Faculty of Electronics and Information Technology

WARSAW UNIVERSITY OF TECHNOLOGY

Graphical User Interfaces (EGUI)

Entity Framework

Julian Myrcha
Institute of Computer Science

October 6, 2024

How to speak with database

History makes a come back ...

- First every provider created his own libraries for C language

How to speak with database

History makes a come back ...

- First every provider created his own libraries for C language
- For readibility `embedded sql` was created - a C preprocesor translates plain SQL into C library function calls

How to speak with database

History makes a come back ...

- First every provider created his own libraries for C language
- For readibility `embedded sql` was created - a C preprocesor translates plain SQL into C library function calls
- Then Microsoft created `ODBC` (it is still procedural approach)

How to speak with database

History makes a come back ...

- First every provider created his own libraries for C language
- For readibility `embedded sql` was created - a C preprocesor translates plain SQL into C library function calls
- Then Microsoft created `ODBC` (it is still procedural approach)
  - After 20 years ODBC drivers are still available for any database solution

How to speak with database

History makes a come back ...

- First every provider created his own libraries for C language
- For readibility `embedded sql` was created - a C preprocesor translates plain SQL into C library function calls
- Then Microsoft created `ODBC` (it is still procedural approach)
  - After 20 years ODBC drivers are still available for any database solution
  - Driver implements part of the (closed) specification

How to speak with database

History makes a come back ...

- First every provider created his own libraries for C language
- For readibility `embedded sql` was created - a C preprocesor translates plain SQL into C library function calls
- Then Microsoft created `ODBC` (it is still procedural approach)
  - After 20 years ODBC drivers are still available for any database solution
  - Driver implements part of the (closed) specification
  - Database accessible using a name (alias) configured by the system admin using separate system tool

How to speak with database

History makes a come back ...

- First every provider created his own libraries for C language
- For readibility `embedded sql` was created - a C preprocesor translates plain SQL into C library function calls
- Then Microsoft created `ODBC` (it is still procedural approach)
  - After 20 years ODBC drivers are still available for any database solution
  - Driver implements part of the (closed) specification
  - Database accessible using a name (alias) configured by the system admin using separate system tool
  - ODBC driver typically uses client database libraries

How to speak with database

History makes a come back ...

- First every provider created his own libraries for C language
- For readibility `embedded sql` was created - a C preprocesor translates plain SQL into C library function calls
- Then Microsoft created `ODBC` (it is still procedural approach)
  - After 20 years ODBC drivers are still available for any database solution
  - Driver implements part of the (closed) specification
  - Database accessible using a name (alias) configured by the system admin using separate system tool
  - ODBC driver typically uses client database libraries
- And later Microsoft introduced many other libraries, like ADO (ActiveX Data Objects)

How to speak with database

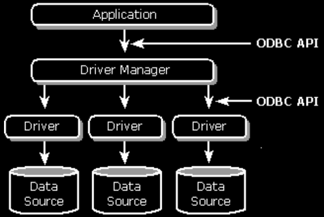
History makes a come back ...

- First every provider created his own libraries for C language
- For readibility `embedded sql` was created - a C preprocesor translates plain SQL into C library function calls
- Then Microsoft created `ODBC` (it is still procedural approach)
  - After 20 years ODBC drivers are still available for any database solution
  - Driver implements part of the (closed) specification
  - Database accessible using a name (alias) configured by the system admin using separate system tool
  - ODBC driver typically uses client database libraries
- And later Microsoft introduced many other libraries, like `ADO` (ActiveX Data Objects)
- When .Net was introduced `ADO` was rewritten as `ADO.NET`

ODBC Architecture

The ODBC architecture has four components:

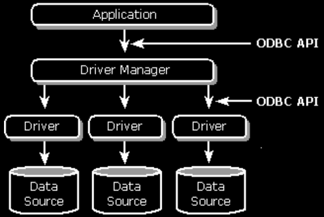
- Application. Performs processing and calls ODBC functions to submit SQL statements and retrieve results.



ODBC Architecture

The ODBC architecture has four components:

- Application. Performs processing and calls ODBC functions to submit SQL statements and retrieve results.

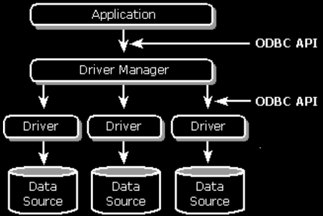


- Driver Manager. Loads and unloads drivers on behalf of an application. Processes ODBC function calls or passes them to a driver.

ODBC Architecture

The ODBC architecture has four components:

- Application. Performs processing and calls ODBC functions to submit SQL statements and retrieve results.

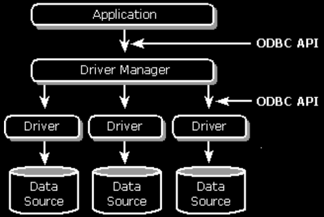


- Driver Manager. Loads and unloads drivers on behalf of an application. Processes ODBC function calls or passes them to a driver.
- Driver. Processes ODBC function calls, submits SQL requests to a specific data source, and returns results to the application. If necessary, the driver modifies an application's request so that the request conforms to syntax supported by the associated DBMS.

ODBC Architecture

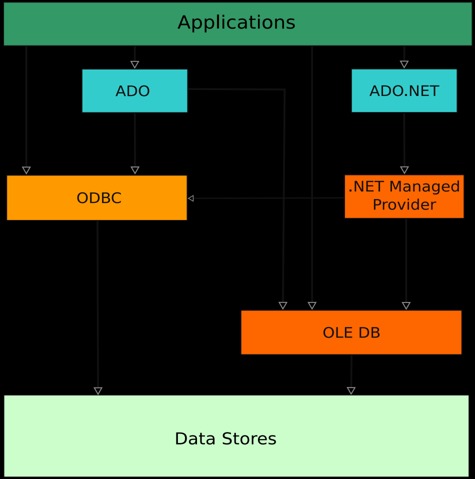
The ODBC architecture has four components:

- Application. Performs processing and calls ODBC functions to submit SQL statements and retrieve results.



- Driver Manager. Loads and unloads drivers on behalf of an application. Processes ODBC function calls or passes them to a driver.
- Driver. Processes ODBC function calls, submits SQL requests to a specific data source, and returns results to the application. If necessary, the driver modifies an application's request so that the request conforms to syntax supported by the associated DBMS.
- Data source. Consists of the data the user wants to access and its associated operating system, DBMS, and network platform (if any) used to access the DBMS.

MDAC


Applications
ADO
ADO.NET
ODBC
.NET Managed Provider
OLE DB
Data Stores

What is JDBC?

- JDBC stands for Java Database Connectivity

What is JDBC?

- JDBC stands for Java Database Connectivity
- Standard Java API for database-independent connectivity between the Java programming language and a wide range of databases.

What is JDBC?

- JDBC stands for Java Database Connectivity
- Standard Java API for database-independent connectivity between the Java programming language and a wide range of databases.
- The JDBC library includes APIs for each of the tasks mentioned below that are commonly associated with database usage.
  - Making a connection to a database.

What is JDBC?

- JDBC stands for Java Database Connectivity
- Standard Java API for database-independent connectivity between the Java programming language and a wide range of databases.
- The JDBC library includes APIs for each of the tasks mentioned below that are commonly associated with database usage.
    - Making a connection to a database.
    - Creating SQL or MySQL statements.

What is JDBC?

- JDBC stands for Java Database Connectivity
- Standard Java API for database-independent connectivity between the Java programming language and a wide range of databases.
- The JDBC library includes APIs for each of the tasks mentioned below that are commonly associated with database usage.
    - Making a connection to a database.
    - Creating SQL or MySQL statements.
    - Executing SQL or MySQL queries in the database.

What is JDBC?

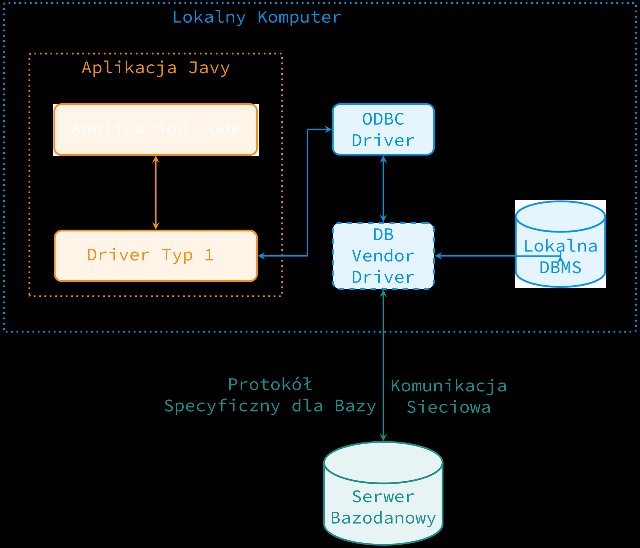
- JDBC stands for Java Database Connectivity
- Standard Java API for database-independent connectivity between the Java programming language and a wide range of databases.
- The JDBC library includes APIs for each of the tasks mentioned below that are commonly associated with database usage.
  - Making a connection to a database.
  - Creating SQL or MySQL statements.
  - Executing SQL or MySQL queries in the database.
  - Viewing & Modifying the resulting records.

What is JDBC?

- JDBC stands for Java Database Connectivity
- Standard Java API for database-independent connectivity between the Java programming language and a wide range of databases.
- The JDBC library includes APIs for each of the tasks mentioned below that are commonly associated with database usage.
    - Making a connection to a database.
    - Creating SQL or MySQL statements.
    - Executing SQL or MySQL queries in the database.
    - Viewing & Modifying the resulting records.
- Fundamentally, JDBC is a specification that provides a complete set of interfaces that allows for portable access to an underlying database.

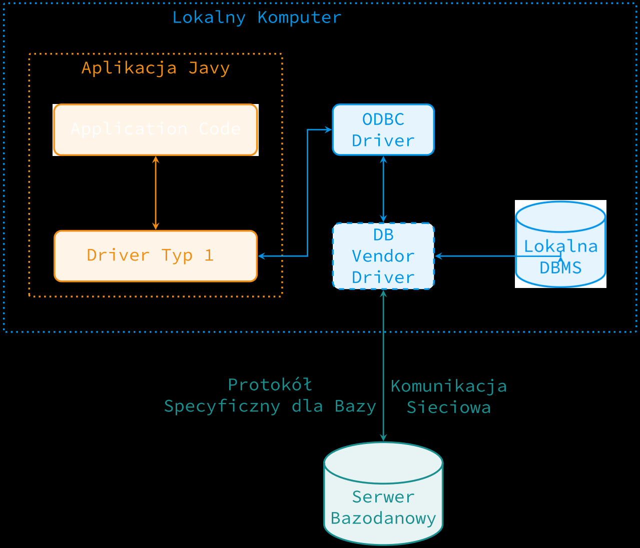
jdbc 1 - most JDBC-ODBC

- that calls native code of the locally available ODBC driver



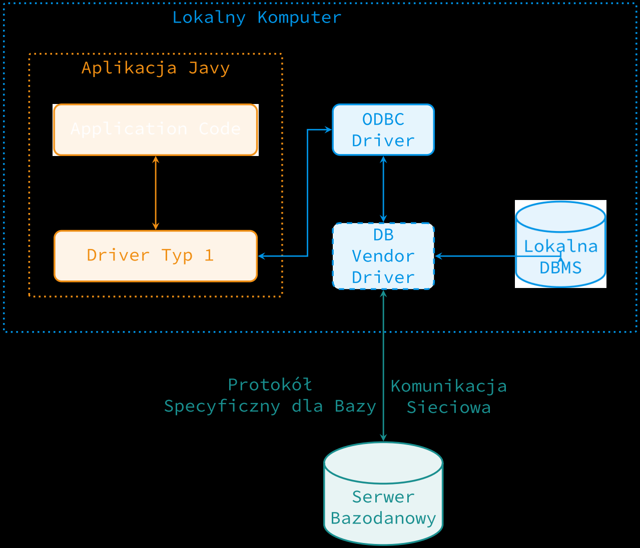
jdbc 1 - most JDBC-ODBC

- that calls native code of the locally available ODBC driver
- The ODBC driver needs to be installed on the client machine.



