Wojskowa Akademia Techniczna

im. Jarosława Dąbrowskiego
w Warszawie

Systemy Baz Danych

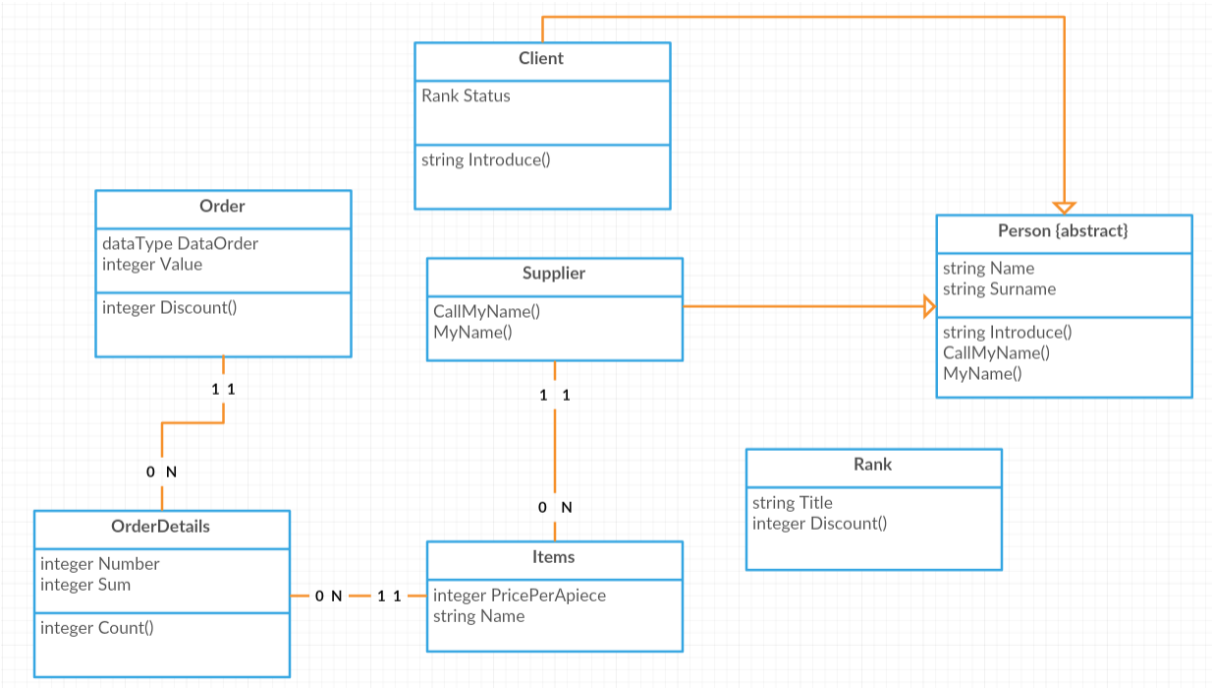
Obiektowe Bazy Danych

Autor: plut. pchor. Joanna Boratyn
kpr. pchor. Rafał Godlewski

Grupa: I7B1S4

Prowadzący: mgr inż. Maciej Szymczyk

1. Model bazy danych

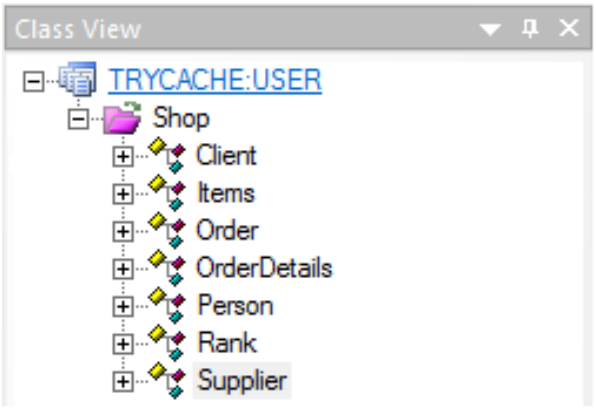


Rysunek 1. Diagram klas obiektowej bazy danych.

