***Wojskowa Akademia Techniczna***

***im. Jarosława Dąbrowskiego***



**Wydział Cybernetyki, kierunek informatyka - inżynieria systemów**

Realizacja zadania laboratoryjnego w ramach przedmiotu:

*Systemy Baz Danych*

Temat laboratorium:

***Obiektowe Bazy Danych***

**Opracował:** Radosław Relidzyński, **Grupa:** WCY23IX3S4

Spis treści

[Wstęp teoretyczny 3](#_Toc168962209)

[Treść zadania 4](#_Toc168962210)

[Środowisko 4](#_Toc168962211)

[Przygotowanie bazy danych 5](#_Toc168962212)

[Diagram klas 8](#_Toc168962213)

[Klasy i związki między nimi 8](#_Toc168962214)

[Klasy 8](#_Toc168962215)

[Związki między klasami 11](#_Toc168962216)

[Opis elementów służących do konstrukcji klas 12](#_Toc168962217)

[Metody 12](#_Toc168962218)

[Dziedziczenie 13](#_Toc168962219)

[Abstrakcja 13](#_Toc168962220)

[Polimorfizm 13](#_Toc168962221)

[Nowo zdefiniowany typ danych 14](#_Toc168962222)

[Wypełnienie bazy danymi 14](#_Toc168962223)

[Uruchomienie metod 15](#_Toc168962224)

[Podsumowanie 16](#_Toc168962225)

[Napotkane problemy 16](#_Toc168962226)

[Refeksja nad środowiskiem 17](#_Toc168962227)

# Wstęp teoretyczny

**Baza danych** – „uporządkowany zbiór danych określających wybrany fragment rzeczywistości lub problemu, które są przechowywane trwale w pamięci komputerowej do której może mieć dostęp wielu użytkowników w dowolnej chwili czasu.”

**System zarządzania bazami danych** – „zorganizowany zbiór narzędzi (programów komputerowych i bibliotek), które umożliwiają wykonanie podstawowych operacji na danych (CRUD) zawartych w jednej lub więcej bazach danych.”

System baz danych – jego definicja wyraża się wzorem:

Gdzie:

[źródło: materiały z wykładu „Temporalne bazy danych” dr inż. Jarosława Koszeli]

**Obiektowa baza danych** – „zbiór obiektów, których zachowanie się, stan oraz związki są określone zgodnie z obiektowym modelem danych. Obiektowy system zarządzania bazą danych jest systemem wspomagającym definiowanie, zarządzanie, utrzymywanie, zabezpieczanie i udostępnianie obiektowej bazy danych.”  
[źródło: <https://pl.wikipedia.org/wiki/Obiektowa_baza_danych>]

# Treść zadania

A black text on a white background

Description automatically generated

# Środowisko

**InterSystems Caché** – „komercyjny operacyjny system zarządzania bazami danych firmy InterSystems, używany do tworzenia aplikacji programowych dla zarządzania opieką zdrowotną, bankowością i usługami finansowymi, administracją rządową oraz innymi sektorami. Oprogramowanie klientów może korzystać z bazy danych za pomocą kodu obiektowego i SQL.”  
[źródło (tłumaczone): <https://en.wikipedia.org/wiki/InterSystems_Cach%C3%A9>]

Realizacja bazy danych zostanie utworzona przy pomocy IDE „Visual Studio Code” z zainstalowanymi rozszerzeniami z paczki „InterSystems ObjectScript Extension Pack”.