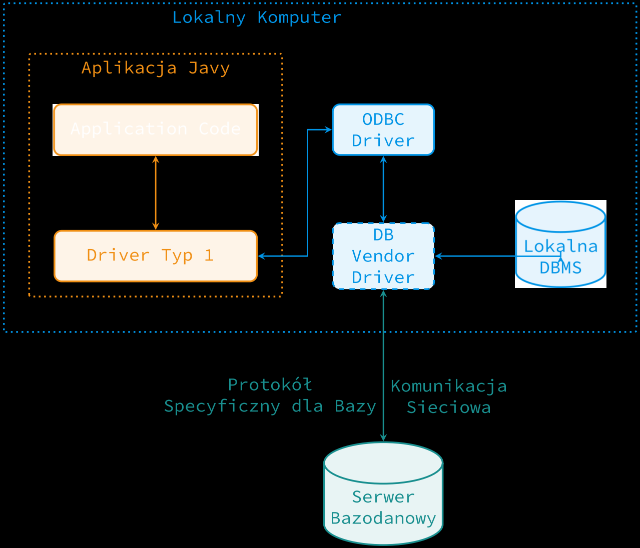
jdbc 1 - most JDBC-ODBC

- that calls native code of the locally available ODBC driver
- The ODBC driver needs to be installed on the client machine.
- In JDBC 4.2, JDBC-ODBC bridge has been removed



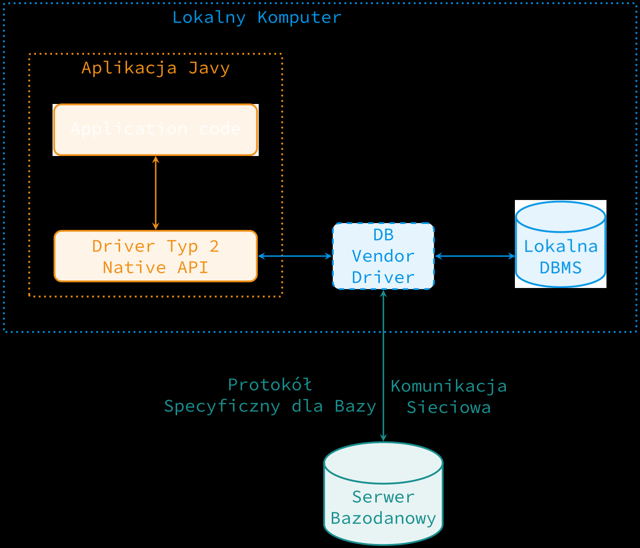
jdbc 1 - most JDBC-ODBC

- that calls native code of the locally available ODBC driver
- The ODBC driver needs to be installed on the client machine.
- In JDBC 4.2, JDBC-ODBC bridge has been removed
- No support from JDK 1.8 (Java 8)



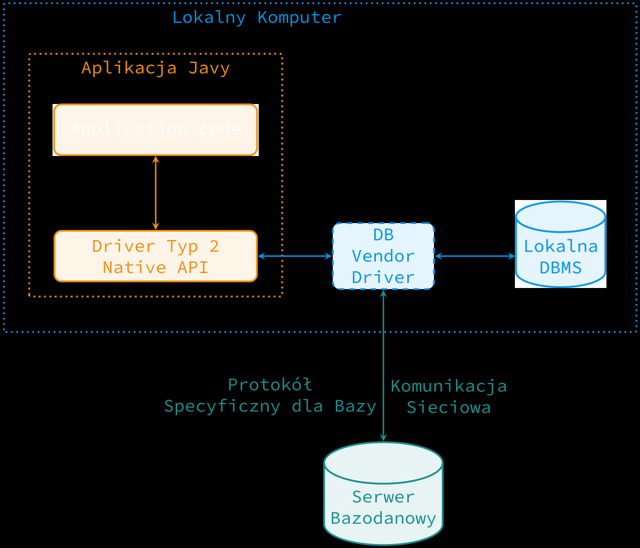
jdbc 2 - natywny driver bazy

- calls database vendor native library on a client side.



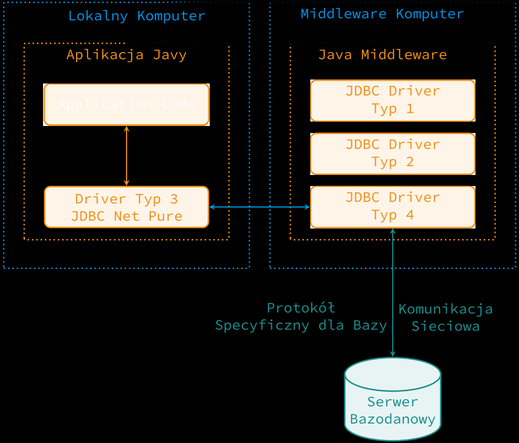
jdbc 2 - natywny driver bazy

- calls database vendor native library on a client side.
- This code then talks to database over the network.



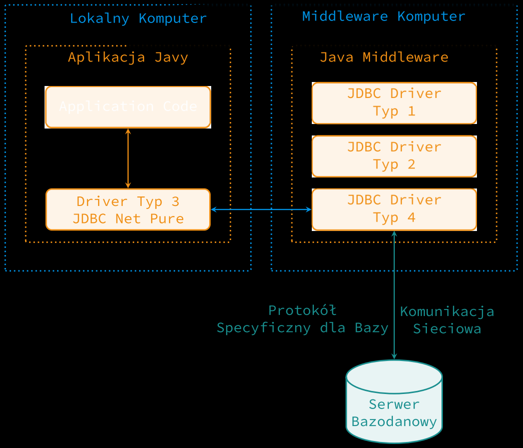
odbc 3 - middleware

- the pure-java driver that talks with the server-side middleware



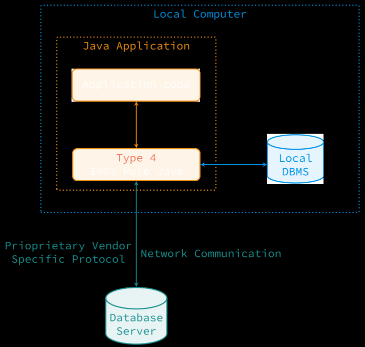
odbc 3 - middleware

- the pure-java driver that talks with the server-side middleware
- middleware then talks to the database.



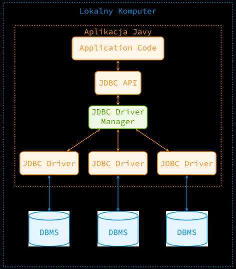
odbc 4 - pure java

- the pure-java driver that uses database native protocol.



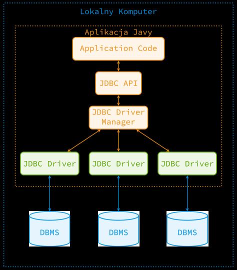
JDBC Architecture

- DriverManager-This class manages a list of database drivers
    - Matches connection requests from the java application with the proper database driver using communication sub protocol.
    - The first driver that recognizes a certain subprotocol under JDBC will be used to establish a database Connection.



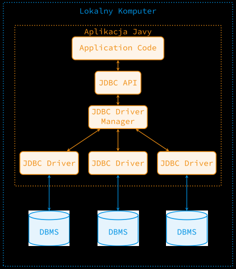
JDBC Architecture

- DriverManager-This class manages a list of database drivers
- Driver- This interface handles the communications with the database server
  - You will interact directly with Driver objects very rarely.
  - Instead, you use DriverManager objects, which manages objects of this type.
  - It also abstracts the details associated with working with Driver objects.



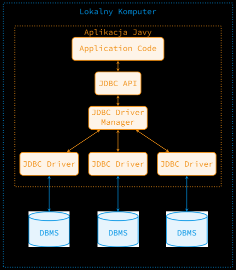
JDBC Architecture

- DriverManager-This class manages a list of database drivers
- Driver- This interface handles the communications with the database server
- Connection- This interface with all methods for contacting a database
  - The connection object represents communication context
  - all communication with database is through connection object only.



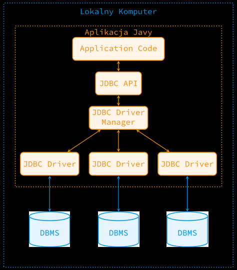
JDBC Architecture

- DriverManager-This class manages a list of database drivers
- Driver- This interface handles the communications with the database server
- Connection- This interface with all methods for contacting a database
- Statement- You use objects created from this interface to submit the SQL statements to the database
    - Some derived interfaces accept parameters in addition to executing stored procedures.



