminen **CListBox**-muuttuja, joka edustaa luetteloruutukontrollia, napauttamalla uudes-taan **Add Variable**.
5. Siirry ClassWizardin **Message Maps** -välilehdelle. Valitse **Object IDs**-luettelosta **CStateDialog** ja **Messages**-ruudusta **WM_INITDIALOG**. Ylikuormita luokassasi oleva **OnInitDialog()** virtuaalifunktio napauttamalla **Add Function**.
6. Siirry muokkaamaan funktiota napauttamalla **Edit Code**. Korvaa // TODO-kommenttirivi seuraavalla koodilla:

```
CDatabase aDB;

try
{
    aDB.OpenEx("DSN=MyDSN");
    // Specify login information if using SQL Server authentication,
    // e.g., aDB.OpenEx("ODBC;DSN=MyDSN;UID=sa;PWD=");

    CRecordset aRS(&aDB);

    aRS.Open(CRecordset::forwardOnly,
        "SELECT DISTINCT state FROM authors");

    while(! aRS.IsEOF())
    {
        CString strValue;
        aRS.GetFieldValue(short(0), strValue);
        m_statelist.AddString(strValue);
        aRS.MoveNext();
    }
```

```
        m_statelist.InsertString(0, “All records”);

        aRS.Close();
        aDB.Close();
    }

    catch(CDBException * ex)
    {
        TCHAR buf[255];
        ex->GetErrorMessage(buf, 255);
        CString strPrompt(buf);
        AfxMessageBox(strPrompt);
    }
```

Huomaa, että tämä funktio käyttää paikallisesti luotuja **CDatabase** ja **CRecordset**-objekteja tietojen noutamiseen tietokannasta. Koodi tekee *forward-only*-tietojoukon, koska sen ei tarvitse käydä tietueita läpi kuin kertaalleen. Tietojoukon avaamisessa käytettyä **SELECT**-komentoa on tarkennettu **DISTINCT**-määreellä, joka määrää, että tietojoukossa voi esiintyä vain yksilöllisiä rivejä. Tästä syystä tieto-joukko sisältää vain yhden tietueen jokaista taulusta löytyvää osavaltiota kohden.

CRecordset::MoveNext()-funktio etenee tietojoukossa kunnes **CRecordset::IsEOF()**-funktio palauttaa arvon tosi. Jokaisen rivin state-sarakkeen arvo noudetaan **CRecordset::GetFieldValue()**-funktiolla. Tämä funktio mahdollistaa tietojen hakemisen dynaamisesti tietojoukon kentästä käyttämällä nollasta lähtevää numeroitua indeksiä. Koska tietojoukko tässä funktiossa palauttaa vain yhden sarakkeen, tiedämme, että haluamamme arvo saadaan antamalla indeksin arvoksi 0. Huomaa, että voit tämän funktion avulla käyttää CRecordset-luokkaa suoraan ilman, että periytät oman luokkasi ja määrittelet RFX-muuttujat.

Tietojoukon riveiltä haetut tiedot lisätään dialogin luetteloruutukontrolliin. **CListBox::InsertString()**-funktion avulla lisätään "All records"-vaihtoehto luettelon alkuun.

Huomaa, että tietokantoja käsittelevä koodi on sijoitettu *try*-lohkon sisään, ja *catch*-käsittelijä hakee tiedot **CDBException**-objektista näytettäväksi käyttäjälle.

Lopuksi tehdään **OnFilterState()**-funktio, joka toteuttaa **Filter**-valikon **State**-komennon. Tämä funktio avaa **Select State** -dialogin ja käyttää käyttäjän valitsemaa osavaltiokoodia suodattaessaan sovelluksen näyttämiä tietueita.

► **OnFilterState()-funktion toteutus**

1. Lisää seuraava rivi MyDBAppDoc.cpp-tiedoston alkuun:

   ```
   #include “StateDialog.h”
   ```

2. Etsi **CMyDBAppDoc::OnFilterState()**-funktio, jonka loit aiemmin. Korvaa // TODO-kommenttirivi seuraavalla koodilla:

```
CStateDialog aSD;

if(aSD.DoModal() == IDOK)
{
    if(aSD.m_strState == “All records”)
        m_myDBAppSet.m_strFilter = “”;
    else
        m_myDBAppSet.m_strFilter.Format(“state = ‘%s’”,
            aSD.m_strState);

    m_myDBAppSet.Requery();

    POSITION pos = GetFirstViewPosition();
    if (pos != NULL)
    {
        CView* pView = GetNextView(pos);
        ASSERT_VALID(pView);
        pView->UpdateData(FALSE);
    }
}
```

CRecordset::Requery()-funktio päivittää tietojoukon suodattimen muuttamisen jälkeen. Uuden kyselyn jälkeen valittuna on uuden tietuejoukon ensimmäinen tietue. On tärkeää varmistaa, että kutsut tietuenäkymän **UpdataData()**-funktiota, joka päivittää kontrollien näyttämät tiedot vastaamaan nykyisen tietueen tietoja.

Voit parametrisoida tietojoukon käyttämällä kysymysmerkkiä (?) suodatusmerkkijonossa parametrinä seuraavaan tapaan:

```
m_myDBAppSet.m_strFilter = “state = ?”;
```

Voit korvata kysymysmerkin suoritusaikaisilla arvoilla RFX-mekanismin avulla. Lisää tietoja tietojoukon parametrisoinnista tällä tavoin saat Visual C++:n ohjeen artikkelista "Recordset: Parameterizing a Recordset (ODBC)". Parametrin käyttäminen tällä tavalla on tehokkaampaa kuin suodatusmerkkijonon yksinkertainen korvaaminen. Tietokannan ei tarvitse suorittaa SQL:n **SELECT**-komentoa kuin kerran parametrisoidulle tietojoukolle. Suodatetulle tietojoukolle, jossa ei ole käytetty parametrejä, täytyy **SELECT**-komento suorittaa joka kerran, kun suodattimelle annetaan uusi arvo.

Oppitunnin yhteenveto

MFC:hen kuuluu kaksi erillistä luokkien ryhmää, jotka mahdollistavat tietokantojen käsittelemisen: toinen on tarkoitettu käytettäväksi DAO-ympäristössä ja toinen ODBC:n kautta. Molemmat luokkaryhmät ovat samanlaisia ja perustuvat yleiseen ohjelmointimalliin.

Kolmen yhteistyössä toimivan ydinluokan avulla voit hakea tietoja tietolähteestä niin, että voit esittää ne dialogipohjaisessa näkymässä selaamista tai päivittämistä varten. Nämä luokat ovat:

- **CDatabase/CDaoDatabase**
- **CRecordset/CDaoRecordset**
- **CRecordView/CDaoRecordView**

Tietokantaluokkia käytetään yhteyden muodostamiseen tietolähteeseen tai tietokantatiedostoon. Tämä tapahtuu luomalla sopivan tyyppinen objekti ja kutsumalla objektin **Open()**-funktiota. **Open()**-funktio antaa sinun määritellä yhteystiedot yhteysmerkkijonoa käyttämällä, ja muodostetaanko yhteys yksinoikeudella tai vain-luku-oikeuksin.

