Коди для роботи з базами даних на платформі .NET Framework.

Підключення бази даних

```
using System;
using System.Data.SqlClient;

namespace CBS.ADO_NET.ConnectionStrings
{
    class Program
    {
        static void Main(string[] args)
        {
           string conStr = @"Data Source=localhost;
                    Initial Catalog=comp_firm;
                    Integrated Security=True";
           SqlConnection connection = new SqlConnection(conStr);

            try
            {
                connection.Open(); // відкриття фізичного підключення до джерел даних
                Console.WriteLine(connection.State);
            }
            catch (Exception ex)
            {
                Console.WriteLine(ex.Message);
            }
            finally
            {
                connection.Close(); //закриття фізичного з'єднання з джерелом даних
                Console.WriteLine(connection.State);
            }
        }
    }
}
```

Пул з'єднань.

```
using System;
using System.Data.SqlClient;

namespace CBS.ADO_NET.Pooling
{
    class Program
    {
        static void Main(string[] args)
        {
            string conStr = @"Data Source=localhost;
                              Initial Catalog=comp_firm;
                              Integrated Security=true;
                              Pooling = true/false"; // включення/вимкнення пула для цього підключення

            DateTime start = DateTime.Now;

            for (int i = 0; i < 1000; i++)
            {
                SqlConnection connection = new SqlConnection(conStr);
                connection.Open();
                connection.Close();
            }

            TimeSpan stop = DateTime.Now - start;

            Console.WriteLine(stop.TotalSeconds);
        }
    }
}
```

Отримання скалярних значень ExecuteScalar()

```
using System;
using System.Data.SqlClient;

// Виконання команд, що повертають скалярні значення

namespace CBS.ADO_NET.ExecuteCommand
{
    class Program
    {
        static void Main(string[] args)
        {
            string conStr = @"Data Source=localhost; Initial Catalog=comp_firm; Integrated
Security=True;";
            SqlConnection connection = new SqlConnection(conStr);
            connection.Open();

            SqlCommand cmd = new SqlCommand("SELECT COUNT(model) FROM PC", connection);

            object result = cmd.ExecuteScalar();  // Отримуємо об'єкт
            string stringResult = result.ToString();  // Перетворимо на рядок


            Console.WriteLine(stringResult);
        }
    }
       }
```

Зміна записів команда ExecuteNonQuery()

```
using System;
using System.Data.SqlClient;

// Виконання команд вставки та видалення ExecuteNonQuery()

namespace CBS.ADO_NET.ExecuteCommand
{
    class Program
    {
        static void Main(string[] args)
        {
            string conStr = @"Data Source=localhost; Initial Catalog=comp_firm; Integrated
Security=True;";

            SqlConnection connection = new SqlConnection(conStr);
            connection.Open();

            // INSERT command

            SqlCommand insertCommand = connection.CreateCommand(); // створення команди на вставку
даних
            insertCommand.CommandText = "INSERT Product VALUES ('HNEU', 'MSIT', '186')";

            int rowAffected = insertCommand.ExecuteNonQuery();
            Console.WriteLine("INSERT command rows affected: "+rowAffected);


            // DELETE command
            /*
            SqlCommand deleteCommand = connection.CreateCommand(); // створення команди на видалення
даних
            deleteCommand.CommandText = "DELETE Product WHERE maker = 'HNEU'";

             int rowAffected = deleteCommand.ExecuteNonQuery();
            Console.WriteLine("DELETE command rows affected: " + rowAffected);
            */
            connection.Close();
        }
    }
}
```

Зчитування даних, що зберігаються в таблиці: метод ExecuteReader().

```
using System;
using System.Data.SqlClient;

// виконання команди ExecuteReader(), що повертають дані у табличному поданні

namespace CBS.ADO_NET.ExecuteTableCommands
{
    class Program
    {
        static void Main(string[] args)
        {
            string conStr = @"Data Source=localhost; Initial Catalog=comp_firm; Integrated
Security=True;";

            SqlConnection connection = new SqlConnection(conStr);
            connection.Open();

            SqlCommand cmd = new SqlCommand("SELECT * FROM Product", connection); // побудова
команди, що повертає дані в табличному поданні

            SqlDataReader reader = cmd.ExecuteReader(); //перевіряємо наявність запитаних даних

            while (reader.Read())
            {
                for (int i = 0; i < reader.FieldCount; i++)
                {
                    Console.WriteLine(reader.GetName(i) + ": " + reader[i]);
                }
                Console.WriteLine(new string('_', 20));
            }
            reader.Close();
            connection.Close();
        }
    }
}
```