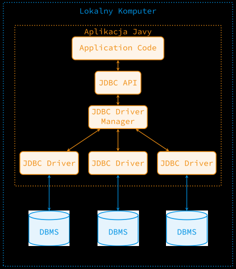
JDBC Architecture

- DriverManager-This class manages a list of database drivers
- Driver- This interface handles the communications with the database server
- Connection- This interface with all methods for contacting a database
- Statement- You use objects created from this interface to submit the SQL statements to the database
- ResultSet- These objects hold data retrieved from a database
  - after you execute an SQL query using Statement objects
  - It acts as an iterator to allow you to move through its data



JDBC Architecture

- DriverManager-This class manages a list of database drivers
- Driver- This interface handles the communications with the database server
- Connection- This interface with all methods for contacting a database
- Statement- You use objects created from this interface to submit the SQL statements to the database
- ResultSet- These objects hold data retrieved from a database
- SQLException- This class handles any errors that occur in a database application



Otwieranie i zamykanie połączenia

- obiekt `Connection` uzyskujemy za pomocą wywołania statycznej metody `getConnection` klasy `DriverManager`

```java
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.SQLException;

Connection conn = null;
try {
  log.info("Opening connection to bookStoreDB");
  conn = DriverManager.getConnection("jdbc:hsqldb:mem:bookStoreDB", "SA", "");
} catch (SQLException ex) {
  log.error("Unable to open connection", ex);
}
```

Otwieranie i zamykanie połączenia

- obiekt `Connection` uzyskujemy za pomocą wywołania statycznej metody `getConnection` klasy `DriverManager`
  - na podstawie connection stringa wybierana jest baza danych do której się łączymy

```java
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.SQLException;

Connection conn = null;
try {
  log.info("Opening connection to bookStoreDB");
  conn = DriverManager.getConnection("jdbc:hsqldb:mem:bookStoreDB", "SA", "");
} catch (SQLException ex) {
  log.error("Unable to open connection", ex);
}
```

Otwieranie i zamykanie połączenia

- obiekt `Connection` uzyskujemy za pomocą wywołania statycznej metody `getConnection` klasy `DriverManager`
  - na podstawie connection stringa wybierana jest baza danych do której się łączymy
  - w przypadku błędu połączenia rzucany jest wyjątek `SQLException`

```java
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.SQLException;

Connection conn = null;
try {
  log.info("Opening connection to bookStoreDB");
  conn = DriverManager.getConnection("jdbc:hsqldb:mem:bookStoreDB", "SA", "");
} catch (SQLException ex) {
  log.error("Unable to open connection", ex);
}
```

Otwieranie i zamykanie połączenia

- obiekt `Connection` uzyskujemy za pomocą wywołania statycznej metody `getConnection` klasy `DriverManager`
  - na podstawie connection stringa wybierana jest baza danych do której się łączymy
  - w przypadku błędu połączenia rzucany jest wyjątek `SQLException`
- zamknięcie połączenia wymaga wywołania metody `close()`

```
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.SQLException;

Connection conn = ...
try {
  log.info("Closing database connection to bookStoreDB");
  conn.close();
} catch (SQLException ex) {
  log.error("Unable to close connection", ex);
}
```

Otwieranie i zamykanie połączenia

- obiekt `Connection` uzyskujemy za pomocą wywołania statycznej metody `getConnection` klasy `DriverManager`
  - na podstawie connection stringa wybierana jest baza danych do której się łączymy
  - w przypadku błędu połączenia rzucany jest wyjątek `SQLException`
- zamknięcie połączenia wymaga wywołania metody `close()`
  - trzeba zrobić ręcznie, bo sprzątaczka zawoła gdy będzie zwalniała pamięć co może się opóźnić

```java
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.SQLException;

Connection conn = ...
try {
  log.info("Closing database connection to bookStoreDB");
  conn.close();
} catch (SQLException ex) {
  log.error("Unable to close connection", ex);
}
```

Otwieranie i zamykanie połączenia

```java
package EGUI;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.SQLException;

public class DBContext implements AutoCloseable {
  private static final Logger log = LoggerFactory.getLogger(DBContext.class);
  private Connection conn = null;

  public void close() {
    if (conn != null) {
      try {
        log.info("Closing database connection to bookStoreDB");
        conn.close();
      } catch (SQLException ex) {
        log.error("Unable to close connection", ex);
      }
      conn = null;
    }
  }

  public Connection getConnection() throws SQLException { // zasob nie w konstruktorze!
    if (conn == null) {
      log.info("Opening connection to bookStoreDB");
      conn = DriverManager.getConnection("jdbc:hsqldb:mem:bookStoreDB", "SA", "");
    }
    return conn;
  }
}
```

Otwieranie i zamykanie połączenia

- obiekt `Connection` uzyskujemy za pomocą wywołania statycznej metody `getConnection` klasy `DriverManager`
  - na podstawie connection stringa wybierana jest baza danych do której się łączymy
  - w przypadku błędu połączenia rzucany jest wyjątek `SQLException`
- zamknięcie połączenia wymaga wywołania metody `close()`
  - trzeba zrobić ręcznie, bo sprzątaczka zawoła gdy będzie zwalniała pamięć co może się opóźnić
  - można wykorzystać AutoCloseable

```
package EGUI;

try(var ctx = new DBContext()) {
    var conn = ctx.getConnection();  // klasa Context implementuje AutoCloseable
    ...
}
```

wykonanie polecenia nie produkującego wyników

```java
Connection conn = ... ;
Statement stmt = null;
try {
    stmt = conn.createStatement();
    stmt.execute(
        "CREATE TABLE bookstore ("+
        "id INT IDENTITY,"+
        " ISBN VARCHAR(30),"+
        " title VARCHAR(30),"+
        " pages INT)");
    log.info("Creating table");
    success = true;
} catch (SQLException e) {
    log.error("Unable to create the database table", e);
} finally {
    if (stmt != null)
        try {
            stmt.close();
        } catch (SQLException e) {
        }
}
```

Connection string

derby - ”jdbc:derby:./data;create=true” - baza w pliku, w katalogu data

pom.xml

```
    <!-- JDBC Connector for DERBY -->
    <dependency>
      <groupId>org.apache.derby</groupId>
      <artifactId>derby</artifactId>
      <version>10.15.2.0</version>
    </dependency>
```

Connection string

derby - `”jdbc:derby:./data;create=true”` - baza w pliku, w katalogu data

sqlserver - `”jdbc:sqlserver://172.17.0.2;databaseName=TestDB;”`

pom.xml

```
<!-- https://mvnrepository.com/artifact/com.microsoft.sqlserver/mssql-jdbc -->
<dependency>
  <groupId>com.microsoft.sqlserver</groupId>
  <artifactId>mssql-jdbc</artifactId>
  <version>9.1.1.jre15-preview</version>
</dependency>
```

Connection string

derby - ”jdbc:derby:./data;create=true” - baza w pliku, w katalogu data
sqlserver - ”jdbc:sqlserver://172.17.0.2;databaseName=TestDB;”
mysql -”jdbc:mysql://localhost:3306/pap”

pom.xml

```
<!-- JDBC Connector for MySQL -->
<dependency>
  <groupId>mysql</groupId>
  <artifactId>mysql-connector-java</artifactId>
  <version>8.0.16</version>
</dependency>
```

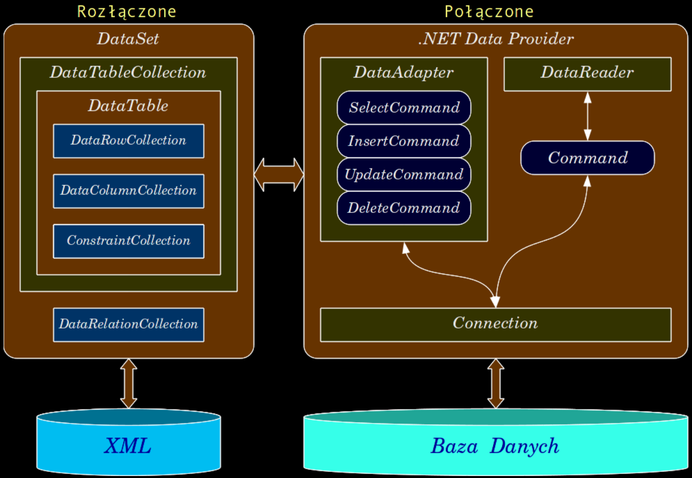
Connection string

derby - "jdbc:derby:./data;create=true" - baza w pliku, w katalogu data
sqlserver - "jdbc:sqlserver://172.17.0.2;databaseName=TestDB;"
mysql -"jdbc:mysql://localhost:3306/pap"
oracle -"jdbc:oracle:thin:@localhost:51521/XEPDB1"

pom.xml

```xml
<dependency>
  <groupId>com.oracle.database.jdbc</groupId>
  <artifactId>ojdbc8-production</artifactId>
  <version>19.7.0.0</version>
  <type>pom</type>
</dependency>
```

ADO.NET - Architecture



ADO.NET - SQL-Server

For each database we have separate class libraries e.g. `OracleConnection` i `SqlConnection`

DataProvider

- SqlConnection
- SqlCommand
- SqlDataReader
- SqlTransaction
- SqlParameter
- SqlParameterCollection
- SqlCommandBuilder
- SqlConnectionStringBuilder
- SqlPermission

ADO.NET-SqlCommand (1)

What we want to do (property CommandText). How to interpret CommandText is set by **CommandType**

- sql
- table Name
- proc Name

execution - 4 possibilities:

- ExecuteNonQuery
- ExecuteScalar
- ExecuteReader
- ExecuteXmlReader

ADO.NET-SqlCommand (2) - pobranie danych

```
public List<Product> GetProducts() {
  SqlConnection con = new SqlConnection(connectionString);
  SqlCommand cmd = new SqlCommand("GetProducts", con);
  cmd.CommandType = CommandType.StoredProcedure;
  List<Product> products = new List<Product>();
  try {
    con.Open();
    SqlDataReader reader = cmd.ExecuteReader();
    while (reader.Read()) {
        Product product = new Product((string)reader["ModelNumber"],
          (string)reader["ModelName"], (decimal)reader["UnitCost"],
          (string)reader["Description"], (string)reader["CategoryName"],
          (string)reader["ProductImage"]);
       products.Add(product);
    }
  } finally {
    con.Close();
  }
return products;
```

ADO.NET-SqlCommand (3) - modyfikacja

```
ALTER PROCEDURE [NTR].[AddStudent]
    @Name nvarchar(50),
    @Surname nvarchar(50),
    @IDStudent int output
AS
BEGIN
    INSERT INTO [dbo].[Student] ([Name],[Surname],[INNO])
        VALUES (@Name,@Surname,NULL);
    SET @IDStudent = @@IDENTITY
END
```

ADO.NET-SqlCommand (4) - dodanie

```
private string ntrConnectionString =
    ConfigurationManager.ConnectionStrings["NTRCS"].ConnectionString;
using (SqlConnection connection=new SqlConnection(ConnectionString)){
  SqlCommand cmd = connection.CreateCommand();
  cmd.CommandText = "[NTR].[AddStudent]";
  cmd.CommandType = CommandType.StoredProcedure;
  cmd.Parameters.Add(new SqlParameter("@Name",SqlDbType.NVarChar,50));
  cmd.Parameters["@Name"].Value = "Jan";
  .. ..
  cmd.Parameters.Add(new SqlParameter("@IDStudent",SqlDbType.SqlInt));
  cmd.Parameters["@IDStudent"].Direction = ParameterDirection.Output;
  try {
      comm.ExecuteNonQuery();
      idStudent = (int)cmd.Parameters["@IDStudent"].Value;
      return null;
  }
  catch (SqlException exc) {
        return exc.Message;
  }
}
```

Ado.Net-Dataset (1)

Container for Data

- DataSet
- DataColumn
- DataRow
- DataRelation

Ado.Net-Dataset (2)

Container for Data

- DataSet
- DataColumn
- DataRow
- DataRelation

Used if:

- we are sending pack of data to other component

Ado.Net-Dataset (3)

Container for Data

- DataSet
- DataColumn
- DataRow
- DataRelation

Used if:

- we are sending pack of data to other component
- We would like to persist (temporary) data to disk

Ado.Net-Dataset (4)

Container for Data

- DataSet
- DataColumn
- DataRow
- DataRelation

Used if:

- we are sending pack of data to other component
- We would like to persist (temporary) data to disk
- We would like to implement scrollable data source

Ado.Net-Dataset (5)

Container for Data

- DataSet
- DataColumn
- DataRow
- DataRelation

Used if:

- we are sending pack of data to other component
- We would like to persist (temporary) data to disk
- We would like to implement scrollable data source
- we would like to operate on joined tables

Ado.Net-Dataset (6)

Container for Data

- DataSet
- DataColumn
- DataRow
- DataRelation

Used if:

- we are sending pack of data to other component
- We would like to persist (temporary) data to disk
- We would like to implement scrollable data source
- we would like to operate on joined tables
- we would like to use database controls

Ado.Net-Dataset (7)

Container for Data

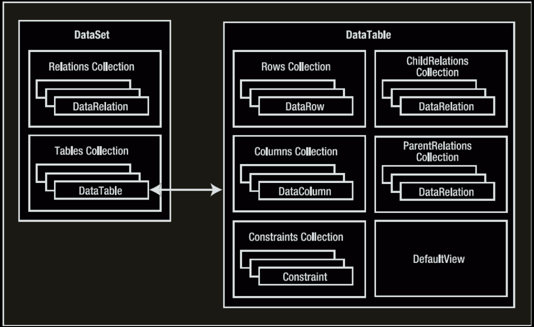
- DataSet
- DataColumn
- DataRow
- DataRelation

Used if:

- we are sending pack of data to other component
- We would like to persist (temporary) data to disk
- We would like to implement scrollable data source
- we would like to operate on joined tables
- we would like to use database controls

but now we have Entity Framework which do things better ....