Klasa	Opis
Person	klasa abstrakcyjna, z której dziedziczą klasy *Client* oraz *Supplier*
Items	Klasa *Items* opisuje towar jakim są przybory szkolne. Posiada atrybuty: cena za sztukę (*PricePerApiece*), nazwa (*Name*). Klasa ta jest powiązana z klasą *Supplier*.
Order	reprezentuje zamówienie, które jest zamawiane przez klienta (*Client*). Klient może posiadać wiele zamówień, ale zamówienie może być przypisane do jednego klienta.
OrderDetalis	zawiera szczegóły opisujące zamówienie (*Order*). Zamówienie może być połączone z wieloma obiektami klasy *OrderDetails*. Obiekty klasy szczegóły mogą być powiązane jedynie z jednym obiektem klasy *Order*.

2. Struktura projektu i poszczególne klasy

Przy pomocy narzędzie CACHÈ powstała poniższa struktura projektu. Utworzyliśmy nowy projekt o nazwie , a następnie utworzyliśmy poszczególne klasy. Każda klasa posiada swoje atrybuty. Zaimplementowaliśmy również metody i powiązania między klasami.



Rysunek 2. Struktura projektu.

Klasa Person

```
Class Shop.Person [ Abstract ]
{

  Property Surname As %String(MINLEN = 2) [ Required ];

  Property Name As %String(MINLEN = 2) [ Required ];

  Method Introduce()
    {
                Write "Person",!
    }

  ClassMethod CallMyName()
    {
                Do ..MyName()
    }

  ClassMethod MyName()
   {
                Write "Person",!
   }

}
```

Klasa Client

```
Class Shop.Client Extends (%Persistent, %Populate, %XML.Adaptor, %ZEN.DataModel.Adaptor,
Shop.Person)
 {

     Property Status As Rank [ Required ];

     Relationship CliOrd As Shop.Order [ Cardinality = many, Inverse = OrdCli ];

     Method Introduce()
      {
              set who="Client"
              Write who,!
      }

}
```

Klasa Items

```
Class Shop.Items Extends (%Persistent, %Populate, %XML.Adaptor, %ZEN.DataModel.Adaptor)
  {

     Property PricePerApiece As %Integer [ Required ];

     Property Name As %String [ Required ];

   Relationship ItemSup As Shop.Supplier [ Cardinality = one, Inverse = SupItem ];

   Index ItemSupIndex On ItemSup;

   Relationship ItemOrd As Shop.OrderDetails [ Cardinality = many, Inverse = OrdDetItem ];

 }
```

Klasa Order

```
Class Shop.Order Extends (%Persistent, %Populate, %XML.Adaptor, %ZEN.DataModel.Adaptor)
 {

  Property DataOrder As %DataType [ Required ];

  Property Value As %Integer;

  Relationship OrdItem As Shop.OrderDetails [ Cardinality = many, Inverse = OrdDetOrd ];

  Relationship OrdCli As Shop.Client [ Cardinality = one, Inverse = CliOrd ];

  Index OrdCliIndex On OrdCli;

  Method Discount(id As %Integer) As %Integer [ Language = cache ]
    {
                Set client = ##class(Shop.Client).%OpenId(id)
                Set rank = client.Status.Discount
                Set dis = ..Value * (rank/100)
                Set value = ..Value - dis
                Set ..Value = value
                return dis
    }

}
```

Klasa OrderDetails

```
Class Shop.OrderDetails Extends (%Persistent, %Populate, %XML.Adaptor,
%ZEN.DataModel.Adaptor)
 {

   Property Sum As %Integer [ Required ];

   Property Number As %Integer [ Required ];

   Relationship OrdDetItem As Shop.Items [ Cardinality = one, Inverse = ItemOrd ];

   Index OrdDetItemIndex On OrdDetItem;

   Relationship OrdDetOrd As Shop.Order [ Cardinality = one, Inverse = OrdItem ];

   Index OrdDetOrdIndex On OrdDetOrd;
```

```
  Method Count(idItem As %Integer, idOrd As %Integer) As %Integer
    {
                Set item = ##class(Items).%OpenId(idItem)
                Set ord = ##class(Order).%OpenId(idOrd)
                Set price = item.PricePerApiece
                Set sum = ..Number * price
                Set ..Sum = sum
                Set value = ord.Value
                Set ord.Value = value + sum
                Return sum
   }

 }
```

