Projekt Podstawy Baz Danych Konferencje

Tomasz Kolbusz

Rafał Lisak

Dokumentacja

1.Lista użytkowników systemu i funkcji, które mogą realizować w systemie:

1.1 Klient firmy:

- Usuwanie zarejestrowanych uczestników
- Rejestrowanie uczestników konferencji
- Rezerwowanie ustalonej liczby miejsc na warsztaty
- Rezerwowanie ustalonej liczby miejsc na konferencje lub poszczególne dni w wypadku konferencji kilkudniowych
- Uzupełnianie danych zarejestrowanych uczestników
- Podgląd nadchodzących konferencji na które zarejestrowaliśmy uczestników oraz listy ich uczestników
- Dostęp do raportów finansowych konferencji dotyczące zarejestrowanych przez klienta uczestników
- Podgląd terminów konferencji w danym miesiącu

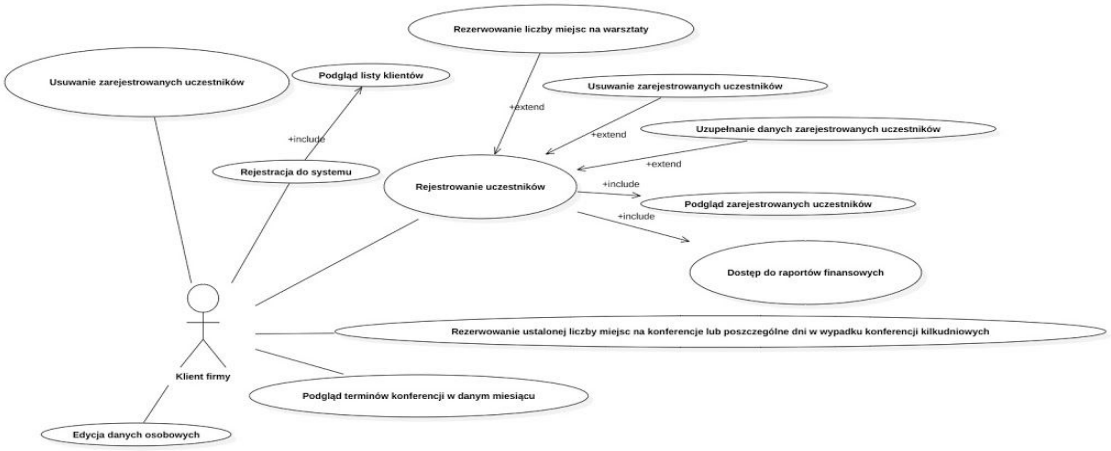
1.2 Uczestnik konferencji:

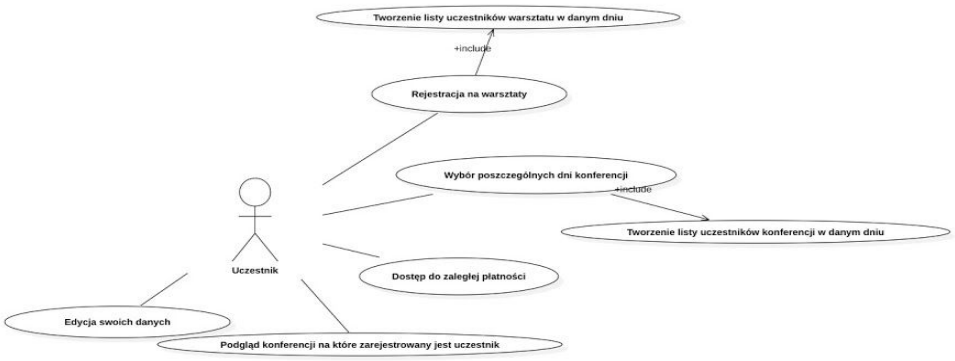
- Rejestracja na warsztaty, pod warunkiem, że uczestnik jest zarejestrowany na daną konferencję, nie jest zarejestrowany na żaden inny warsztat, który odbywa się w tym samym czasie oraz są wolne miejsca.
- Wybór poszczególnych dni konferencji
- Dostęp do należności, które zalega uczestnik
- Podgląd konferencji na, które zarejestrowany jest uczestnik

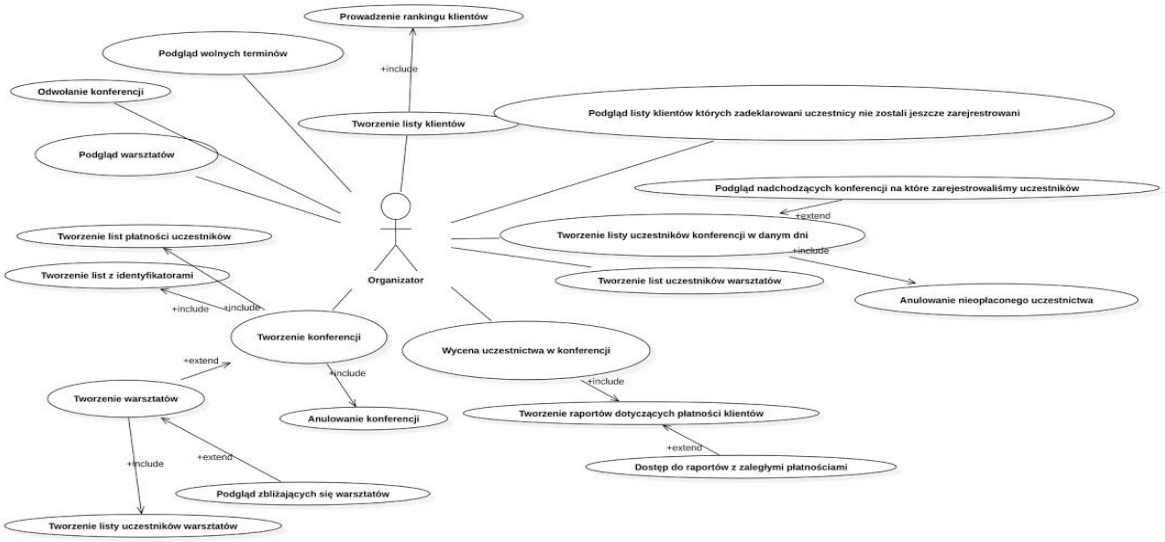
1.3 Organizator:

- Podgląd listy klientów, których zadeklarowani uczestnicy nie zostali jeszcze zarejestrowaniu.
- Tworzenie list uczestników konferencji w danym dniu
- Tworzenie list uczestników warsztatów
- Spisywanie raportu dotyczących płatności klientów
- Wycenienie udziału w konferencji w zależności od terminu rezerwacji oraz zarezerwowanych usług
- Anulowanie nieopłaconych rezerwacji po upływie wyznaczonego terminu
- Prowadzenie rankingu klientów zależnego od ilości zorganizowanych konferencji
- Tworzenie warsztatów
- Tworzenie konferencji jedno- lub kilkudniowych.
- Tworzenie list z identyfikatorami uczestników danej konferencji
- Tworzenie listy płatności uczestników danej konferencji
- Podgląd zbliżających się warsztatów
- Podgląd wolnych terminów

1.4 Diagram UseCase

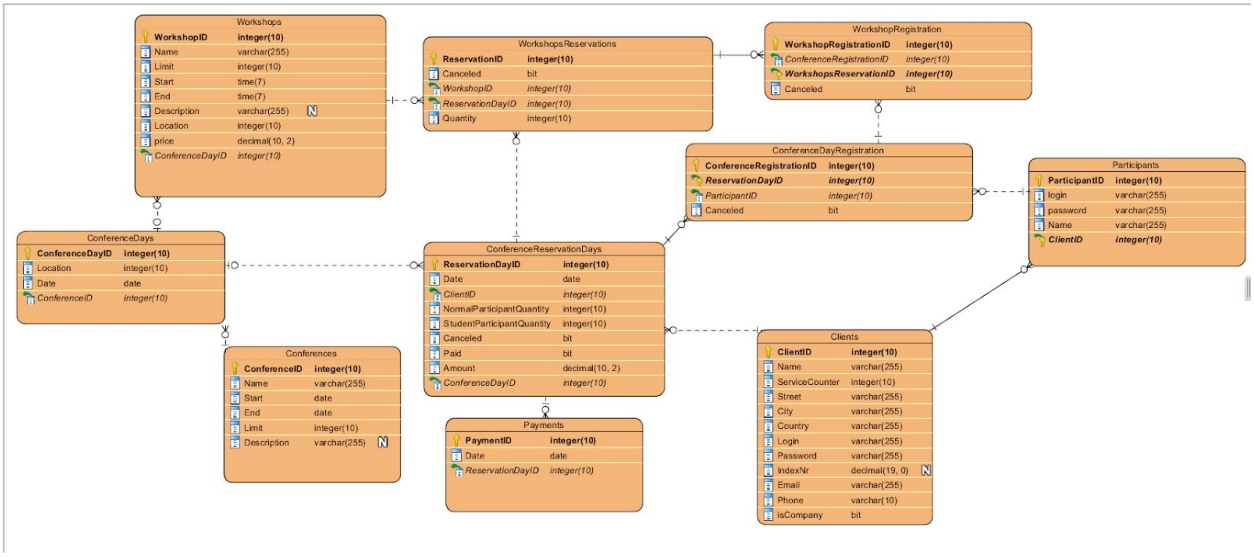






2 Baza danych

2.1 Diagram ERD



3. Tabele

Clients - tabela przechowująca dane dotyczące klientów takie jak dane kontaktowe, adresy, nazwa (nazwa firmy lub imię i nazwisko klienta), licznik ilości wykorzystanych usług, login i hasło - potrzebny do autentykacji internetowej, a także oznaczenie czy dany klient jest firmą czy osobą prywatną.

```
create table Clients (
  ClientID       int          not null primary key identity (1, 1),
  Name           varchar(255) not null,
  ServiceCounter int          not null             default 0 check (ServiceCounter > 0),
  Street         varchar(255) not null,
  City           varchar(255) not null,
  Country        varchar(255)                      default 'Polska',
  Login          varchar(255) not null,
  Password       varchar(255) not null,
  Email          varchar(255) not null check (Email like '%@%'),
  Phone          varchar(20) check (Phone not like '%[A-Za-z]%'),
  isCompany      bit                               default 0
)
go

create nonclustered index ClientsName_index
  on Clients (Name)
go
```

Participants - tabela zawiera wszystkich uczestników konferencji. Każdy klient zakłada przez siebie zarejestrowanym uczestnikom konta dzięki którym mogą przez internet zapisywać się na wybrane przez siebie warsztaty. Name zawiera imię i nazwisko uczestnika.W tabeli można także wpisać numer indeksy w razie gdyby uczestnik był studentem.

```
create table Participants (
  ParticipantID int          not null primary key identity (1, 1),
  login         varchar(255) not null,
  password      varchar(255) not null,
  Name          varchar(255) not null,
  StudentIndex  decimal(20),
  ClientId      int          not null foreign key references Clients (ClientID)
)
go

create nonclustered index ClientIdFK_index
  on Participants (ClientId)
go
```

Conferences - tabela zawiera podstawowe informacje na temat konferencji organizowanych przez firmę. Nazwę konferencji, datę rozpoczęcia i zakończenia, maksymalną ilość miejsc, bazowy koszt oraz opis.

```
create table Conferences (
   ConferenceID int          not null primary key identity (1, 1),
   [Name]       varchar(255) not null,
   Start        datetime     not null,
   [End]        datetime     not null,
   Limit        int          not null check (Limit > 0),
   Description  varchar(400),
   Price        money        not null
)
go

create nonclustered index ConferencesClientsName_index
   on Conferences (Name)
go
create nonclustered index ConferencesSE_index
   on Conferences (Start, [End])
go


alter table Conferences
   with check add
   constraint [CK_ConferenceStart] check
   (Start < [End])
go
```

Conference Days - zawiera konkretne dni poszczególnych konferencji, czas trwania konferencji w poszczególnym dniu, jak i miejsca w których się odbędą.

```
create table ConferenceDays (
   ConferenceDayID int          not null primary key identity (1, 1),
   Location        varchar(255) not null,
   Date            date         not null,
   Start           time         not null,
   [End]           time         not null,
   ConferenceID    int          not null foreign key references [Conferences] (ConferenceID)
)
go

create nonclustered index ConferenceDaysSE_index
   on ConferenceDays (Start, [End])
go

create nonclustered index ConferenceDaysFK_index
   on ConferenceDays (ConferenceID)
go
```