Microsoft suosittelee, että käytät **CDatabase:: OpenEx()**-funktiota ODBC-yhteyksissä. Tämän funktion lippujen avulla voit antaa ODBC:n käyttämiä lisätietoja.

Kun lopetat työskentelyn tietokantaobjektin kanssa, sinun tulisi käyttää aina **Close()**-funktiota sen sulkemiseen.

Tietojoukko kapseloi joukon tietokannasta poimittuja tietueita. Tietojoukko mahdollistaa tietojen läpikäymisen tietue tietueelta, tietueiden päivittämisen, suodattamisen ja järjestämisen. Tietojoukko-objektit sisältävät jäsenmuuttujia, jotka vastaavat tietokannan taulusta valittuja sarakkeita. Kun käyttäjä selaa tietojoukkoa, RFX päivittää vuorossa olevaa tietuetta vastaavat tiedot jäsenmuuttujiin.

Tietojoukko-objekti voidaan luoda yhdessä tietokantaobjektin kanssa. Sinun täytyy välittää olemassaolevan tietokantaobjektin osoite tietojoukon muodostimelle. Periytetyn tietojoukkoluokan muodostimelle voit antaa parametriksi NULL-arvon objektia luotaessa, jolloin framework luo väliaikaisen tietokantaobjektin tietojoukkoa varten. Yhteystiedot välitetään väliaikaiselle tietokantaobjektille ylikuormittamalla **GetDefaultConnect()**-jäsenfunktio.

Kun olet luonut tietojoukko-objektin, voit hakea tiedot tietojoukkoon kutsumalla **Open()**-jäsenfunktiota, joka suorittaa kyselyn tietokantaan.

Open()-funktio määrittelee tietojoukkoa varten luotavan tietokantaosoittimen tyypin. **CRecordset**-objektille tämä tietojoukko voi olla jokin seuraavista:

- **dynaset** Tietojoukko on luonteeltaan dynaaminen ja sitä voidaan vierittää molempiin suuntiin.
- **snapshot** Tietojoukko ei näytä muiden käyttäjien tietolähteeseen tekemiä muutoksia ja sitä voidaan vierittää molempiin suuntiin.
- **dynamic** Tietojoukko on luonteeltaan aidosti dynaaminen. Tietojoukkoa voidaan vierittää molempiin suuntiin ja se järjestetään uudelleen aina kun käyttäjä tekee muutoksia tietolähteen sisältöön.

- **forwardonly** Vain-luku-tietojoukko, jota voidaan selata vain eteenpäin.

CDaoRecordset-objekti tukee vain-luku-, dynasets- ja taulukkotietojoukkoja, jotka pohjautuvat yhteen kokonaiseen tauluun .mdb-tietokannassa.

Tietojoukon **Open()**-funktio käyttää SQL-komentojonoa tietojoukon tietueiden valitsemiseen. **CDaoRecordset**-objekti voi ottaa DAO-taulun tai kyselymäärittelyn nimen.

Käytä **Close()**-funktiota tietojoukkojen sulkemiseen, kun lopetat niiden kanssa työskentelyn.

Tietuenäkymäluokka sisältää keinot tietojoukon tietojen esittämiseen näkymäpohjaisessa dialogimalliresurssissa. Tietuenäkymän kontrollit on yhdistetty siihen liittyvän tietojoukon jäsenmuuttujiin. Tietojen vaihtoa nykyisen tietueen ja tietuenäkymän kontrollien välillä käsitellään DDX/DDV mekanismilla.

CDatabase::Open() ja **CRecordset::Open()**-funktiot nostavat **CDBException**-poikkeuksen virhetilanteessa. **CDaoDatabase::Open()** ja **CDaoRecordset::Open()**-funktiot nostavat **CDaoException**-poikkeuksen.

MFC AppWizard (.exe) toiminnon vaihe 2 antaa sinulle mahdollisuuden määritellä sovellukseen sisällytettävän tietokantatuen tason. Voit yksinkertaisesti lisätä pelkät tietokantaluokkien header-tiedostot projektiisi ja käyttää tietokantaluokkia manuaalisesti. Vaihtoehtoisesti voit luoda dokumentti/näkymäpohjaisen sovelluksen, jonka dokumenttiluokka sisältää tietojoukon ja näkymäluokka on periytetty joko **CRecordView** tai **CDaoRecordView** -luokasta. Voit lisätä sovellukseen tiedostojenkäsittelyrutiinit tai yksinkertaisen lomakepohjaisen sovelluksen, jolla voi katsella ja/tai päivittää tietokannassa olevia tietoja.

Kun valitset tietokantanäkymän sovellukseesi, sinun täytyy määritellä sovelluksen tietojoukkoluokan käyttämä tietolähde. AppWizard luo projektiisi tietojoukko- ja tietonäkymäluokat sekä tyhjän dialogimallin, johon voit lisätä tietojoukon tietojen esittämiseen tarvittavat kontrollit. Tietojoukkoluokka lisää työkaluriville joukon painikkeita, joiden avulla voit liikkua tietojoukossa. Voit suodattaa tietojoukkoa asettamalla **WHERE**-ehdon määrittävän merkkijonon tietojoukkoluokan m_strFilter-jäsenmuuttujaan. Voit myös lajitella tietojoukon tietueet asettamalla **ORDER BY** -määreen määrittävän merkkijonon m_strSort-jäsenmuuttujaa. Parhaan mahdollisen tehon saavuttamiseksi kanattaa m_strFilter-merkkijono parametrisoida ja käyttää RFX-mekanismia parametrien määrittelyyn.

Oppitunti 3: ADO:n esittely

Microsoft suosittelee nykyisin ADO:n käyttämistä standardirajapintana kaikenlaisiin ulkoisiin tietolähteisiin. Koska ADO perustuu COM-tekniikkaan, et voi op-pia kuinka ADO APIa käytetään ennen kuin tutustut COM:iin tarkemmin. Luvus- sa 10, *COM-asiakkaat*, kerrotaan, kuinka voit luoda ADO-objekteja omassa C++-sovelluksessasi.

Johdantona ADO-tekniikkaan tällä oppitunnilla kerrotaan, kuinka ADO Data -kontrollia käytetään OLE DB -tietolähteestä haettujen tietojen näyttämiseen dialogipohjaisessa sovelluksessa.

Tämän oppitunnin jälkeen:

- Tiedät, mitä etua ADO-objektien käyttämisestä on.
- Osaat kuvata ADO:n objektimallin rakenteen ja siihen kuuluvat osat.
- Tiedät, kuinka ADO Data -kontrollia ja tietokantasidottuja ActiveX-kontrolleja käytetään OLE DB -tietolähteen tietojen näyttämiseen.