ReaderWork

Для отримання результатів SqlDataReader використовувався метод GetValue, який повертав значення певного стовпця у поточному осередку у вигляді об'єкта типу object. Однак у ряді випадків такий спосіб не є оптимальним. Наприклад, ми знаємо, що в третьому стовпці зберігається вік користувача, який представляє ціле число, і в програмі ми хотіли б використовувати його як ціле число. Так як GetValue повертає об'єкт типу об'єкта, то, щоб його використовувати, наприклад, як число, нам треба його привести до типу int. Однак ми можемо вибрати інший шлях – використовувати типізовані методи.

```
using System;
using System.Data.SqlClient;

// отримання даних об'єкта DataReader за допомогою типізованих засобів доступу

namespace CBS.ADO_NET.ExecuteTableCommands
{
    class Program
    {
        static void Main(string[] args)
        {
            string conStr = @"Data Source=localhost; Initial Catalog=comp_firm; Integrated
Security=True";

            SqlConnection connection = new SqlConnection(conStr);

            SqlCommand cmd = new SqlCommand("SELECT * FROM Warehouse", connection); // побудова
команди, що повертає дані в табличному поданні

            connection.Open();
            SqlDataReader reader = cmd.ExecuteReader();

            while (reader.Read())
            {
                Console.WriteLine(reader.GetString(3));                    // виведення на екран
"Тип продукту"
                Console.WriteLine(reader.GetInt32(4));                     // виведення на екран "ID
клієнта" використовуючи метод GetInt
                //Console.WriteLine(reader.GetFieldValue<DateTime>(5));    // виведення на екран
поля "SalesData клієнта"
                Console.WriteLine("{0:D}",reader.GetDateTime(5));          // виведення на екран
відформатованого поля "SalesData клієнта"


                Console.WriteLine(new string('_',20));
            }
            reader.Close();
            connection.Close();
        }
    }
       }
```

Виконання пакетних запитів

При створенні SqlCommand властивості CommandText можна присвоїти відразу кілька операторів SQL. У разі використання об'єкта DataReader для такої команди після перегляду даних першого набору рядків потрібно перейти до наступного набору за допомогою методу NextResult() об'єкта DataReader.

```
using System;
using System.Data.Common;
using System.Data.SqlClient;

// виконання пакета операторів SQL за допомогою об'єкта SqlCommand

namespace CBS.ADO_NET.PackageCommands
{
    class Program
    {

        public static void WriteReaderData(DbDataReader reader)
        {
            while (reader.Read())                       // виведення даних, що повертаються запитом
            {
                for (int i = 0; i < reader.FieldCount; i++)
                    Console.WriteLine(reader.GetName(i)+": "+reader[i]);
                    Console.WriteLine(new string('_', 20));
            }
        }

        static void Main(string[] args)
        {
            string conStr = @"Data Source=localhost; Initial Catalog=comp_firm; Integrated
Security=True";

            SqlConnection connection = new SqlConnection(conStr);
            connection.Open();

            // створення пакету запитів
            SqlCommand cmd = new SqlCommand("SELECT * FROM Warehouse WHERE id = 2; SELECT * FROM PC
WHERE model = '1233'", connection);
            SqlDataReader reader = cmd.ExecuteReader();

            Console.WriteLine("натисніть будь-яку клавішу, щоб переглянути дані з таблиці
Warehouse");
            Console.ReadKey();

            WriteReaderData(reader);  // виведення на екран даних

            Console.WriteLine("press any key to see data from PC");
            Console.ReadKey();

            reader.NextResult();      // перехід до наступного запиту

            WriteReaderData(reader);

            connection.Close();
            reader.Close();
            Console.ReadKey();
        }
    }
      }
```