Workshops - zawiera dane poszczególnych warsztatów. Nazwę, ilość miejsc, datę rozpoczęcia i zakończenia, opis, miejsce w którym się odbywają, a także cenę. Domyślnie warsztat jest darmowy. Cena warsztatu jest stała .

```
create table Workshops (
  WorkshopID      int          not null primary key identity (1, 1),
  Name            varchar(255) not null,
  Limit           int          not null check (Limit > 0),
  Start           time         not null,
  [End]           time         not null,
  Description     varchar(255),
  Location        varchar(50)  not null,
  price           money        not null            default 0,
  ConferenceDayID int          not null foreign key references ConferenceDays (ConferenceDayID)
)
go

create nonclustered index WorkshopsSE_index
  on Workshops (Start, [End])
go

create nonclustered index WorkshopsFK_index
  on Workshops (ConferenceDayID)
go

alter table Workshops
  with check add
  constraint [CK_WorkshopsStart] check
  (Start < [End])
go
```

Conference Reservation Days - tabela zawiera rezerwację składane przez klientów na poszczególne dni . Podczas składania rezerwacji klient musi podać ilość osób z podziałem na studentów i osoby którym zniżka nie przysługuje. Biorąc pod uwagę, że cena konferencji rośnie wraz z przybliżaniem się daty jej rozpoczęcia, cena ustalona podczas robienia rezerwacji zostaje w niej zapisana. Baza odnotowuje także czy dana rezerwacja nie została anulowana.

```
create table ConferenceReservationDays (
  ReservationDayID          int      not null primary key identity (1, 1),
  Date                      datetime not null,
  ClientID                  int      not null foreign key references Clients (ClientID),
  NormalParticipantQuantity int      not null,
  StudentParticipantQuantity int     not null,
  Canceled                  bit                          default 0,
  Paid                      bit                          default 0,
  Amount                    money    not null,
  ConferenceDayID           int      not null foreign key references ConferenceDays (ConferenceDayID)
)
go
create nonclustered index ConferenceReservationDaysDayIDFK_index
  on ConferenceReservationDays (ConferenceDayID)
go

create nonclustered index ConferenceReservationDaysClientFK_index
  on ConferenceReservationDays (ClientID)
go
```

Payments - tabela zawiera wpłaty za daną rezerwację.

```
create table Payments (
  PaymentID        int  not null primary key identity (1, 1),
  Date             date not null,
  ReservationDayID int  not null foreign key references ConferenceReservationDays (ReservationDayID)
)
go

create nonclustered index PaymentsFK_index
  on Payments (ReservationDayID)
go
```

Workshops Reservations - zawiera rezerwacje składane przez klientów na dane warsztaty. Klienci muszą podczas rezerwacji podać liczbę osób, które będą uczestniczyć w warsztacie.

```
create table WorkshopsReservations (
  ReservationID    int not null primary key identity (1, 1),
  Canceled         bit                    default 0,
  WorkshopID       int not null foreign key references Workshops (WorkshopID),
  ReservationDayID int not null foreign key references ConferenceReservationDays (ReservationDayID),
  Quantity         int not null
)
go

create nonclustered index WorkshopsReservationsWIDFK_index
  on WorkshopsReservations (WorkshopID)
go

create nonclustered index WorkshopsReservationsDIDFK_index
  on WorkshopsReservations (ReservationDayID)
go
```

Conference Day Registration - zawiera rejestrację poszczególnych uczestników na dni konferencji.

```
create table ConferenceDayRegistration (
  ConferenceRegistrationID int not null primary key identity (1, 1),
  ReservationDayID         int not null foreign key references ConferenceReservationDays (ReservationDayID),
  ParticipantID            int not null foreign key references Participants (ParticipantID),
  Canceled                 bit not null
)
go

create nonclustered index ConferenceDayRegistrationDIDFK_index
  on ConferenceDayRegistration (ReservationDayID)
go

create nonclustered index ConferenceDayRegistrationPIDFK_index
  on ConferenceDayRegistration (ParticipantID)
go
```

Workshop Registration - pozwala na rejestrację poszczególnym uczestnikom na wybrane przez siebie warsztaty.

```
create table WorkshopRegistration (
  WorkshopRegistrationID   int not null primary key identity (1, 1),
  ConferenceRegistrationID int not null foreign key references ConferenceDayRegistration (ConferenceRegistrationID),
  WorkshopsReservationID   int not null foreign key references WorkshopsReservations (ReservationID),
  Canceled                 bit not null
)
go

create nonclustered index WorkshopRegistrationCFK_index
  on WorkshopRegistration (ConferenceRegistrationID)
go

create nonclustered index WorkshopRegistrationRFK_index
  on WorkshopRegistration (WorkshopsReservationID)
go
```

4.Funkcje

F_AvailableWorkshopPlaces - funkcja obliczająca ilość wolnych miejsc na dany warsztat. Jako argumenty przyjmuje ID warsztatu.

```
create function F_AvailableWorkshopPlaces
  (
    @workshopID INT
  )
  returns int
as begin
  declare @limit int = (select Limit from Workshops w where w.WorkshopID = @workshopID)
  declare @taken int = (select sum(Quantity)
                        from WorkshopsReservations ws
                        where ws.WorkshopID = @workshopID
                          and ws.Canceled = 0)
  declare @output int = @limit - @taken
  if (@output is null)
    begin
      set @output = @limit
    end
  return @output
end
go
```

F_DayCost - funkcja przeznaczona do wyznaczania kosztu podanej rezerwacji na dany dzień konferencji. Jako argumenty przyjmuje ID rezerwacji na dzień konferencji.

```
create function F_DayCost
  (
    @ConferenceReservationDayID int
  )
  returns int
as begin
  declare @ConfDayID int = (select ConferenceDayID
                            from ConferenceReservationDays crd
                            where crd.ReservationDayID = @ConferenceReservationDayID)
  declare @ConfID int = (select cd.ConferenceID
                         from ConferenceDays cd
                         where cd.ConferenceDayID = @ConfDayID)
  declare @NQuantity int = (select NormalParticipantQuantity
                            from ConferenceReservationDays crd
                            where crd.ReservationDayID = @ConferenceReservationDayID)
  declare @SQuantity int = (select StudentParticipantQuantity
                            from ConferenceReservationDays crd
                            where crd.ReservationDayID = @ConferenceReservationDayID)
  declare @price int = (select c.Price from Conferences c where c.ConferenceID = @ConfID)
  declare @sPrice int = (select c.Price * 0.8 from Conferences c where c.ConferenceID = @ConfID)

  declare @cost int = @NQuantity * @price + @SQuantity * @sPrice * (4 - DateDiff(week, getdate(), (
                                                                                  select c.Start
                                                                                  from Conferences c
                                                                                  where ConferenceID = @ConfID)))

  return @cost
end

go
```