Oppitunnin arvioitu kesto: 30 minuuttia

ADO:n käytöstä saatavat hyödyt

ADO suunniteltiin toimimaan OLE DB:n helppokäyttöisenä sovellusrajapintana. ADO on helppokäyttöinen, koska se sisältää OLE DB -rajapinnat abstraktoivat Automaatio-objektit. Tämä antaa ohjelmoijalle mahdollisuuden kes-kittyä OLE DB:n monimutkaisuuden sijasta ratkaistavana olevaan ongelmaan. Missä tahansa kehitysympäristössä, joka tukee COM ja Automation-tekniikoita ja sisältää skriptikielen kuten Microsoft Visual Basic Scripting Edition (VBScript) ja Microsoft JScript, voidaan käyttää ADO-objekteja. Tämä tarkoittaa sitä, että ADOa voidaan käyttää Web-pohjaisessa ohjelmoinnissa, käytettäessä esimerkiksi Active Server Pages (ASP) tekniikkaa, aivan vastaavalla tavalla kuin käytettäessä perinteisiä sovelluskehittimiä kuten Visual C++ ja Visual Basic.

Lisätietoja Automation-tekniikasta saat luvun 3 oppitunnilta 8.

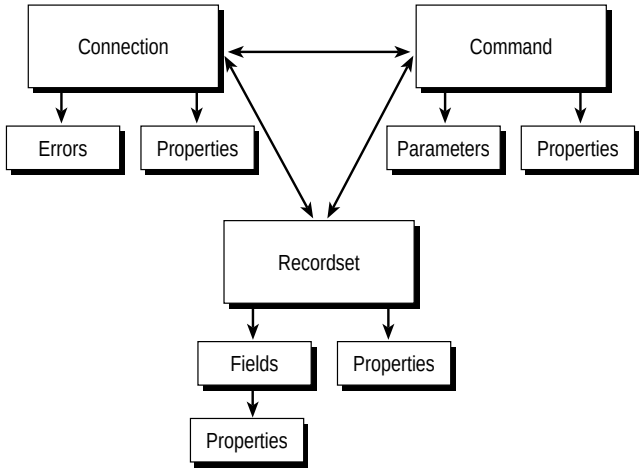
ADO-tietojoukot ovat ainutlaatuisia, koska ne voivat toimia erillään tietolähteestä. Irrotettu tietojoukko voidaan siirtää toiselle sovellukselle ja siihen voidaan tehdä muutoksia ilman verkkoliikennöintiä tai yhteyttä tietolähteeseen. Tämä ominaisuus on erityisen hyödyllinen Web-pohjaisissa sovelluksissa.

ADOn objektimallissa on vähemmän objekteja ja se on helpompi käyttää verrattuna muihin tietolähdeobjekteihin kuten DAO tai RDO.

ADO:n objektimalli

ADO:n objektimalli on suunniteltu niin, että se sisältää OLE DB:n kaikkein eniten käytetyt ominaisuudet. Kuten näet kuvasta 7.14, ADO:n objektimallissa on kolme pääkomponenttia, **Connection**-objekti, **Command**-objekti ja **Recordset**-objekti:

- **Connection**-objekti muodostaa yhteyden sovelluksen ja ulkoisen tietolähteen kuten SQL Server välillä. **Connection**-objekti sisältää myös mekanismit yhteyden alustamiseen ja ylläpitämiseen, kyselyiden tekemiseen ja tiedonsiirtoon. Se on ADO-objektimallin korkeimmalla tasolla.
- **Command**-objekti muodostaa kyselyt, joita tarvitaan tietojen keräämiseen tietolähteestä, käyttäjän määrittelemät parametrit mukaan lukien. Tyypillisesti nämä tietueet palautetaan **Recordset**-objektissa. **Command**-objektit luodaan joko tietokannan taulusta tai SQL-kyselystä. Voit myös luoda yhteyksiä **Command**-objektien välille hakeaksesi joukon yhteenkuuluvia tietoja.
- **Recordset**-objekti kokoaa SQL-kyselyn palauttamat tietueet. Voit antaa käyttäjille mahdollisuuden muuttaa, lisätä tai poistaa tietueita tietolähteestä **Recordset**-objektien avulla.



Kuva 7.14 ADO:n objektimalli

ADO:n objektimalli poikkeaa DAO ja RDO -malleista, koska siinä on mahdollista luoda suuri osa objekteista täysin toisista objekteista riippumatta. ADO-objektit on varastoitu hierarkkiseen muotoon, mutta hierarkia ei edellytä sellaisten objektien luomista, joita sovelluksessa ei tarvita. Voit luoda **Recordset**, **Connection** tai **Command** -objektit suoraan luomatta ensin niiden kantaluokkien objekteja. Voit esimerkiksi luoda **Recordset**-objektin ilman, että luot ensin **Connection**-objektin. ADO luo tarvittavan **Connection**-objektin puolestasi.

ADO-kokoelmat

Kolmen pääobjektin lisäksi ADO tukee kolmea kokoelmaa, jotka voivat tuoda lisätoiminnallisuutta sovelluksiisi:

- **Errors Collection** Mikä tahansa operaatio, johon ADO-objektit osallistuvat, voi aiheuttaa yhden tai useamman virhetilanteen. Kun virhe tapahtuu, yksi tai useampi error-objekti sijoitetaan **Connection**-objektin **Errors**-kokoelmaan. Jokainen **Error**-objekti edustaa tiettyä lähdevirhettä, ei ADO-virhettä.
- **Parameters Collection** **Command**-objektilla on **Parameter**-objekteista koostuva **Parameters**-kokoelma. Parameters-kokoelmaa käytetään välitettäessä määrättyjä tietoja parametrisoidulle kyselylle tai **Command**-objektin kapseloimalle proseduurille. Se on erityisen käytännöllinen, jos sinun täytyy hakea tulosparametrin arvo varastoidusta proseduurista.
- **Fields Collection** **Recordset**-objektilla on **Field**-objekteista muodostuva **Fields**-kokoelma. Jokainen **Field**-objekti vastaa yhtä tietojoukon saraketta. **Fields**-kokoelmaa käytetään, kun halutaan käsitellä tiettyä kenttää olemassa-olevassa **Recordset**-objektissa.

Omien objektiensa ominaisuuksien lisäksi ADO-objektit tukevat **Properties**-kokoelmaa. Se on kokoelma **Property**-objekteja, jotka sisältävät palvelusta riippuvia tietoja objektista. Esimerkiksi **Connection**-objektin **Properties**-kokoelma sisältää **Property**-objektit, jotka määrittävät nykyisen palvelun sarakkeiden mak-simimäärän ja suurimman sallitun rivikoon.

ADO Data -kontrollin käyttäminen

ADO Data -kontrolli on graafinen ActiveX-kontrolli, täydennettynä tietueiden välillä liikuttaessa tarvittavilla painikkeilla, joka tarjoaa helppokäyttöisen käyttöliittymän, jonka avulla voit luoda tietokantasovelluksia mahdollisimman pienellä koodi määrällä. ADO Data -kontrolli käyttää ADO:n nopeasti luotavia yhteyksiä datatietoisten komponenttien ja tietolähteiden välillä. Datatietoiset kontrollit ovat ActiveX-käyttöliittymäkontrolleja, joilla on kaksi tärkeää piirrettä:

- *DataSource*-ominaisuus, johon voidaan asettaa ADO Data -kontrollin tunniste
- Kyky näyttää tietoja, jotka ne vastaan ottavat niihin sidotuilta ADO Data -kontrolleilta

Kun sidot kontrollin ADO Data -kontrolliin, jokainen kenttä näytetään ja päivitetään automaattisesti sitä mukaa kun tietueita selataan. Tämän toiminnan kontrollit toteuttavat itsenäisesti - sinun ei tarvitse kirjoittaa lainkaan koodia.