Źródło IDE: <https://code.visualstudio.com/>  
Źródło paczki InterSystems: <https://marketplace.visualstudio.com/items?itemName=intersystems-community.objectscript-pack>

Dodatkowo utworzony zostanie serwer bazodanowy „InterSystems IRIS” przy pomocy dockera wykonując poniższe polecenie z poziomu wsl:  
docker run -d --name my-iris -p 52773:52773 -p 1972:1972 intersystemsdc/iris-community:latest

fd2ca7e289abd0a1448b4e83453a7b0719aca515f15dd83a32276905bd5aef1d

Serwer dostępny będzie lokalnie pod takim linkiem:  
<http://localhost:52773/csp/sys/UtilHome.csp>

Widok strony serwera po pomyślnym zalogowaniu się:

A screenshot of a computer

Description automatically generated

# Przygotowanie bazy danych

Kompilowanie klas

A screen shot of a computer

Description automatically generated

Wypełnianie bazy danymi

A screen shot of a computer program

Description automatically generated

# Diagram klas

A screenshot of a computer

Description automatically generated

# Klasy i związki między nimi

### Klasy

1. **Person (abstract)**

* Atrybuty:
  + FirstName: %String(MAXLEN = 20)
  + LastName: %String(MAXLEN = 20)
  + BirthDate: %DateTime
  + Gender: %String(MAXLEN = 1)

1. **Customer**

* Atrybuty:
  + CustomerNumber: %Integer
  + DiscountType: %String(MAXLEN = 20)
* Związki:
  + Tickets: Ticket (Cardinality = many, Inverse = Customer)
  + Discount: Discount (Cardinality = one, Inverse = Customers)

1. **Employee**

* Atrybuty:
  + EmployeeNumber: %Integer
* Związki:
  + Licenses: License (Cardinality = many, Inverse = Employee)
  + Roles: RoleInFlight (Cardinality = many, Inverse = Employee)

1. **Aircraft**

* Atrybuty:
  + AircraftNumber: %Integer
  + ProductionYear: %Integer
  + ModelNumber: %Integer
* Związki:
  + Model: Model (Cardinality = one, Inverse = Aircrafts)
  + Flights: Flight (Cardinality = many, Inverse = Aircraft)
  + Seats: Seat (Cardinality = many, Inverse = Aircraft)

1. **Airport**

* Atrybuty:
  + AirportNumber: %Integer
  + Name: %String(MAXLEN = 40)
  + Country: %String(MAXLEN = 20)
* Związki:
  + FlightsFrom: Flight (Cardinality = many, Inverse = StartAirportObj)
  + FlightsTo: Flight (Cardinality = many, Inverse = DestinationAirportObj)

1. **Discount**

* Atrybuty:
  + DiscountType: %String(MAXLEN = 20)
  + DiscountPercentage: %Integer(MAXVAL = 100, MINVAL = 0)
* Związki:
  + Customers: Customer (Cardinality = many, Inverse = Discount)

1. **Flight**

* Atrybuty:
  + FlightNumber: %Integer
  + FlightDate: %DateTime
  + FlightDurationMinutes: %Integer(MINVAL = 1)
  + StartAirport: %Integer
  + DestinationAirport: %Integer
  + AircraftNumber: %Integer
* Związki:
  + StartAirportObj: Airport (Cardinality = one, Inverse = FlightsFrom)
  + DestinationAirportObj: Airport (Cardinality = one, Inverse = FlightsTo)
  + Tickets: Ticket (Cardinality = many, Inverse = Flight)
  + Aircraft: Aircraft (Cardinality = one, Inverse = Flights)
  + Roles: RoleInFlight (Cardinality = many, Inverse = Flight)

1. **JobRole**

* Atrybuty:
  + VALUELIST: ",pilot,steward,mechanik"

1. **License**

* Atrybuty:
  + EmployeeNumber: %Integer
  + ModelNumber: %Integer
* Związki:
  + Employee: Employee (Cardinality = one, Inverse = Licenses)
  + Model: Model (Cardinality = one, Inverse = Licenses)

1. **Model**

* Atrybuty:
  + ModelNumber: %Integer
  + Name: %String(MAXLEN = 40)
* Związki:
  + Aircrafts: Aircraft (Cardinality = many, Inverse = Model)
  + Licenses: License (Cardinality = many, Inverse = Model)

1. **RoleInFlight**

* Atrybuty:
  + EmployeeNumber: %Integer
  + FlightNumber: %Integer
  + JobRole: JobRole
* Związki:
  + Employee: Employee (Cardinality = one, Inverse = Roles)
  + Flight: Flight (Cardinality = one, Inverse = Roles)