F_AvailableConferenceDayPlaces - funkcja obliczająca ilość wolnych miejsc na daną konferencję .Jako argumenty przyjmuje ID warsztatu.

```
create function F_AvailableConferenceDayPlaces
  (
    @confDayID int
  )
  returns int
as begin
  declare @limit int = (select limit
                        from Conferences c
                               inner join ConferenceDays cd on cd.ConferenceID = c.ConferenceID
                        where ConferenceDayID = @confDayID)

  declare @taken int = (select sum(NormalParticipantQuantity) + sum(StudentParticipantQuantity)
                        from ConferenceReservationDays
                        where Canceled = '0'
                          and ConferenceDayID = @confDayID)
  declare @output int = @limit - @taken
  if (@output is null)
    begin
      set @output = @limit
    end
  return @output
end

go
```

5.Widoki

V_ClientsConferencesWithFreeTerms - jest to podgląd przeznaczony dla klientów, który ma za zadanie prezentować zbliżające się konferencję na, które można się zarejestrować. Założyliśmy że informację o dostępnych konferencjach są udostępniane 3 tygodnie przed ich rozpoczęciem. Widok służy głównie do wyboru konferencji przez klienta. Podawana tu cena jest to cena którą klient musiałby zapłacić gdyby w tym momencie złożył rezerwację. Cena biletu za konferencji zwiększa się wraz z zbliżaniem się daty rozpoczęcia konferencji. Cena biletu studenckiego uwzględnia też zniżkę 20%.

```
create view V_ClientsConferencesWithFreeTerms
  as
    select c.Name                                                    [Nazwa konferencji],
           cd.Date                                                   [Data wydarzenia],
           dbo.F_AvailableConferenceDayPlaces(c.ConferenceID)      as [Wolne miejsca],
           cd.Location,
           c.Description                                             [Opis],
           convert(money,
                   c.Price * 0.8 * (4 - DateDiff(week, getdate(), c.Start)))as [Bilet studencki],
           convert(money, c.Price * (4 - DateDiff(week, getdate(), c.Start)))  [Bilet normalny]
    from ConferenceDays cd
           inner join Conferences c on c.ConferenceID = cd.ConferenceID
      where DATEDIFF(d, GETDATE(), c.Start) >= 0
      and DATEDIFF(d, GETDATE(), c.Start) <= 21
      and dbo.F_AvailableConferenceDayPlaces(c.ConferenceID) > 0
go
```

Dostęp:
-Client
-Admin

V_ClientsWorkshopWithFreeTerms - podobnie jak wyżej jest to widok przeznaczony dla klienta, który ma za zadanie prezentować zbliżające się warsztaty .Założenia dostępności zostały przyjęte na takich samych zadaniach co dostępności konferencji.Cena za warsztat jest stała.

```
create view V_ClientsWorkshopWithFreeTerms as
  select w.Name                                                    [Nazwa warsztatu],
         co.Name                                                   [Nazwa konferencji],
         cd.Date                                                   [Data konferencji],
         cd.Location                                               [Miejce],
         dbo.F_AvailableWorkshopPlaces(w.WorkshopID)                 [Wolne miejsca],
         w.Limit,
         IIF(w.price = 0, 'darmowa', convert(varchar, w.price)) as [Cena]
  from Workshops w
         inner join ConferenceDays cd on cd.ConferenceDayID = w.ConferenceDayID
         inner join Conferences co on co.ConferenceID = cd.ConferenceID
  where DATEDIFF(d, GETDATE(), cd.Date) >= 0
    and DATEDIFF(d, GETDATE(), cd.Date) <= 21
    and dbo.F_AvailableWorkshopPlaces(w.WorkshopID) > 0

go
```

Dostęp:
-Client
-Admin

V_OrgConferences - widok przeznaczony dla firmy, który pozwala na wgląd do poszczególnych dni konferencji zapisanych w systemie. Widok pozwala także na przeprowadzenie statystyki dotyczącej ilości osób zarezerwowanych na poszczególne dni.

```
create view V_OrgConferences
  as
    select c.Name                                                  [Nazwa konferencji],
           cd.Date                                                 [Dzien],
           c.Start                                                 [Poczatek wydarzenia],
           c.[End]                                                 [Koniec wydarzenia],
           cd.Location                                             [Miejsce],
           dbo.F_AvailableConferenceDayPlaces(cd.ConferenceDayID)  [Wolne miejsca],
           c.Limit                                                 [Limit osob],
           isNull((select sum(crd.StudentParticipantQuantity)
                   from ConferenceReservationDays crd
                   where crd.ConferenceDayID = cd.ConferenceDayID
                   group by ConferenceDayID), 0)                   [Ilosc studentow],
           isNull((select sum(crd.NormalParticipantQuantity)
                   from ConferenceReservationDays crd
                   where crd.ConferenceDayID = cd.ConferenceDayID
                   group by ConferenceDayID),
                  0)                                               [Ilosc normalnych uczestnikow],
           isNULL((select sum(crd.ConferenceDayID)
                   from ConferenceReservationDays crd
                   where crd.ConferenceDayID = cd.ConferenceDayID), 0) [ilosc rezerwacji],
           c.Price                                                 [Bazowa cena konferencji],
           c.Description                                           [Opis]
    from Conferences c
           inner join ConferenceDays cd on cd.ConferenceID = c.ConferenceID

go
```

Dostęp:
-Worker
-Admin

V_OrgNotRegistredParticipants - widok przeznaczony dla pracowników firmy zajmującymi się kontaktowaniem z klientami w razie nie zarejestrowania uczestników. Przedstawia listę klientów którzy nie zarejestrowali wszystkich swoich uczestników, liczbę zarejestrowanych uczestników, a także dane kontaktowe do klienta.

```
create view V_OrgNotRegistredParticipants
  as
    select c.Name,
           c.Phone,
           crd.NormalParticipantQuantity + crd.StudentParticipantQuantity -
           (select count(cdr.ParticipantID)
            from ConferenceDayRegistration cdr
            where cdr.ReservationDayID = crd.ReservationDayID) as [Ilosc_niezarejestrowanych],
           crd.Date                                            [Dzien konferencji],
           (select co.Name
            from Conferences co
                   inner join ConferenceDays cd on cd.ConferenceID = co.ConferenceID
            where cd.ConferenceDayID = crd.ConferenceDayID)     [Nazwa konferecji]
    from Clients c
           inner join ConferenceReservationDays crd on crd.ClientID = c.ClientID
    where crd.NormalParticipantQuantity + crd.StudentParticipantQuantity -
          (select count(cdr.ParticipantID)
           from ConferenceDayRegistration cdr
           where cdr.ReservationDayID = crd.ReservationDayID) > 0
go
```