Visual C++ sisältää useita datatietoisia ActiveX-kontrolleja kuten Microsoft DataGrid ja Microsoft DataList -kontrollit. Voit myös luoda omia datatietoisia kontrolleja tai hankkia niitä muilta toimittajilta.

Seuraavassa harjoituksessa teet yksinkertaisen dialogipohjaisen sovelluksen, joka käyttää ADO Data -kontrollia ja DataGrid kontrollia *pubs*-tietokannan *authors*-taulun tietueiden esittämiseen. Opit myös kuinka voit asettaa noiden kontrollien ominaisuuksia sovelluksesi lähdekoodissa.

► **ViewDB-sovelluksen luominen**

1. Aloita uusi **MFC AppWizard (.exe)** -projekti ja luo **ViewDB**-niminen sovellus.
2. Valitse **Dialog based** AppWizardin vaiheessa 1 ja napauta sitten **Finish**.
3. Vahvista valinta napauttamalla **OK**.

Dialogipohjainen sovellus on yksinkertaisin mahdollinen sovellus, joka AppWizardilla voidaan luoda; tämän tyyppinen ohjelma ei ole dokumentti/näkymä-sovellus. Sovelluksen pääikkuna on modaalinen dialogi ja sovellus sulkeutuu, kun tämä dialogi suljetaan.

AppWizard luo dialogimallin ja dialogiluokan pääikkunadialogia varten. ViewDB-sovellukseen se on luonut **IDD_VIEWDB_ DIALOG**-mallin ja **CViewDBDlg-** luokan. **IDD_VIEWDB_DIALOG** -dialogimalliresurssin pitäisi olla parhaillaan avattuna dialogieditorissa.

ADO Data -kontrolli ja DataGrid-kontrolli kuuluvat Visual C++:n perusasennukseen. Sinun täytyy kuitenkin lisätä kontrollit projektiisi Components and Controls Gallerya käyttämällä ennen kuin voit käyttää niitä.

ActiveX-kontrollin lisääminen projektiin mahdollistaa sen käyttämisen:

- **Controls**-työkaluriviin lisätään kuvake, jota käyttämällä voit sijoittaa kontrollin dialogimalliin hiirtä käyttämällä.
- Projektiisi lisätään C++-luokat, jotka peittävät kontrollin käyttämät Automation-rajapinnat. Voit tutkia ja asettaa kontrollien ominaisuuksia ja kutsua kontrollien tarjoamia metodeja näiden luokkien kautta.

► **ActiveX-kontrollin lisääminen projektiin**

1. Valitse **Components and Controls Project** -valikon **Add to Project** -toiminnosta.
2. Saat näkyviin kaikki järjestelmääsi rekisteröidyt ActiveX-kontrollit kaksoisnapautamalla **Registered ActiveX Controls** -kansiota **Components and Controls Gallery**:ssa.
3. Valitse **Microsoft ADO Data Control, version 6.0 (OLE DB)** ja napauta **Insert**.
4. Lisää komponentti napauttamalla **OK**. Vahvista, että haluat luoda kontrollien tarvitsemat luokat napauttamalla **OK**.

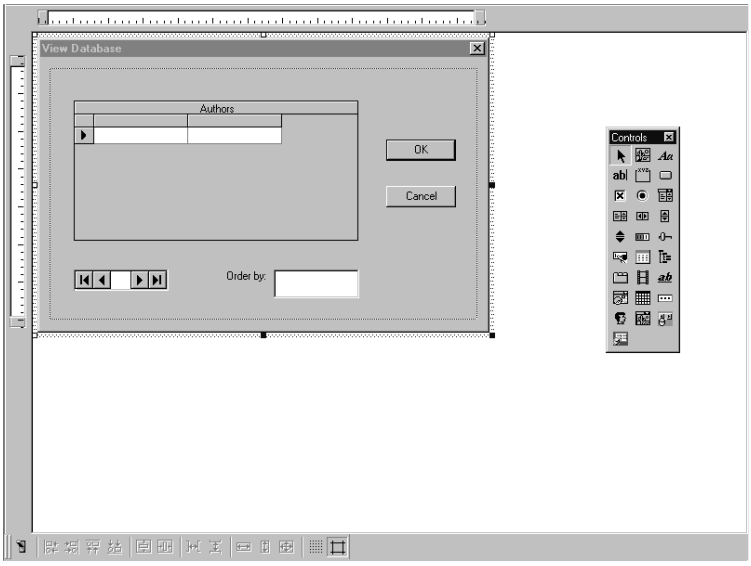
5. Lisää projektiisi **Microsoft DataGrid Control, Version 6.0 (OLE DB)** toistamalla äskeiset vaiheet.
6. Sulje Components and Controls Gallery napauttamalla **Close**.

Näet nyt **Controls**-työkaluriviin lisätyt ADO Data Control ja DataGrid Control -kuvakkeet.

Käytä mallina seuraavalla sivulla olevaa kuvaa 7.15 muokatessasi IDD_VIEWDB _DIALOG-mallia seuraavien vaiheiden mukaisesti:

► **IDD_VIEWDB_DIALOG-mallin muokkaaminen**

1. Poista selitekontrolli, jossa lukee **TODO: Place dialog controls here**.
2. Aseta dialogin otsikoksi teksti **View Database**.
3. Aseta ADO Data -kontrolli kuvan 7.15 mukaisesti. Poista otsikko muokkaamalla ominaisuuksia. Säilytä **IDC_ADODC1** kontrollintunnisteena.
4. Sijoita DataGrid-kontrolli kuvan 7.15 mallin mukaisesti. Muuta caption-tekstiksi **Authors**. Säilytä **IDC_DATAGRID1** tunnisteena. Muuta kontrolli vain-luku-tyyppiseksi tyhjentämällä **AllowUpdate**-valintaruutu DataGrid-kontrollin **Properties**-sivulta.
5. Lisää seliteteksti, jossa lukee Order by:. Lisää sen viereen luetteloruutu, joka on tarpeeksi korkea kahdelle tekstiriville, kuten kuvassa 7.15. Yhdistä luetteloruutu tunnisteeseen **IDC_ORDERLIST**.
6. Järjestä **OK** ja **Cancel** -painikkeet kuten kuvassa 7.15.