Ado.Net-Dataset (8)



Ado.Net-Dataset (9) załadowanie danych

Pobranie obiektu DataSet zawierającego kilka tabelek - tutaj druga tabelka przechowuje rozmiar tabeli przed filtrowaniem

```
public static DataSet GetItems(SqlConnection cn,
    int assignedUserID, int itemID,
    string itemName, DateTime? purchaseDate ) {
    DataSet ds = new DataSet();
    try {
      SqlDataAdapter da = new SqlDataAdapter("Store.GetItems", cn);
      SqlCommand cmd = da.SelectCommand;
      cmd.CommandType = CommandType.StoredProcedure;
      cmd.Parameters.AddWithValue("@AssignedUserID", assignedUserID);
      cmd.Parameters.AddWithValue("@ItemID", claimTypeID);
      cmd.Parameters.AddWithValue("@ItemName", itemName);
      cmd.Parameters.AddWithValue("@PurchaseDate", purchaseDate ??
                                                   SqlDateTime.Null);
      da.Fill(ds);
      TableDef(ds, 0, Names.Items, false);
      TableDef(ds, 1, Names.TotalRows, false);
    } catch (Exception ex) { throw ex; }
    return ds;
}
```

Ado.Net-Dataset (10) załadowanie danych

Pobranie DataSetu zawierającego jeden rekord z tabelki

```
public static DataSet GetItem(SqlConnection cn, int itemID) {
    DataSet ds = new DataSet();
    try {
      using (SqlDataAdapter da=new SqlDataAdapter("Store.GetItem",cn)){
        da.SelectCommand.CommandType = CommandType.StoredProcedure;
        da.SelectCommand.Parameters.AddWithValue("@ItemID", itemID);
        da.Fill(ds);
        // nazwy
        TableDef(ds, 0, Names.Items, true);
        TableDef(ds, 1, Names.Notes, true);
        RelationDef(ds, Names.Items, Names.Notes);
      }
    } catch (Exception ex) { throw ex; }
    return ds;
}
```

ADO.NET- SqlDataAdapter (1)

Zapis zmian z Datasetu do bazy

```
public static DataSet UpdateItem(SqlConnection cn,int actionUserID,
                                                                DataSet ds){
  DataTable  tblItems = ds.Tables[Names.Items];
  DataTable  tblNotes = ds.Tables[Names.Notes];
  SqlDataAdapter daItems = DataAdapterDef(ds, "[Store].[Items]", cn);
  SqlDataAdapter daNotes = DataAdapterDef(ds, "[Store].[Notes]", cn);
  // INSERT (master->details)
  daItems.Update(tblItems.Select("", "", DataViewRowState.Added));
  daNotes.Update(tblNotes.Select("", "", DataViewRowState.Added));
  // UPDATE (bez znaczenia)
  daLogs.Update(tblLogs.Select("","",DataViewRowState.ModifiedCurrent));
  daNotes.Update(tblNotes.Select("","",DataViewRowState.ModifiedCurrent));
  // DELETE (detail->master)
  daNotes.Update(tblNotes.Select("", "", DataViewRowState.Deleted));
  daItems.Update(tblItems.Select("", "", DataViewRowState.Deleted));
  return ds;
}
```

ADO.NET- SqlDataAdapter (1) - przykład

```
public static void TableDef(
  DataSet ds,
  int pos,
  String tableName,
  bool autoIncrement
){
    ds.Tables[pos].TableName = tableName;
    DataColumn[] primaryKey = new DataColumn[]
                           { ds.Tables[tableName].Columns["ID"] };

    if (autoIncrement) {
        primaryKey[0].AutoIncrement = true;
        primaryKey[0].AutoIncrementStep = -1;
        primaryKey[0].AutoIncrementSeed = -1;
    }
    ds.Tables[tableName].PrimaryKey = primaryKey;
}
```

ADO.NET- SqlDataAdapter (2) - przykład

```
public static SqlCommand CreateCommand(String procName,
                                       SqlConnection cn {
    List<SqlParameter> param = GetParameters(procName, cn);
    SqlCommand cmd = new SqlCommand(procName, cn);
    cmd.CommandType = CommandType.StoredProcedure;
    SqlParameterCollection pc = cmd.Parameters;
    foreach (SqlParameter p in param)
        pc.Add(p);
    return cmd;
}
```

```
public static void RelationDef(DataSet ds, string masters,
                                           string details) {
    string master = masters.Substring(0, masters.Length - 1);
// cut last character - s
    DataColumn pk = ds.Tables[masters].Columns["ID"];
    DataColumn fk = ds.Tables[details].Columns[master+"ID"];
    DataRelation rel = new DataRelation(master+details, pk, fk);
    ds.Relations.Add(rel);
}
```

ADO.NET- SqlDataAdapter (3) - przykład

obiekt do zarządzania DataSet-em

```
public static SqlDataAdapter DataAdapterDef(
                                     DataSet ds, string tableName, SqlConnection cn
) {
  SqlDataAdapter da = new SqlDataAdapter();
  da.InsertCommand  = CreateCommand( tableName.Replace("].[","].[Insert"), cn);
  da.UpdateCommand  = CreateCommand( tableName.Replace("].[","].[Update"), cn);
  da.DeleteCommand  = CreateIDCommand(tableName.Replace("].[","].[Delete"), cn);
  return da;
}
```

- Utworzenie komendy biorącej jako parametr tylko identyfikator

```
public static SqlCommand CreateIDCommand(String procName, SqlConnection cn) {
    SqlCommand cmd = new SqlCommand(procName, cn);
    cmd.CommandType = CommandType.StoredProcedure;
    SqlParameterCollection pc = cmd.Parameters;
    SqlParameter par = new SqlParameter("@ID", SqlDbType.Int, 4, "ID");
    pc.Add(par);
    return cmd;
}
```

ADO.NET- SqlCommand (4) - przykład

zapis do bazy

```
public static SqlCommand CreateUpdateCommand(String procName,
                                             SqlConnection cn) {
    List<SqlParameter> param = GetParameters(procName, cn);
    SqlCommand cmd = new SqlCommand(procName, cn);
    cmd.CommandType = CommandType.StoredProcedure;
    SqlParameterCollection pc = cmd.Parameters;
    foreach (SqlParameter p in param)
        pc.Add(p);
    foreach (SqlParameter p in param) {
        if (p.SourceColumn == "ID")
            continue;
        int size = p.Size;
        SqlParameter par =
             new SqlParameter(p.ParameterName, p.SqlDbType,
                                    size, p.SourceColumn);
        par.SourceVersion = DataRowVersion.Original;
        pc.Add(par);
    }
    return cmd;
}
```

Instead of introduction

Sql-Server versions

- `Sql Server` installed on the same or remote system
- `Sql Server Express` installed on the same or remote system
- `Sql Server Localdb` - file based version working on local account

Tools

- command line utilities (Windows/Linux)
- `Sql Server Management Studio`
- `Sql Server Management Studio Express`
- `Visual Studio` - pages `Server Explorer` and `Sql Server`
- `Visual Studio Code` - plugin `mssql`
- `Sql server Configuration Manager` - protocols

there may be several instances of several versions of SQL-Server working on the same computer. Only one is default (.) or localhost, others are named like `./SqlExpress`

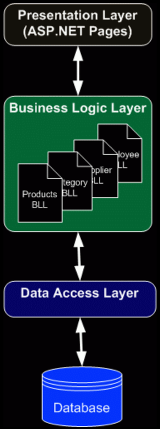
Layer architecture - pros and cons

- presentation layer (`ModelView, View`)
- business logic layer `BLL Model`)
- data access layer `DAL` (`Model`)

such split enables scaffolding part of the code

ORM - Object Relational Mapping

- NHibernate
- Entity Framework



Entity Framework advantages

- partial code generation
- database schema generation (sometime)
- model visualisation (sometime)
- Solution standarisation - we avoid prioprietary solutions which should be maintained
- Compatibility with other technologies (MVC, ASP.NET, Forms, LINQ)
- support for different application scenarios

  Code first nowa baza Code first new database - databese build from classes

  Code first istniejąca baza code first existing database - classes build from database

  Model first nowa baza Model first new base - we create model from which base is created

  Model first istniejąca baza model first existing database - model created from base

DBContext example 2

Lets have an data items

```
public class TodoItem {
    public long Id { get; set; }
    public string Name { get; set; }
    public bool IsComplete { get; set; }
}
```

And Data context to manage it:

```
public class TodoContext : DbContext {
  public TodoContext(DbContextOptions<TodoContext> options): base(options) {}
  public DbSet<TodoItem> TodoItems { get; set; }
}
```

then we can write:

```
_context.TodoItems.Any(e => e.Id == id); // returns true if exist element with id
_context.TodoItems.Remove(todoItem);
await _context.SaveChangesAsync();       // save changes in database
var todoItem = await _context.TodoItems.FindAsync(id);
```

Convention over Configuration

- If you follow convention you can avoid configuration
- but you may configure things you choose
- In Entity Framework conventions are configurable

```
public class BlogMap : EntityTypeConfiguration<Blog> {
  public BlogMap() {
    // Primary Key
    this.HasKey(t => t.BlogId);
    // Properties
    this.Property(t => t.Name).HasMaxLength(200);
    this.Property(t => t.Url).HasMaxLength(200);
    // Table & Column Mappings
    this.ToTable("Blogs");
    this.Property(t => t.BlogId).HasColumnName("BlogId");
    this.Property(t => t.Name).HasColumnName("Name");
    this.Property(t => t.Url).HasColumnName("Url");
  }
}
```

Convention examples (1)

Klucz główny Primary Key - name like `ID` or `ClassName+ID` then first such field becomes primary key. If its type is numeric or GUID then it has IDENTITY in `SqlServer`

Klucz obcy Foreign Key - we provide navigation and the key

```
public class Department{
    public int DepartmentID { get; set; }
    public virtual ICollection<Course> Courses { get; set; }
}
public class Course{
    public int CourseID { get; set; }
    public int DepartmentID { get; set; }
    public virtual Department Department { get; set; }
}
```

