Dr. Hajnal Éva: Haladó programozás

HALADÓ PROGRAMOZÁS

Adatbázis-elérési módszerek összehasonlítása
Adatbázisok elérése DbConnection/DbReader módszerrel
SQL server elérése DataSet módszerrel
SQL server elérése Entity Framework módszerrel

Adatbázis-kezelés módszerek szempontjai

Kódfejlesztés átláthatósága	• Debug lehetőségek • Kódszinezés
Hatékonyság	• Idő és tárbonyolultság • Kód felhasználhatósága (SQL, NOSQL adatbázisok)
Migrálhatóság	• Adatbázismotor cseréje
Biztonsági kérdések	• Tranzakció kezelés (ACID elvek) • Adatbázis biztonság (pl. SQL injection elkerülése)

Haladó Programozás

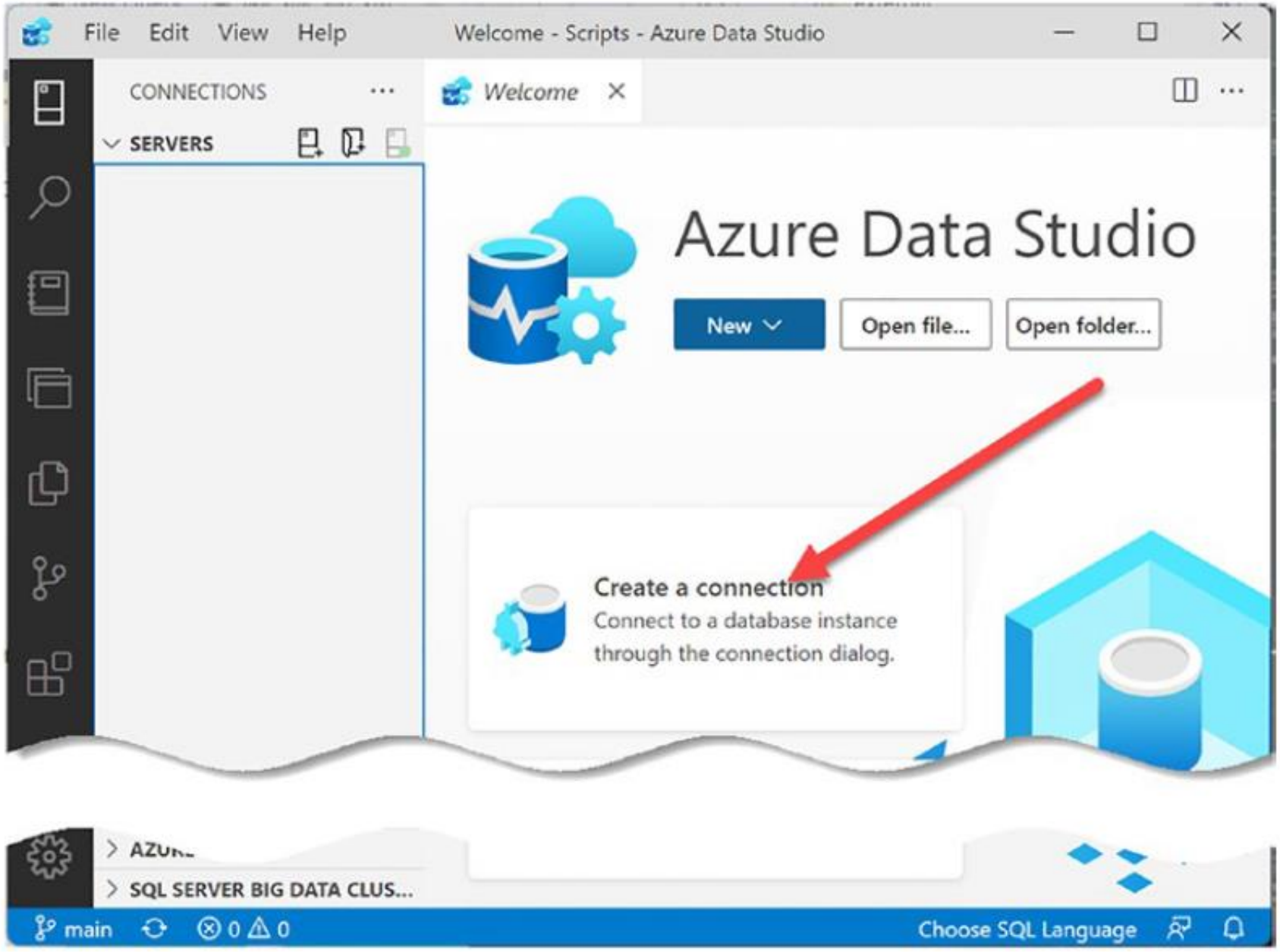
- Adatbázis-elérési módszerek összehasonlítása
- Adatbázisok elérése DbConnection/DbReader módszerrel
- SQL server elérése DataSet módszerrel
- SQL server elérése Entity Framework módszerrel