Kuva 7.15 IDD_VIEWDB_DIALOG-malli

► **ADO Data -kontrollin yhdistäminen tietolähteeseen**

1. Muokkaa **IDC_ADODC1** ADO Data -kontrollin ominaisuuksia. Varmista, että **Control**-sivulla **Use Connection String** on valittu. Luo yhteysmerkkijono napauttamalla **Build**.
2. **Data Link Properties** -dialogi avautuu. Valitse **Provider**-sivulta **Microsoft OLE DB Provider for SQL Server**. Siirry **Connection**-sivulle napauttamalla **Next**.
3. Kirjoita palvelimen nimeksi **(local)**. Valitse **Use Windows NT integrated security** tai kirjoita SQL Server -tilin nimi ja salasana asiaan kuuluvalla tavalla.
4. Valitse pelvelimelta **pubs**-tietokanta.
5. Napauta **Test Connection**. Jos testi onnistuu, tallenna **Data Link** ominaisuudet napauttamalla **OK**.
6. Avaa uudelleen **ADO Data Control Properties** -dialogi. Valitse **RecordSource**-sivu.
7. Valitse **Command Type** -alasvetovalikosta vaihtoehto **1 – adCmdText**, joka on ADO:n määrittelyvakio, joka määrittelee, että käytät SQL-komentojen tekstimuotoisia määrittelyjä. Huomaa, että muut vaihtoehdot osoittavat, että määrittelet taulun tai varastoidun proseduurin nimen.
8. Kirjoita **Command Text (SQL)** -ruutuun seuraava komento:

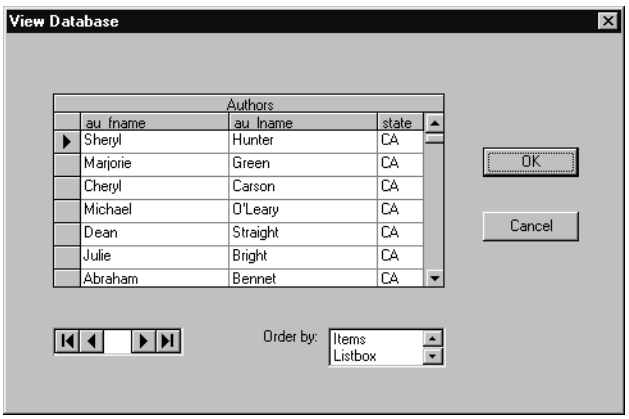
   ```
   SELECT au_fname,au_lname,state FROM authors
   ```

9. Sulje **ADO Data Control Properties** -dialogi.

► **DataGrid kontrollin yhdistäminen ADO Data -kontrolliin**

1. Muokkaa **IDC_DATAGRID1** DataGrid-kontrollin ominaisuuksia. Valitse **All**-sivu, joka näyttää luettelon kaikista muutettavista kontrollin ominaisuuksista. Saadaksesi tämän sivun näkyviin sinun täytyy ehkä vierittää näkymää oikealle **Properties**-dialogin oikeassa yläkulmassa olevalla nuolella.
2. Napauta DataSource-ominaisuuden **Value**-saraketta (jossa on tällä hetkellä teksti **<Not bound to a DataSource>**). Napautuksen jälkeen ilmestyy näkyville avattava valikko. Valitse tästä luettelosta ADO Data -kontrollin tunniste: **IDC_ADODC1**.

   Voit kokeilla dialogia painamalla CTRL+T tai napauttamalla valokatkaisijakuva-ketta **Dialog**-työkalurivin vasemmassa reunassa. Dialogin tulisi näyttää sa-manlaiselta kuin kuvassa 7.16 ja authors-taulusta valittujen sarakkeiden tulisi näkyä DataGrid-kontrollissa. Voit käyttää ADO Data -kontrollin navigointipainikkeita tietojoukon selaamiseen, tai voit valita DataGrid-kontrollin ja selata tietueita käyttämällä hiirtä ja nuolinäppäimiä.



Kuva 7.16 IDD_VIEWDB_DIALOG-dialogin testaaminen

3. Tarkista, että ViewDB-sovellus toimii odotetulla tavalla kääntämällä ja suorittamalla se.

Kontrollien ominaisuuksien asettaminen ohjelmakoodissa

Jos tutkit ViewDB-projektia ClassView-näkymässä huomaat, että se näyttää kaikki luokat, jotka lisättiin projektiisi, kun lisäsit ADO Data -kontrollin ja DataGrid-kontrollin. Erityisen kiinnostavia ovat ***CAdodc*** ja ***CDataGrid*** -luokat, jotka edustavat itse kontrolleja. Jos avaat nämä luokat ClassViewissä, näet niiden sisältävän jäsenfunktiot, joiden avulla voit tutkia ja muuttaa kontrollien ominaisuuksia. Näiden funktioiden nimet muodostuvat ominaisuuden nimestä ja etuliitteestä ***Get*** tai ***Set***. Luokat sisältävät myös jäsenfunktiot, jotka käärivät kontrollien julkistamat metodit.

Seuraavassa harjoituksessa opit, kuinka voit käyttää näitä funktioita omassa ohjelmakoodissasi kontrollien ominaisuuksien asettamiseen ja kontrollien metodien kutsumiseen. Lisäät **Order By** -luetteloruutuun kaksi riviä, joiden avulla käyttäjä voi valita, järjestetäänkö tietueet tekijän sukunimen vai kotivaltion perusteella. Kun käyttäjä muuttaa valintaa, tietolähteestä valittujen tietueiden järjestys muuttuu ja DataGrid-kontrollin otsikko päivitetään.

► **Jäsenmuuttujien lisääminen dialogin kontrolleille**

1. Avaa ClassWizard. Napauta **Member Variables** -välilehteä.
2. Valitse **CViewDBDialog**-luokka. Lisää taulukossa 7.4 luetellut jäsenmuuttujat.

Taulukko 7.4 CViewDBDialog-jäsenmuuttujat

Resource ID	Kategoria	Muuttujan tyyppi	Muuttujan nimi
IDC_ADODC1	kontrolli	CAdodc	m_adodc
IDC_DATAGRID1	kontrolli	CDataGrid	m_datagrid
IDC_ORDERLIST	arvo	CString	m_lbOrder
IDC_ORDERLIST	kontrolli	CListBox	m_strOrder

3. Sulje ClassWizard ja tallenna muutokset napauttamalla **OK**.

► **Order By -luetteloruudun alustaminen**

Paikanna **CViewDBDlg::OnInitDialog()**-funktio. Lisää seuraavat ohjelmarivit funktion loppuun ennen return-komentoa:

```
m_lbOrder.AddString("By last name");
m_lbOrder.AddString("By state");
m_lbOrder.SetCurSel(0);
OnSelchangeOrderlist();
```

Tämä ohjelma lisää luetteloruutuun kaksi uutta riviä ja asettaa valinnan listan ensimmäiselle riville. Voit nyt luoda **OnSelchangeOrderlist()**-funktion käsittelemään **LBN_SELCHANGE**-sanomaa, joka lähetetään käyttäjän muuttaessa luetteloruudun valintaa.