Створення параметрізованих запитів.

```
using System;
using System.Data.SqlClient;

// створення параметризованих запитів за допомогою конкатенації рядків

namespace CBS.ADO_NET.ParametrizedCommands
{
    class Program
    {
        static void Main(string[] args)
        {
            string conStr = @"Data Source=localhost; Initial Catalog=comp_firm; Integrated
Security=True";

            Console.WriteLine("Введіть ID виробника");

            var customerNo = Console.ReadLine();

            // для створення параматизованого запиту використовується метод string.Format
            string commandStr = string.Format("SELECT * FROM Warehouse WHERE id = {0};",
customerNo);

            using (SqlConnection connection = new SqlConnection(conStr)) // створення підключення
            {
                connection.Open();

                SqlCommand cmd = new SqlCommand(commandStr, connection);

                using (SqlDataReader reader = cmd.ExecuteReader()) // виконання запиту та читання
результатів
                {
                    while (reader.Read())
                    {
                        for (int i = 0; i < reader.FieldCount; i++)
                            Console.WriteLine("{0}: {1}", reader.GetName(i), reader[i]);
                        Console.WriteLine(new string('-', 20));
                    }
                }
            }
        }
    }
}
```

Процедури що зберігаються

```
using System;
using System.Data.SqlClient;

// Виклик збереженої процедури за допомогою команди EXECUTE T-SQL

namespace CBS.ADO_NET.ParametrizedCommands
{
    class Program
    {
        static void Main(string[] args)
        {
            /* код збереженої процедури GetLaptopsFromWarehouse:
               SELECT* FROM Warehouse WHERE type = 'Laptop';
            */

            string conStr = @"Data Source=localhost; Initial Catalog=comp_firm; Integrated
Security=True";
            SqlConnection connection = new SqlConnection(conStr);

            SqlCommand cmd = new SqlCommand("EXECUTE GetLaptopsFromWarehouse", connection);  //
створення команди, що виконує збережену процедуру selectEmp

            connection.Open();

            SqlDataReader reader = cmd.ExecuteReader();

            while (reader.Read())
            {
                for (int i = 0; i < reader.FieldCount; i++)
                    Console.WriteLine("{0}: {1}", reader.GetName(i), reader[i]);

                Console.WriteLine();
            }

            connection.Close();
        }
    }
}
```

Виконання процедури що приймає параметри

```
using System;
using System.Data.SqlClient;

// виконання збереженої процедури, що приймає параметри

namespace CBS.ADO_NET.ParametrizedCommands
{
    class Program
    {
        static void Main(string[] args)
        {
            /* код збереженої процедури selectEmp:
            CREATE PROCEDURE GetItemsByType+
                    @ItemType NVARCHAR(50)
            AS
            BEGIN
                    SET NOCOUNT ON;
                    SELECT * FROM Warehouse WHERE type = @ItemType;
            END;
            */

            string conStr = @"Data Source=localhost; Initial Catalog=comp_firm; Integrated
Security=True";
            SqlConnection connection = new SqlConnection(conStr);

            Console.WriteLine("Введите тип продукта: PC / Laptop / Printer");
            string ItemType = Console.ReadLine(); // отримання даних від користувача

            // створення команди, що викликає процедуру, що зберігається
            SqlCommand cmd = new SqlCommand("GetItemsByType", connection) { CommandType =
System.Data.CommandType.StoredProcedure };


            cmd.Parameters.AddWithValue("@ItemType", ItemType); // додавання одного параметра

            connection.Open();

            SqlDataReader reader = cmd.ExecuteReader();

            while (reader.Read())
            {
                for (int i = 0; i < reader.FieldCount; i++)
                    Console.WriteLine("{0}: {1}", reader.GetName(i), reader[i]);

                Console.WriteLine();
            }

            connection.Close();
        }
    }
       }
```