SQL Server / SQL Express / LocalDB

- MSSQL: tipikusan kis- és középvállalatok által használt adatbázis-kezelő szerver
  - SQL Express: kisebb változat: max 10GB/database, max 1 CPU, max 1GB RAM
- Korábbi VS verziók (x<2012): SQL Express integrálva volt a fejlesztőeszköz telepítőjébe

  (Data Source=.\SQLEXPRESS)
- VS 2010 óta elérhető, VS 2012-től default: LocalDB

  (Data Source=(localdb)\v11.0)
  - Szerver-szolgáltatás helyett igény szerint induló library, ami egy adatbázis-file-t használ



Letölthető eszközök

- Azure Data Studio
- Server 20xx Developer, Express Studio

DbConnection vs DataSet vs Entity Framework

- Különböző absztrakciós szintek
- DbConnection
  - Alap SQL-szintű elérés, string SQL utasításokkal, object tömb eredményekkel
- DataSet
  - Az SQL réteg fölé egy erősen típusos, GUI központú réteg kerül, a műveleteket típusos metódusok végzik
- Entity Framework
  - Az SQL réteg fölé egy MVC elvű ORM (Object Relational Mapping) réteget helyezünk: a táblákat mint objektumok általános gyűjteményét kezeljük

- Alapvető műveletek: Kapcsolódás/inicializáció; beszúrás; módosítás; törlés; adatlekérés; GUI-hoz kapcsolás

Haladó Programozás

Adatbázis-elérési módszerek összehasonlítása

Adatbázisok elérése DbConnection/DbReader módszerrel

SQL server elérése DataSet módszerrel

SQL server elérése Entity Framework módszerrel

Data providers: ADO.NET vs ODBC, Microsoft SQL Server

- ODBC a régebbi technológia
  - SQL lekérdezések támogatása
  - Szükséges hozzá egy odbc driver
  - Lekérdezések futtatása kevésbé hatékony

- ADO.NET újabb technológia
  - SQL és NOSQL lekérdezések támogatása
  - Ado provider szükséges hozzá

ADO alaposztályok

Base Class	Relevant Interfaces	Meaning in Life
`DbConnection`	`IDbConnection`	Provides the ability to connect to and disconnect from the data store. Connection objects also provide access to a related transaction object.
`DbCommand`	`IDbCommand`	Represents a SQL query or a stored procedure. Command objects also provide access to the provider's data reader object.
`DbDataReader`	`IDataReader`, `IDataRecord`	Provides forward-only, read-only access to data using a server-side cursor.
`DbDataAdapter`	`IDataAdapter`, `IDbDataAdapter`	Transfers `DataSet`s between the caller and the data store. Data adapters contain a connection and a set of four internal command objects used to select, insert, update, and delete information from the data store.
`DbParameter`	`IDataParameter`, `IDbDataParameter`	Represents a named parameter within a parameterized query.
`DbTransaction`	`IDbTransaction`	Encapsulates a database transaction.