1. **Seat**

* Atrybuty:
  + SeatNumber: %Integer
  + Class: %String(MAXLEN = 20)
  + AircraftNumber: %Integer
* Związki:
  + Aircraft: Aircraft (Cardinality = one, Inverse = Seats)
  + Ticket: Ticket (Cardinality = one, Inverse = Seat)

1. **Ticket**

* Atrybuty:
  + TicketCode: %String(MAXLEN = 10)
  + Price: %Integer(MINVAL = 0)
  + SeatNumber: %Integer
  + FlightNumber: %Integer
  + AircraftNumber: %Integer
  + CustomerNumber: %Integer
* Związki:
  + Customer: Customer (Cardinality = one, Inverse = Tickets)
  + Flight: Flight (Cardinality = one, Inverse = Tickets)
  + Seat: Seat (Cardinality = one, Inverse = Ticket)

### Związki między klasami

1. Aircraft jest związany z Model (one-to-one), Flight (one-to-many) i Seat (one-to-many).
2. Airport jest związany z Flight jako lotnisko startowe (one-to-many) i docelowe (one-to-many).
3. Customer jest związany z Ticket (one-to-many) i Discount (one-to-one).
4. Discount jest związany z Customer (one-to-many).
5. Employee jest związany z License (one-to-many) i RoleInFlight (one-to-many).
6. Flight jest związany z Airport jako lotnisko startowe i docelowe (one-to-one), Ticket (one-to-many), Aircraft (one-to-one) i RoleInFlight (one-to-many).
7. License jest związany z Employee (one-to-one) i Model (one-to-one).
8. Model jest związany z Aircraft (one-to-many) i License (one-to-many).
9. RoleInFlight jest związany z Employee (one-to-one) i Flight (one-to-one).
10. Seat jest związany z Aircraft (one-to-one) i Ticket (one-to-one).
11. Ticket jest związany z Customer (one-to-one), Flight (one-to-one) i Seat (one-to-one).

# Opis elementów służących do konstrukcji klas

### Metody

Metoda służąca do rezerwowania biletu przez klienta:

Method BookAFlight(SeatNumber As %Integer, FlightNumber As %Integer, Price As %Integer, TicketCode As %String) As %Status

{

    Set NewTicket = ##class(EuropeAirports.Ticket).%New()

    Set NewTicket.SeatNumber = SeatNumber

    Set NewTicket.FlightNumber = FlightNumber

    Set NewTicket.Price = Price

    Set NewTicket.CustomerNumber = ..CustomerNumber

    Set NewTicket.TicketCode = TicketCode

    Set Flight = ##class(EuropeAirports.Flight).%OpenId(FlightNumber)

    If Flight = $$$NULLOREF {

        Return $$$ERROR($$$GeneralError, "Invalid FlightNumber")

    }

    Set NewTicket.AircraftNumber = Flight.AircraftNumber

    Set sc = NewTicket.%Save()

    If $$$ISERR(sc) {

        Return sc

    }

    Set sc = ..Tickets.Relate(NewTicket)

    If $$$ISERR(sc) {

        Return sc

    }

    Return $$$OK

}

Metoda służąca do zliczania przychodu z lotu:

Method calculateIncome() As %Integer

{

    Set total = 0

    For i=1:1:..Tickets.Count()

    {

        Set total = total + ..Tickets.GetAt(i).Price

    }

    Return total

}

### Dziedziczenie

Dziedziczenie pracownika po klasie człowieka:

Class EuropeAirports.Employee Extends EuropeAirports.Person

### Abstrakcja

Klasa abstrakcyjna człowiek:

Class EuropeAirports.Person Extends %Persistent [ Abstract ]

{

Property FirstName As %String(MAXLEN = 20) [ Required ];

Property LastName As %String(MAXLEN = 20) [ Required ];

Property BirthDate As %DateTime [ Required ];

Property Gender As %String(MAXLEN = 1) [ Required ];