Przykłady konwencji (2)

- Za pomocą FluentApi można konwencje usuwać lub definiować własne

```
public class SchoolEntities : DbContext {
    . . .
  protected override void OnModelCreating(DbModelBuilder modelBuilder)  {
    // Configure Code First to ignore PluralizingTableName convention
    // If you keep this convention, the generated tables
    // will have pluralized names.
    modelBuilder.Conventions.Remove<PluralizingTableNameConvention>();
  }
}
```

Adnotation

- Instead of `FluentApi` we could use adnotations

```
public class Blog
{
    [Key]
    public int PrimaryTrackingKey { get; set; }
    [Required]
    public string Title { get; set; }
    [MaxLength(10),MinLength(5)]
    public string BloggerName { get; set;}
    [NotMapped]
    public string BlogCode {
      get {
        return
          Title.Substring(0,1)+":"+BloggerName.Substring(0, 1);
      }
    }
    public virtual ICollection<Post> Posts { get; set; }
}
```

obiekt DbContext (1)

- we derive from DbContext and we declare one or more DbSet<obiekt>

```
namespace StudentsList.Model {
  public class StorageContext : DbContext {
    public DbSet<Student> Students { get; set; }
    public DbSet<Group> Groups { get; set; }
  }
}
```

- In configuration file we could put DataSource declaration

```
<connectionStrings>
    <add name="StudentsList.Model.StorageContext"
      connectionString="Data Source=(localdb)\v11.0;
        AttachDbFilename=C:\tmp\StudentsList.Model.StorageContext.mdf;
        Initial Catalog=Model.StorageContext;Integrated Security=True"
      providerName="System.Data.SqlClient" />
</connectionStrings>
```

object DbContext (2)

- manages group of objects
- remembers what was changed
- we try to create as short as possible

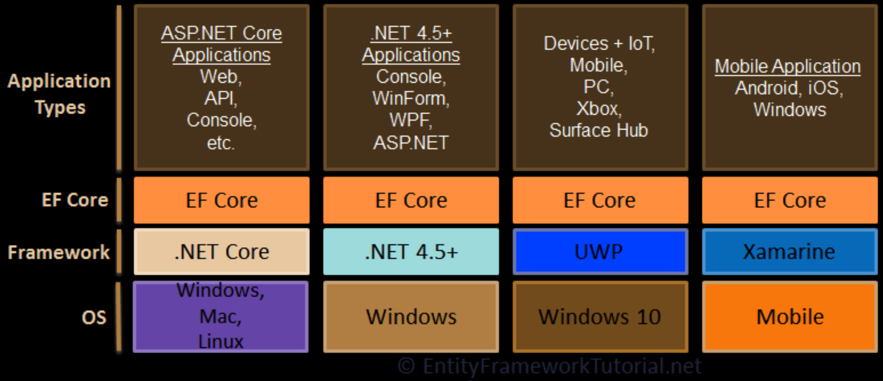
```
void createStudent(string firstName, string lastName,
                       string indexNo, int groupId) {
  using (var db = new StorageContext()) {
    var group = db.Groups.Find(groupId);
    var student = new Student { FirstName = firstName,
           LastName=lastName, IndexNo = indexNo, Group=group };
    db.Students.Add(student);
    db.SaveChanges();
  }
}
```

object DbContext

```
void updateStudent(Student st) {
  using (var db = new StorageContext()) {
    var original = db.Students.Find(st.StudentId);
    if (original != null) {
      original.FirstName = st.FirstName;
      original.LastName = st.LastName;
      db.SaveChanges();
    }
  }
}
void deleteStudent(Student st) {
  using (var db = new StorageContext()) {
    var original = db.Students.Find(st.StudentId);
    if (original != null) {
      db.Students.Remove(original);
      db.SaveChanges();
    }
  }
}
```

Entity Framework Core

cross-platform ORM



Application Types	ASP.NET Core Applications Web, API, Console, etc.	.NET 4.5+ Applications Console, WinForm, WPF, ASP.NET	Devices + IoT, Mobile, PC, Xbox, Surface Hub	Mobile Application Android, iOS, Windows
EF Core	EF Core	EF Core	EF Core	EF Core
Framework	.NET Core	.NET 4.5+	UWP	Xamarine
OS	Windows, Mac, Linux	Windows	Windows 10	Mobile

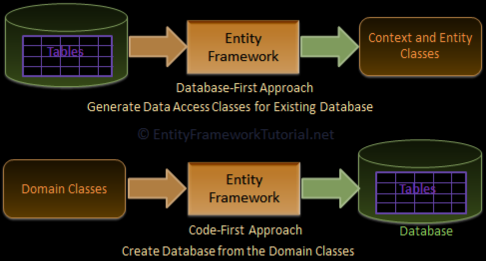
Versions of Entity Framework Core

version	Release date
EF Core 5.0	January 2021
EF Core 3.1	December 2019
EF Core 3.0	September 2019
EF Core 2.0	August 2017
EF Core 1.0	June 2016

```
var orders =
  from o in context.Orders
  where o.Status == OrderStatus.Pending
  select o;

await foreach(var o in orders.AsAsyncEnumerable()) {
  Process(o);
}
```

What is common with EF6


Tables
Entity Framework
Context and Entity Classes
Database-First Approach
Generate Data Access Classes for Existing Database
© EntityFrameworkTutorial.net
Domain Classes
Entity Framework
Tables
Database
Code-First Approach
Create Database from the Domain Classes

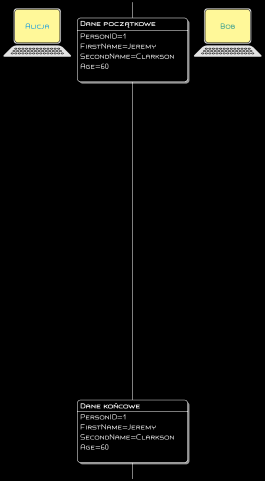
- DbContext & DbSet
- Data Model
- Querying using Linq-to-Entities
- Change Tracking
- SaveChanges
- Migrations

Suported databases

Database	NuGet Package
SQL Server	Microsoft.EntityFrameworkCore.SqlServer
MySQL	MySql.Data.EntityFrameworkCore
PostgreSQL	Npgsql.EntityFrameworkCore.PostgreSQL
SQLite	Microsoft.EntityFrameworkCore.SQLite
SQL Compact	EntityFrameworkCore.SqlServerCompact40
In-memory	Microsoft.EntityFrameworkCore.InMemory

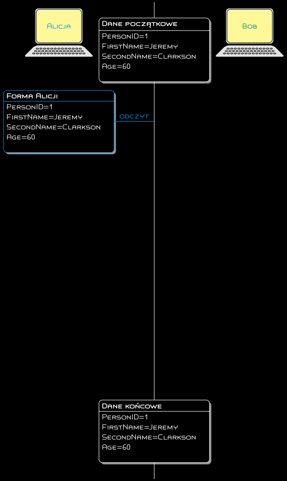
concurrency

- Alicja i Bob edytują



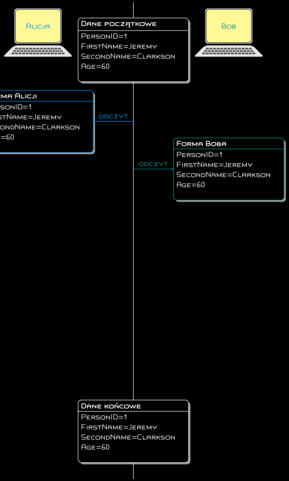
concurrency

- Alicja i Bob edytują
- Alicja czyta



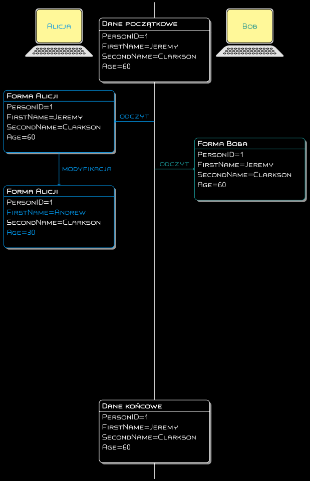
concurrency

- Alicja i Bob edytują
- Alicja czyta
- Bob czyta



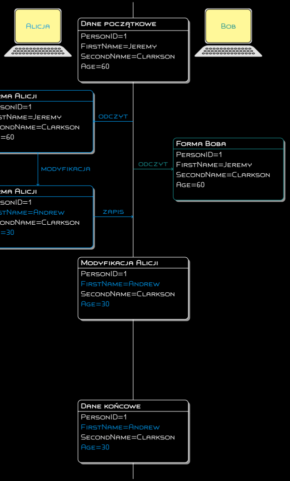
concurrency

- Alicja i Bob edytują
- Alicja czyta
- Bob czyta
- Alicja modyfikuje



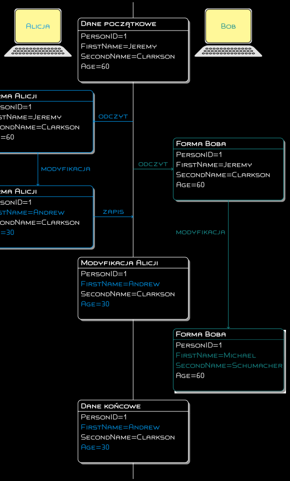
concurrency

- Alicja i Bob edytują
- Alicja czyta
- Bob czyta
- Alicja modyfikuje
- Alicja zapisuje



concurrency

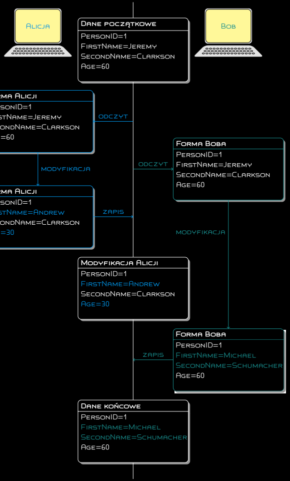
- Alicja i Bob edytują
- Alicja czyta
- Bob czyta
- Alicja modyfikuje
- Alicja zapisuje
- Bob modyfikuje



concurrency

- Alicja i Bob edytują
- Alicja czyta
- Bob czyta
- Alicja modyfikuje
- Alicja zapisuje
- Bob modyfikuje
- Bob zapisuje
    - cały rekord

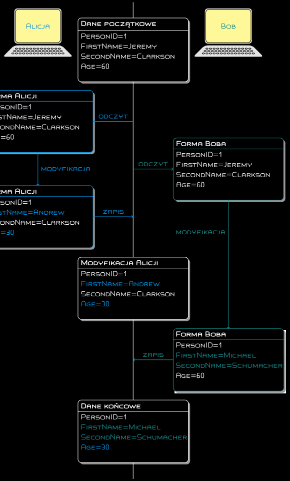
```
UPDATE Persons SET
  FirstName=?, SecondName=?,
  Age = ?
WHERE PersonID = ?
```



concurrency

- Alicja i Bob edytują
- Alicja czyta
- Bob czyta
- Alicja modyfikuje
- Alicja zapisuje
- Bob modyfikuje
- Bob zapisuje
    - cały rekord
    - tylko zmienione pola