► **OnSelchangeOrderlist()-funktion luominen**

1. Avaa ClassWizard. Napauta **Message Maps** -välilehteä.
2. Valitse **CViewDBDlg**-luokka. Valitse **IDC_ORDERLIST** objektitunniste.
3. Valitse **LBN_SELCHANGE**-sanoma. Napauta **Add Function** ja määritä nimeksi **OnSelchangeOrderlist**.
4. Aloita funktion muokkaaminen napauttamalla **Edit Code**. Korvaa // TODO-kommentti seuraavilla ohjelmariveillä:

```
if(m_lbOrder.GetCurSel() == 0)
{
        m_adodc.SetRecordSource("SELECT au_fname,au_lname,\
        state FROM authors ORDER BY au_lname");
        m_datagrid.SetCaption("Authors by name");
}
else
{
        m_adodc.SetRecordSource("SELECT au_fname,au_lname,\
        state FROM authors ORDER BY state");
        m_datagrid.SetCaption("Authors by state");
}

m_adodc.Refresh();
```

Huomaa, kuinka **CAdodc** ja **CDataGrid** -luokkien jäsenfunktioita käytetään ADO Data -kontrollin ja DataGrid-kontrollin ominaisuuksien asettamiseen ja metodien kutsumiseen. Koodi asettaa ADO Data -kontrollin **RecordSource**-ominaisuuden ja DataGrid-kontrollin **Caption**-ominaisuuden. Koodi kutsuu myös ADO Data -kontrollin **Refresh()**-metodia.

Käännä ja aja DBView-sovellus. Muuta valintaa **Order By** -luetteloruudussa. Tarkista, että DataGrid-kontrollin otsikko ja sen näyttämien tietueiden järjestys muuttuvat odotetulla tavalla.

Oppitunnin yhteenveto

ADO on suunniteltu helppokäyttöiseksi käyttöliittymäksi OLE DB:hen. ADO julkistaa Automation-objektit, jotka abstraktoivat OLE DB -rajapinnan. Tästä syystä skriptikielet kuten VBScript ja JScript voivat käyttää ADOa aivan samalla tavoin kuin Visual C++:n ja Visual Basicin kaltaiset kehitysympäristöt. ADO tukee yhteydettömiä tietojoukkoja, joita voidaan käsitellä vaikka ilman yhteyttä tietolähteeseen ja jotka voidaan lähettää muille sovelluksille.

ADO:n objektimallissa objektit on järjestetty hierarkkisesti itsensä korjaavaan rakenteeseen. Tämä tarkoittaa sitä, että sinun tarvitsee luoda vain ne objektit, joita sovelluksessasi tarvitset. Jos muita objekteja tarvitaan, ne luodaan automaattisesti.

ADO-objektimallin kolme pääkomponenttia ovat:

- **Connection**-objekti muodostaa yhteyden sovelluksen ja ulkoisen tietolähteen kuten SQL Server välillä.
- **Command**-objekti muodostaa kyselyt, joita tarvitaan tietojen keräämiseen tietolähteestä, käyttäjän määrittelemät parametrit mukaan lukien.
- **Recordset**-objekti kokoaa SQL-kyselyn palauttamat tietueet.

Näiden kolmen pääobjektin lisäksi ADO tukee kolmea kokoelmaa, jotka voivat lisätä sovelluksesi toiminnallisuutta:

- **Errors**-kokoelma liittyy **Connection**-objektiin. Varastoi toimittajan virheet.
- **Parameters**-kokoelma liittyy **Command-**objektiin ja varastoi kyselylle tai talletetulle proseduurille välitettävät parametrit.
- **Fields** kokoelma liittyy **Recordset**-objektiin ja sisältää yhteyden tietolähteen sarakkeisiin.

ADO-objektit tukevat lisäksi **Properties**-kokoelmaa, joka sisältää valmistajasta riippuvaa lisäinformaatiota objektista.

ADO Data -kontrolli on graafinen ActiveX-kontrolli, joka sisältää helppokäyttöisen käyttöliittymän, jonka avulla voit luoda tietokantasovelluksia mahdollisimman vähällä ohjelmoinnilla. ADO Data -kontrolli luo yhteydet datatietoisten kontrollien ja tietolähteiden välille. Datatietoiset kontrollit ovat ActiveX-käyttöliittymäkontrolleja, jotka voivat olla yhteydessä ADO Data -kontrolliin ja osaavat näyttää automaattisesti vastaanottamansa tiedot.

ADO Data -kontrolli ja joukko datatietoisia kontrolleja toimitetaan Visual C++:n mukana. Kuten kaikki muutkin ActiveX-kontrollit, ne täytyy lisätä projektiin ennen kuin voit käyttää niitä; tee lisääminen Components and Controls Gallery -toimintoa käyttäen. Kontrollin lisäämisen jälkeen voit käyttää dialogieditoria ActiveX-kontrollien sijoittamiseen aivan samoin kuin muidenkin kontrollien kohdalla. Kun kontrolli lisätään, luodaan projektiisi myös C++-luokat, jotka käärivät kontrollin paljastamat Automation-käyttöliittymät. Voit käyttää näitä luokkia kontrollin ominaisuuksien tutkimiseen ja asettamiseen, sekä kutsua niiden kautta kontrollin metodeja.

Laboratorio 7: Kyselyn tekeminen tietokannasta

Tässä laboratoriossa toteutat STUpload-sovelluksen **Query Database** -toiminnon. Tämän toiminnon avulla käyttäjä voi suorittaa ennakoimattoman kyselyn keskus-tietokantaan. Käyttäjä määrittelee osakkeen nimen, aloituspäivämäärän ja loppupäivämäärän **Query**-dialogissa ja saa vastauksena määrittelemänsä osakkeen hintakehityksen määräämältään aikaväliltä. Käyttäjä tutkii kyselyn tulosta **Results**-dialogissa, jossa on ADO tietolähteeseen yhdistetty DataGrid-kontrolli.

Query Database -toiminto on käytettävissä vain, jos STUpload-sovelluksessa on esillä osakkeen hintakehitys. **Query Database** -toimintoa käytetään lähinnä silloin, kun käyttäjä haluaa tarkistaa, että tiedot, joita hän aikoo tietokantaan tallentaa, eivät jo ole siellä. Tästä syystä **Query**-dialogissa tarjotaan oletuksena esillä olevaa osaketta ja nykyistä aikajaksoa.

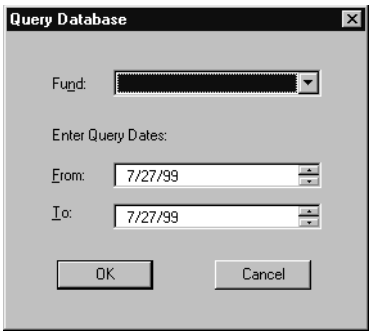
Tässä laboratoriossa oletetaan, että olet asentanut SQL Serverin ja Stocks-tietokannan Tästä kirjasta -luvun ohjeiden mukaan.

Query-dialogin tekeminen

Ensimmäisenä sinun täytyy luoda dialogimalli ja dialogiluokka **Query**-dialogia varten.

► **Query-dialogimallin luominen**

1. Tee **Query Database** -dialogimalli kuvan 7.17 mallin mukaisesti. Resurssin ID on **IDD_QUERYDIALOG**, ja siihen kuuluu yhdistelmäruutu ja kaksi **Date Time Picker** -kontrollia. (Käytä työkaluvihjettä etsiessäsi näitä kontrolleja **Controls**-työkaluriviltä.)