Method IsAdult() As %Boolean

{

    Set ageInDays = $ZDATETIMEH($HOROLOG, 3) - $ZDATETIMEH(..BirthDate, 3)

    If ageInDays >= (18 \* 365) { Return 1 }

    Else { Return 0 }

}

Method GetDetails() As %String [ Abstract ]

{

}

…

### Polimorfizm

Nadpisywanie metody klasy nadrzecznej człowiek przez klase pracownik

Method GetDetails() As %String

{

    Quit ..CustomerNumber \_ " " \_ ..FirstName \_ " " \_ ..LastName \_ " " \_ ..Discount

}

### Nowo zdefiniowany typ danych

Służy do ograniczania możliwych opcji przy deklaracji rodzaju roli pracownika

Class EuropeAirports.JobRole Extends %Library.String

{

Parameter VALUELIST = ",pilot,steward,mechanik";

}

# Wypełnienie bazy danymi

Na przykładzie dodawania zniżek:

ClassMethod FillDiscounts() As %Status

{

    Set sc = $$$OK

    Try {

        Set DiscountsArray = ##class(%DynamicArray).%New()

        Do DiscountsArray.%Push({"DiscountType": "rodzinna", "DiscountPercentage": 37})

        Do DiscountsArray.%Push({"DiscountType": "student", "DiscountPercentage": 50})

        Do DiscountsArray.%Push({"DiscountType": "krwiodawca", "DiscountPercentage": 75})

        For i=0:1:DiscountsArray.%Size() -1 {

            Set DiscountDetails = DiscountsArray.%Get(i)

            Set NewDiscount = ##class(EuropeAirports.Discount).%New()

            Set NewDiscount.DiscountType = DiscountDetails.%Get("DiscountType")

            Set NewDiscount.DiscountPercentage = DiscountDetails.%Get("DiscountPercentage")

            Set sc = NewDiscount.%Save()

            If $$$ISERR(sc) {

                Write "Error saving NewDiscount: ", $System.Status.GetErrorText(sc),!

                Return sc

            }

        }

        Write "Discounts filled with data",!

    } Catch ex {

        Write "Error FillDiscounts: ", ex.DisplayString(),!

        Set sc = $$$ERROR($$$GeneralError, ex.DisplayString())

    }

    Quit sc

}

Ogólna klasa do dodawania wszystkich obiektów:

ClassMethod FillAllData() As %Status

{

    Set sc = $$$OK

    Try {

        Set sc = ..ClearData()

        Set sc = ..FillDiscounts()

        Set sc = ..FillCustomers()

        Set sc = ..FillAirports()

        Set sc = ..FillModels()

        Set sc = ..FillEmployees()

        Set sc = ..FillLicenses()

        Set sc = ..FillAircrafts()

        Set sc = ..FillFlights()

        Set sc = ..FillRolesInFlight()

        Set sc = ..FillSeats()

        Set sc = ..FillTickets()

        Write "FillAllData ended",!

    } Catch ex {

        Write "Error FillAllData: ", ex.DisplayString(),!

        Set sc = $$$ERROR($$$GeneralError, ex.DisplayString())

    }

    Quit sc

}

# Uruchomienie metod

Uruchomienie klasy sprawdzającej działanie 3 przykładowych metod:

Przykładowe sprawdzanie dla metody rezerwującej bilet

ClassMethod TestCustomerBookAFlight() As %Status

{

    Set sc = $$$OK

    Try {

        Set CustomerObj = ##class(EuropeAirports.Customer).%OpenId(41)

        If CustomerObj '= $$$NULLOREF {

            Set SeatNumber = 12

            Set FlightNumber = 10

            Set Price = 150

            Set TicketCode = "ABC123"

            Set sc = CustomerObj.BookAFlight(SeatNumber, FlightNumber, Price, TicketCode)

            If $$$ISERR(sc) {

                Write "Error booking flight: ", ##class(%SYSTEM.Status).GetErrorText(sc), !

            } Else {

                Write "Flight booked successfully", !