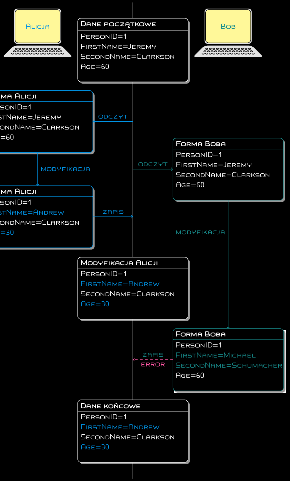
```
UPDATE Persons SET
  FirstName=?,
  SecondName=?
WHERE PersonID = ?
```



concurrency

- Alicja i Bob edytują
- Alicja czyta
- Bob czyta
- Alicja modyfikuje
- Alicja zapisuje
- Bob modyfikuje
- Bob zapisuje
    - cały rekord
    - tylko zmienione pola
    - sprawdzając wszystkie pola

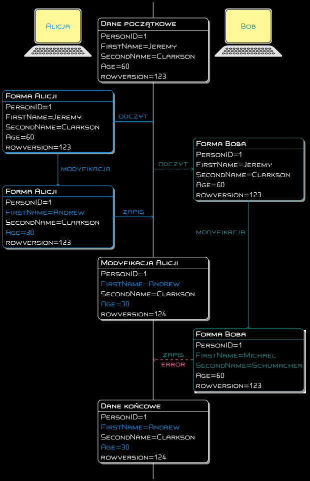
```
UPDATE Persons SET
  FirstName=?, SecondName=?
WHERE PersonID = ? AND FirstName = ?
  AND SecondName = ? AND Age = ?
```



concurrency

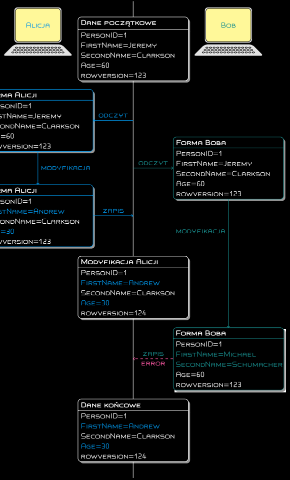
- Alicja i Bob edytują
- Alicja czyta
- Bob czyta
- Alicja modyfikuje
- Alicja zapisuje
- Bob modyfikuje
- Bob zapisuje
    - cały rekord
    - tylko zmienione pola
    - sprawdzając wszystkie pola
    - tylko pole RowVersion

```sql
UPDATE Persons SET
  FirstName=?, SecondName=?
WHERE PersonID = ? AND RowVersion = ?
```



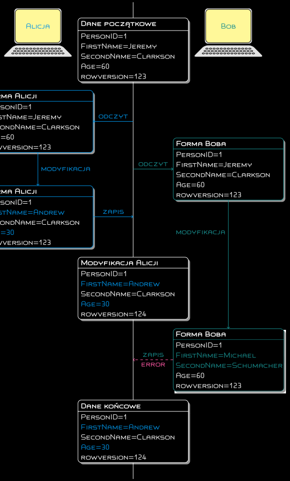
concurrency

- Alicja i Bob edytują
- Alicja czyta
- Bob czyta
- Alicja modyfikuje
- Alicja zapisuje
- Bob modyfikuje
- Bob zapisuje
- musimy przechować stare wartości pól do porównania



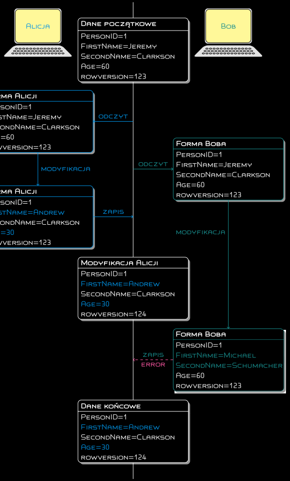
concurrency

- Alicja i Bob edytują
- Alicja czyta
- Bob czyta
- Alicja modyfikuje
- Alicja zapisuje
- Bob modyfikuje
- Bob zapisuje
- musimy przechować stare wartości pól do porównania
    - przy modyfikacji pól zmienionych



concurrency

- Alicja i Bob edytują
- Alicja czyta
- Bob czyta
- Alicja modyfikuje
- Alicja zapisuje
- Bob modyfikuje
- Bob zapisuje
- musimy przechować stare wartości pól do porównania
    - przy modyfikacji pól zmienionych
    - przy sprawdzeniu równoległości



Conflicts (1)

```
public class BankAccount {
    public int Id { get; set; }
    public decimal Balance { get; set; }
    public string FirstName { get; set; }
    public string LastName { get; set; }

    public void Credit(decimal amount)  {
        Console.WriteLine($"Balance before credit:{Balance,5}");
        Console.WriteLine($"Amount to add        :{amount,5}");
        Balance += amount;
        Console.WriteLine($"Balance after credit :{Balance,5}");
    }

    public decimal Debit(decimal amount) {
        if (Balance >= amount) {
            Console.WriteLine($"Balance before debit :{Balance,5}");
            Console.WriteLine($"Amount to remove     :{amount,5}");
            Balance -= amount;
            Console.WriteLine($"Balance after debit  :{Balance,5}");
            return amount;
        }
        return 0;
    }
}
```

Conflicts (2)

```
using (var dbContext = new MyDbContext()) {
    var account = await dbContext.BankAccounts.FindAsync(1);
    account.Credit(100);
    await dbContext.SaveChangesAsync();
}
```

Concurrency Detection via ConcurrencyToken

```
internal class BankAccountEntityTypeConfigurationSqlite : IEntityTypeConfiguration<BankAccount>
{
    public void Configure(EntityTypeBuilder<BankAccount> builder)
    {
        builder.ToTable("BankAccounts");
        builder.HasKey(x => x.Id);
        builder.Property(x => x.Id).HasColumnName("Id").ValueGeneratedOnAdd();
        builder.Property(x => x.Balance).HasColumnName("Balance").HasConversion<double>()
               .IsConcurrencyToken();
    }
}
```

every update uses it as a part of the key:

```
nfo: Microsoft.EntityFrameworkCore.Database.Command[20101]
      Executed DbCommand (1ms) [Parameters=[ @p1='?', @p0='?', @p2='?'], CommandType='Text',
      ↪   CommandTimeout='30']
      UPDATE "BankAccounts" SET "Balance" = @p0
      WHERE "Id" = @p1 AND "Balance" = @p2;
      SELECT changes();
```

Concurrency Detection via ConcurrencyToken (3)

```
using (var dbContext = new MyDbContext())
{
    var account = await dbContext.BankAccounts.FindAsync(1);
    account.Credit(100);
    try
    {
        await dbContext.SaveChangesAsync();  // Attempt to save changes to the database
    }
    catch (DbUpdateConcurrencyException e)
    {
        Console.WriteLine(e.Message);   // Handle the exception here.
    }
}
```

Concurrency Detection via RowVersion (4)

```
public class BankAccount {
    public int Id { get; set; }
    public decimal Balance { get; set; }
    public byte[] Timestamp { get; set; }   // add a new property
    // ...
}
```

```
internal class BankAccountEntityTypeConfigurationSqlite : IEntityTypeConfiguration<BankAccount>
{
    public void Configure(EntityTypeBuilder<BankAccount> builder) {
        builder.ToTable("BankAccounts");
        builder.HasKey(x => x.Id);
        builder.Property(x => x.Id).HasColumnName("Id").ValueGeneratedOnAdd();
        builder.Property(x => x.Balance).HasColumnName("Balance").HasConversion<double>();
        builder.Property(x => x.Timestamp).HasColumnName("Timestamp")
            .HasColumnType("BLOB")
            .IsRowVersion();
    }
}
```

LINQ - solving conflicts with Merge (5)

Fields:	FirstName	LastName	Balance
U1 and U2 read	Jan	Kowalski	10000
U2 write		Nowak	20000
U1 write	Alfred		30000
Result	Alfred	Nowak	30000

```
try {
    db.SubmitChanges(ConflictMode.ContinueOnConflict);
}
catch (ChangeConflictException e) {
    foreach (ObjectChangeConflict occ in db.ChangeConflicts) {
        occ.Resolve(RefreshMode.KeepChanges);
        // fields refreshed from database
    }
}
// now should be the success
db.SubmitChanges(ConflictMode.FailOnFirstConflict);
```

Merge will success only if different fields will be modified

Solving conflicts by overwrite (6)

Fields	FirstName	LastName	Balance
U1 and U2 read	Jan	Kowalski	10000
U2 write		Nowak	20000
U1 write	Alfred		30000
Result	Alfred	Kowalski	30000

```
try {
    db.SubmitChanges(ConflictMode.ContinueOnConflict);
}
catch (ChangeConflictException e) {
    foreach (ObjectChangeConflict occ in db.ChangeConflicts)  {
        occ.Resolve(RefreshMode.KeepCurrentValues);
        // we ignore database
    }
}
// now should be the success
db.SubmitChanges(ConflictMode.FailOnFirstConflict);
```

Solving conflicts by preserve database (7)

Fields	FirstName	LastName	Balance
U1 i U2	Jan	Kowalski	10000
U2 write		Nowak	20000
U1 write	Alfred		30000
Result	Jan	Nowak	20000

```
try {
    db.SubmitChanges(ConflictMode.ContinueOnConflict);
}
catch (ChangeConflictException e) {
    foreach (ObjectChangeConflict occ in db.ChangeConflicts)  {
        occ.Resolve(RefreshMode.OverwriteCurrentValues);
        // Ignore our changes
    }
}
// now should be the success
db.SubmitChanges(ConflictMode.FailOnFirstConflict);
```

Checking modified fields (8)

```
Checking whether a property is marked as modified

using (var context = new BloggingContext()) {
    var blog = context.Blogs.Find(1);
    var nameIsModified1 = context.Entry(blog).Property(u => u.Name).IsModified;
    // Use a string for the property name
    var nameIsModified2 = context.Entry(blog).Property("Name").IsModified;
}

Marking a property as modified

using (var context = new BloggingContext()) {
    var blog = context.Blogs.Find(1);
    context.Entry(blog).Property(u => u.Name).IsModified = true;

    context.Entry(blog).Entry(blog).Property(p => p.Name).IsModified = true;
    // Use a string for the property name
    context.Entry(blog).Property("Name").IsModified = true;
}
```

Conflicts reading values (9)

```
using (var context = new BloggingContext()){
    var blog = context.Blogs.Find(1);

    blog.Name = "My Cool Blog";    // Make a modification to Name in the tracked entity

    // Make a modification to Name in the database
    context.Database.SqlCommand("update dbo.Blogs set Name = 'My Boring Blog' where Id = 1");

    // Print out current, original, and database values
    Console.WriteLine("Current values:");
    PrintValues(context.Entry(blog).CurrentValues);
    Console.WriteLine("\nOriginal values:");
    PrintValues(context.Entry(blog).OriginalValues);
    Console.WriteLine("\nDatabase values:");
    PrintValues(context.Entry(blog).GetDatabaseValues());
}

public static void PrintValues(DbPropertyValues values) {
    foreach (var propertyName in values.PropertyNames) {
        Console.WriteLine("Property {0} has value {1}",
                              propertyName, values[propertyName]);
    }
}
```

Conflicts setting original values (10)

```
public class BlogDto {
    public int Id { get; set; }
    public string Name { get; set; }
}
using (var context = new BloggingContext()) {
    var blog = context.Blogs.Find(1);
    var coolBlog = new Blog { Id = 1, Name = "My Cool Blog" };
    var boringBlog = new BlogDto { Id = 1, Name = "My Boring Blog" };

    // Change the current and original values by copying the values from other objects
    var entry = context.Entry(blog);
    entry.CurrentValues.SetValues(coolBlog);
    entry.OriginalValues.SetValues(boringBlog);

    // Print out current and original values
    Console.WriteLine("Current values:");
    PrintValues(entry.CurrentValues);

    Console.WriteLine("\nOriginal values:");
    PrintValues(entry.OriginalValues);
}
```

Language Integrated Query -> LINQ (1)

Do yoy remember Embedded Queries?