ADO.NET: DbConnection/DbReader

- „Kapcsolt" adatbázis-elérés (Connected Data-Access Architecture)
- Előny: gyors, egyszerű
- Hátrány: nehéz módosítani és technológiát/tárolási módszert váltani; kapcsolat elveszését kezelni kell
- A különböző adatbázis-szerverekhez különböző implementációk
- Közös ősosztályok a különféle feladatokhoz
  - Adatbázis-kapcsolat: DbConnection
  - SQL/RPC utasítás végrehajtása: DbCommand
  - Utasítás eredményének beolvasása: DbDataReader
- Specifikus utódosztályok a különféle adatbázis-szerverekhez
  - SqlConnection (MSSQL System.Data.SqlClient)
  - MySqlConnection (MySQL MySql.Data.MySqlClient)
  - NpgsqlConnection (PostgreSQL - Npgsql)
  - OracleConnection (Oracle System.Data.OracleClient)

1. Inicializálás

```
string connStr = @"Data
 Source=(LocalDB)\v11.0;AttachDbFilename=path\to\empdept.mdf;Integrated
 Security=True;";
SqlConnection conn;
private void button15_Click(object sender, EventArgs e)
{
    conn = new SqlConnection(connStr);
    conn.Open();
    MessageBox.Show("CONNECTED");
}
```

2. INSERT

```
private void button18_Click(object sender, EventArgs e)
{
    SqlCommand comm = new SqlCommand("insert into EMP (ENAME, MGR,
 DEPTNO, EMPNO) values ('BELA', NULL, 20, 1000)", conn);
    SqlDataReader reader=comm.ExecuteReader();
    MessageBox.Show(reader.RecordsAffected.ToString());
    reader.Close();
}
```

3. UPDATE

```
private void button18_Click(object sender, EventArgs e)
{
    SqlCommand comm = new SqlCommand("update EMP set ENAME='JOZSI'
 where EMPNO=1000", conn);
    SqlDataReader reader=comm.ExecuteReader();
    MessageBox.Show(reader.RecordsAffected.ToString());
    reader.Close();
}
```

4. DELETE

```
private void button18_Click(object sender, EventArgs e)
{
    SqlCommand comm = new SqlCommand("delete from EMP where
 empno=1000", conn);
    SqlDataReader reader=comm.ExecuteReader();
    MessageBox.Show(reader.RecordsAffected.ToString());
    reader.Close();
}
```

5. SELECT

```
private void button14_Click(object sender, EventArgs e)
{
    listBox1.Items.Clear();
    SqlCommand comm = new SqlCommand("select * from EMP where sal>=3000 order by ename",
  conn);
    SqlDataReader reader = comm.ExecuteReader();
    while (reader.Read())
    {
        listBox1.Items.Add(reader["ENAME"].ToString());
    }
    reader.Close();
}
```

6. Megjelenítés GUI-n

```
private void button13_Click(object sender, EventArgs e)
{
    dataGridView1.DataSource = null;
    dataGridView1.Rows.Clear();
    dataGridView1.Columns.Clear();
    dataGridView1.AllowUserToAddRows = false;
    SqlCommand comm = new SqlCommand("select * from EMP order by ename", conn);
    SqlDataReader reader = comm.ExecuteReader();
    while (reader.Read())
    {
        [GRIDVIEW FELTÖLTÉSE]
    }
    reader.Close();
}
```

6. Megjelenítés GUI-n - GRIDVIEW FELTÖLTÉSE

```
if (dataGridView1.Columns.Count == 0)
{
    for (int i = 0; i < reader.FieldCount; i++) {
        string coltext = reader.GetName(i).ToLower();
        dataGridView1.Columns.Add(coltext, coltext);
    }
}
dataGridView1.Rows.Add();
int rowid = dataGridView1.Rows.Count - 1;
for (int i = 0; i < reader.FieldCount; i++)
{
  dataGridView1.Rows[rowid].Cells[i].Value =       reader[i].ToString();
}
```