Dostęp:
-Worker
-Admin

V_OrgTopClients - widok pozwalający na prowadzenie rankingu klientów którzy najczęściej korzystają z usług firmy.

```
create view V_OrgTopClients
  as
    select top 100 c.Name       [Nazwa Klienta],
                   c.ServiceCounter [Ilosc konferencji w ktorych uczestniczyl]
    from Clients c
    order by ServiceCounter desc
go
```

Dostęp:
-Worker
-Admin

V_OrgClientPayments - widok pozwalający na sporządzanie raportów dotyczących płatności poszczególnych klientów z rozbiciem na lata i miesiące wraz z danymi klientów.

```
create view V_OrgClientPayments
  as
    select c2.Name,
           x1.rok,
           x1.miesiac,
           x1.[Zaplacona kwota],
           c2.Street,
           c2.City,
           c2.Country,
           c2.Email,
           c2.Phone,
           c2.isCompany
    from Clients c2
           right join (select crd.ClientID,
                              year(crd.Date)as   rok,
                              MONTH(crd.Date) as miesiac,
                              sum(crd.Amount)    [Zaplacona kwota]
                       from ConferenceReservationDays crd
                       group by crd.ClientID, year(crd.Date), MONTH(crd.Date) with rollup) x1
             on c2.ClientID = x1.ClientID
go
```

Dostęp:
-Worker
-Admin

V_OrgClientDebts - lista klientów oraz ich zaległości uwzględniającą koszt korzystania z warsztatów i datą.

```
create view V_OrgClientDebts
  as
    select c.ClientID,
           c.Name,
           crd.ReservationDayID,
           (crd.Amount + isNull(ws.Quantity * w.price, 0)) as [Zaleglosci],
           crd.Date
    from Clients c
           inner join ConferenceReservationDays crd on crd.ClientID = c.ClientID
           left join WorkshopsReservations ws on ws.ReservationDayID = crd.ReservationDayID
           left join Workshops w on w.WorkshopID = ws.WorkshopID
    where crd.Paid = 0
      and crd.Canceled = 0
go
```

Dostęp:
-Worker
-Admin

V_OrgParticipantWorkshops - widok wyświetla uczestników , warsztaty na które są zapisani oraz informacje o nich.

```
create view V_OrgParticipantWorkshops
  as
    select p.ClientId,p.ParticipantID,wr.Canceled, p.Name , w.Name[Nazwa warsztatu], w.Start, w.[End], cd.Date
    from Participants p
           inner join ConferenceDayRegistration cdr on cdr.ParticipantID = p.ParticipantID
           left join WorkshopRegistration wr
             on wr.ConferenceRegistrationID = cdr.ConferenceRegistrationID
           inner join WorkshopsReservations wrs on wrs.ReservationID = wr.WorkshopsReservationID
           inner join Workshops w on w.WorkshopID = wrs.WorkshopID
           inner join ConferenceDays cd on cd.ConferenceDayID = w.ConferenceDayID

go
```

Dostęp:
-Worker
-Admin

6.Triggery

T_CancelingReservation - trigger anuluje wszystkie powiązane rezerwacje i rejestrację uczestników po anulowaniu głównej rezerwacji na dzień konferencji.

```
create trigger T_CancelingReservation
  on ConferenceReservationDays
  after update
as
  begin
    if UPDATE(Canceled)
      begin
        declare @reservationID Int = (select top 1 ReservationDayID from inserted)
        update WorkshopsReservations set Canceled = 1 where ReservationDayID = @reservationID

        update WorkshopRegistration
        set Canceled = 1
        where WorkshopsReservationID in (select top 1 ReservationID
                                         from WorkshopsReservations
                                         where ReservationDayID = @reservationID)

        update ConferenceDayRegistration set Canceled = 1 where ReservationDayID = @reservationID
      end
  end
go
```

T_WorkshopRegistration - trigger zapewnia, że żaden uczestnik nie zarejestruje się na warsztaty, które trwają w tym samym czasie.

```
create trigger T_WorkshopRegistration
  on WorkshopRegistration
  for insert
as
  begin
    declare @ConferenceRegistrationID int = (select ConferenceRegistrationID from inserted)
    declare @participantID int = (select ParticipantID
                                  from ConferenceDayRegistration
                                  where ConferenceRegistrationID =
                                        (select top 1 ConferenceRegistrationID from inserted))

    declare @WorkshopID int =
    (select top 1 w.WorkshopID
     from inserted i
            left join WorkshopsReservations wr on wr.ReservationID = i.WorkshopsReservationID
            left join Workshops w on w.WorkshopID = wr.WorkshopID)

    declare @Start time = (select Start from Workshops where WorkshopID = @WorkshopID)
    declare @End time = (select [End] from Workshops where WorkshopID = @WorkshopID)
    if ((select count(ParticipantID)
         from V_OrgParticipantWorkshops
         where (@Start between Start and [End])
            or (@End between Start and [End]) and ParticipantID = @participantID) > 1)
      begin

        rollback transaction;
        declare @msg nvarchar(200)
        set @msg =
        'Nie mozna rejestrowac sie na warsztaty ktore trwaja w tym samym czasie ConferenceRegistrationID = '
        + convert(nvarchar(100), @ConferenceRegistrationID);
        throw 52000, @msg, 1
      end
  end
go
```

T_PaymentsPaid - trigger, który oznacza rezerwacje jako opłaconą po zamieszczeniu opłaty i zwiększa licznik ilości wykorzystanych usług klienta o jeden.

```
create trigger T_PaymentsPaid
  on Payments
  for insert
as
  begin
    declare @ReservationDayID int = (select ReservationDayID from inserted)
    update ConferenceReservationDays set Paid = 1 where ReservationDayID = @ReservationDayID

    declare @CounterService int = (select c.ServiceCounter
                                   from inserted i
                                          left join ConferenceReservationDays cr
                                            on cr.ReservationDayID = i.ReservationDayID
                                          left join Clients c on c.ClientID = cr.ClientID)
    declare @ClientID int = (select cr.ClientID
                             from inserted i
                                    left join ConferenceReservationDays cr
                                      on cr.ReservationDayID = i.ReservationDayID)
    update Clients set ServiceCounter = (@CounterService + 1) where ClientID = @ClientID

  end
go
```