Створення екземплярів DataColumn та додавання їх до колекції стовпців об'єкту DataTable

```
using System;
using System.Data;

// створення екземплярів DataColumn та додавання їх до колекції стовпців об'єкту DataTable

namespace DataSetBasics
{
    class Program
    {
        static void Main(string[] args)
        {
            DataTable table = new DataTable("MyFirstTable");

            DataColumn firstColumn = new DataColumn("First Column", typeof(int));
            DataColumn secondColumn = new DataColumn("Second column", typeof(string));

            DataColumnCollection columnCollection = table.Columns;
            columnCollection.AddRange(new DataColumn[]{firstColumn, secondColumn});

            foreach (DataColumn column in table.Columns)
                Console.WriteLine("{0}: {1};", column.ColumnName, column.DataType);
        }
    }
       }
```

Створення рядків та додавання їх до об'єкта DataTable

```
using System;
using System.Data;

// створення рядка для таблиці та додавання до колекції рядків об'єкта DataTable

namespace BasicsDataRow
{
    class Program
    {
        static void Main(string[] args)
        {
            DataTable table = new DataTable();

            table.Columns.Add(new DataColumn("Column1", typeof(int)));
            table.Columns.Add(new DataColumn("Column2"));

            DataRow newRow = table.NewRow();

            newRow["Column1"] = 1; // індексатор об'єкта DataRow в якості рядкового індексу приймає
ім'я поля в рядку до якого потрібно звернутися
            //newRow[0] = 1;      // альтернативний запис того ж рядка

            newRow["Column2"] = "One"; // індексатор об'єкта DataRow в якості цілочисленного індексу
приймає індекс поля в рядку до якого потрібно звернутися
            //newRow[1] = "One";


            Console.WriteLine("table.Rows.Count: " + table.Rows.Count); // виведеться 0

            table.Rows.Add(newRow); // рядок додається до таблиці

            Console.WriteLine("table.Rows.Count: " + table.Rows.Count); // виведеться 1

            Console.WriteLine();

            foreach (DataRow row in table.Rows)
            {
                foreach (DataColumn column in table.Columns)
                    Console.WriteLine("{0}: {1}", column.ColumnName, row[column]);
            }
        }
    }
}
```

Створення таблиці має таку ж структуру, як і таблиця у джерелі бази даних.

```
using System;
using System.Data;
using System.Data.SqlClient;

// створення методів, що дозволяють створювати нову таблицю на основі даних, що надаються об'єктом
DataReader

namespace TableWork
{
    class Program
    {

        // Цей метод створює новий об'єкт DataTable зі схемою, як і у SqlDataReader.
        private static DataTable CreateSchemaFromReader(SqlDataReader reader, string tableName)
        {
            DataTable table = new DataTable(tableName);

            for (int i = 0; i < reader.FieldCount; i++)
                table.Columns.Add(new DataColumn(reader.GetName(i), reader.GetFieldType(i)));

            return table;
        }

        // Цей метод записує дані в об'єкт DataTable з тією ж схемою, що DataReader.
        private static void WriteDataFromReader(DataTable table, SqlDataReader reader)
        {
            while (reader.Read())
            {
                DataRow row = table.NewRow();

                for (int i = 0; i < reader.FieldCount; i++)
                    row[i] = reader[i];

                table.Rows.Add(row);
            }
        }


        static void Main(string[] args)
        {
            string conStr = @"Data Source=localhost; Initial Catalog=comp_firm; Integrated
Security=True";

            SqlConnection connection = new SqlConnection(conStr);
            connection.Open();

            SqlCommand cmd = new SqlCommand("SELECT * FROM PC", connection);

            SqlDataReader reader = cmd.ExecuteReader();
            DataTable table = CreateSchemaFromReader(reader, "Customers"); // створення нової
таблиці на основі схеми DataReader

            foreach (DataColumn column in table.Columns)
                Console.WriteLine("{0}: {1}", column.ColumnName, column.DataType);

            WriteDataFromReader(table, reader); // запис даних у таблицю за допомогою DataReader
            Console.WriteLine();

            foreach (DataRow row in table.Rows)
            {
                foreach (DataColumn column in table.Columns)
                    Console.WriteLine("{0}: {1}", column.ColumnName, row[column]);

                Console.WriteLine();
```

```
            }

            reader.Close();
            connection.Close();
        }
    }
}
```