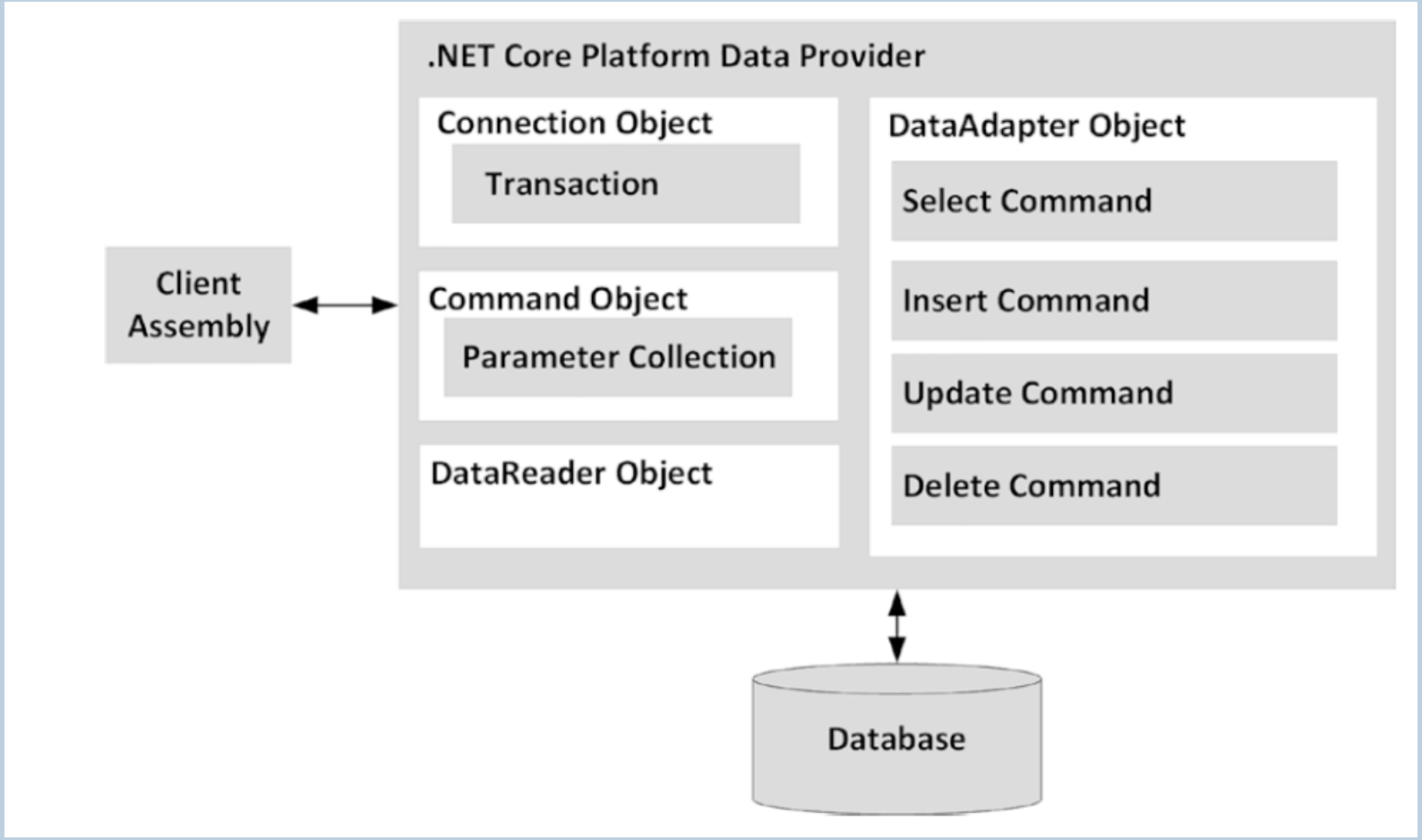
Haladó Programozás

Adatbázis-elérési módszerek összehasonlítása
Adatbázisok elérése DbConnection/DbReader módszerrel
ADO.NET-SQL server elérése DataSet módszerrel
SQL server elérése Entity Framework módszerrel