T_WorkshopLimit - trigger, który nie pozwala na zrobienie rezerwacji na warsztat kiedy nie ma na niego wystarczająco dużo miejsca.

```
create trigger T_WorkshopLimit
  on WorkshopsReservations
  for insert
as
  begin

    declare @WorkshopID int = (select WorkshopID from inserted where canceled = 0)

    if (dbo.F_AvailableWorkshopPlaces(@WorkshopID) < 0)
      begin
        rollback transaction;
        declare @msg nvarchar(200)
        set @msg = 'Przekroczony limit osob na warsztaty dla WorkshopID = ' +
                   convert(nvarchar(100), @WorkshopID);
        throw 53000, @msg, 1
      end
  end
go
```

T_ConferenceLimit - trigger, który nie pozwala na zrobienie rezerwacji na konferencję na, którą nie ma wystarczająco dużo miejsca.

```
create trigger T_ConferenceLimit
  on ConferenceReservationDays
  for insert
as
  begin

    declare @ConferenceDayID int = (select ConferenceDayID from inserted)
    if (dbo.F_AvailableConferenceDayPlaces(@ConferenceDayID) < 0)
      begin
        rollback transaction;
        declare @msg nvarchar(100)
        set @msg = 'Przekroczony limit osob na konferencje ConferenceDayID = ' +
                   convert(nvarchar(100), @ConferenceDayID);
        throw 52300, @msg, 1
      end
  end
go
```

T_ConferenceDayRegistrationQuantity - trigger, który nie pozwala na zarejestrowanie się większej ilości uczestników niż przewiduje to rezerwacja.Gdyby go nie było wszyscy uczestnicy zarejestrowani przez klienta mogliby się zarejestrować na dany dzień konferencji niezależnie od tego na ile osób została zrobiona rezerwacja.

```
create trigger T_ConferenceDayRegistrationQuantity
  on ConferenceDayRegistration
  for insert
as
  begin


    declare @ReservationDayID int = (select ReservationDayID from inserted)
    declare @limit int = (select NormalParticipantQuantity + StudentParticipantQuantity
                          from ConferenceReservationDays
                          where ReservationDayID = @ReservationDayID)
    if ((select count(ReservationDayID)
         from ConferenceDayRegistration
         where ReservationDayID = @ReservationDayID) > @limit)
      begin
        rollback transaction;
        declare @msg nvarchar(200)
        set @msg = 'Rezerwacja pelna ReservationDayId = ' +
                   convert(nvarchar(100), @ReservationDayID);
        throw 56000, @msg, 1
      end
  end
go
```

T_WorkshopRegistrationQuantity - trigger, który nie pozwala na przekroczenie ilości osób podanej na rezerwacji podczas rejestrowania się uczestników na warsztaty. Problem jest analogiczny dla wcześniejszego problemu z rezerwacjami na konferencje.

```
create trigger T_WorkshopRegistrationQuantity
  on WorkshopRegistration
  for insert
as
  begin

    declare @ReservationID int = (select WorkshopsReservationID from inserted)
    declare @limit int = (select Quantity
                          from WorkshopsReservations
                          where ReservationID = @ReservationID)
    if ((select count(WorkshopRegistrationID)
         from WorkshopRegistration
         where WorkshopsReservationID = @ReservationID) > @limit)
      begin
        rollback transaction;
        declare @msg nvarchar(200)
        set @msg = 'Rezerwacja pelna ReservationID = ' + convert(nvarchar(100), @ReservationID);
        throw 56000, @msg, 1
      end
  end
go
```

T_ConferenceDayAdd - trigger sprawdzający czy dodawany dzień konferencji mieści się w czasie trwania konferencji, a także czy nie jest dodany już dzień konferencji o takiej samej dacie.

```
create trigger T_ConferenceDayAdd
  on ConferenceDays
  for insert
as
  begin

    declare @DayDate date = (select Date from inserted)
    declare @DayStartTime time = (select Start from inserted)
    declare @DayEndTime time = (select [End] from inserted)
    declare @ConferenceID int = (select ConferenceID from inserted)
    declare @ConfStartDate date = (select cast(c.Start as date)
                                   from Conferences c
                                   where c.ConferenceID = @ConferenceID)
    declare @ConfEndDate date = (select cast(c.[End] as date)
                                 from Conferences c
                                 where c.ConferenceID = @ConferenceID)
    declare @ConfStartTime time = (select cast(c.Start as time)
                                   from Conferences c
                                   where c.ConferenceID = @ConferenceID)
    declare @ConfEndTime time = (select cast(c.[End] as time)
                                 from Conferences c
                                 where c.ConferenceID = @ConferenceID)

    declare @Msg nvarchar(200)
    if (@DayDate < @ConfStartDate or @DayDate > @ConfEndDate)
      begin
        rollback transaction;
        set @Msg = 'Data dnia konferencji nie pokrywa się z konferencja: DayDate: ' +
                   convert(nvarchar(100), @DayDate) + ' ConfStartDate: ' +
                   convert(nvarchar(100), @ConfStartDate) + ' ConfEndDate: ' +
                   convert(nvarchar(100), @ConfEndDate);
        throw 56002, @Msg, 1
      end
    if (@DayStartTime < @ConfStartTime or @DayEndTime > @ConfEndTime)
      begin
        rollback transaction;
        set @Msg = 'Godzina dnia konferencji nie pokrywa się z konferencja: DayStartTime: ' +
                   convert(nvarchar(100), @DayStartTime) + ' ConfStartTime: ' +
                   convert(nvarchar(100), @ConfStartTime) + ' DayEndTime: ' +
                   convert(nvarchar(100), @DayEndTime) + ' ConfEnd: ' +
                   convert(nvarchar(100), @ConfEndTime);
        throw 56003, @Msg, 1
      end

    if ((select Count(cd.Date) from ConferenceDays cd where cd.Date = @DayDate) > 1)
      begin
        rollback transaction;
        throw 56004, 'Dzien konferencji juz istnieje', 1
      end
  end
go
```