Kuva 7.17 Query Database -dialogi

2. Yhdistä yhdistelmäruutu tunnukseen **IDC_QUERY_FUND**. Varmista, että **Styles**-sivulla on valittuna **Sort**-valintaruutu. Valitse **Type**-ruudusta **Drop List**.
3. Napauta yhdistelmäruudun avausnuolta. Valinnan avulla voit määrittää, kuinka paljon luettelo laajenee. Raahaa yhdistelmäruudun luetteloa niin, että se laajenee juuri **OK** ja **Cancel** -painikkeiden yläpuolelle.
4. Anna ensimmäiselle **Date Time Picker** -kontrollille nimeksi **IDC_FROMDATE**. Aseta **Styles**-sivun **Format** asetuksen arvoksi **Short Date**. Valitse **Use Spin Control** -valintaruutu.
5. Tee samat toimenpiteet toiselle **Date Time Picker** -kontrollille ja anna sille nimeksi **IDC_TODATE**.

► **CQueryDialog dialogiluokan luominen**

1. Avaa ClassWizard painamalla CTRL+W. Luo **CQueryDialog**-dialogiluokka.
2. Lisää taulukossa 7.5 luetellut jäsenmuuttujat **Member Variables** -välilehdellä.

Taulukko 7.5 CQueryDialog-jäsenmuuttujat

Resource ID	Kategoria	Muuttujan tyyppi	Muuttujan nimi
IDC_FUND	Arvo	int	m_nFund
IDC_FUND	Kontrolli	CComboBox	m_dtFund
IDC_FROMDATE	Arvo	CTime	m_fromdate
IDC_FROMDATE	Kontrolli	CDateTimeCtrl	m_dtFrom
IDC_TODATE	Arvo	CTime	m_todate
IDC_TODATE	Kontrolli	CDateTimeCtrl	m_dtTo

Seuraavaksi toteutat **CQueryDialog::OnInitDialog()**-funktion, joka alustaa **Query Dialog** -dialogin. Tämä funktio hakee yhdistelmäruutuun tällä hetkellä

tiedostossa olevat osakkeet ja asettaa tämänhetkisen valinnan. Se alustaa myös **Date Time Picker** -kontrollit tiedoston alku- ja loppupäivämäärillä.

► **Query dialogin alustaminen**

1. Ensiksi **Select Fund** -dialogi täytyy saada **CQueryDialog::OnInitDialog()**-funktion käytettäväksi. Avaa **Mainfrm.h**-tiedosto. Lisää **CMainFrame**-luokan määrittelyn public-osaan seuraava inline-funktio:

```
const CFundDialog * GetFundDialog()
{return &m_wndFundDialog;}
```

2. Lisää seuraavat rivit QueryDialog.cpp-tiedoston alkuun:

```
#include “Mainfrm.h”
#include “STUploadDoc.h”
#include “FundDialog.h”
```

3. Luo **CQueryDialog::OnInitDialog()**-funktio ClassWizardilla (käsittele **WM_INITDIALOG**-sanoma **CQueryDialog**-luokassa). Korvaa generoidun funktion //TODO-kommentti seuraavalla koodilla:

   (Tämä listaus on oheisrompulta asennetussa CH7_01.cpp-tiedostossa.)

```
CMainFrame * pWnd =
dynamic_cast<CMainFrame *>
(AfxGetAPP()->m pMainWnd();

ASSERT_VALID(pWnd);

CSTUploadDoc * pDoc =
      dynamic_cast<CSTUploadDoc *>(pWnd->GetActiveDocument());

ASSERT_VALID(pDoc);

const CFundDialog * pFD = pWnd->GetFundDialog();

ASSERT_VALID(pFD);

// Fill combo box with current fund names
for(int n = 0; n < pFD->m_listBox.GetCount(); n++)
{
      CString strBuf;
      pFD->m_listBox.GetText(n, strBuf);
      m_cbFund.AddString(strBuf);
}

// Set listbox selection to strCurrentFund parameter
int iPos =
```

```
        m_cbFund.FindStringExact(-1, pDoc->GetCurrentFund());
        m_cbFund.SetCurSel(iPos);

    // Setup Date Time Pickers
    m_dtFrom.SetFormat("d MMM yyy");
    m_dtTo.SetFormat("d MMM yyy");
```

Sinun täytyy ylikuormittaa **CQueryDialog**-luokan **OnOK()**-funktio niin, että se noutaa käyttäjän valitseman osakkeen:

► **CQueryDialog::OnOK()-funktion toteutus**

1. Avaa QueryDialog.h-tiedosto, ja lisää seuraava muuttuja **CQueryDialog**-luokan määrittelyn **public**-osaan:

```
CString m_strFund;
```

2. Luo ClassWizardilla **OnOK()**-funktio käsittelemään **IDOK**-objektin **BN_CLICKED**-sanoma. Korvaa // TODO-kommentti seuraavalla koodilla:

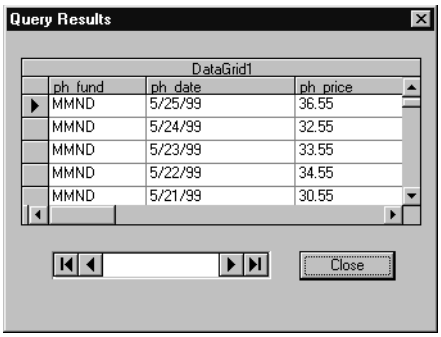
```
int nChoice = m_cbFund.GetCurSel();

if(nChoice >= 0)
    m_cbFund.GetLBText(nChoice, m_strFund);
```

Query Results -dialogin tekeminen

Seuraavaksi luot dialogimallin ja dialogiluokan **Query Results** -dialogille.

► **Query Results -dialogimallin luominen**

1. Lisää **Microsoft ADO Data Control, version 6.0 (OLEDB)** ja **Microsoft DataGrid Control**, **Version 6.0 (OLEDB)** projektiin käyttämällä Components and Controls Gallerya. Huolehdi siitä, että luot kaikki näihin kontrolleihin liittyvät luokat.
2. Tee **Results**-dialogi kuvan 7.18 mallin mukaan. Resurssitunniste on **IDD_RESULTSDIALOG**, ja siinä on ADO Data -kontrolli, jonka ID on oletuksena oleva **IDC_ADODC1**, ja DataGrid-kontrolli, jonka tunnus on myös oletusarvon mukainen **IDC_ADODC1**. **OK**-painikkeen otsikoksi (Caption) on muutettu teksti **Close**. **Cancel**-painike on poistettu.



Kuva 7.18 **Query Results** -dialogi

3. Tee **Microsoft OLE DB Provider for SQL Server** -yhteyden luomiseen tarvittava yhteysmerkkijono **ADO Data Control Properties** -dialogin **Control**-sivulla. Aseta **Connection**-sivulla määritys **(local)** SQL Server ja valitse tietokannaksi **Stocks**.
4. Avaa uudelleen **ADO Data Control Properties** -dialogi. Valitse **RecordSource**-sivulta **Command Type** -luetteloruudusta **1 – adCmdText**. Kirjoita **Command Text (SQL)** ruutuun, seuraava komento:

   ```
   SELECT * FROM pricehistory
   ```

5. Muokkaa DataGrid-kontrollin ominaisuuksia. Poista valinta **AllowUpdate** ruudusta **Control**-sivulla. Muuta **All**-sivulla **DataSource** ominaisuudeksi ADO Data -kontrollin tunniste **IDC_ADODC**.