Ключи, відносини та обмеження

```
using System;
using System.Data;

// Властивість Reaonly об'єкта DataColumn дозволяє встановити значення, що вказує на допустимість
зміни стовпця після додавання рядка в таблицю.

namespace CBS.ADO_NET.ColumnProperties
{
    class Program
    {
        static void Main(string[] args)
        {
            DataTable table = new DataTable();

            DataColumn column = table.Columns.Add("ReadonlyColumn", typeof(string));
            column.ReadOnly = true; // столбец таблицы с именем ReadonlyColumn доступен только для
чтения

            DataRow newRow = table.NewRow();

            newRow[0] = "ReadonlyValue";

            table.Rows.Add(newRow);

            Console.WriteLine(table.Rows[0][0]);

26            //table.Rows[0][0] = "NewValue"; // ОШИБКА выполнения
        }
    }
}
```

Запустимо програму і бачимо назву нашого стовпця.

Тепер розкоментуємо 26 рядок і спробуємо записати до неї нове значення. Отримуємо помилку, тому що наш стовпець припускає лише читання даних.

Обмеження AllowDBNull. Властивість AllowDBNull об'єкта DataColumn повертає або задає значення, що вказує на допустимість нульових значень у цьому стовпці для рядків, що належать до таблиці.

```
using System;
using System.Data;

/* Властивість AllowDBNull об'єкта DataColumn повертає або задає значення,
   що вказує на допустимість нульових значень у цьому стовпці для рядків, що належать до таблиці.
*/
namespace CBS.ADO_NET.ColumnProperties
{
    class Program
    {
        static void Main(string[] args)
        {
            DataTable table = new DataTable();

            DataColumn column = table.Columns.Add("AllowDBNullColumn", typeof(int));
            column.AllowDBNull = false;

            DataRow newRow = table.NewRow();

            newRow[0] = DBNull.Value;

            table.Rows.Add(newRow);

            Console.WriteLine(table.Rows[0][0]);
        }
    }
       }
```

Обмеження MaxLength. Свойтсво MaxLength об'єкту DataColumn повертає або задає максимальну довжину текстового стовпця.

```
using System;
using System.Data;

// Властивість MaxLength об'єкта DataColumn повертає або задає максимальну довжину текстового
стовпця.

namespace CBS.ADO_NET.ColumnProperties
{
    class Program
    {
        static void Main(string[] args)
        {
            DataTable table = new DataTable();

            DataColumn column = table.Columns.Add("MaxLengthConstraintColumn", typeof(string));
            column.MaxLength = 5;

            DataRow newRow = table.NewRow();

            //newRow[0] = "Some value";
            newRow[0] = "Some";

            table.Rows.Add(newRow);

            Console.WriteLine(table.Rows[0][0]);
        }
    }
}
```

Після запуску ми бачимо помилку, оскільки наше значення перевищує 5 символів. Спробуємо ввести значення менше 5 символів.

Обмеження `Unique`. Властивість Unique об'єкта DataColumn повертає або задає значення, що показує, чи мають значення у кожному рядку стовпця бути унікальними.

```
using System;
using System.Data;

// властивість Unique об'єкта DataColumn повертає або задає значення, що показує, чи мають значення
у кожному рядку стовпця бути унікальними.

namespace CBS.ADO_NET.ColumnProperties
{
    class Program
    {
        static void Main(string[] args)
        {
            DataTable table = new DataTable();

            DataColumn column = table.Columns.Add("UniqueColumn", typeof(string));
            column.Unique = true;

            DataRow newRow = table.NewRow();
            newRow[0] = "NonUniqueValue";
            table.Rows.Add(newRow);

            newRow = table.NewRow();
            newRow[0] = "NonUniqueValue";
            //newRow[0] = "UniqueValue";
            table.Rows.Add(newRow); // помилка виконання при порушенні обмеження Unique

            Console.WriteLine(table.Rows[0][0]);
            Console.WriteLine(table.Rows[1][0]);
        }
    }
       }
```

Під час запуску настає виняткова ситуація. Якщо ми змінимо друге значення, все буде працювати.