            }

        } Else {

            Write "Customer object not found", !

            Set sc = $$$ERROR($$$GeneralError, "Customer object not found")

        }

    } Catch ex {

        Write "Error TestCustomerBookAFlight: ", ex.DisplayString(), !

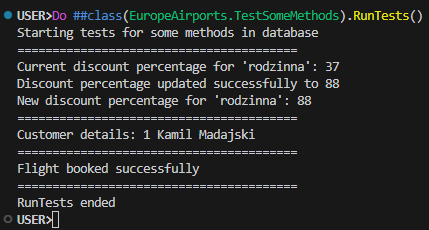
        Set sc = $$$ERROR($$$GeneralError, ex.DisplayString())

    }

    Quit sc

}

Wywołanie wszystkich klas sprawdzających:



# Podsumowanie

### Napotkane problemy

#### Identyfikatory dla klas dziedziczonych

Błąd pojawiał się w momencie zastosowania takiej deklaracji identyfikatora dla klasy dziedziczącej:

Property AircraftNumber As %Integer [ Identity, Required ];

Rozwiązaniem było dodanie tego identyfikatora w osobnej instrukcji:

Property EmployeeNumber As %Integer [ Required ];

Index EmployeeNumberIndex On EmployeeNumber [ Unique ];

#### Operacja quit wewnątrz obsługi błędów

Dodając quit wewnątrz catch kompilator podkreślał, że ta operacja jest niemożliwa, rozwiązaniem było stworzenie zmiennej, która w przypadku błędu była nadpisywana, tak jak tutaj:

ClassMethod FillAllData() As %Status

{

    Set sc = $$$OK

    Try {

        Set sc = ..ClearData()

        Set sc = ..FillDiscounts()

        Set sc = ..FillCustomers()

        Set sc = ..FillAirports()

        Set sc = ..FillModels()

        Set sc = ..FillEmployees()

        Set sc = ..FillLicenses()

        Set sc = ..FillAircrafts()

        Set sc = ..FillFlights()

        Set sc = ..FillRolesInFlight()

        Set sc = ..FillSeats()

        Set sc = ..FillTickets()

        Write "FillAllData ended",!

    } Catch ex {

        Write "Error FillAllData: ", ex.DisplayString(),!

        Set sc = $$$ERROR($$$GeneralError, ex.DisplayString())

    }

    Quit sc

}

#### Relacja jeden do jeden

W przypadku zastosowania po obu stronach relacji „Cardinality = one” wyskakiwał błąd:

ERROR #5494: Inverse cardinality, 'one' is not valid, 'EuropeAirports.Seat:Ticket'

Rozwiązaniem było zastosowanie relacji jeden do wielu z ograniczeniem unikalności:

Relationship Seat As EuropeAirports.Seat [ Cardinality = many, Inverse = Ticket ];

Relationship Ticket As EuropeAirports.Ticket [ Cardinality = one, Inverse = Seat ];

Index TicketAIndex On Ticket [ Unique ];

### Refeksja nad środowiskiem

Praca z zadanym środowiskiem była dość nowym doświadczeniem ze względu na połączenie mechanizmów, które podczas dotychczasowych projektów były realizowane przez niezależne technologie. Mimo, że wszystkie były znajome, zarówno te wynikające z obiektowości bazy danych, jak i te wynikające z samego faktu bycia bazą danych, mogącą wyświetlać wartości w tabeli i posiadająca relacje między nimi, tak działanie z nimi połączonymi okazało się być czymś zupełnie nowym.

„Caché InterSystems” to środowisko pozwalające na integrację obiektowych i relacyjnych sposobów przechowywania danych. To pozwala na bardziej naturalne i intuicyjne modelowanie rzeczywistości, dzięki temu tworząc bazę danych można dokonać szerszego przekładu rzeczywistości na bazę danych. Jest ona również elastyczna, można ją stale rozbudowywać, rozszerzając o kolejne atrybuty klas lub kolejne metody, co może zastąpić w dużym stopniu standardowy backend.