   Testaa **Query Results** -dialogi painamalla CTRL+T. Tarkista, että tiedot Stocks-tietokannan hintaseuranta-taulusta tulevat näkyviin.

► CResultsDialog-dialogiluokan luominen

1. Avaa ClassWizard painamalla CTRL+W. Luo **CResultsDialog**-dialogiluokka.
2. Lisää taulukossa 7.5 luetellut jäsenmuuttujat **Member Variables** -välilehdellä.

Taulukko 7.6 **CResultsDialog jäsenmuuttujat**

Resource ID	Kategoria	Muuttujan tyyppi	Muuttujan nimi
IDC_ADODC1	Kontrolli	CAdodc	m_adodc
IDC_DATAGRID1	Kontrolli	CDataGrid	m_datagrid

3. Sulje ClassWizard. Avaa ResultsDialog.h-tiedosto, ja lisää seuraavat muuttujat **CResultsDialog**-luokan määrittelyn **public**-osaan:

```
CString m_strQuery;
CString m_strCaption;
```

Seuraavaksi toteutetaan **CResultsDialog::OnInitDialog()**-funktio, joka alustaa **Query Results** -dialogin. Tämä funktio täyttää yhdistelmäruudun tällä hetkellä tiedostossa olevilla osakkeilla ja asettaa oletusvalinnan. Se alustaa myös **Date Time Picker** -kontrollit alku- ja päättymispäivämäärillä.

► **Query Results -dialogin alustaminen**

1. Lisää seuraava koodirivi ResultsDialog.cpp-tiedoston alkuun saadaksesi **C_Recordset**-luokan käyttöön. Luokka luotiin, kun lisäsit ADO Data -kontrollin projektiin:

```
#include "_recordset.h"
```

2. Luo **CResultsDialog::OnInitDialog()**-funktio ClassWizardilla. Korvaa luodun funktion // TODO-kommentti seuraavalla koodilla:

   (Tämä koodi on asennettu oheisrompulta CH7_02.cpp-tiedostoon.)

```
m_adodc.SetRecordSource(m_strQuery);
m_adodc.Refresh();

C_Recordset cRS = m_adodc.GetRecordset();

long lRecs = cRS.GetRecordCount();

if(lRecs < 1)
{
    AfxMessageBox("No records match this query");
    EndDialog(IDCANCEL);
}

m_datagrid.SetCaption(m_strCaption);
```

Query Database -valikkokomennon käsitteleminen

Seuraavaksi lisätään tapahtumakäsittelijä ja funktio, joka huolehtii käyttöliittymän päivittämisestä **Data**-valikon **Query Database** -toimintoa varten.

► **OnDataQueryDatabase()-komennon käsittelijän lisääminen**

1. Lisää seuraavat rivit STUploadDoc.cpp-tiedoston alkuun:

```
#include "QueryDialog.h"
#include "StockDataList.h"
#include "ResultsDialog.h"
```

2. Lisää komennonkäsittelijä **CSTUploadDoc**-luokan **ID_QUERY_DATABASE** -tunnisteelle ClassWizardia käyttämällä. Funktion nimeksi tulisi antaa **OnDataQuerydatabase()**.

3. Korvaa // TODO-kommentti seuraavalla koodilla:

   (Tämä koodi on asennettu oheisrompulta tiedostoon CH7_03.cpp.)

```
CQueryDialog aQDlg;

// Set the default values for the Date Time Picker controls
// with first and last date on file (all funds)
CStockData sdFirst = m DocList.GetHead();
CStockData sdLast = m DocList.GetTail();

aQDlg.m fromdate = sdFirst.GetDate();
aQDlg.m todate = sdLast.GetDate();

if(aQDlg.DoModal() == IDOK)
{
      // Construct query
      CString strQuery =
            "select * from PriceHistory where ph_fund = '";
      strQuery += aQDlg.m strFund;
      strQuery += "' and ph date between '";
      strQuery += aQDlg.m fromdate.Format("%Y/%m/%d");
      strQuery += "' and '";
      strQuery += aQDlg.m todate.Format("%Y/%m/%d");
      strQuery += "'";

      // Construct caption string
      CString strCaption = aQDlg.m strFund;
      strCaption += " Prices ";
      strCaption += aQDlg.m fromdate.Format("%d/%m/%Y");
      strCaption += " - ";
      strCaption += aQDlg.m todate.Format("%d/%m/%Y");

      CResultsDialog rd;
      rd.m strQuery = strQuery;
      rd.m strCaption = strCaption;

      rd.DoModal();
}
```

Käyttöliittymän päivittämisestä huolehtiva tapahtumakäsittelijä huolehtii siitä, että **Query Database** -toiminto on käytettävissä vain, jos sovelluksessa on parhaillaan esillä osakkeen hintaseuranta.

► Käyttöliittymän päivittämisestä huolehtivan tapahtumakäsittelijän lisääminen

1. Lisää **CSTUploadDoc**-luokkaan **ID_QUERY_DATABASE**-tunnisteelle **UPDATE_COMMAND_UI**-käsittelijä ClassWizardia käyttämällä. Funktiolle tulee antaa nimi **OnUpdateDataQuerydatabase()**.
2. Korvaa // TODO -kommentti seuraavalla koodilla:

   (Tämä koodi on asennettu oheisrompulta tiedostoon CH7_04.cpp.)

```
// Enable the Query Database command only if there is
// data on file and a fund currently selected for viewing

BOOL bEnable = FALSE;
bEnable = m strCurrentFund.IsEmpty() ? FALSE : TRUE;

pCmdUI->Enable(bEnable);
```

Voit nyt kääntää ja ajaa STUpload-sovelluksen. Nouda Ch7Test.dat-tiedosto \Chapter 7\Data-kansiosta. Poimi yksi osake tutkittavaksi ja valitse **Query Database** -komento, joka on nyt valittavissa. Arvot Ch7Test.dat-tiedossa vastaavat tietoja, jotka on jo tallennettu Stocks-tietokantaan. Hyväksy kontrolleiden oletusarvot ja siirry selaamaan tietokannan tietueita napauttamalla **OK**-painiketta. Voit kokeilla toimintoa lähettämällä kyselyitä, jotka poimivat osan tietueista tai eivät poimi tietueita lainkaan.

Kertaus

1. Kuinka voit yhdistää ADO-sovelluksen ODBC-tietolähteeseen?

2. Mitä vikaa on seuraavassa SQL-lauseessa?

```
SELECT * FROM authors WHERE au_lname LIKE M%
```

3. Sinun täytyy avata **CRecordset**-objekti niin, että voit tulostaa kirjoittimella kaikki tietokannan taulussa olevat tietueet. Minkä arvon asetat **Open()**-funktion ensimmäiseksi parametriksi?

4. Mitä **CRecordset::GetDefaultConnect()**-funktio määrittelee?

5. Mihin ADO-objektiin **Errors**-kokoelma on yhdistetty? Mitä se pitää sisällään?

6. Mitä ADO Data -kontrollin ominaisuutta käytetään haettavat tietueet määräävän SQL-komennon määrittelemiseen?