Обмеження UniqueConstraint. Клас UniqueConstraint надає обмеження на набір стовпців, у яких усі значення мають бути унікальними. Слід користуватися цим обмеженням у разі, коли необхідно гарантувати унікальність комбінацій значень різних полів таблиці

```
using System.Data;

/* клас UniqueConstraint надає обмеження на набір стовпців, у яких усі значення мають бути
унікальними.
   Слід користуватися цим обмеженням у разі, коли необхідно гарантувати унікальність комбінацій
   значень різних полів таблиці
*/

namespace CBS.ADO_NET.TableConstraints
{
    static class TableExtentions
    {
          public static void AddRow(this DataTable table, string column1Val, string column2Val)
        {
            var newRow = table.NewRow();

            newRow[0] = column1Val;
            newRow[1] = column2Val;

            table.Rows.Add(newRow);
        }
    }

    class Program
    {
        static void Main(string[] args)
        {
            DataTable table = new DataTable();

            DataColumn column1 = table.Columns.Add("Column1", typeof(string));
            DataColumn column2 = table.Columns.Add("Column2", typeof(string));

            table.Constraints.Add("tableUniqueConstraint", new[] { column1, column2 }, false);

            table.AddRow("Unique", "Unique");
            table.AddRow("Unique", "Unique");
        }
    }
   }
```

Нам видає помилку

Встановлення первинного ключа. PrimaryKey – особливий вид обмеження унікальності. Первинний ключ таблиці може бути лише один

```
using System;
using System.Data;

//PrimaryKey – особливий вид обмеження унікальності. Первинний ключ таблиці може бути лише один

namespace CBS.ADO_NET.TableConstraints
{
    class Program
    {
        static void Main(string[] args)
        {
            DataTable table = new DataTable();

            DataColumn column1 = table.Columns.Add("Column1", typeof(string));
            DataColumn column2 = table.Columns.Add("Column2", typeof(string));

            table.Constraints.Add(new UniqueConstraint(column1, true));

            Console.WriteLine("is unique: " + table.Columns[0].Unique);
            Console.WriteLine("Primary key columns count: "+ table.PrimaryKey.Length);
            Console.WriteLine("Primary key column name: " + table.PrimaryKey[0].ColumnName);


        }
    }
      }
```

Після запуску, бачимо, що ми маємо первинний ключ, і він встановлений на стовпець `Column 1.`

Створення зовнішнього ключа. ForeignKeyConstraint – обмеження, що гарантує, що не можна створити рядок у дочірній таблиці, яка посилається на неіснуючий рядок батьківської таблиці.

```
using System.Data;

/* ForeignKeyConstraint – обмеження, що гарантує, що не можна створити рядок у дочірній таблиці,
   яка посилається на неіснуючий рядок батьківської таблиці.
 */


namespace CBS.ADO_NET.TableConstraints
{
    class Program
    {
        static void Main(string[] args)
        {
            DataSet ds = new DataSet();

            DataTable parentTable = new DataTable(); // батьківська таблиця
            DataTable childTable = new DataTable(); // дочірня таблиця

            DataColumn childColumn = childTable.Columns.Add("ChildColumn", typeof(int));
            DataColumn parentColumn = parentTable.Columns.Add("ParentColumn", typeof(int));

            // обмеження ForeignKeyConstraint буде працювати, якщо батьківська та дочірня таблиця
знаходяться в одному об'єкті DataSet
            ds.Tables.AddRange(new DataTable[] { childTable, parentTable });

            parentTable.Constraints.Add(new UniqueConstraint(parentColumn));
            childTable.Constraints.Add(new ForeignKeyConstraint(parentColumn, childColumn));

            DataRow parentRow = parentTable.NewRow();
            parentRow[0] = 1;
            parentTable.Rows.Add(parentRow);

            DataRow childRow = childTable.NewRow();
            childRow[0] = 1;
            //childRow[0] = 0;
            childTable.Rows.Add(childRow);
        }
    }
       }
```

Після запуску все працює. Спробуємо вставити рядок, який не збігається з батьківським. Як бачимо, видає помилку.