- First every provider created his own libraries for C language
- For readibility embedded sql was created - a C preprocesor translates plain SQL into C library function calls
- ...
- And then Microsoft created LINQ, which looks like ... embedded sql

```
var studenci = from s in Data.students
               where s.FirstName == "Adam"
               select new { s.FirstName, s.LastName } ;
foreach (var s in studenci)
   Console.WriteLine(s);
```

Language Integrated Query -> LINQ (2)

```
public void f2m() {
    IEnumerable<Student> stu =
                Data.students.Where(s => s.FirstName == "Adam");
        // List of Student objects
    foreach (var v in stu)
                Console.WriteLine(v);

    // string list
    var stu1 = Data.students.Where(s => s.FirstName == "Adam")
                .Select(s => s.LastName);
    foreach (var v in stu1)
        Console.WriteLine(v);

    // list of objects of anonymous class
    var stu2 = Data.students.Where(s => s.FirstName == "Adam")
                .Select(s => new { s.FirstName, s.LastName });
    foreach (var v in stu2)
        Console.WriteLine(v);
```

Language Integrated Query -> LINQ (3)

```
LINQ syntax
    var stu3 = from s in Data.students
                where s.FirstName == "Adam"
                select new { s.FirstName, s.LastName } ;
    foreach (var v in stu3)
        Console.WriteLine(v);

    var stu4 = from s in Data.students
                 where s.FirstName == "Adam"
                 select s.LastName ;
    foreach (var v in stu4)
        Console.WriteLine(v);

    // result may be a source for the next query
    var stu5 = (from s in Data.students
                where s.FirstName == "Adam"
                select s).Select(s=>s.LastName);
    foreach (var v in stu5)
        Console.WriteLine(v);
    Console.ReadKey();
}
```

Language INtegrated Query

Syntax independent from source of the data
Which results in query language being a part of C#

We can ask different source of data

- LINQ to Object
- LINQ to Dataset
- LINQ to SQL
- LINQ to XML
- LINQ to Entities

Immediate operators (1)

First

```
public static T First<T>(this IEnumerable<T> source);
public static T First<T>(this IEnumerable<T> source,Func<T, bool> predicate);
```

FirstOrDefault

```
public static T FirstOrDefault<T>(this IEnumerable<T> source);
public static T FirstOrDefault<T>(this IEnumerable<T> source,Func<T, bool> predicate);
```

Last

```
public static T Last<T>(this IEnumerable<T> source);
public static T Last<T>(this IEnumerable<T> source,Func<T, bool> predicate);
```

LastOrDefault

```
public static T LastOrDefault<T>(this IEnumerable<T> source);
public static T LastOrDefault<T>(this IEnumerable<T> source,Func<T, bool> predicate);
```

Immediate operators (2)

Single

```
public static T Single<T>(this IEnumerable<T> source);
public static T Single<T>(this IEnumerable<T> source,Func<T, bool> predicate);
```

ElementAt

```
public static T ElementAt<T>(this IEnumerable<T> source,int index);
```

ElementAtOrDefault

```
public static T ElementAtOrDefault<T>(this IEnumerable<T> source,int index);
```

Immediate predictors (3)

Any

```
public static bool Any<T>(this IEnumerable<T> source);
public static bool Any<T>(this IEnumerable<T> source,Func<T, bool> predicate);
```

All

```
public static bool All<T>(this IEnumerable<T> source,Func<T, bool> predicate);
```

Contains

```
public static bool Contains<T>(this IEnumerable<T> source,T value);
public static bool Contains<T>(this IEnumerable<T> source,T value,IEqualityComparer<T> comparer);
```

Immediate predictors (4)

Count

```
public static int Count<T>(this IEnumerable<T> source);
public static int Count<T>(this IEnumerable<T> source,Func<T, bool> predicate);
```

Sum

```
public static Numeric Sum(this IEnumerable<Numeric> source);
public static Numeric Sum<T>(this IEnumerable<T> source,Func<T, Numeric> selector);
```

Min

```
public static Numeric Min(this IEnumerable<Numeric> source);
public static T Min<T>(this IEnumerable<T> source);
public static Numeric Min<T>(this IEnumerable<T> source,Func<T, Numeric> selector);
public static S Min<T, S>(this IEnumerable<T> source,Func<T, S> selector);
```

- Max
- Average
- Agregate

Whrere (restriction), Select(projection)

```
public static IEnumerable<T> Where<T>(this IEnumerable<T> source,
                                      Func<T, bool> predicate);
public static IEnumerable<T> Where<T>(this IEnumerable<T> source,
                                      Func<T, int, bool> predicate);
// int is a number (0, 1, ...) of the record->so we can have a parity condition
var stu4a = from s in Data.students
            where s.FirstName == "Adam"
            select s;
var stu4b = Data.students.Where((Student c)=>c.FirstName == "Adam");//.Select(s=>s);
IEnumerable<Student> stu4c = Data.students.Where(delegate(Student c) {
  return c.FirstName == "Adam";
  });
var stu4d = from s in Data.students
            where s.FirstName == "Adam"
            select new {s.FirstName, s.LastName};
var stu4e = Data.students.Where((Student c) => c.FirstName == "Adam")
            .Select(s=>new{s.FirstName,s.LastName});
var stu4f = Data.students.Where(delegate(Student c){ return c.FirstName == "Adam";})
            .Select(delegate(Student c){ return new{s.FirstName,s.LastName});
foreach (var v in stu4c) Console.WriteLine(v);
```

When we return anonymous objects we must return var, because it is not possible to write `IEnumerable<>`

Operatory - OrderBy ThenBy

```
public static IOrderedEnumerable<T> OrderBy<T, K>(
    this IEnumerable<T> source,Func<T, K> keySelector
    ) where K : IComparable<K>;

public static IOrderedEnumerable<T> OrderBy<T, K>(
    this IEnumerable<T> source,Func<T, K> keySelector,
    IComparer<K> comparer);

var u1 = from stu in Data.students
            orderby stu.FirstName, stu.LastName
            select stu;

// jest rownowazne
var u2 = Data.students.OrderBy(a=>a.FirstName)
                      .ThenBy(a=>a.LastName)
                      .Select(a=>a);
```

we have `OrderByDescending` and `ThenByDescending`

Operators - join

```
public static IEnumerable<V> Join<T, U, K, V>(
this IEnumerable<T> outer,IEnumerable<U> inner,
Func<T, K> outerKeySelector,Func<U, K> innerKeySelector,
Func<T, U, V> resultSelector);

var x = from s in Data.realisations
            join su in Data.subjects on s.SubjectId equals su.Id
            select new { s.Id, s.SemesterId, s.SubjectId, su.Name };
```

Operators - selectmany

```
var sentences = new List<string> {"Bob is quite excited.",
                                  "Jim is very upset."};
var words1 = sentences.SelectMany(w => w.TrimEnd('.')
                      .Split(' ')).ToList();
var words2 = from s in sentences
            from w in s.TrimEnd('.').Split(' ')
            select w;
```

```
var y = from rel in Data.realisations
        from gra in Data.grades
        where rel.Id == gra.RealisationId
        select new { rel.Id, GradeId=gra.Id };

var z = from rel in Data.realisations
        .SelectMany(e => Data.grades
                   .Where(g => g.RealisationId == e.Id)
                   .Select(eo => new { e.Id, GradeId = eo.Id })
                   )
        select new { rel.Id, rel.GradeId } ;
```

Operators - DefaultIfEmpty

Return sequence containing default value if sequence is empty

```
public static IEnumerable<T> DefaultIfEmpty<T>(
              this IEnumerable<T> source);
public static IEnumerable<T> DefaultIfEmpty<T>(
              this IEnumerable<T> source,T defaultValue);
```

For empty sequence return default value

```
var x = (from s in Data.students
    where s.LastName == "Di Caprio"
    select s.LastName).DefaultIfEmpty("brak");
```

Operatory - DefaultIfEmpty

Left outer join with multielement key

```
var z2 = from gr in Data.grades
    join rel in Data.realisations on gr.RealisationId equals rel.Id
    join reg in Data.registrations on rel.Id equals reg.RealisationId
    join stu in Data.students on reg.StudentId equals stu.Id
    join sub in Data.subjects on rel.SubjectId equals sub.Id
    join sem in Data.semesters on rel.SemesterId equals sem.Id
    orderby reg.Id
    join gv in Data.gradeValues on
       new { RegistrationId=reg.Id, GroupId = gr.Id }
    equals
    new { RegistrationId=gv.RegistrationId,GroupId=gv.GradeId } into g
      from o in g.DefaultIfEmpty()
      select new { StudentId = stu.Id,
         RealisationId = rel.Id,  RegistrationId = reg.Id,
         GradeId = gr.Id, SubjectId = sub.Id,
         SemesterId = sem.Id, GradeValueExist = o != null ? 1 : 0,
         Value = o!= null?o.Value:0, FirstName = stu.FirstName,
         LastName=stu.LastName, Semester=sem.Name,Subject = sub.Name
        };
```

Operators: group and group into

```
var z = from s in Data.gradeValues
        group s by s.RegistrationId into g
        select new { g.Key, Suma=g.Sum<GradeValue>(a=>a.Value) };
foreach (var v in z)
  Console.WriteLine(v);
var z1 = from g in
         from s in Data.gradeValues
         group s by s.RegistrationId
         select new { g.Key, Suma = g.Sum<GradeValue>(a => a.Value) };
foreach (var v in z1) Console.WriteLine(v);
```

LINQ to DataSet

data access

```
        DataSet ds = LoadDataSetUsingDataAdapter();
        DataTable orders = ds.Tables["Orders"];
        DataTable orderDetails = ds.Tables["OrderDetails"];
        var query =        from    o in orders.AsEnumerable()
            where   o.Field<DateTime>( "OrderDate" ).Year >= 1998
            orderby o.Field<DateTime>( "OrderDate" ) descending
            select  o;
```