Klasa Supplier

```
Class Shop.Supplier Extends (%Persistent, %Populate, %XML.Adaptor, %ZEN.DataModel.Adaptor,
Shop.Person)
 {

  Relationship SupItem As Shop.Items [ Cardinality = many, Inverse = ItemSup ];

  ClassMethod CallMyName()
    {
                Do ..MyName()
    }

 ClassMethod MyName()
   {
                Write "Person",!
   }

}
```

Klasa Rank

```
Class Shop.Rank Extends (%Persistent, %Populate, %XML.Adaptor, %ZEN.DataModel.Adaptor)
 {

   Property Title As %String [ Required ];

   Property Discount As %Integer [ Required ];

 }
```

3. Wypełnienie bazy danych testowymi danymi

Aby wprowadzić dane do tabel klas należy skorzystać z terminala. Po zalogowaniu w celu wprowadzenia utworzenia obiektu korzystamy z formuły:

SET zmienna =##class(projekt.nazwaklasy).%New()

Aby wypełnić pola w tabeli należy podać formułę:

SET zmienna.atrybut = wartość

Aby zapisać dany obiekt wpisujemy w terminalu:

SET sc = zmienna.%Save()

Aby wyświetlić tabele należy przejść do trybu SQL wpisując komendę:

Do $SYSTEM.SQL.Shell()

Następnie prostym zapytaniem SQL możemy wyświetlić tabele.

Poniżej zaprezentowano 3 przykładowe tabele z danymi

```
USER>>select * from Shop.Rank
4.      select * from Shop.Rank

ID      Discount        Title
1       5       Bronze
2       10      Silver
3       20      Gold

3 Rows(s) Affected
statement prepare time(s)/globals/lines/disk: 0.0004s/22/579/0ms
          execute time(s)/globals/lines/disk: 0.0003s/4/827/0ms
                          cached query class: %sqlcq.USER.cls14
```

```
USER>>select * from Shop.Items
2.      select * from Shop.Items

ID      ItemSup Name    PricePerApiece
1               ołówek  2
2               długopis        5
3               linijka 10
4               cyrkiel 15
5               plecak  100

5 Rows(s) Affected
statement prepare time(s)/globals/lines/disk: 0.0004s/22/584/0ms
          execute time(s)/globals/lines/disk: 0.0003s/6/1141/0ms
                          cached query class: %sqlcq.USER.cls13



USER>Do $SYSTEM.SQL.Shell()
SQL Command Line Shell
----------------------------------------------------

The command prefix is currently set to: <<nothing>>.
Enter q to quit, ? for help.
USER>>select * from Shop.Client
1.      select * from Shop.Client

ID      Name    Status  Surname
1       Wojtek  1       Kowalczyk

1 Rows(s) Affected
statement prepare time(s)/globals/lines/disk: 0.0020s/31/584/0ms
          execute time(s)/globals/lines/disk: 0.0003s/14/781/0ms
                          cached query class: %sqlcq.USER.cls4
```

4. Wykonanie metod obiektów w terminalu

Klasy Client i Supplier dziedziczą po klasie abstrakcyjnej Person. Dziedziczą między innymi metodę Introduce() która zwraca na terminalu nazwę klasy.

```
Method Introduce()
  {
            Write "Person",!
  }
```

```
Username: Admin
Password: *****
USER>set x = ##class(Shop.Client).%OpenId(1)

USER>do ##class(Shop.Client)x.Introduce()
Client
```