7.Procedury

P_GetConferenceCostForClient - procedura która zwraca konferencje za, które dany klient musiał zapłacić wraz z ceną tej konferencji, razem z podsumowaniem cen za wszystkie. Jako argument przyjmuje ID klienta.

```
create procedure P_GetConferenceCostForClient(
  @ClientID int
)
as
  begin
    select C.ConferenceID, isNull(sum(CRD.Amount), 0) as [Do zaplaty]
    from Conferences C
           join ConferenceDays CD on C.ConferenceID = CD.ConferenceID
           join ConferenceReservationDays CRD on CD.ConferenceDayID = CRD.ConferenceDayID
    where CRD.ClientID = @ClientID
    group by C.ConferenceID with rollup
    having (sum(CRD.Amount) > 0)
  end
go
```

Dostęp:
-Worker
-Admin

P_MoneyToPayForConference - procedura która zwraca sumę którą klient musi zapłacić za daną konferencję. Jako argument przyjmuje ID klienta i konferencji.

```
create procedure P_MoneyToPayForConference(
  @ClientID     int,
  @ConferenceID int)
as
  begin
    select isNULL(sum(CRD.Amount), 0)
    from Conferences C
           join ConferenceDays CD on C.ConferenceID = CD.ConferenceID
           join ConferenceReservationDays CRD on CD.ConferenceDayID = CRD.ConferenceDayID
    where C.ConferenceID = @ConferenceID
      and CRD.ClientID = @ClientID
      and paid = 0
  end
go
```

Dostęp:
-Worker
-Admin

P_ParticipantIdentificatorData - procedura która zwraca Imiona i Nazwiska uczestników danego dnia konferencji. Jako argument przyjmuje ID konferencji i dnia konferencji. Wykorzystywana głównie do tworzenia identyfikatorów dla uczestników konferencji.

```
create procedure P_ParticipantIdentificatorData(
  @ConferenceID    int,
  @ConferenceDayID int
) as
  begin
    select P.Name as [Imie Nazwisko uczestnika]
    from ConferenceDays CD
           join ConferenceReservationDays CRD on CD.ConferenceDayID = CRD.ConferenceDayID
           join ConferenceDayRegistration CDR on CRD.ReservationDayID = CDR.ReservationDayID
           join Participants P on CDR.ParticipantID = P.ParticipantID
    where CD.ConferenceDayID = @ConferenceDayID
      and CD.ConferenceID = @ConferenceID
  end
go
```

Dostęp:
-Worker
-Admin

P_ParticipantsForWorkshops - procedura która zwraca uczestników danego warsztatu. Jako argument przyjmuje ID warsztatu.

```
create procedure P_ParticipantsForWorkshops(
  @WorkshopID int
)
as
  begin
    select P.name
    from Participants P
           join ConferenceDayRegistration CDR on CDR.ParticipantID = P.ParticipantID
           join WorkshopRegistration R on CDR.ConferenceRegistrationID = R.ConferenceRegistrationID
           join WorkshopsReservations WR on R.WorkshopsReservationID = WR.ReservationID
    where WR.WorkshopID = @WorkshopID
  end
go
```

Dostęp:
-Worker
-Admin

P_GetFreeWorkshops - procedura która zwraca warsztaty na które są jeszcze wolne miejsca dla danej konferencji. Jako argument przyjmuje ID konferencji.

```
create procedure P_GetFreeWorkshops(
  @ConferenceID int
)
as
  begin
    select workshopid, Name
    from Workshops
           join ConferenceDays C on Workshops.ConferenceDayID = C.ConferenceDayID
    where C.ConferenceID = @ConferenceID
      and dbo.F_AvailableWorkshopPlaces(WorkshopID) > 0

  end
go
```

Dostęp:
-Worker
-Admin
-Clinet

P_CancelReservation - procedura anulowania rezerwacji na dany dzień konferencji. Jako argument przyjmuje ID dnia rezerwacji.

```
create procedure P_CancelReservation
    @ReservationDayID int
as
  begin
    update ConferenceReservationDays
    set Canceled = 1
    where ConferenceReservationDays.ReservationDayID = @ReservationDayID
  end
go
```

Dostęp:
-Worker
-Admin

P_ConferencesForParticipant - procedura wyświetlająca informację o konferencjach na, które zarejestrowany jest podany uczestnik.

```
create procedure P_ConferencesForParticipant
    @ParticipantId int
as
  begin
    select distinct C2.ConferenceID, C2.Name, C2.Start, C2.[End], C2.Description
    from ConferenceDayRegistration
           join ConferenceReservationDays on ConferenceDayRegistration.ReservationDayID =
                                             ConferenceReservationDays.ReservationDayID
           join ConferenceDays C on ConferenceReservationDays.ConferenceDayID = C.ConferenceDayID
           join Conferences C2 on C.ConferenceID = C2.ConferenceID
    where ParticipantID = @ParticipantId
  end
go
```

Dostęp:
-Worker
-Admin

P_Refresh - procedura która anuluje wszystkie nieopłacone rezerwację starsze niż jeden tydzień. Procedura powinna być uruchamiana każdego dnia.

```
create procedure P_Refresh
as
  begin
    update ConferenceReservationDays
    set Canceled = 1
    where ReservationDayID in
          (select ReservationDayID from V_OrgClientDebts where DATEDIFF(week, Date, GETDATE()) >= 1)
  end
go
```

Dostęp:
-Worker
-Admin

P_CreateClient - procedura do dodawania klienta.

```
create procedure P_CreateClient
    @Name    varchar(255), @ServiceCounter int, @Street varchar(255), @City varchar(255),
    @Country varchar(255), @Login varchar(255), @Password varchar(255), @Email varchar(255),
    @Phone   varchar(255), @isCompany bit
as begin
  insert into Clients (Name,
                       ServiceCounter,
                       Street,
                       City,
                       Country,
                       Login,
                       Password,
                       Email,
                       Phone,
                       isCompany)
  values (@Name,
          @ServiceCounter,
          @Street,
          @City,
          @Country,
          @Login,
          @Password,
          @Email,
          @Phone,
          @isCompany);
end
go
```

Dostęp:
-Worker
-Admin

P_CreateConference - procedura do dodawania konferencji

```
create procedure P_CreateConference
    @Name  varchar(255), @Start datetime, @End datetime, @Limit int, @Description varchar(255),
    @Price money
as begin
  insert into Conferences ([Name], Start, [End], Limit, Description, Price)
  values (@Name, @Start, @End, @Limit, @Description, @Price)
end go
```