LINQ to DataSet-joining two entities

```
DataSet ds = LoadDataSetUsingDataAdapter();
DataTable orders = ds.Tables["Orders"];
DataTable orderDetails = ds.Tables["OrderDetails"];
var query = from    o in orders.AsEnumerable()
    join    od in orderDetails.AsEnumerable()
            on o.Field<int>( "OrderID" ) equals od.Field<int>( "OrderID" )
            into orderLines
    where   o.Field<DateTime>( "OrderDate" ).Year >= 1998
    orderby o.Field<DateTime>( "OrderDate" ) descending
    select new { OrderID = o.Field<int>( "OrderID" ),
                 OrderDate = o.Field<DateTime>( "OrderDate" ),
                 Amount = orderLines.Sum(
                   od => od.Field<decimal>( "UnitPrice" )
                          * od.Field<short>( "Quantity" ) ) };
```

Przykłady

Podsumowując operatory

```
        foreach( var student in query ) {
            Console.WriteLine( student.FirstName );
        }
        ---------------
        IEnumerator<string> enumerator = query.GetEnumerator();
                while (enumerator.MoveNext()) {
                    Console.WriteLine( enumerator.Current );
        }
```

możemy zapisać w liście

```
                var studentNames = query.ToList();
```

pobranie jednego rekordu - Single

```
var student = qStudents.Single( s => S.ID == 1 );
Console.WriteLine( "{0} {1}", student.ID, student.FirstName );
```

MySQL Database installation

- package installation on Ubuntu 20.04

```
sudo apt install mysql-server
```

MySQL Database installation

- package installation on Ubuntu 20.04
- `mysql_secure_installation` that can perform several security-related operations.

```
sudo mysql_secure_installation
```

MySQL Database installation

- package installation on Ubuntu 20.04
- `mysql_secure_installation` that can perform several security-related operations.
    - VALIDATE PASSWORD PLUGIN installation - enforces password rules

```
sudo mysql_secure_installation
```

MySQL Database installation

- package installation on Ubuntu 20.04
- mysql_secure_installation that can perform several security-related operations.
  - VALIDATE PASSWORD PLUGIN installation - enforces password rules
  - set a password for the MySQL root user.

```
sudo mysql_secure_installation
```

MySQL Database installation

- package installation on Ubuntu 20.04
- `mysql_secure_installation` that can perform several security-related operations.
    - VALIDATE PASSWORD PLUGIN installation - enforces password rules
    - set a password for the MySQL root user.
    - Once you do that the script will also ask you to remove the anonymous user

```
sudo mysql_secure_installation
```

MySQL Database installation

- package installation on Ubuntu 20.04
- `mysql_secure_installation` that can perform several security-related operations.
    - VALIDATE PASSWORD PLUGIN installation - enforces password rules
    - set a password for the MySQL root user.
    - Once you do that the script will also ask you to remove the anonymous user
    - restrict root user access to the local machine

```
sudo mysql_secure_installation
```

MySQL Database installation

- package installation on Ubuntu 20.04
- `mysql_secure_installation` that can perform several security-related operations.
  - VALIDATE PASSWORD PLUGIN installation - enforces password rules
  - set a password for the MySQL root user.
  - Once you do that the script will also ask you to remove the anonymous user
  - restrict root user access to the local machine
  - remove the test database.

```
sudo mysql_secure_installation
```

MySQL Database installation

- package installation on Ubuntu 20.04
- `mysql_secure_installation` that can perform several security-related operations.
- change a root authentication method
    - connection to database

```
sudo mysql
```

- In Ubuntu systems running MySQL 5.7 (and later), the root user is authenticated by the auth_socket plugin by default.
- The auth_socket plugin authenticates users that connect from the localhost through the Unix socket file.
- This means that you can't authenticate as root by providing a password.

MySQL Database installation

- package installation on Ubuntu 20.04
- `mysql_secure_installation` that can perform several security-related operations.
- change a root authentication method
  - connection to database
  - change the authentication method from auth_socket to mysql_native_password

```
ALTER USER 'root'@'localhost' IDENTIFIED WITH mysql_native_password BY 'very_strong_password';
FLUSH PRIVILEGES;
```

MySQL Database installation

- package installation on Ubuntu 20.04
- `mysql_secure_installation` that can perform several security-related operations.
- change a root authentication method
- create a new administrative user with access to all databases

```
CREATE USER 'administrator'@'localhost' IDENTIFIED BY '<very strong password>';
GRANT ALL ON *.* TO 'administrator'@'localhost';
```

MySQL Database installation

- package installation on Ubuntu 20.04
- `mysql_secure_installation` that can perform several security-related operations.
- change a root authentication method
- create a new administrative user with access to all databases
- phpmyadmin installation (apache)
    - gdy nie uruchomiliśmy wtyczki Validate Password

```
sudo apt install phpmyadmin
```

MySQL Database installation

- package installation on Ubuntu 20.04
- `mysql_secure_installation` that can perform several security-related operations.
- change a root authentication method
- create a new administrative user with access to all databases
- phpmyadmin installation (apache)
    - gdy nie uruchomiliśmy wtyczki Validate Password
    - when the Validate Password plugin has not been launched, it must be disabled

```
sudo mysql      #  lub: mysql -u root -p
mysql>UNINSTALL COMPONENT "file://component_validate_password";
mysql>exit
sudo apt install phpmyadmin
sudo mysql      #  lub: mysql -u root -p
mysql>INSTALL COMPONENT "file://component_validate_password";
mysql>exit
```

MySQL Database creation

```
CREATE USER 'EGUI20Z'@'localhost' IDENTIFIED WITH caching_sha2_password BY '***';
GRANT ALL PRIVILEGES ON *.* TO 'EGUI20Z'@'localhost' WITH GRANT OPTION;
ALTER USER 'EGUI20Z'@'localhost' REQUIRE NONE
  WITH MAX_QUERIES_PER_HOUR 0
  MAX_CONNECTIONS_PER_HOUR 0
  MAX_UPDATES_PER_HOUR 0
  MAX_USER_CONNECTIONS 0;
CREATE DATABASE IF NOT EXISTS `EGUI20Z`;
GRANT ALL PRIVILEGES ON `EGUI20Z`.* TO 'EGUI20Z'@'localhost';
```

Create ef console project

```
dotnet tool install --global dotnet-ef --version 3.1.9
mkdir mysql
cd mysql
dotnet new console
dotnet add package Microsoft.EntityFrameworkCore.Design --version 3.1.9
dotnet add package Microsoft.EntityFrameworkCore --version 3.1.9
dotnet add package MySql.Data.EntityFrameworkCore --version 8.0.22
# to dodajemy pliki zrodlowe
dotnet build
dotnet-ef migrations add initial
```

Program.cs

```
using System;
using System.Text;
using Microsoft.EntityFrameworkCore;
using System.Linq;
namespace mysql {
  class Program {
    static void Main(string[] args) {
      removeAllData();
      insertData();
      modifyData();
      printData();
    }
    private static void removeAllData() {
      // slow but always ok
      using (var context = new LibraryContext()) {
        // Creates the database if not exists
        context.Database.EnsureCreated();
        context.Book.RemoveRange(context.Book);
        context.Publisher.RemoveRange(context.Publisher);
        context.SaveChanges();
      }
      // faster but problems with Foreign keys
      using (var context = new LibraryContext()) {
        // Creates the database if not exists
        context.Database.EnsureCreated();
        context.Database.ExecuteSqlRaw("TRUNCATE TABLE Book");
        context.Database.ExecuteSqlRaw("DELETE FROM Publisher");
      }
    }
```

Program.cs

```
    private static void modifyData() {
      using (var context = new LibraryContext()) {
        // modify first book
        var books = from b in context.Book
                    where b.Title == "The Lord of the Rings"
```

Program.cs

```
            select b;
      var book = books.Single();  // only one expected
      book.Language = "Polish";
      book.Title = "Wladca Pierscieni";

      // remove second book
      books = context.Book.Where(b=>b.ISBN=="978-0547247762");
      book = books.Single();  // only one expected
      context.Book.Remove(book);
      context.SaveChanges();
     }
    }
    private static void deleteData() {
     using (var context = new LibraryContext()) {
      var books = from b in context.Book
            where b.ISBN == "978-0547247762"
            select b;
      var book = books.Single();  // only one expected
      context.Book.Remove(book);
      context.SaveChanges();
     }
    }
```

Program.cs

```
    private static void insertData() {
      using (var context = new LibraryContext()) {
        // Creates the database if not exists
        context.Database.EnsureCreated();
        // Adds a publisher
        var publisher = new Publisher {
          Name = "Mariner Books"
        };
        context.Publisher.Add(publisher);
        // Adds some books
        context.Book.Add(new Book {
          ISBN = "978-0544003415",
          Title = "The Lord of the Rings",
```

Program.cs

```
          Author = "J.R.R. Tolkien",
          Language = "English",
          Pages = 1216,
          Publisher = publisher
        });
        context.Book.Add(new Book {
          ISBN = "978-0547247762",
          Title = "The Sealed Letter",
          Author = "Emma Donoghue",
          Language = "English",
          Pages = 416,
          Publisher = publisher
        });
        context.SaveChanges();  // Saves changes
      }
    }
```

Program.cs

```
        private static void printData() {
          // Gets and prints all books in database
          using (var context = new LibraryContext()) {
            // loads also items from Publisher -> otherwise they will be NULL
            var books = context.Book.Include(p => p.Publisher);
            foreach (var book in books) {
              var data = new StringBuilder();
              data.AppendLine($"ISBN: {book.ISBN}");
              data.AppendLine($"Title: {book.Title}");
              data.AppendLine($"Publisher: {book.Publisher.Name}");
              Console.WriteLine(data.ToString());
            }
          }
        }
      }
    }
    /* program drukuje:
    ISBN: 978-0544003415
    Title: Wladca Pierscieni
```

Program.cs

```
Publisher: Mariner Books
*/
```

LibraryModel.cs

```
using System.Collections.Generic;
namespace mysql {

  public class Book {
    public string ISBN { get; set; }
    public string Title { get; set; }
    public string Author { get; set; }
    public string Language { get; set; }
    public int Pages { get; set; }
    public virtual Publisher Publisher { get; set; }
  }

  public class Publisher {
    public int ID { get; set; }
    public string Name { get; set; }
    public virtual ICollection<Book> Books { get; set; }
  }
}
```

LibraryContext.cs

```
using Microsoft.EntityFrameworkCore;
namespace mysql {
  public class LibraryContext : DbContext {
    public DbSet<Book> Book { get; set; }
    public DbSet<Publisher> Publisher { get; set; }
    protected override void OnConfiguring(DbContextOptionsBuilder optionsBuilder) {
      optionsBuilder.UseMySQL("server=localhost;database=NTR20Z;user=NTR20Z;password=***");
    }
    protected override void OnModelCreating(ModelBuilder modelBuilder) {
      base.OnModelCreating(modelBuilder);
      modelBuilder.Entity<Publisher>(entity => {
        entity.HasKey(e => e.ID);
        entity.Property(e => e.Name).IsRequired();
      });
      modelBuilder.Entity<Book>(entity => {
        entity.HasKey(e => e.ISBN);
        entity.Property(e => e.Title).IsRequired();
        entity.HasOne(d => d.Publisher).WithMany(p => p.Books);
      });
    }
  }
}
```