Metoda Discount() oblicza zniżkę która należy się dla klienta, z powodu przynależności do jakiejś grupy rabatowej. Argumentem metody jest identyfikator obiektu Client. Na podstawie atrybutów klienta wiemy jaką ma zniżkę i dokonujemy obliczeń. Aktualizujemy wartość zamówienia po uwzględnieniu zniżki. Zwracana jest kwota rabatu.

```
Method Discount(id As %Integer) As %Integer [ Language = cache ]
  {
            Set client = ##class(Shop.Client).%OpenId(id)
            Set rank = client.Status.Discount
            Set dis = ..Value * (rank/100)
            Set value = ..Value - dis
            Set ..Value = value
            return dis
  }
```

```
USER>set ord =##class(Shop.Order).%OpenId(1)

USER>set x=ord.Discount(1)

USER>zwrite x
x=9.85
```

Metoda Count() liczy kwotę na jaką opiewa zamówienie na jedną z pozycji zamówienia w określonej w zamówieniu ilości sztuk. Aktualizuje kwotę całego zamówienia. Metoda szuka obiektów Items i Orders, następnie na podstawie atrybutu PricePerApiece oblicza wartość zamówienia poprzez pomnożenie ilości razy cenę za sztukę.

```
Method Count(idItem As %Integer, idOrd As %Integer) As %Integer
  {
        Set item = ##class(Items).%OpenId(idItem)
        Set ord = ##class(Order).%OpenId(idOrd)
        Set price = item.PricePerApiece
        Set sum = ..Number * price
        Set ..Sum = sum
        Set value = ord.Value
        Set ord.Value = value + sum
        Return sum
  }
```

```
USER>set det =##class(Shop.OrderDetails).%OpenId(1)

USER>set mes =det.Count(1,1)

USER>write mes
190
USER>
```

5. Wnioski

W czasie realizacji projektu musieliśmy poznać nowe środowisko oraz składnie języka, które przyniosły nam najwięcej problemów. Środowisko wymaga poświęcenia sporej ilości czasu tak aby użytkownik mógł się w nim poruszać w stopniu pozwalającym na wykonanie zadania. Obiektowe bazy danych nie są tak rozpowszechnione więc mieliśmy mniejszy dostęp do informacji o nich. Błędy z którymi spotkaliśmy się w czasie realizacji projekty były niezrozumiałe i nieczytelne dla użytkownika. Kolejnym problemem była niekompletna dokumentacja, która nie pomagała w rozwiązaniu występujących problemów. Uważamy, że dokumentacja powinna zostać poprawiona, tak aby stała się bardziej intuicyjna dla użytkownika.

Jak środowisko realizuje zagadnienia związane z:	Odpowiedź
Obiektowością - Metodami	dostatecznie, opcje pracy w różnych językach
Obiektowością – Dziedziczeniem, typami abstrakcyjnymi	dobrze, nie wystąpiły znaczące problemy
Obiektowością – Związkami między klasami (asocjacja, kompozycja)	dobrze, nie wystąpiły znaczące problemy
Obiektowością – Typy danych – proste, złożone	dobrze, typy danych złożone i proste definiują klasy
Obiektowością - Polimorfizmem	dobrze, nie wystąpiły znaczące problemy
Obiektowością – Tożsamością danych	Obiekty identyfikowane są za pomocą systemowego identyfikatora OID
Obiektowością – Enkapsulacją	bezpośredni dostęp do atrybutów, metody zdefiniowane w jego klasie
Obiektowością – Trwałością danych	np. persistence. Obiekty w ciągu cyklu życia mogą być przenoszone między trwałym i nietrwałym obszarem składowania
Administracja – Zarządzanie środowiskiem	dostatecznie - niekompletna, nieintuicyjna dokumentacja
Interfejs – Czy narzędzie posiada API? Dla jakich języków?	Tak, Java, C++, SQL
Środowisko – Czy narzędzie zawiera w sobie środowisko programistyczne?	tak zawiera – „Studio”
Skalowalność – Czy narzędzie umożliwia horyzontalne skalowanie środowiska? (rozproszone przetwarzanie, magazynowanie, replikacja)	Magazynowania i replikacja są możliwe o rozproszonym przetwarzaniu nie znaleźliśmy informacji
Multi-model – Czy narzędzie zapewnia inne rodzaje bazy danych?	tak, relacyjne bazy danych