Dostęp:
-Worker
-Admin

P_CreateConferenceDay - procedura do dodawania dnia konferencji

```
create procedure P_CreateConferenceDay(
  @Location varchar(255), @Date date, @Start time, @End time, @ConferenceId int
)
as begin
  insert into ConferenceDays (Location, Date, Start, [End], ConferenceId)
  values (@Location, @Date, @Start, @End, @ConferenceId)
end go
```

Dostęp:
-Worker
-Admin

P_CreateWorkshop - procedura do dodawania warsztatu

```
create procedure P_CreateWorkshop(
  @Name     varchar(255), @Limit int, @Start time, @End time, @Description varchar(255),
  @Location varchar(255), @Price money, @ConferenceDayID int
) as begin
  insert into Workshops (Name, Limit, Start, [End], Description, Location, price, ConferenceDayID)
  values (@Name, @Limit, @Start, @End, @Description, @Location, @Price, @ConferenceDayID)
end go
```

Dostęp:
-Worker
-Admin

P_CreateConferenceReservationsDays - procedura do tworzenia rezerwacji na dany dzień konferencji

```
create procedure P_CreateConferenceReservationDays(
  @Date datetime, @ClientID int, @NormalParticipantQuantity int, @StudentParticipantQuantity int,
  @Paid bit, @Amount money, @ConferenceDayID int
) as begin
  insert into ConferenceReservationDays (Date,
                                         ClientID,
                                         NormalParticipantQuantity,
                                         StudentParticipantQuantity,
                                         Paid,
                                         Amount,
                                         ConferenceDayID)
  values (@Date,
          @ClientID,
          @NormalParticipantQuantity,
          @StudentParticipantQuantity,
          @Paid,
          @Amount,
          @ConferenceDayID)
end
go
```

Dostęp:
-Worker
-Admin

P_CreateWorkshopReservation - procedura do tworzenia rezerwacji na warsztaty

```
create procedure P_CreateWorkshopReservation
    @Canceled int, @WorkshopID int, @ReservationDayID int, @Quantity int
as
  begin
    insert into WorkshopsReservations (Canceled, WorkshopID, ReservationDayID, Quantity)
    values (@Canceled, @WorkshopID, @ReservationDayID, @Quantity)
  end
go
```

Dostęp:
-Worker
-Admin

P_CreatePayment - procedura do tworzenia płatności

```
create procedure P_CreatePayment(@Date date, @ReservationDayID int)
as begin
  insert into Payments (Date, ReservationDayID) values (@Date, @ReservationDayID);
end
go
```

Dostęp:
-Worker
-Admin

P_CreateConferenceDayRegistration - procedura do tworzenia rejestracji uczestnika na dzień konferencji

```
create procedure P_CreateConferenceDayRegistration(
  @ReservationDayID int, @ParticipantID int, @Canceled bit
) as
  begin
    insert into ConferenceDayRegistration (ReservationDayID, ParticipantID, Canceled)
    values (@ReservationDayID, @ParticipantID, @Canceled)
  end
go
```

Dostęp:
-Worker
-Admin

P_CreateWorkshopRegistration - procedura do tworzenia rejestracji uczestnika na warsztaty

```
create procedure P_CreateWorkshopRegistration(@ConferenceRegistrationID int,
                                              @WorkshopsReservationID   int, @Canceled bit) as
  begin
    insert into WorkshopRegistration (ConferenceRegistrationID, WorkshopsReservationID, Canceled)
    values (@ConferenceRegistrationID, @WorkshopsReservationID, @Canceled);
  end
go
```

Dostęp:
-Worker
-Admin

P_RegisterToConferenceDaysInRange - procedura która tworzy rejestracje na kilka dni konferencji. Jako argument przyjmuje zakres dat, ID konferencji oraz dane potrzebne do rejestracji.

```
create procedure P_RegisterToConferenceDaysInRange
    @ConferenceId              int,
    @DateStart                 date,
    @DateEnd                   date,
    @ClientID                  int,
    @NormalParticipantQuantity  int,
    @StudentParticipantQuantity int,
    @Paid                      bit,
    @Amount                    money
as begin
  while (abs(datediff(day, @DateStart, @DateEnd)) != 0)
    begin
      declare @ConfDayID int = (select ConferenceDayID
                                from ConferenceDays
                                where DATEDIFF(day, ConferenceDays.Date, @DateStart) = 0
                                  and ConferenceID = @ConferenceId)
      set @DateStart = DATEADD(day, 1, @DateStart)
      if (@ConfDayID is not null)
        begin
          insert into ConferenceReservationDays (ClientID,
                                                 NormalParticipantQuantity,
                                                 StudentParticipantQuantity,
                                                 Paid,
                                                 Amount,
                                                 ConferenceDayID)
          values (@ClientID,
                  @NormalParticipantQuantity,
                  @StudentParticipantQuantity,
                  @Paid,
                  @Amount,
                  @ConfDayID)
        end
    end
end
go
```

Dostęp :
-Worker
-Admin

P_CreateParticipant - procedura do tworzenia uczestnika konferencji

```
create procedure P_CreateParticipant
    @Name         varchar(255),
    @Login        varchar(255),
    @Password     varchar(255),
    @StudentIndex int,
    @ClientId     int
as begin
  insert into Participants (login, password, Name, StudentIndex, ClientId)
  values (@Login, @Password, @Name, @StudentIndex, @ClientId)
end
go
```

Dostęp :
-Worker
-Admin

P_GetClientDebtByParticipant - procedura która zwraca zagległości jakie musi uiścić klient na podstawie uczestnika jego konferencji. Jako argument przyjmuje ID uczestnika konferencji.

```
create procedure P_GetClientDebtByParticipant(
  @ParticipantId int
)
as
  begin
    select isNULL(sum(debts.Zaleglosci), 0)
    from dbo.V_OrgClientDebts debts
           join ConferenceDayRegistration CDR on debts.ReservationDayID = CDR.ReservationDayID
    where CDR.ParticipantID = @ParticipantId
      and CDR.Canceled = 0
  end
go
```

Dostęp :
-Worker
-Admin

8.Role

Client - posiada dostęp do wszystkich widoków i procedur prezentujących ofertę firmy.
Participant - posiada dostęp do widoku na warsztaty z wolnymi miejscami ,aby mógł wybrać interesujący go warsztat.
Worker - pracownik firmy posiada uprawnienia do wszystkich procedur pozwalających na obsługę klienta a także widoków związanych z prowadzeniem firmy ,generowaniem raportów, oraz przygotowywaniem konferencji i warsztatów.
Admin - opiekuje się systemem ,więc posiada pełną władzę nad systemem.