ADO.NET:
DataAdapter/DataSet/DataTable

- „Kapcsolat nélküli" adatbázis-elérés (Disconnected D-A. A.)
- Előny: nem kell konkrét SQL utasításokat írni, könnyebb adatkapcsolódás a GUI elemeihez
- Hátrány: nehéz módosítani és technológiát/tárolási módszert váltani; nagy memóriaigény
- Automatikusan generált, adatfüggő típusozott osztályok keletkeznek (~200KB teljesen normális méret egy két táblás adatbázishoz)
- Az osztályok legenerálása után egyedi osztályok vannak a táblákhoz/adatmezőkhöz:
  - TableAdapter: az adatbázishoz való kapcsolatot építi fel, eredményeket olvas
  - DataSet: az adatbázis reprezentációja
  - BindingSource: adatforrás-kapcsolat az ablak felé

Data Adapter



DataSet létrehozása

- Server explorer (Express Edition-ben: Database Explorer) és Data Sources (Shift+Alt+D) fül kell
- Server explorer: Adatbázis/connection létrehozása
  - Későbbi dián részletezzük
- Data sources: Add new data source
  - Database / DataSet
  - Connection kiválasztása, „Include sensitive data”
  - „Save connection string to the application file”, Táblák kiválasztása
- Ezután a DataSources-ból az ablakra egyszerűen ráhúzzuk a táblát, máris kész az alkalmazás

1. Inicializálás

```
private void button5_Click(object sender, EventArgs e)
{
    eMPTableAdapter.Fill(nikdbDataSetVariable.EMP);
    MessageBox.Show("CONNECTED");
}
```

2. INSERT

```
private void button8_Click(object sender, EventArgs e)
{
    DataRow ujsor = nikdbDataSetVariable.EMP.NewRow();
   //NewEmpRow() is használható lenne
    ujsor["ENAME"] = "BELA";
    ujsor["MGR"] = DBNull.Value;
    ujsor["DEPTNO"] = 20;
    ujsor["EMPNO"] = 1000;
    nikdbDataSetVariable.EMP.Rows.Add(ujsor);
    eMPTableAdapter.Update(nikdbDataSetVariable);
    MessageBox.Show("DONE");
}
```

3. UPDATE

```
private void button7_Click(object sender, EventArgs e)
{
    DataRow dr = nikdbDataSetVariable.EMP.Rows[0];
    dr.BeginEdit();
    dr["ENAME"] = "JOZSI";
    dr.EndEdit();
    nikdbDataSet valtozas=(nikdbDataSet)nikdbDataSetVariable.
              GetChanges(DataRowState.Modified);
    if (valtozas.HasErrors) {
        nikdbDataSetVariable.RejectChanges();
    } else {
        nikdbDataSetVariable.AcceptChanges();
        eMPTableAdapter.Update(valtozas);
```

4. DELETE

```
private void button6_Click(object sender, EventArgs e)
{
    DataRow dr = nikdbDataSetVariable.EMP.Rows[0];
    dr.Delete();
    nikdbDataSetVariable.AcceptChanges();
    eMPTableAdapter.Update(nikdbDataSetVariable);
}
DataRow dr = nikdbDataSetVariable.EMP.Select("empno=1000")[0];
  // EZ NEM A LINQ EXTENSION METHOD, CSAK UGYANAZ A NEVE!
```

5. SELECT

```
private void button12_Click(object sender, EventArgs e)
{
    listBox1.Items.Clear();
    foreach (DataRow dr in
                nikdbDataSetVariable.EMP.Select("sal>=3000"))
    {
        listBox1.Items.Add(dr["ENAME"].ToString());
    }
}
```

6. Megjelenítés GUI-n

```
private void button11_Click(object sender, EventArgs e)
{
    dataGridView1.Columns.Clear();
    dataGridView1.Rows.Clear();
    dataGridView1.DataSource = null;
    //dataGridView1.DataSource = nikdbDataSetVariable.EMP;
    dataGridView1.DataSource = eMPBindingSource;
}
```

Köszönöm a figyelmet!